

Db2 13 for z/OS

SQL Data Insights User Guide
Last updated: 2023-11-17



Notes

Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

Subsequent editions of this PDF will not be delivered in IBM Publications Center. Always download the latest edition from [IBM Documentation](#).

2023-11-17 edition

This edition applies to Db2[®] 13 for z/OS[®] (product number 5698-DB2[®]), Db2 13 for z/OS Value Unit Edition (product number 5698-DBV), and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© **Copyright International Business Machines Corporation 2022, 2023.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this information.....	V
Who should read this information.....	v
Terminology and citations.....	v
Accessibility features for Db2 for z/OS.....	vi
How to send comments.....	vi
 Chapter 1. Overview of SQL Data Insights.....	 1
 Chapter 2. Installing and configuring SQL DI (Roadmap).....	 5
Preparing SQL DI installation.....	6
Configuring system resources for SQL DI.....	8
Configuring network ports for SQL DI.....	8
Configuring setup user ID for SQL DI.....	9
Configuring a keyring-based keystore (JCERACFKS) for SQL DI.....	12
Configuring Db2 for SQL DI.....	13
Installing SQL DI.....	14
Verifying the installation and configuration of SQL DI.....	16
 Chapter 3. Upgrading SQL DI.....	 19
 Chapter 4. Enabling and running AI queries.....	 21
Creating a connection to Db2.....	21
Adding an AI object.....	22
Enabling AI query.....	22
Viewing an AI object model.....	24
Running an AI query.....	25
Analyzing data.....	26
 Chapter 5. Administering SQL DI.....	 29
Modifying your SQL DI settings.....	29
Creating a started task for the SQL DI application.....	30
Creating started tasks for the Spark cluster.....	31
 Chapter 6. Db2 tables for SQL DI.....	 33
 Chapter 7. Db2 subsystem parameter for SQL DI.....	 41
DSNTIP81: Performance and optimization panel 2.....	41
MAX AI DATA CACHING field (MXAIDTCACH subsystem parameter).....	41
 Chapter 8. Db2 built-in functions for SQL DI.....	 43
AI_ANALOGY.....	43
AI_COMMONALITY.....	45
AI_SIMILARITY.....	47
AI_SEMANTIC_CLUSTER.....	49
 Chapter 9. Db2 SQL statements for SQL DI.....	 51
CREATE FUNCTION (sourced).....	51
CREATE FUNCTION (inlined SQL scalar).....	62
CREATE FUNCTION (SQL table).....	71

CREATE INDEX.....	79
CREATE MASK.....	108
CREATE PERMISSION.....	117
CREATE TABLE.....	124
CREATE VIEW.....	193
DELETE.....	200
SET CURRENT TEMPORAL BUSINESS_TIME.....	216
SET CURRENT TEMPORAL SYSTEM_TIME.....	217
UPDATE.....	219
Chapter 10. Db2 queries for SQL DI.....	241
table-reference.....	241
Chapter 11. Db2 SQL codes for SQL DI.....	253
Information resources for Db2 for z/OS and related products.....	261
Notices.....	263
Trademarks.....	264
Terms and conditions for product documentation.....	264
Privacy policy considerations.....	265
Glossary.....	267
Index.....	269

About this information

Throughout this information, "Db2" means "Db2 13 for z/OS". References to other Db2 products use complete names or specific abbreviations.

Important: To find the most up to date content for Db2 13 for z/OS, always use [IBM® Documentation](#) or download the latest PDF file from [PDF format manuals for Db2 13 for z/OS \(Db2 for z/OS in IBM Documentation\)](#).

For more about how to use this information, see ["About this information" in the online product documentation](#).

Who should read this information

This information is for data scientists and data engineers who want to enable and run AI queries against Db2 data to extract hidden patterns and derive business insights.

This information is also for Db2 application architects and developers who are familiar with Structured Query Language (SQL) and who want to develop AI-based applications that can quickly analyze complex Db2 data for explainable insights in real time.

Terminology and citations

When referring to a Db2 product other than Db2 for z/OS, this information uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

Db2

Represents either the Db2 licensed program or a particular Db2 subsystem.

IBM OMEGAMON® for Db2 Performance Expert on z/OS

Refers to any of the following products:

- IBM IBM OMEGAMON for Db2 Performance Expert on z/OS
- IBM Db2 Performance Monitor on z/OS
- IBM Db2 Performance Expert for Multiplatforms and Workgroups
- IBM Db2 Buffer Pool Analyzer for z/OS

C, C++, and C language

Represent the C or C++ programming language.

CICS®

Represents CICS Transaction Server for z/OS.

IMS

Represents the IMS Database Manager or IMS Transaction Manager.

MVS™

Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®

Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for Db2 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including Db2 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: IBM Documentation (which includes information for Db2 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the Db2 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

See the *IBM Accessibility Center* at <http://www.ibm.com/able> for more information about the commitment that IBM has to accessibility.

How to send your comments about Db2 for z/OS documentation

Your feedback helps IBM to provide quality documentation.

Send any comments about Db2 for z/OS and related product documentation by email to db2zinfo@us.ibm.com.

To help us respond to your comment, include the following information in your email:

- The product name and version
- The address (URL) of the page, for comments about online documentation
- The book name and publication date, for comments about PDF manuals
- The topic or section title
- The specific text that you are commenting about and your comment

Chapter 1. Overview of SQL Data Insights

SQL Data Insights (SQL DI) is an AI-powered Db2 feature. It combines deep learning in artificial intelligence (AI) with advanced IBM Z technologies to infuse the Db2 engine with SQL-based semantic queries on user tables and views.

The existing data model in a relational database, SQL queries, text extensions, and user-defined functions are incapable of capturing the semantic relationships among value entities within and across columns or rows. SQL DI uses database embedding, a self-supervised learning approach in deep learning and AI, to train a neural network model and infer semantic meanings for the unique values in a relational table. The inferred meanings in the form of numeric vectors encapsulate the inter-column and intra-column data relationships. SQL DI then uses the trained model that consists of numeric vectors to run AI queries that discover, match, and cluster semantic similarities and dissimilarities in your Db2 data.

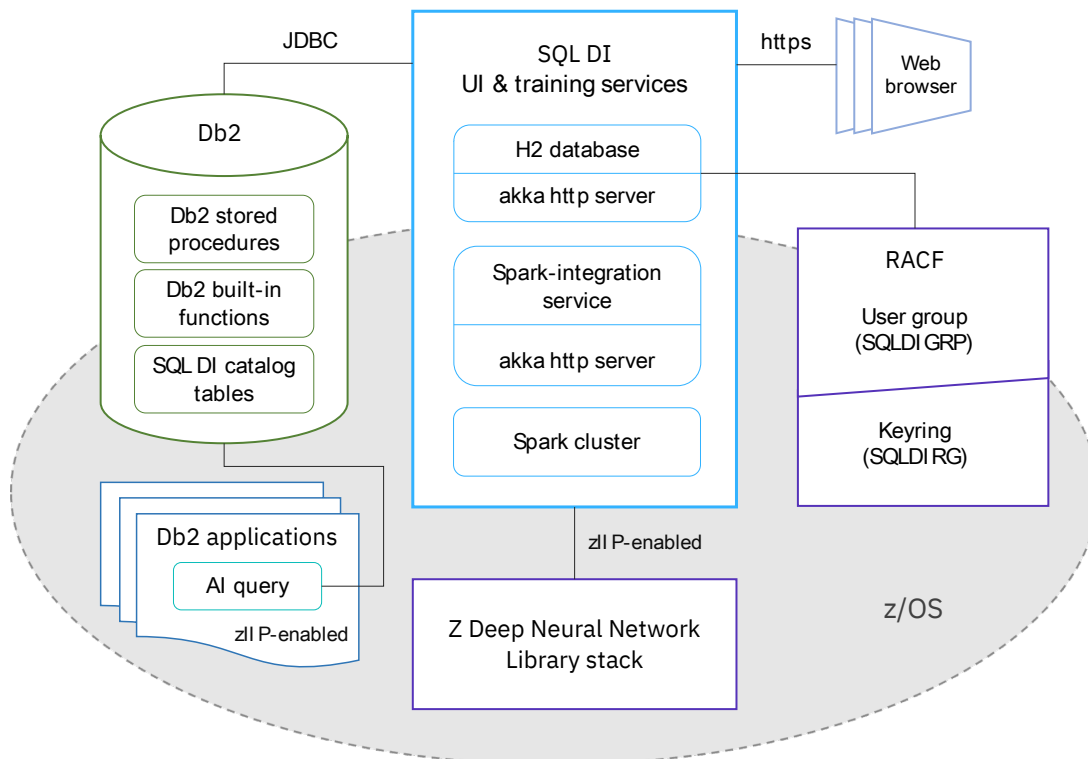


Figure 1. SQL DI architecture

SQL DI is seamlessly integrated into your Db2 for z/OS environment. The feature is comprised of an AI library stack, a data and query engine, and a web application. The AI library stack is provided by the Z Deep Neural Network Library component that resides natively in z/OS. The stack consists of the zDNN, zAIO, and zADE libraries, which enables SQL DI to take full advantage of IBM Z processors.

The data and query engine is built into Db2 for z/OS and provides services for processing data and semantic queries. The core of this processing engine is the set of AI_SIMILARITY, AI_SEMANTIC_CLUSTER, and AI_ANALOGY scalar functions. You can use these built-in functions in SQL statements to ask semantic questions about your data.

The SQL DI web application leverages the analytic framework of IBM Watson® Machine Learning for z/OS and the runtime engines of z/OS Spark. The application provides the primary user interface for you to create AI objects, enable AI queries, run AI queries, and visualize query results.

With SQL DI, you can quickly tap into the vast amount of mission-critical data in your Db2 and easily uncover the hidden information across tables and views for actionable insights. You can achieve all these

without the costly effort of moving massive data across platforms, procuring expensive AI infrastructures and acquiring advanced AI skills.

How does SQL DI work?

In the web user interface (UI), you connect SQL DI to your Db2, create an AI object from selected Db2 tables and views, enable the object for AI queries, and run queries on the object at any time.

To enable the object for AI queries, SQL DI preprocesses the data in the object, creates a neural network model on the data, and loads the model into Db2. For data preprocessing, SQL DI uses an embedded Spark cluster to convert Db2 columns selected for enablement as text, known as the AI object text data. After data transformation, SQL DI clusters numeric data type values and replace all numeric values with cluster identifiers. In text format, every numeric or categorical data type value in the data is tagged with its column name. In addition, every numeric data type value is associated a cluster identifier.

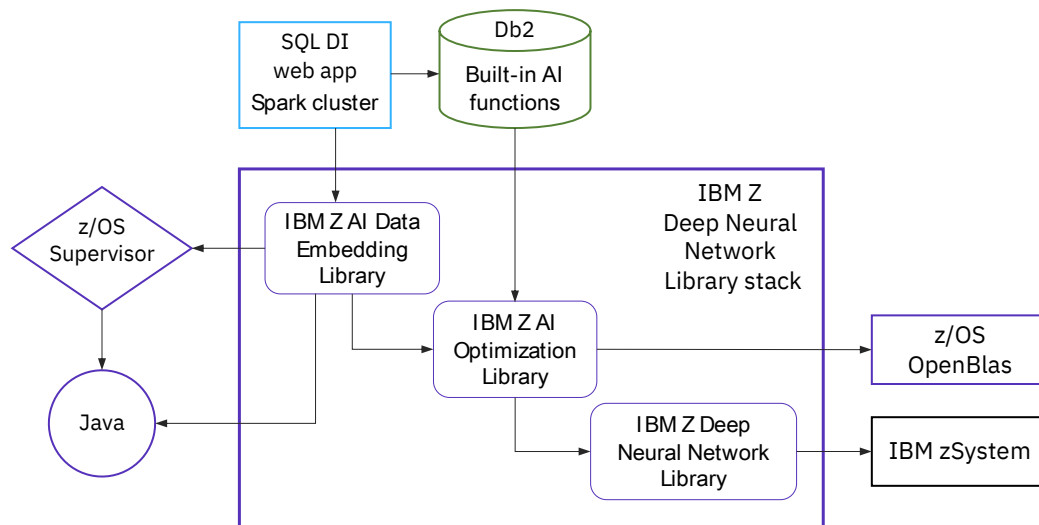


Figure 2. SQL DI AI query enablement

After the data is preprocessed, SQL DI uses z/OS AI capabilities provided by the zDNN stack to train the model with input from the AI object text data set. The training process generates numeric vectors for every unique value in the data set. The numeric vectors represent the inter-column and intra-column semantic relationships between different unique values, also referred as the vocabulary, in the object. SQL DI uses the Db2 LOAD utility to load the trained model into a Db2 table.

After successful enablement, you can run similarity, dissimilarity, clustering, or analogy queries on the object at any time. Simply select a query type, enter a SQL statement, and run the query. Depending on the query type that you select, SQL DI uses a corresponding Db2 AI function to process your query. It displays the first 50 rows of the query results in the UI and loads the remaining rows into the Db2 table. You can export the displayed rows from the SQL DI UI or download the entire result set from Db2.

During model training, SQL DI also collects key data statistics and renders them into column influence and discriminator scores for the object and the model. A column influence score correlates to the number of user-specified NULL values in a column and indicates the column's influence on the training of the object model. On the other hand, a discriminator score correlates to the number of unique values in a column and measures the column's ability to semantically distinguish its values from other values in the table. You can use the visualized scores to help you understand the results of your AI queries on the object.

SQL DI concepts and definitions

The following concepts and definitions can help you learn, use, and administer SQL DI:

AI object

A SQL DI asset that you can create to contain a Db2 user table or view. You can then enable the object for AI query in the SQL DI UI.

AI object text data

The data in an AI object that is transformed and formatted as text for AI query enablement. This text data is the input to model training. The number of tokens in the data are the values in every categorical and numeric column that are selected for model training for the AI object.

Vocabulary

The total number of unique values in the selected SQL DI categorical columns and the number of clusters associated with the selected numeric columns in the AI object for AI query enablement.

SQL DI data type

A SQL DI-specific data category descriptor that you can assign to a column in an AI object for column configuration and model training. You can assign the column one of the following SQL DI data types:

- *Categorical*: The SQL DI categorical data type is used for columns with discrete values, each of which is its own entity.
- *Numeric*: The SQL DI numeric data type is used for columns with continuous values.
- *Key*: The SQL DI key data type is used to indicate that a column represents an entire row.

See [“Enabling AI query” on page 22](#) for more information.

AI query

A semantic query that you can run on an AI object to infer hidden relationships between entities in the object. You can select from the following AI query types:

SQL DI AI query types and corresponding Db2 AI functions		
SQL DI query type	Description	Corresponding Db2 AI function
Semantic similarity	A similarity query identifies groups of similar records or entities in records.	AI_SIMILARITY
Semantic dissimilarity	A dissimilarity query finds the outliers from the norm in records.	AI_SIMILARITY
Semantic clustering	A clustering query forms a cluster of entities in records and evaluates whether or not an additional entity belongs in the cluster.	AI_SEMANTIC_CLUSTER
Semantic analogy	An analogy query determines if the relationship between two entities applies to that of a second pair of entities.	AI_ANALOGY
Semantic commonality	A commonality query identifies the entities in records that exhibit the most common or uncommon patterns.	AI_COMMONALITY

See [“Running an AI query” on page 25](#) for more information.

Vector prefetch

The advanced processes that SQL DI uses to upload the numeric vectors to the zAIO library for calculating the similarity scores. The vectors are generated from an AI object text data set. The existing query processing architecture dictates that Db2 AI functions submit the vectors, record-by-record, to z/OS for processing. To significantly improve the query performance, SQL DI implements vector prefetch, which enables Db2 to upload multiple vectors in a batch at a time.

ibm-data2vec

A z/OS native AI function that SQL DI uses for model training. The `ibm-data2vec` function is an implementation of a self-supervised database embedding algorithm. The database embedding takes as input a text file that is created from a multi-modal relational table and builds a relationship map

between text tokens by using the relational data model. The input training document generated from a relational table consists of string tokens that represent different relational entities in the original Db2 table or view. The `ibm-data2vec` function views the training document as a set of sentences, where each sentence represents a relational table row.

After model training is completed, `ibm-data2vec` generates a numeric vector of a pre-defined length (dimension) for each token, and the vector encodes the meaning of that token. The core numerical computations of the training process are paralleled by using multiple threads and accelerated by using hardware-accelerated numerical computations. The final trained model is stored as a binary file that uses the Db2 ZLOAD utility. See [ibm-data2vec](#) for more information.

base10Cluster

A z/OS numerical clustering algorithm that clusters together numerically closer items in different buckets, each of which represents a distinct cluster. SQL DI uses the `base10Cluster` algorithm to process an input data set and generate an output file that lists the number of buckets and their corresponding minimum values. See [base10Cluster](#) for more information.

Column influence score

A score that correlates to the number of user-specified and SQL NULL values in a column and indicates the column's influence on the training of the object model. The fewer NULL values the column has, the higher influence score it generates.

Column discriminator score

A score that correlates to the number of unique values in a column and measures the column's ability to semantically distinguish its values from other values in the table. The more unique values the column has, the higher discriminator score it generates.

Chapter 2. Installing and configuring SQL DI (Roadmap)

Getting SQL DI up and running involves a sequence of tasks that might be best performed by people in different roles at different times. Create a high level action plan for planning, installing, and configuring SQL DI.

Consider using the following roadmap to manage your planning, installation, and configuration activities and progresses.

Sequence	Task and instructions	Priority	Type	Role and skills
Step 1	Preparing for SQL DI installation <ul style="list-style-type: none">• Meet hardware requirements by allocating a IBM z16, z15™, z14, z13®, or zEnterprise® EC12 system.• Meet OS requirements by installing IBM z/OS 2.5 or 2.4.• Install the IBM Z Deep Neural Network Library (zDNN) stack.• Install z/OS Supervisor.• Install IBM OpenBLAS.• Install IBM Db2 13 for z/OS.• Obtain SQL DI product code packages.	Required	Planning, installation, configuration	z/OS system administrator or programmer, Db2 system or database administrator, SQL DI setup user
Step 2	Configuring system capacity and resources for SQL DI	Required	Planning, configuration	z/OS system administrator, MVS system administrator
Step 3	“Configuring network ports for SQL DI” on page 8	Required	Planning, configuration	z/OS system programmer with Unix service system skills, network administrator
Step 4	“Configuring setup user ID for SQL DI” on page 9	Required	Planning, configuration	z/OS system administrator or programmer with Unix service system skills, security administrator
Step 5	“Configuring a keyring-based keystore (JCERACFKS) for SQL DI” on page 12	Required	Configuration	z/OS system administrator or programmer with Unix service system skills and z/OS keyring and keystore skills, network administrator, security administrator
Step 6	“Configuring Db2 for SQL DI” on page 13	Required	Configuration	Db2 system or database administrator

Sequence	Task and instructions	Priority	Type	Role and skills
Step 7	Installing and configuring SQL DI	Required	Installation, configuration	SQL DI setup user, z/OS system programmer with Unix shell skills
Step 8	“Verifying the installation and configuration of SQL DI” on page 16	Required	Installation, configuration	z/OS system administrator or programmer, Db2 system or database administrator, SQL DI setup user
Step 9	Chapter 3, “Upgrading SQL DI,” on page 19	Required (for installing the latest enhancements)		z/OS system administrator or programmer, Db2 system or database administrator, SQL DI setup user

The roadmap consists of the following elements:

Sequence

Prescribes the order in which a task must be performed in the overall installation and configuration process. It will be noted if a particular task can be executed concurrently with another one or otherwise out of the order.

Task and instructions

Specifies the name of a task in the sequence and links to step-by-step instructions for performing the task.

Priority

Specifies whether a task is required or optional.

Type

Indicates the type of a task that can be planning, installation, or configuration. Some planning tasks, such as allocating system capacity and procuring prerequisite hardware and software, require longer lead time to complete. Quickly identify those tasks so that you can factor in the required time in your own action plan.

Role and skills

Recommends the IT role, skills, and access levels required for performing a particular task. For example, a z/OS system administrator with UNIX shell programming skills is one of the roles required for installing and configuring SQL DI. While the roles of database administrator, security administrator, network administrator, and UNIX shell programmer are optional, their skills and knowledge are much wanted.

Preparing SQL DI installation

Preparing for installation involves obtaining the SQL Data Insights (SQL DI) product code and readying your system environment. SQL DI has specific system, network, user access, and security requirements. You must satisfy these requirements before you install SQL DI.

Required product code packages

IBM distributes the SQL DI product code in SMP/E RELFILE format. Make sure that you obtain the following SQL DI code packages:

- SQL Data Insights 1.1.0 (HDBDD18) for Db2 13 for z/OS.
- APAR PH46563 (and any latest maintenance packages if available).

Important: Db2 releases new and enhanced SQL DI functions as they become available. To exploit these functions, install the latest maintenance packages (APARs/PTFs) for your Db2, z/OS, and SQL DI UI. See Chapter 3, “Upgrading SQL DI,” on page 19 for details.

Hardware and software requirements

SQL DI requires the following hardware, software, and integrated development tools. If you decide to enable the feature, make sure that you meet all the prerequisites before the installation. Consider planning the system requirements for both your Db2 system and the SQL DI feature together, particularly if you decide to install the SQL DI feature on the same LPAR where your Db2 system runs.

- IBM z16, z15, z14, z13, or zEnterprise EC12 system.

For best performance, consider running SQL DI on the latest models of Z.

- IBM z/OS 2.5 or 2.4.

Verify that data set SYS1.SIEALNKE and CEE.SCEERUN2 are APF authorized and accessible by Db2. See [z/OS 2.5 program directory](#) or [z/OS 2.4 program directory](#) for instructions.

- IBM Z Deep Neural Network Library (zDNN), including the Z AI Optimization Library (zAIO) and the Z AI Data Embedding Library (zADE), by applying the following APARs for z/OS:
 - For z/OS 2.5, apply OA62901, OA62902, and OA62903.
 - For z/OS 2.4, apply OA62849, OA62886, and OA62887.
- z/OS Supervisor with APAR OA62728 for both z/OS 2.5 and 2.4.
- IBM OpenBLAS by applying the following APARs for z/OS:
 - For z/OS 2.5, apply PH45672, PH44479, and PH46862.
 - For z/OS 2.4, apply PH45663, PH44479, and PH46862.
- IBM Db2 13 for z/OS (5698-DB2 or 5698-DBV) with APAR PH49781 applied.
- z/OS OpenSSH. See [z/OS OpenSSH](#) for instructions.
- IBM 64-bit SDK for z/OS Java™ Technology Edition Version 8 SR7 FP11 or later.

Browser requirements

SQL DI features a web-based user interface (UI). Make sure that you run the UI with the following standard or desktop version of Mozilla Firefox and Google Chrome:

- Firefox version 54 or higher
- Chrome version 60 or higher.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

[“Verifying the installation and configuration of SQL DI” on page 16](#)

Before you and your business start to use SQL Data Insights (SQL DI), complete a quick procedure to verify that the feature is properly installed and configured.

Configuring system resources for SQL DI

Preparing for installation involves obtaining the SQL Data Insights (SQL DI) product code and readying your system environment. SQL DI has specific system, network, user access, and security requirements. You must satisfy these requirements before you install SQL DI.

System capacity requirements

System capacity for SQL DI varies based on several key workload factors, including the size of source data, the number of unique values, and the data type of selected columns. As the number of rows and columns increase, more CPU, memory, and storage are required for enabling and running AI queries. The number of unique column values and the size of an AI object model correspond proportionally. The more distinct column values there are, the larger the resulting object model becomes and the more system resources are needed for training the model.

Take for example the system resource usage for processing a small SQL DI AI object. The AI object is 2.2 GB in size with 26 columns and 10 million rows. While 14 columns are of the SQL DI numeric data type, the remaining 12 columns are of the categorical type. It requires 8 threads on 10 CPUs, up to 17 GB of memory, and 20 GB file system storage to enable the object for AI query while achieving adequate performance goals. The total of 4 million unique values contributes to the final size of the resulting model, which requires 13 GB of disk space in the Db2 storage group.

MVS resource workload requirements

When you enable an AI object for AI query, SQL DI creates and trains a machine learning model for the object. Model training can consume all the resources available for your OMVS subsystem. Consider defining your SQL DI workload in z/OS Workload Manager (WLM) and assign a service class for Spark associated with this workload. For the service class, specify the default qualifier names SQLD% and SQLDAPPS with your performance goals and resource requirements for the workload. Also, consider assigning the service class for your SQL DI workload a lower priority than for your Db2 workloads. See [z/OS workload management for Apache Spark](#) for more information.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

[“Verifying the installation and configuration of SQL DI” on page 16](#)

Before you and your business start to use SQL Data Insights (SQL DI), complete a quick procedure to verify that the feature is properly installed and configured.

Configuring network ports for SQL DI

Preparing for installation involves obtaining the SQL Data Insights (SQL DI) product code and readying your system environment. SQL DI has specific system, network, user access, and security requirements. You must satisfy these requirements before you install SQL DI.

Network requirements

SQL DI requires dedicated networks and ports for communications across systems and services. Some of the ports are predefined while others can be user-defined. You must configure the following ports in your firewall before the SQL DI installation.

System or service	Port number	Outbound system	Inbound system	Default address space
SQL DI UI	15001 or user-defined	Your network	z/OS system	SQLDAPPS
z/OS Spark master	7077 or user-defined	z/OS system	z/OS Spark system	SQLDSPKM
z/OS Spark master REST API	6066 or user-defined	z/OS system	z/OS Spark system	SQLDSPKM
z/OS Spark master UI	8080 or user-defined	Your network	z/OS Spark system	SQLDSPKM
z/OS Spark worker	System-assigned* or user-defined	z/OS system	z/OS Spark system	SQLDSPKW
z/OS Spark worker UI	8081 or user-defined	Your network	z/OS Spark system	SQLDSPKW
z/OS Spark driver	System-assigned* or user-defined	z/OS system	z/OS Spark system	SQLDSPKD
z/OS Spark block manager	System-assigned* or user-defined	z/OS system	z/OS Spark system	SQLDSPKX
z/OS driver-specific port for Spark block manager	System-assigned* or user-defined	z/OS system	z/OS Spark system	SQLDSPKD

Notes:

* If you manage port assignments and access in your sysplex, do not use system-assigned port numbers for Spark worker, Spark driver, Spark block manager, or z/OS driver-specific port for Spark block manager. Also, a Spark cluster requires a port range, instead of a single port, at runtime. The actual range depends on the specified maximum number of retries for binding to a port.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

[“Verifying the installation and configuration of SQL DI” on page 16](#)

Before you and your business start to use SQL Data Insights (SQL DI), complete a quick procedure to verify that the feature is properly installed and configured.

Configuring setup user ID for SQL DI

The installation and configuration of SQL Data Insights (SQL DI) requires that you have sufficient privileges to access your z/OS system, allocate system resources, and customize system environment variables. Consider creating a multipurpose SQL DI setup user ID, grant it required permissions, and customize your z/OS environment for it.

Procedure

1. If you have not done so, create a multipurpose `<sqldi_setup_userid>`, which you will use to install, configure, and run your SQL DI.

You can create the required `<sqldi_setup_userid>` in different ways. For example, you can customize and run the following sample JCL job to create the ID:

```
//CREATE JOB (0),SQLDI RACF',CLASS=A,REGION=0M,
//MSGCLASS=H,NOTIFY=&SYSUID
//*-----*/
//RACF      EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
ADDGROUP SQLDIGRP OMVS(GID(<group-identifier>)) OWNER(SYS1)
ADDUSER <sqldi_setup_userid> DFLTGRP(SQLDIGRP) OMVS(UID(<user-identifier>)) -
HOME(/u/<sqldi_setup_userid>) -
PROGRAM($SQLDI_INSTALL_DIR/tools/bin/bash)) -
NAME('SQLDI ID') PASSWORD(<password>) NOOICARD
/*
```

where

- `<sqldi_setup_userid>` is the user ID that you will use to configure and run your SQL DI.
 - SQLDIGRP is a RACF group that you will use to associate SQL DI users and manage their access.
 - `<group-identifier>` is the identifier for SQLDIGRP.
 - `<user-identifier>` is the identifier for `<sqldi_setup_userid>`. Do not use UID 0 for `<sqldi_setup_userid>`.
 - \$SQLDI_INSTALL_DIR is the directory where SQL DI is installed. The default is `/usr/lpp/IBM/db2sqldi/`.
2. Allocate a minimum of 500 MB disk space to the home directory for `<sqldi_setup_userid>`.
 3. Create a SQLDI_HOME directory where SQL DI will store all the configuration, customization, and log files as well as runtime data.
 - a. Create the SQLDI_HOME directory. Make sure that SQLDI_HOME is mounted to a zFS file system with at least 100 GB storage available.

Tip: Avoid creating or configuring the SQLDI_HOME directory with automount management. Automount might unmount a directory if it is not referenced for a period of time. Any unplanned unmount of the SQLDI_HOME directory will cause SQL DI to fail.
 - b. If you use another user ID to create the SQLDI_HOME directory, make sure to change the directory owner to `<sqldi_setup_userid>` by issuing the following command:

```
chown -R sqldi_setup_userid:SQLDIGRP SQLDI_HOME/
```

- c. To allocate zFS data sets for SQLDI_HOME that are larger than 100 GB, make sure that you specify the DFSMS data class with extended format and extended addressability.
4. Configure your z/OS UNIX shell environment for `<sqldi_setup_userid>`
 - a. Copy the \$SQLDI_INSTALL_DIR/templates/profile.template directory into \$HOME/.profile for `<sqldi_setup_userid>`.
 - b. Customize the following environment variables in the profile template:
 - Set \$JAVA_HOME to the directory of your IBM Java 8 SR7 installation.
 - Set \$SQLDI_INSTALL_DIR to the directory where your SQL DI is installed. The default is `/usr/lpp/IBM/db2sqldi/`.
 - Set \$BLAS_INSTALL_DIR to the directory where the IBM OpenBLAS is located on your z/OS system. The default is `/usr/lpp/cbclib`.
 - c. Verify that the PATH environment variable is correctly set in the profile template as shown in the following example:

```
PATH=/bin:
PATH=$SQLDI_INSTALL_DIR/sql-data-insights/bin:$PATH
PATH=$SQLDI_INSTALL_DIR/tools/bin:$PATH
PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH
```

Where /tools/bin is home to bash, OpenSSL, and other tools.

5. Configure `<sqldi_setup_userid>` access to your z/OS UNIX shell environment.

`<sqldi_setup_userid>` must have the following permissions to install, configure, and run your SQL DI:

- Permission to read and write to the SQLDI_HOME directory.
 - Permission to read and execute to the \$SQLDI_INSTALL_DIR directory used by the SMP/E installation.
 - \$JAVA_HOME/bin defined in the \$PATH environment variable in the user's profile.
 - \$IBM_JAVA_OPTIONS environment variable set to -Dfile.encoding=UTF-8 in the user's profile.
 - \$_BPXK_AUTOCVT environment variable set to ON in the user's profile.
 - READ access to the RACF BPX.JOBNAME facility class so that SQL DI default address space names can take effect and that you can assign default job names with the SQLDI prefix to SQL DI started services.
 - READ access to resources CSFDSG, CSFDSV, CSFEDH, CSFIQA, CSFIQF, CSFOWH, CSFPKG, CSFPKI, CSFPKX, CSFRNG, and CSFRNGL for ICSF services in the CSFSERV class if your system is CryptoCard-enabled.
6. Update system resource settings, including CPUTIMEMAX, MEMLIMIT, and ASSIZEMAX values in the OMVS segment of the RACF profile for `<sqldi_setup_userid>`.

If needed, issue the **ALTUSER** command to update the CPUTIMEMAX, MEMLIMIT, and ASSIZEMAX settings as shown in the following example:

```
ALTUSER <sqldi_setup_userid> OMVS(ASSIZEMAX(address-space-size)
                                MEMLIMIT(nonshared-memory-size) CPUTIMEMAX(cpu-time))
```

SQL DI requires sufficient system memory to function properly. You can use the MEMLIMIT and ASSIZEMAX parameters to control the amount of memory for the address space started by `<sqldi_setup_userid>`. At the minimum, set MEMLIMIT initially to 32 GB or greater and ASSIZEMAX to 1 GB.

SQL DI also requires sufficient system CPU to run unimpeded. Consider setting the CPUTIMEMAX parameter to unlimited to ensure uninterrupted operations.

You can issue the **ulimit** command in a z/OS UNIX shell session to verify CPUTIMEMAX, MEMLIMIT, and ASSIZEMAX settings. The command returns a message that is similar to the following example:

```
/bin/ulimit -a
core file             8192b
cpu time              unlimited
data size             unlimited
file size             unlimited
stack size            unlimited
file descriptors      520000
address space         1048576k
memory above bar      24576m
```

Where

- "cpu time" is the value of the CPUTIMEMAX parameter.
- "address space" is the value of the ASSIZEMAX parameter.
- "memory above bar" is the value of the MEMLIMIT parameter.

See [ALTUSER \(Alter user profile\)](#) and [ulimit](#) for more information.

7. Verify that the required Java is installed on the z/OS system where you will install the SQL DI and available to `<sqldi_setup_userid>`.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

[“Configuring a keyring-based keystore \(JCERACFKS\) for SQL DI” on page 12](#)

SQL Data Insights (SQL DI) uses SSL to secure network communications and RACF to authenticate users. You must configure a RACF keyring and an associated keystore (JCERACFKS) to manage your SSL certificates and SQL DI user authentication.

Configuring a keyring-based keystore (JCERACFKS) for SQL DI

SQL Data Insights (SQL DI) uses SSL to secure network communications and RACF to authenticate users. You must configure a RACF keyring and an associated keystore (JCERACFKS) to manage your SSL certificates and SQL DI user authentication.

Before you begin

A RACF keyring is a set of digital certificates, private keys, and key mappings that defines your network trust policy, and a RACF keystore (JCERACFKS) collects and manages all the keyrings. To configure a RACF keystore, make sure that you grant your user ID with the RACF `SPECIAL` authority or sufficient authority as described in [RACDCERT command](#).

Procedure

1. Create a keyring by issuing the following RACF command:

```
RACDCERT ADDRING(SQLDIRG) ID(SQLDIID)
```

Where SQLDIID is the owner of the RACF keyring.

2. Generate a CA (certificate authority) certificate by issuing the following command:

```
RACDCERT GENCERT CERTAUTH +  
SUBJECTSDN( +  
    CN('PLEXE2') +  
    C('US') +  
    SP('CA') +  
    L('SAN JOSE') +  
    O('IBM') +  
    OU('SQLDI') +  
) +  
ALTNAME( +  
    EMAIL('user1@ibm.com') +  
) +  
WITHLABEL('SQLDICert') +  
NOTAFTER(DATE(2030/01/01))
```

If you decide to use an existing CA-signed certificate used by your business, make sure that you add and import the root CA certificate to RACF. See instructions in [RACDCERT command](#) for using the **RACDCERT ADD** and **RACDCERT IMPORT** commands.

3. Generate and sign a user certificate for `<sqldi_setup_userid>` by issuing the following command:

```
RACDCERT GENCERT ID(SQLDIID) +  
SUBJECTSDN( +  
    CN('PLEXE2') +  
    C('US') +  
    SP('CA') +  
    L('SAN JOSE') +  
    O('IBM') +  
    OU('SQLDI') +  
) +  
ALTNAME( +  
    EMAIL('user1@ibm.com') +  
) +  
WITHLABEL('SQLDICert_SQLDIID') +  
SIGNWITH(CERTAUTH LABEL('SQLDICert')) +  
NOTAFTER(DATE(2022/01/01))
```

Where SQLDIID is the owner of the RACF keyring.

4. Connect the user certificate and the CA certificate to the keyring you created and add usage options by issuing the following commands:

```
RACDCERT ID(SQLDIID) CONNECT(CERTAUTH LABEL('SQLDICert') +  
RING(SQLDIRG))  
  
RACDCERT ID(SQLDIID) CONNECT(ID(SQLDIID) LABEL('SQLDICert_SQLDIID') +  
RING(SQLDIRG) USAGE(PERSONAL))
```

5. Grant `<sqldi_setup_userid>` permission to access the keyring and the CA certificate.

`<sqldi_setup_userid>` must have the READ or UPDATE authority to the `IRR.DIGTCERT.LISTRING` resource in the FACILITY class. While the READ access enables the listing of your own keyring, the UPDATE access enables the listing of keyrings owned by others.

Issue the following commands:

```
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)  
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(<sqldi_setup_userid>) ACCESS(READ)  
SETROPTS RACLIST(FACILITY) REFRESH
```

`<sqldi_setup_userid>` must also have the READ or UPDATE authority to the `<ringOwner>.<ringName>.LST` resource in the RDATALIB class. While the READ access enables the retrieval of your private keys, the UPDATE access enables the retrieval of keys by others.

Issue the following commands:

```
RDEFINE RDATALIB SQLDIID.SQLDIRG.LST UACC(NONE)  
SETROPTS CLASSACT(RDATALIB) RACLIST(RDATALIB)  
SETROPTS CLASSACT(RDATALIB)  
PERMIT SQLDIID.SQLDIRG.LST CLASS(RDATALIB) ID(<sqldi_setup_userid>) ACCESS(READ)  
SETROPTS RACLIST(RDATALIB) REFRESH
```

See [“Configuring setup user ID for SQL DI” on page 9](#) for the full list of the privileges required for `<sqldi_setup_userid>`.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

Related information

[RACDCERT command](#)

Configuring Db2 for SQL DI

To enable SQL Data Insights (SQL DI), you must customize and submit the DSNTIJAI job to create the required database and tables in Db2 for z/OS for SQL DI.

Before you begin

The DSNUTILU stored procedure must be configured in Db2. See [DSNUTILU stored procedure](#).

Note: DSNTIJAI uses program DSNTIAD, the package of which must be bound with APPLCOMPAT V12R1M500 or higher in order to run.

Procedure

1. Copy and customize the DSNTIJAI sample job member in the Db2 SDSNSAMP library according to your needs.

When you create STOGROUP DSNAIDSG in the DSNTIJAI sample job, use a catalog alias for the VCAT option. Make sure that the alias is assigned to SMS-managed data sets that have allocation for extended format and extended addressability. The allocation will accommodate models that are larger than 4 GB in size.

2. Submit DSNTIJAI.

Note: The last step, DSNTIAI4, of the DSNTIJAI job grants the necessary database permissions to SQL DI users. You must repeat step DSNTIAI4 for each user.

3. Review the load template and add permissions.

This step involves loading a Db2 table. SQL DI provides a template with the utility control statements for the zLoad process.

Follow the instructions described in step “3” on page 29 of “[Modifying your SQL DI settings](#)” on page 29 to review the load template file. Ensure that SQL DI users have access to the data sets referred to in the template, and that adequate temporary space has been allocated, based on the size of your source table.

Note: The default high-level qualifier (HLQ) specified in the TEMPLATE statements in the load template is the z/OS job name indicated by the "&JO" variable. Review whether it is appropriate for SQL DI users to run the zLOAD utility in your environment.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

[“Modifying your SQL DI settings” on page 29](#)

During the installation and configuration, default values are automatically set to some of your SQL Data Insights (SQL DI) parameters, including the minimum amount of memory to run Spark jobs and the maximum number of rows to load for AI queries. Depending on the size of your data and the need of your workload, you can modify the default settings on the Settings page of the SQL DI user interface.

Installing SQL DI

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

Before you begin

Make sure that you have obtained the SQL DI code packages and met all system requirements as described in [“Preparing SQL DI installation” on page 6](#).

Procedure

1. Transfer the SQL DI program directory and product code packages onto your Z system.
2. Follow the instructions in the program directory and use SMP/E to install SQL DI and apply fixes.

The SMP/E program installs SQL DI in the \$SQLDI_INSTALL_DIR directory. The default \$SQLDI_INSTALL_DIR directory is /usr/lpp/IBM/db2sqldi/.
3. Verify that SMP/E was successfully executed for all the product code packages.
4. Create a separate SQLDI_HOME directory, with a minimum of 100 GB free disk space, to store the SQL DI configuration and log files.
5. Install the SQL DI web application.
 - a) In a bash session, change to the \$SQLDI_INSTALL_DIR/bin directory.
 - b) Execute the installation script by issuing the following command:

```
./sqldi.sh create
```
 - c) For each prompt, respond by entering requested information or accepting the default.
 - Enter the SQLDI_HOME directory where your SQL DI configuration and log files will be stored.
 - Enter the IP address or hostname for your SQL DI application.

- Enter the port number for your SQL DI application or press **Enter** to use the default port of 15001.
- Enter your keystore information.

SQL DI requires one of the following keystore types:

1. JCERACFKS (for managing RACF certificates and keys)
2. JCECCARACFKS (for managing RACF certificates and keys and exploiting hardware cryptography)

Select your keystore type and then enter the keyring name, the keyring owner, and the label of your SSL certificate.

- Enter the IP address or hostname of your Spark master.
- Enter the port number of your Spark master or press **Enter** to use the default port of 7077.
- Enter the port number of your Spark master REST API or press **Enter** to use the default port of 6066.
- Enter the port number of your Spark web UI or press **Enter** to use the default port of 8080.
- Enter the port number of your Spark worker or press **Enter** to use a system-assigned port.
- Enter the port number of your Spark worker web UI or press **Enter** to use the default port of 8081.
- Enter the port number of your Spark driver or press **Enter** to use a system-assigned port.
- Enter the port number of your Spark block manager or press **Enter** to use a system-assigned port.
- Enter the driver-specific port for the Spark block manager to listen on or press **Enter** to use a system-assigned port as the default.
- Enter the maximum number of retries when binding to a port or press **Enter** to use the default number of 16.

The installation process continues. You will see a message similar to the following example when it completes:

```
Congratulations! You have successfully installed SQL Data Insights.
```

- Confirm or decline to start your SQL DI automatically. If you confirm, the current command process will start SQL DI automatically. If you decline, continue to the next step to start SQL DI manually.

6. Start your SQL DI by issuing the following command:

```
./sqldi.sh start
```

Your SQL DI is successfully started if you see a message similar to the following example:

```
SQL Data Insights will use SQLDI_HOME to store configuration files and logs.
Bash version is 4.3
Starting SQL Data Insights ...
Reading configurations ...
Generating required configuration files ...
Launching SQL Data Insights ...

.....

SQL Data Insights is successfully started.

You can access it at https://<SQLDI-IPAddress>:<SQLDI-PortNumber>
```

Where *SQLDI-IPAddress* and *SQLDI-PortNumber* are either the IP address and port number that you entered or the defaults you accepted earlier. Make note of this URL and distribute it to your users.

7. Verify that the SQL DI user interface (UI) is up and running.

Sign in the UI at the following address with a valid RACF user ID that belongs to the SQLDIGRP group:

<https://<SQLDI-IPAddress>:<SQLDI-PortNumber>>

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation” on page 6](#) for details.

Tip: When the installation and configuration process completes successfully, the `sqldi.sh` script appends a set of command aliases to your `$HOME/.profile`. After you execute a **source** `$HOME/.profile` command, you can use the aliases to manage the lifecycle of SQL DI application and related Spark processes as follows:

- **start_sqldi** used for starting the SQL DI application.
- **stop_sqldi** used for stopping the SQL DI application.
- **start_spark** used for starting the embedded Spark cluster.
- **stop_spark** used for stopping the embedded Spark cluster.

Related tasks

[“Verifying the installation and configuration of SQL DI” on page 16](#)

Before you and your business start to use SQL Data Insights (SQL DI), complete a quick procedure to verify that the feature is properly installed and configured.

Related reference

[“Preparing SQL DI installation” on page 6](#)

Preparing for installation involves obtaining the SQL Data Insights (SQL DI) product code and readying your system environment. SQL DI has specific system, network, user access, and security requirements. You must satisfy these requirements before you install SQL DI.

Verifying the installation and configuration of SQL DI

Before you and your business start to use SQL Data Insights (SQL DI), complete a quick procedure to verify that the feature is properly installed and configured.

Before you begin

- Complete all the pre-installation tasks as described in [“Preparing SQL DI installation” on page 6](#).
- Complete the installation and configuration of SQL DI as described in [“Installing SQL DI” on page 14](#).

Procedure

Your SQL DI is properly installed and configured and ready for use if you can successfully complete the following sequence of tasks.

1. Customize and run the DSNTIJAV sample job in the Db2 SDSNSAMP data set.
The JCL job creates the sample DSNAIDB.CHURN table.
2. Create a connection to the Db2 system where the DSNAIDB.CHURN table is stored, as described in [“Creating a connection to Db2” on page 21](#).
3. Create an AI object named CHURN from the DSNAIDB.CHURN table and then enable it for AI query as described in [“Enabling AI query” on page 22](#).
 - a) For column configuration, assign SQL DI key data type to the CustomerID and retain the pre-assigned SQL DI data types for all other columns. You don't need to set column filter values.
 - b) Click **Enable AI query** to start the enablement process.

When the AI query enabling process completes successfully, the status of object CHURN is changed to Enabled.
4. Run AI query on object CHURN as described in [“Running an AI query” on page 25](#).
 - a) Enter the following statement in the SQL editor:

```
SELECT AI_SIMILARITY(X.customerID, '3668-QPYBK') AS SimilarityScore, X.*
FROM DSNAIDB.CHURN X
WHERE X.customerID <> '3668-QPYBK'
```

```
ORDER BY SimilarityScore DESC  
FETCH FIRST 10 ROWS ONLY
```

The purpose of this SQL statement is to identify top 10 customers who share similar characteristics with customer with ID 3668-QPYBK at a banking service. Customer 3668-QPYBK closed all accounts and left the service. The CHURN object and this sample query are intended to identify other customers who might also churn so that the service can act on this insight to mitigate the risk and try to retain those potential churners.

b) Click **Run** to start the query.

As specified, the query displays 10 rows of the result set in the **Results** section.

Chapter 3. Upgrading SQL DI

Db2 releases new and enhanced SQL DI functions as they become available. If you have already installed SQL DI, install the latest maintenance packages (APARs/PTFs) to upgrade your SQL DI and exploit all the new functions.

Before you begin

Important:

- The upgrade procedure assumes that you have already installed SQL DI. Make sure that you have met all system requirements as described in [“Preparing SQL DI installation” on page 6](#).
- Db2 APAR PH55212 includes the new AI_COMMONALITY function, and the function requires Db2 application compatibility level V13R1M504.
- The new Db2 AI_COMMONALITY function requires that models be trained with the latest zADE updates. The zADE updates are available in APAR OA64845 for z/OS 3.1 or OA64844 for z/OS 2.5. The function is not supported on z/OS 2.4. Consider upgrading your z/OS system to 2.5 or later if you want to use the AI_COMMONALITY function.

Procedure

1. Apply the latest APARs in the following order:
 - a. For zAIO and zADE in the zDNN stack on z/OS:
 - Apply OA64845 on z/OS 3.1 (HZAI310).
 - Apply OA64844, OA63950, and OA63952 for z/OS 2.5 (HZAI250).
 - Apply OA63949 and OA63951 for z/OS 2.4 (HBB77C0).
 - b. For OpenBLAS on z/OS:
 - Apply PH49807 and PH50872 for both z/OS 2.5 and z/OS 2.4 (HTV77C0).
 - Apply PH50881 for z/OS 2.5 (HLE77D0).
 - Apply PH50880 for z/OS 2.4 (HLE77C0).
 - c. For Db2 13 for z/OS, apply PH55212 and PH51892.
 - d. For SQL Data Insights 1.1.0 (HDBDD18), apply PH55943, PH55964, PH54368, and PH54661.
2. Verify that your SQL DI is successfully upgraded by completing some quick tasks as described in [“Verifying the installation and configuration of SQL DI” on page 16](#).

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

[“Verifying the installation and configuration of SQL DI” on page 16](#)

Before you and your business start to use SQL Data Insights (SQL DI), complete a quick procedure to verify that the feature is properly installed and configured.

Related reference

[“Preparing SQL DI installation” on page 6](#)

Preparing for installation involves obtaining the SQL Data Insights (SQL DI) product code and readying your system environment. SQL DI has specific system, network, user access, and security requirements. You must satisfy these requirements before you install SQL DI.

Chapter 4. Enabling and running AI queries

After successful installation and configuration, you can use the web user interface of your SQL DI to connect it to your Db2, create an AI object from selected Db2 tables and views, enable the object for AI queries, and run queries on the object at any time.

Creating a connection to Db2

To access data and enable AI query, your SQL Data Insights (SQL DI) must be connected to your Db2 system or data sharing group. You can create and activate a required JDBC connection on the Connections page of the SQL DI user interface.

Procedure

1. Sign in your SQL DI user interface with a valid RACF user ID (associated with the SQLDIGRP group) on the LPAR where your SQL DI is installed:

`https://<SQLDI-IPAddress>:<SQLDI-PortNumber>`

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation” on page 6](#) for details.
2. On the Connections page, click **Add connection**.
3. On the Add connection page, specify a name and details for the connection, including the hostname or IP address, port number, location, JDBC properties (optional), and special registers (optional) of your Db2 system or data sharing group.

4. Optionally, check **Port enabled for SSL connections** and enter the SSL certificate content in the input field.

If your Db2 system or data sharing group uses SSL for network communications, you must configure the new connection with the required SSL certificate information. You can provide your SSL certificate information in one of the following ways:

- Option 1: Check the box for **Port enabled for SSL connections** and provide the required SSL certificate information. Your SSL certificate must use Base64 ASCII encoding in PEM (.pem) format. Make sure that the certificate content is bound by the -----BEGIN CERTIFICATE----- header and the -----END CERTIFICATE----- footer.
- Option 2: Specify the following value in the JDBC properties field in the previous step:


```
sslConnection=true;sslCertLocation=<path_to_trusted_certificate>
```


Where *<path_to_trusted_certificate>* is the full path and the file name of your SSL certificate on the system where your SQL DI runs. The certificate must use Base64 ASCII or binary encoding in ARM (.arm), PEM (.pem), CERT (.cert), CRT (.crt), or DER (.der) format. If the certificate file uses Base64 ASCII encoding, make sure that the certificate content is bound by the plain-text -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- lines.

5. Enter your Db2 username and password.

Make sure that your username or ID has sufficient privileges to access the Db2 system or the Db2 data sharing group.

6. Click **Add** to create the new connection.
7. Back on the Connections page, verify that the new connection shows up.
8. Optionally, activate the new connection.

A new connection is activated (green-checked) when it is created. If necessary, you can deactivate the connection by selecting **Disconnect** from the  action menu of the connection. To re-activate the connection:

- a) Click the action menu  and select Connect.
- b) Enter your Db2 username and password.
- c) Click **Connect** to activate the connection.

The connection is successfully activated when it's green-checked.

If necessary, you can deactivate, edit, or remove the connection.

Adding an AI object


SQL Data Insights (SQL DI) executes AI functions on a Db2 user table or view through a corresponding AI object and that is enabled for AI query. AI objects are associated with a specific connection. You can add and manage AI objects for a connection on the AI objects page of the SQL DI user interface.

Procedure

1. Sign in your SQL DI user interface with a valid RACF user ID (associated with the SQLDIGRP group) on the LPAR where your SQL DI is installed:

`https://<SQLDI-IPAddress>:<SQLDI-PortNumber>`

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation” on page 6](#) for details.

2. On the Connections page, select a connection and click the  action menu.
3. Select List AI objects to open the AI objects page for the connection.
4. On the AI objects page, click **Add object**.
5. On the Add object page, choose one or more schemas to display all associated Db2 tables and views.
6. Select one or more Db2 tables or views.
7. Click **Add object** to create one or more AI objects.

If you select one Db2 table or view and click the **Add object** button, SQL DI will create a single AI object. If you select multiple Db2 tables or views, SQL DI will create multiple AI objects with each object corresponding to a specific Db2 table or view.

If you select only one Db2 table or view to create a corresponding AI object and want to enable the object for AI query, click **Enable AI query** to accomplish both in a single step. See [“Enabling AI query” on page 22](#) for instructions on column configurations.

8. Back on the AI objects page, verify that the newly added object show up and their statuses are Created.

Enabling AI query

You can enable an AI object for AI query when or after the object is created. Enabling AI query requires column configuration and model training. You can enable an AI object for AI query on the AI objects page of the SQL Data Insights (SQL DI) user interface.


Procedure

1. Sign in your SQL DI user interface with a valid RACF user ID (associated with the SQLDIGRP group) on the LPAR where your SQL DI is installed:

`https://<SQLDI-IPAddress>:<SQLDI-PortNumber>`

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation” on page 6](#) for details.

2. On the Connections page, select a connection and click the  action menu.

3. Select **List AI objects** to open the **AI objects** page for the connection.
4. Select an AI object and from the  action menu, select **Enable AI query**.
5. On the **Enable AI query** page, select and configure the columns that you want to include for your AI queries.

- a) Select columns and assign SQL DI data types to create a column configuration for the AI object.

You can select one or more columns and assign each column a SQL DI categorical, numeric, or key data type. SQL DI uses your selections to create a column configuration and train a machine learning model for the object.

- **Categorical:** The SQL DI categorical data type is used for columns with discrete values, each of which is its own entity. Type `categorical` is common in columns of many SQL data types. Columns with character or datetime SQL data types, such as `CHAR`, `VARCHAR`, `DATE`, `TIME`, `TIMESTAMP`, and `TIMESTAMP WITH TIMEZONE`, are a SQL DI categorical type, and so are columns with numeric values representing social security or ID numbers.
- **Numeric:** The SQL DI numeric data type is used for columns with continuous values. Columns with numeric SQL data types, such as `SMALLINT`, `INTEGER`, `BIGINT`, `DECIMAL`, `REAL`, `FLOAT`, and `DECFLOAT`, are a SQL DI numeric type. SQL DI uses clustering to group numeric values that are close together during the AI query enablement process.
- **Key:** The SQL DI key data type is used to indicate that a column represents an entire row. A customer ID column is a SQL DI key type. When processing an AI query that includes a column with the key type, SQL DI evaluates the affected rows in their entirety and effectively compares all the values in one row to those in another, not just the values in the column of the key type.

Important:

- SQL data types `BINARY`, `LOB`, `XML`, and `ROWID` are currently not supported for AI query enablement. Any `BINARY`, `LOB`, `XML` or `ROWID` column of an AI object is designated as "unsupported" and cannot be selected for the column configuration.
- While you can specify SQL DI data type `categorical` or `numeric` to as many columns as you want, assign type `key` to only one column.
- Do not assign SQL DI data type `numeric` to any non-numeric column, such as a `CHAR` or `VARCHAR` column. The conversion of non-numeric data to float might produce unpredictable results, which causes model training to fail.
- Treat a column of numeric SQL data type as a SQL DI categorical type if the column contains 10 or fewer distinct values. A grade column in a class schedule table and an interest rate in a bank loan table are good examples. If the grade column has only 6-10 unique values, set the column as a SQL DI categorical column.
- When a SQL DI categorical or key column is trained, SQL DI will automatically change the data type to `VARCHAR`. However, typecasting might not work properly if the data in the column is `BINARY`, such as `FOR BIT DATA`. In this case, create a view of the table and convert the column data from `BINARY` to `HEX`, which will ensure the correct working of your AI queries.
- Make sure that the name of a selected column does not contain any exclamation mark or whitespace character. If you want to include columns with names that don't meet the criteria, create a Db2 view on these columns and include the view in the column configuration.

Optionally, you can **import** the column configuration of an AI object that is already enabled for AI query. You must first export the column configuration of the object into a `.json` file. When importing a column configuration, make sure the columns defined in the JSON file are consistent with those in the AI object that you currently select.

- b) Click **Next** to continue.
- c) Optionally, specify column values as `NULL` for model training.


You can specify column values of your choice, such as `N/A`, `n/a`, `na`, `NR`, `invalid`, and `empty`, as `NULL` values. SQL DI ignores these user-specified and SQL `NULL` values during model training.


As the field names indicate, the value that you specify in the Specify NULL values that apply to all columns field applies to one or more columns with matching records. The value you specify in the Specify NULL values that apply to a specific column field applies to any matching record within a specific column only. All records that match the specified NULL values are ignored when SQL DI trains the machine model for the AI object.

You can specify multiple values separated by semicolons. For example, you can specify N/A;n/a;na;NR;invalid;empty as NULL values, and SQL DI will ignore all matching records from model training.

6. Click **Enable** to start the model training in the background.

If you create your AI objects one at a time, you have the option to enable the object for AI query during the creation process. On the Add object page, click **Enable AI query** to add the object and enable it for AI query in a single step. See [“Adding an AI object”](#) on page 22 for more information.

SQL DI starts the enabling process in the background. The entire process may take some time to complete depending on the size of your table or view and the number of selected columns. You can monitor the progress by refreshing the page and then clicking the  details arrow to the left of the object name.

The AI query enabling process completes successfully when the object's status is changed to Enabled with a green check mark. The Enabled status indicates that the model for the object is successfully created and trained. If needed, export the column configuration of this object for future use by clicking Export column configuration from the  action menu of the object. The column configuration is saved into a .json file.

If the status is Failed with a red triangle, repeat steps [“4”](#) on page 23 - [“6”](#) on page 24 to restart the enabling process. Make sure that you review your column configuration and eliminate any error.

SQL DI uses the Db2 zLoad utility to upload object model data during the AI query enablement. If the enablement process fails during the zLoad phase, check the Spark log and resolve any data loading errors. When you select to enable AI query on the object again, you will have the option to resume the previously failed process without having to start a new one. When the resumed process completes successfully, the object's status will be changed to Enabled after you refresh the UI page.

If needed, you can disable the object for AI query. Afterward, the status of the object is changed to Disabled.

If an object is never initiated for AI query enablement, its status remains Created.

Viewing an AI object model



When an AI object is enabled for AI query, SQL Data Insights (SQL DI) creates and trains a model for the object. You can view the model on the Model details page of the SQL DI user interface.

Procedure

1. Sign in your SQL DI user interface with a valid RACF user ID (associated with the SQLDIGRP group) on the LPAR where your SQL DI is installed:

`https://<SQLDI-IPAddress>:<SQLDI-PortNumber>`

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation”](#) on page 6 for details.

2. On the Connections page, select a connection and click the  action menu.
3. Select List AI objects to open the AI objects page for the connection.
4. On the AI objects page, select an object and click the  action menu.
5. Select View model to open the Model details page.

You can toggle between Training history and Cluster center tabs to view the details of an object model.

Running an AI query


After an AI object is enabled for AI query, you can run queries on the object on the AI objects page of the SQL Data Insights (SQL DI) user interface.

Procedure

1. Sign in your SQL DI user interface with a valid RACF user ID (associated with the SQLDIGRP group) on the LPAR where your SQL DI is installed:

`https://<SQLDI-IPAddress>:<SQLDI-PortNumber>`

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation”](#) on page 6 for details.

2. On the Connections page, select a connection and click the  action menu.
3. Select List AI objects to open the AI objects page for the connection.
4. Click **Run query** to open the Run query page with the query editor.
5. On the Run query page, select a query type from the options menu and then enter a query in the SQL editor.
 - a) Optionally, select one of the following query types based on the insights you want your query to discover:
 - *Semantic similarity*: A similarity query identifies groups of similar records or entities in records. Consider selecting semantic similarity if your query intends to identify the similarities of customer characteristics and behaviors in industries, such as commerce, finance, and insurance.
 - *Semantic dissimilarity*: A dissimilarity query finds the outliers from the norm in records. Consider selecting semantic dissimilarity if your query intends to detect operational anomalies, fraudulent activities, and other patterns of deviation.
 - *Semantic clustering*: A clustering query forms a cluster of entities in records and evaluates whether or not an additional entity belongs in the cluster. Consider selecting semantic clustering if your query intends to examine similarities or dissimilarities across multiple entities in a broader context.
 - *Semantic analogy*: An analogy query determines if the relationship between two entities applies to that of a second pair of entities. Consider selecting semantic analogy if your query intends to discover your customers' preference for a specific product and the degree of their affinity for other products.
 - *Semantic commonality*: A commonality query identifies the entities in records that exhibit the most common or uncommon patterns. Consider using semantic commonality if your query intends to detect the normal or aberrant characteristics and behaviors of your customers.

Note: Selecting a query type is optional, and not all available semantic query types, such as semantic commonality, are listed on the options menu. With or without a selected query type, SQL DI will process your query and invoke the AI function you specify. You can specify the AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY function in your query. See [Chapter 8, “Db2 built-in functions for SQL DI,”](#) on page 43 for details about the AI functions.

- b) In the SQL editor, customize the sample query or enter a new one.

When you select a query type, SQL DI populates the corresponding tab of the SQL editor with a sample query. You can customize the sample query based on your need.

Alternatively, enter a new query as shown in the following example. This simple query specifies the AI_COMMONALITY function and intends to find the top 20 customers by their CUSTOMERID who exhibit the most common pattern of behaviors:

```
SELECT AI_COMMONALITY(CUSTOMERID) AS SCORE, C.*  
FROM DSNAIDB.CHURN C
```

```
ORDER BY SCORE DESC  
FETCH FIRST 20 ROWS ONLY;
```

If you want to find the top 10 customers by their CUSTOMERID who exhibit the most uncommon pattern of behaviors, enter the following query that also specifies the AI_COMMONALITY function:

```
SELECT AI_COMMONALITY(CUSTOMERID) AS SCORE, C.*  
FROM DSNADB.CHURN C  
ORDER BY SCORE ASC  
FETCH FIRST 10 ROWS ONLY;
```

The most common pattern means that the scores of the top 20 customers converge toward the centroid value of the whole data set, and the most uncommon pattern indicates that the scores of the top 10 customers deviate the most from the centroid value.

SQL DI retains and caches the SQL statement on each tab of the editor. If needed, click **Add SQL +** to add a new tab or click **X** to remove a tab from the editor.

You can open up to 10 tabs of the query editor. When the limit is reached, you must close some existing tabs in order to open new ones. When you close a tab, the SQL statement on the tab and in the cache is deleted.

6. Click **Run** to run the query and review the results in the **Result set** section.

- You can run multiple queries specified on multiple tabs at the same time. A query run on each tab returns the results in the corresponding **Result set** section of the tab. Any subsequent run repopulates and refreshes the result set. If there is no matching record for a query, you will see an "Unable to retrieve query results" message in the section. In this case, verify that there are matching records for the specified query or update the SQL statement in the editor and run the query again.
- By default, SQL DI fetches and displays 50 rows for a query result set on this page. If you want to see fewer than 50 rows, you can specify the `fetch * rows` option in your SQL statement. You have the option to export the displayed rows into a CSV file.
- SQL DI loads the remaining rows of a query result set in the backend. You have the option to download the remaining rows or the entire set of your query result. The default value for the maximum number of loaded rows is 1000. If you want more rows loaded, change the default value on the **Settings** page as described in [“Modifying your SQL DI settings” on page 29](#).
- An AI query returns an SQL null value if it includes functions with arguments of null, filtered, or unseen values. Filtered values result from the application of the NULL values that you specify for all columns or a specific column. In the AI model, they are represented with the DB2_GENERATED_EMPTY string. Unseen values are those that are not present in the AI object when it's enabled for AI query. If your query includes arguments with null, filtered, or unseen values, SQL DI does not compute any result and thus returns an SQL null value.
- For best query results from the AI_SEMANTIC_CLUSTER function, consider specifying constant or unchanging values for three clustering-arguments.

Related information

[Db2 built-in scalar functions](#)

Analyzing data




When training the model for an AI object, SQL Data Insights (SQL DI) collects key data statistics and renders them into metric scores for the model. The visualized scores can help you understand the results of AI queries on the object. You can view the data statistics and model scores on the **Analyze data** page of the SQL DI user interface.

Procedure

1. Sign in your SQL DI user interface with a valid RACF user ID (associated with the SQLDIGRP group) on the LPAR where your SQL DI is installed:

`https://<SQLDI-IPAddress>:<SQLDI-PortNumber>`

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation” on page 6](#) for details.

2. On the Connections page, select a connection and click the  action menu.
3. Select List AI objects to open the AI objects page for the connection.
4. Select an AI object and from the  action menu, select Analyze data.
5. On the Analyze data page, toggle between the Object details, Data statistics, Column influence, and Column discriminator tabs.
 - The Object details tab displays the column configuration information, including the name, Db2 data type, and SQL DI data type of a column.
 - The Data statistics tab displays the column value distribution information, including the most common value, the number of most common value, the number of unique value, standard deviation, mean, max, and min values of a column.
 - The Column influence tab, available after successful AI query enablement on the object, displays the column influence scores. An influence score correlates to the number of user-specified and SQL NULL values in a column and indicates the column's influence on the training of the object model. The fewer NULL values the column has, the higher influence score it generates.
 - The Column discriminator tab, available after successful AI query enablement on the object, displays the column discriminator scores. A discriminator score correlates to the number of unique values in a column and measures the column's ability to semantically distinguish its values from other values in the table. The more unique values the column has, the higher discriminator score it generates. The generally high discriminator score of the primary key column is not included on the tab because it may skew the representation of the scores of other columns.
6. Click the  icon to reload the page to display data updates.

Chapter 5. Administering SQL DI

After SQL Data Insights (SQL DI) is up and running, it is important to keep it that way. You can adjust your SQL DI settings based on your workload. You can also manage the SQL DI application and the embedded Spark cluster with z/OS started tasks.

SQL Data Insights supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation” on page 6](#) for details.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

Related reference

[“Preparing SQL DI installation” on page 6](#)

Preparing for installation involves obtaining the SQL Data Insights (SQL DI) product code and readying your system environment. SQL DI has specific system, network, user access, and security requirements. You must satisfy these requirements before you install SQL DI.

Modifying your SQL DI settings


During the installation and configuration, default values are automatically set to some of your SQL Data Insights (SQL DI) parameters, including the minimum amount of memory to run Spark jobs and the maximum number of rows to load for AI queries. Depending on the size of your data and the need of your workload, you can modify the default settings on the Settings page of the SQL DI user interface.

Procedure

1. Sign in your SQL DI user interface with a valid RACF user ID (associated with the SQLDIGRP group) on the LPAR where your SQL DI is installed:

`https://<SQLDI-IPAddress>:<SQLDI-PortNumber>`

The SQL DI UI supports the standard or desktop version of Mozilla Firefox and Google Chrome. See [“Preparing SQL DI installation” on page 6](#) for details.

2. On the upper right corner of the SQL DI framework, click the  (gear) icon to open the Settings page.
3. On the Settings page, review and adjust the settings for the following parameters:
 - For Spark, specify the number of cores and the amount of memory for your Spark driver and executor. SQL DI uses Spark jobs to transform the data in the tables or views in your AI object. Increase the number of cores and the amount of memory to improve performance if the size of the data is large.
 - For CPU threads, specify the number of CPU threads for data preprocessing and model training. Increase the number of CPU threads to reduce the model training time.
 - For Db2 load utility, use the default LOAD utility control statement or customize it for loading trained models. SQL DI uses the ZLOAD command to upload the model training results to your Db2 system. Verify that the Db2 user ID has the permissions to run the ZLOAD command and to access the data sets referenced in the load utility control statement. See [“Configuring Db2 for SQL DI” on page 13](#) for details.

Customize the LOAD control statement based on your Db2 settings. For a very large data, increase the space allocation and specify a data class with the extended addressability attribute, allowing the data sets to grow in size beyond 4 GB.

- For AI query, specify the maximum number of rows to load for a query result set. By default, SQL DI loads up to 1000 rows of a query result set. You can change the default value if needed.

Creating a started task for the SQL DI application

After you have successfully installed and configured SQL Data Insights (SQL DI), consider running the application as a z/OS started task. You can quickly create a started task by customizing the SQLDAPPS sample JCL job.

Before you begin

- Plan, install, and configure SQL DI as described in [“Preparing SQL DI installation”](#) on page 6 and [“Installing SQL DI”](#) on page 14.

Procedure

1. Locate the following files in the \$SQLDI_INSTALL_DIR/templates/started-task-samples directory on the z/OS system where your SQL DI runs:
 - SQLDAPPS (sample JCL job)
 - SQLDSTRT-STDPARM.template (sample MVS data set content)
 - SQLDSTOP-STDPARM.template (sample MVS data set content)
 - stdenvs-STDENV.template (sample z/OS Unix text file content)
2. Copy SQLDAPPS into a data set in your PROCLIB concatenation, such as SYS1.PROCLIB, and customize them based on your system environment.
3. Define a new MVS data set to be used by ddname STDPARM.

- a) Create member SQLDSTRT for SQLDAPPS by copying the following lines from SQLDSTRT-STDPARM.template:

```
PGM /usr/lpp/IBM/db2sqldi/v1r1/tools/bin/bash
/usr/lpp/IBM/db2sqldi/v1r1/sql-data-insights/bin/sqldi.sh start
```

- b) Create member SQLDSTOP for SQLDAPPS by copying the following lines from SQLDSTOP-STDPARM.template:

```
PGM /usr/lpp/IBM/db2sqldi/v1r1/tools/bin/bash
/usr/lpp/IBM/db2sqldi/v1r1/sql-data-insights/bin/sqldi.sh stop
```

- c) If necessary, replace the default /usr/lpp/IBM/db2sqldi/v1r1 segment in each directory path with \$SQLDI_INSTALL_DIR where your SQL DI is installed.
4. Create a new z/OS Unix text file stdenvs to be used by ddname STDENV.

You can use the same STDENV file for all your SQL DI started tasks.

- a) Copy the following lines from stdenvs-STDENV.template:

```
_BPXK_AUTOCVT=ON
SQLDI_INSTALL_DIR=/usr/lpp/IBM/db2sqldi/v1r1
BLAS_INSTALL_DIR=/usr/lpp/cbclib
SQLDI_HOME=/path/to/sqldi-home
SPARK_HOME=/usr/lpp/IBM/db2sqldi/v1r1/spark24x
SPARK_CONF_DIR=/path/to/sqldi-home/spark/conf
JAVA_HOME=/java8_64/J8.0_64
PATH=/bin:/usr/lpp/IBM/db2sqldi/v1r1/tools/bin:/java8_64/J8.0_64/bin
LIBPATH=/lib:/usr/lib:/java8_64/J8.0_64/bin/classic:/java8_64/J8.0_64/bin/j9vm:
        /java8_64/J8.0_64/lib/s390x:/java8_64/J8.0_64/lib:/usr/lpp/cbclib/lib
IBM_JAVA_OPTIONS="-Dfile.encoding=UTF-8"
_ENCODE_FILE_NEW=ISO8859-1
_ENCODE_FILE_EXISTING=UNTAGGED
_CEE_RUNOPTS="FILETAG(AUTOCVT,AUTOTAG) POSIX(ON)"
SQLDI_SERVICE_ONLY=TRUE
```

- b) If necessary, replace the default /usr/lpp/IBM/db2sqldi/v1r1 segment in each directory path with \$SQLDI_INSTALL_DIR where your SQL DI is installed.
- c) If necessary, replace the default /usr/lpp/cbclib segment in each directory path with \$BLAS_INSTALL_DIR where the IBM OpenBLAS is installed.

- d) Set all environment variables based on your z/OS system environment and your SQL DI installation. See [“Configuring setup user ID for SQL DI”](#) on page 9 for instructions.
5. Define a RACF profile for the new SQLDAPPS started task and assign `<sqldi_setup_userid>` as the owner by issuing the following commands:

```
RDEFINE STARTED SQLDAPPS.* STDATA(USER(<sqldi_setup_userid>) GROUP(SQLDIGRP))
SETROPTS RACLIST(STARTED) REFRESH
```

6. Run the SQLDAPPS started task to start the SQL DI application as shown in the following example:

```
/S SQLDAPPS
```

7. If necessary, run the SQLDAPPS started task to stop the UI services by issuing the following command:

```
/S SQLDAPPS,OPTION='SQLDSTOP'
```

The OPTION value is case-sensitive. Make sure that you issue the command in your SDSF system command extension to retain the lower case of your input.

Related tasks

[“Creating started tasks for the Spark cluster”](#) on page 31

The SQL Data Insights (SQL DI) application is powered by an embedded Spark cluster. After you have successfully installed SQL DI, consider managing the cluster by creating and running z/OS started tasks. You can quickly create the started tasks for the Spark master and worker by customizing the SQLDSPKM and SQLDSPKW sample JCL jobs.

Creating started tasks for the Spark cluster

The SQL Data Insights (SQL DI) application is powered by an embedded Spark cluster. After you have successfully installed SQL DI, consider managing the cluster by creating and running z/OS started tasks. You can quickly create the started tasks for the Spark master and worker by customizing the SQLDSPKM and SQLDSPKW sample JCL jobs.

Before you begin

- Plan, install, and configure SQL DI as described in [“Preparing SQL DI installation”](#) on page 6 and [“Installing SQL DI”](#) on page 14.

Procedure

- Navigate to the `$SQLDI_INSTALL_DIR/templates/started-task-samples` directory on the z/OS system where your SQL DI runs.
- Copy the SQLDSPKM and SQLDSPKW sample JCL files into a data set in your PROCLIB concatenation, such as `SYS1.PROCLIB`.
- Follow the instructions in the sample procedures to customize the environment variables based on your system environment.

For example, set `$SPARK_CONF_DIR` to `SQLDI_HOME/spark/conf`.

- Copy the `spark-zos-started-tasks.sh.template` file to the `SQLDI_HOME/spark/conf` directory by issuing the following command:

```
cp $SQLDI_INSTALL_DIR/templates/started-task-samples/spark-zos-started-tasks.sh.template
SQLDI_HOME/spark/conf/spark-zos-started-tasks.sh
```

- Update the `spark-zos-started-tasks.sh` script in the `SQLDI_HOME/spark/conf` directory as shown in the following example:

```
# Java environment variable - REQUIRED
# Default: /usr/lpp/java/J8.0_64
export JAVA_HOME=<PATH_TO_JAVA_HOME>
```

```
# SQL DI installation directory - REQUIRED
# Default: /usr/lpp/IBM/db2sqlcli/
export SQLDI_INSTALL_DIR=<PATH_TO_SQLDI_INSTALL_DIR>

# OpenBLAS installation directory - REQUIRED
# Default: /usr/lpp/cbclib
export BLAS_INSTALL_DIR=<PATH_TO_BLAS_INSTALL_DIR>
```

6. Define a RACF profile for the new SQLDSPKM and SQLDSPKW started tasks and assign `<sqlcli_setup_userid>` as the owner by issuing the following commands:

```
RDEFINE STARTED SQLDSPKM.* STDATA(USER(<sqlcli_setup_userid>) GROUP(SQLDIGRP))
RDEFINE STARTED SQLDSPKW.* STDATA(USER(<sqlcli_setup_userid>) GROUP(SQLDIGRP))
SETROPTS RACLIST(STARTED) REFRESH
```

7. Start the SQLDSPKM and SQLDSPKW started tasks by issuing the following MVS commands without any parameter:

```
start SQLDSPKM
start SQLDSPKW
```

To run the Spark started tasks manually, make sure that you start SQLDSPKM before SQLDSPKW. If you automate the run, you can start them in parallel in which the processes triggered by SQLDSPKW will start right after those by SQLDSPKM.

8. If necessary, stop the SQLDSPKM and SQLDSPKW started tasks by issuing the following MVS commands without any parameter:

```
stop SQLDSPKM
stop SQLDSPKW
```

See [Stopping z/OS started tasks](#) for more information about stopping Spark started tasks.

Related tasks

[“Installing SQL DI” on page 14](#)

The installation of SQL Data Insights (SQL DI) involves a script-driven sequence of interactive tasks. Make sure that you follow the step-by-step instructions and successfully complete each task.

Related reference

[“Preparing SQL DI installation” on page 6](#)

Preparing for installation involves obtaining the SQL Data Insights (SQL DI) product code and readying your system environment. SQL DI has specific system, network, user access, and security requirements. You must satisfy these requirements before you install SQL DI.

Chapter 6. Db2 tables for SQL DI

When you run the sample DSNTIJAI job to configure your Db2 system for SQL DI, the job creates a set of tables that are used to record and store metadata for AI objects, object models, and tables.

The sample DSNTIJAI job creates the following Db2 tables, tablespaces, and indexes for SQL DI:

Table 1. Db2 tables, tablespaces, and indexes for SQL DI

Table	Description	Tablespace	Index	Index field
SYSAIDB.SYSAIOBJECTS	Contains a row for each Db2 table or view you select for an SQL DI AI object	SYSTSAIO	SYSAIOBJECTSIX1	OBJECT_ID
			SYSAIOBJECTSIX2	SCHEMA, NAME
SYSAIDB.SYSAICONFIGURATIONS	Contains a row for each configuration ID for an AI object and related attributes	SYSTSAIC	SYSAICONFIGURATION SIX1	CONFIGURATION_ID
SYSAIDB.SYSAICOLUMNCONFIG	Contains a row for each column and related attributes within a column configuration	SYSTSAID	SYSAICOLUMNCONFIG IX1	CONFIGURATION_ID, COLUMN_NAME, COLUMN_AISQL_TYPE
SYSAIDB.SYSAIMODELS	Contains a row for each AI object model and related table and state information	SYSTSAIM	SYSAIMODELSIX1	MODEL_ID
SYSAIDB.SYSAICOLUMNCENTERS	Contains a row for each column centroid for a trained model	SYSTSAIE	SYSAICOLUMNCENTER SIX2	MODEL_ID, COLUMN_NAME, CENTROID
SYSAIDB.SYSAITRAININGJOBS	Contains a row for each training job that you initiate and job status information	SYSTSAIT	SYSAITRAININGJOBSI X1	TRAINING_JOB_ID
			SYSAITRAININGJOBSI X2	OBJECT_ID, CONFIGURATION_ID, MODEL_ID

SYSAIDB.SYSAIOBJECTS

The SYSAIDB.SYSAIOBJECTS table contains a row for each Db2 table or view you select for an SQL DI AI object.

Column name	Data type	Description	Usage
OBJECT_ID	BIGINT NOT NULL	A unique identifier for the AI object.	

Column name	Data type	Description	Usage
OBJECT_NAME	VARCHAR(32)	A user-defined name for the AI object.	
OBJECT_TYPE	CHAR(1)	An identifier that identifies a Db2 table or view: T Specifies a Db2 table V Specifies a Db2 view	
SCHEMA	VARCHAR(128) NOT NULL	The schema of the AI object.	
NAME	VARCHAR(128) NOT NULL	The name of the AI object.	
STATUS	VARCHAR(16) NOT NULL	The status of the AI query enabling process: Enabled Indicates that the AI object is enabled with AI query and that the row for the AI object model is populated. Disabled Indicates that the AI object is not enabled with AI query and that the row for the AI object model is not populated. Training Indicates that the AI object is being enabled with AI query and that the row for the AI object model is being updated. Failed Indicates that the AI query enabling process for the AI object failed.	
CONFIGURATION_ID	BIGINT	The identifier for the configuration used for the active model. A null value indicates that there is no active configuration yet.	
MODEL_ID	BIGINT	The identifier for the active model. A null value indicates that there is no active model table created yet.	
CREATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user to which the object is registered.	
CREATED_DATE	VARCHAR(32) (With CURRENT TIMESTAMP as default)	The timestamp when the object was registered.	
LAST_UPDATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user who last updated the object.	

Column name	Data type	Description	Usage
LAST_UPDATED_DATE	TIMESTAMP (With ROW CHANGE TIMESTAMP as default)	The timestamp when the object was last updated.	
DESCRIPTION	VARCHAR(256)	A user-specified description of the object.	

SYSAIDB.SYSAICONFIGURATIONS

The SYSAIDB.SYSAICONFIGURATIONS table contains a row for each configuration ID for an AI object and related attributes.

Column name	Data type	Description	Usage
CONFIGURATION_ID	BIGINT NOT NULL	A unique identifier for this configuration.	
NAME	VARCHAR(32)	A user-defined name for the configuration.	
OBJECT_ID	BIGINT NOT NULL	An identifier of the object for which this configuration is created.	
RETRAIN_INTERVAL	INTEGER	The interval at which retraining occurs.	
KEEP_ROWIDENTIFIER_KEY	CHAR(1) NOT NULL	An indicator for the presence of the row identifier key in a model: Y Indicates that the row identifier key is kept in the model. N Indicates that the row identifier key is not kept in the model.	
NEGLECT_VALUES	VARCHAR(1024)	A semicolon-separated string of values to be treated as null in the model.	
CREATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user to which the object is registered.	
CREATED_DATE	VARCHAR(32) (With CURRENT TIMESTAMP as default)	The timestamp when the object was registered.	
LAST_UPDATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user who last updated the object.	
LAST_UPDATED_DATE	TIMESTAMP (With ROW CHANGE TIMESTAMP as default)	The timestamp when the object was last updated.	

SYSAIDB.SYSAICOLUMNCONFIG

The SYSAIDB.SYSAICOLUMNCONFIG table contains a row for each column and related attributes within a column configuration.

Column name	Data type	Description	Usage
CONFIGURATION_ID	BIGINT NOT NULL	A unique identifier for the column configuration.	
COLUMN_AISQL_TYPE	CHAR(1) NOT NULL	<p>A SQL DI data type that you assign to a column in the column configuration:</p> <p>K Indicates that a column is assigned the key data type.</p> <p>C Indicates that a column is assigned the categorical data type.</p> <p>N Indicates that a column is assigned the numeric data type.</p> <p>I Indicates that a column is not assigned a data type.</p> <p>U Indicates that a column is assigned an unsupported data type</p>	
COLUMN_NAME	VARCHAR(128) NOT NULL	The name of the column in the column configuration.	
COLUMN_PRIORITY	CHAR(1)	<p>(Reserved) The processing priority that you assign to a column in the column configuration:</p> <p>H Indicates an high priority.</p> <p>M Indicates a medium priority.</p> <p>L Indicates a low priority.</p>	
COLUMN_VECTOR_CARDINALITY	BIGINT NOT NULL (With -1 as default)	The cardinality of the vectors in the VECTOR column in the vector table.	
MAX_DATA_VALUE_LEN	INTEGER NOT NULL (With -1 as default)	The maximum length of the value for the column in the vector table.	
NEGLECT_VALUES	VARCHAR(1024)	A semicolon-separated string of values to be treated as null in the model.	

SYSAIDB.SYSAIMODELS

The SYSAIDB.SYSAIMODELS table contains a row for each AI object model and related table and state information.

Column name	Data type	Description	Usage
CONFIGURATION_ID	BIGINT NOT NULL	A unique identifier for the configuration that is used to create this model.	

Column name	Data type	Description	Usage
MODEL_CODE_LEVEL	CHAR(32) NOT NULL (With ' ' as default)	The code level for training the model.	
MODEL_ID	BIGINT NOT NULL	A unique identifier for the model.	
NAME	VARCHAR(32)	A user-defined name for the model.	
OBJECT_ID	BIGINT NOT NULL	An identifier of the object for which this configuration is created.	
VECTOR_TABLE_CREATOR	VARCHAR(128)	The name of the user who created the vector table.	
VECTOR_TABLE_NAME	VARCHAR(128) NOT NULL	The name of the vector table.	
VECTOR_TABLE_STATUS	CHAR(2) NOT NULL	The status of the vector table. I Indicates that the table is initialized for the current process. L Indicates that the table is loading. A Indicates that the table is available for use. E Indicates that the table is in error state.	
VECTOR_TABLE_DBID	SMALLINT NOT NULL	The internal identifier of the vector table database.	
VECTOR_TABLE_OBID	SMALLINT NOT NULL	The internal identifier of the vector table.	
VECTOR_TABLE_IXDBID	SMALLINT NOT NULL	The internal identifier of the vector table index database.	
VECTOR_TABLE_IXOBID	SMALLINT NOT NULL	The internal identifier of the vector table index.	
VECTOR_TABLE_VERSION	SMALLINT NOT NULL	The internal format number of the vector table.	
METRICS	CLOB(500K)	A JSON object to store metrics about the model for display in the user interface.	
INTERPRETABILITY_OCCURENCE_STRUCT	BLOB(2G)	Reserved.	
CREATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user who created the model.	
CREATED_DATE	TIMESTAMP (With CURRENT_TIMESTAMP as default)	The timestamp when the model was created.	

Column name	Data type	Description	Usage
LAST_UPDATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user who last updated the model.	
LAST_UPDATED_DATE	TIMESTAMP (With ROW CHANGE TIMESTAMP as default)	The timestamp when the model was last updated.	
MODEL_ROWID	ROWID NOT NULL	A rowid column to support a LOB table.	

SYSAIDB.SYSAICOLUMNCENTERS

The SYSAIDB.SYSAICOLUMNCENTERS table contains a row for each column centroid for a trained model.

Column name	Data type	Description	Usage
MODEL_ID	BIGINT NOT NULL	The unique identifier of the model to which the centroid belongs.	
COLUMN_NAME	VARCHAR(128) NOT NULL	The name of the column to which the centroid belongs.	
CLUSTER_MIN	FLOAT NOT NULL	The numeric center of a cluster.	
LABEL	VARCHAR(5) NOT NULL	The label of the vector corresponding to the cluster.	

SYSAIDB.SYSAITRAININGJOBS

The SYSAIDB.SYSAITRAININGJOBS table contains a row for each training job that you initiate and job status information.

Column name	Data type	Description	Usage
TRAINING_JOB_ID	BIGINT NOT NULL	A unique identifier for the model training job.	
OBJECT_ID	BIGINT NOT NULL	The identifier for the object for which the model is being trained.	
CONFIGURATION_ID	BIGINT NOT NULL	The identifier for the configuration that is used for the model training.	
MODEL_ID	BIGINT NOT NULL	The identifier for the model that is created as a result of training.	

Column name	Data type	Description	Usage
STATUS	CHAR(2) NOT NULL	<p>The status of the model training for the object:</p> <p>I Indicates that the training process is being initialized.</p> <p>L Indicating that the data is being loaded for the training job.</p> <p>P Indicates that the data is being processed.</p> <p>T Indicates that the training is started.</p> <p>C Indicates that the training process is completed.</p> <p>F Indicates that the training process failed.</p>	
PROGRESS	SMALLINT NOT NULL	The percentage of the training process completed.	
RESOURCE	VARCHAR(512) NOT NULL	A JSON object that describes the resources allocated to the training job.	
MESSAGES	CLOB(8K)	The output of the training job.	
START_TIME	TIMESTAMP NOT NULL	The start time of the training job.	
END_TIME	TIMESTAMP	The end time of the training job. A null value indicates that the training job has not yet completed.	
CREATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user who initiated the training job.	
CREATED_DATE	TIMESTAMP (With CURRENT_TIMESTAMP as default)	The timestamp when the training job started.	
LAST_UPDATED_BY	VARCHAR(32) (With SESSION_USER as default)	The SQLID of the user who last updated the training job.	
LAST_UPDATED_DATE	TIMESTAMP (With ROW CHANGE_TIMESTAMP as default)	The timestamp when the training job was last updated.	

Related tasks

[“Configuring Db2 for SQL DI” on page 13](#)

To enable SQL Data Insights (SQL DI), you must customize and submit the DSNTIJAI job to create the required database and tables in Db2 for z/OS for SQL DI.

Chapter 7. Db2 subsystem parameter for SQL DI

Db2 updates the DSNTIP81 CLIST panel and adds the MXAIDTCACH subsystem parameter to support SQL DI.

DSNTIP81: Performance and optimization panel 2

The DSNTIP81 panel is a continuation of the DSNTIP8 panel. It is used to set application programming default values pertaining to performance and optimization panel.

```
DSNTIP81      INSTALL DB2 - PERFORMANCE AND OPTIMIZATION (PANEL 2)
====>

Enter data below:
 1 CURRENT DEGREE      ===> 1          1 or ANY
 2 MAX DEGREE          ===> 0          Maximum degree of parallelism. 0-254
 3 MAX DEGREE FOR DPSI ===> 0          Maximum degree of parallelism for data
                                         partitioned secondary indexes. 0-254
                                         or DISABLE

 4 STAR JOIN QUERIES   ===> DISABLE   DISABLE, ENABLE, 1-32768
 5 MAX DATA CACHING    ===> 20        0-512
 6 MAX AI DATA CACHING ===> 0         0-512

Enter default settings for maintained query special registers:
 7 CURRENT REFRESH AGE ===> 0          0 or ANY
 8 CURRENT MAINT TYPES  ===> SYSTEM    NONE, SYSTEM, USER, ALL

PRESS:  ENTER to continue   RETURN to exit   HELP for more information
```

Figure 3. Performance and optimization panel: DSNTIP81

MAX AI DATA CACHING field (MXAIDTCACH subsystem parameter)

The MXAIDTCACH subsystem parameter specifies the maximum amount of memory, in MB, that is to be allocated for AI data caching for each thread.

Acceptable values:	0 - 512
Default:	0
Update:	option 30 on panel DSNTIPB
DSNZPxxx:	DSN6SPRM MXAIDTCACH
Data sharing scope	Member
Online changeable	Yes

MXAIDTCACH controls memory allocation for AI queries that use sparse index access.

0

Specifies the default value. Db2 does not allocate any additional memory for AI data caching and disables batching for vector fetching.

1 - 512

Specifies a value between 1 and 512 (in MB). Db2 allocates the specified memory from above the 2 GB bar pool for AI data caching and enables batching for vector fetching.

If vector prefetching is enabled and if a query consists of multiple threads, Db2 allocates the specified amount of memory to each thread for AI data caching.

Db2 dynamically chooses between vector prefetching and row-by-row processing based on the AI object (table vs. view) and the AI cache size. If the MXAIDTCACH parameter is set to a value greater than 0 and a query invokes a SQL DI function on a table, Db2 automatically disables vector prefetching to optimize the CPU usage of the function.

Related reference

[MAX DATA CACHING field \(MXDTCACH subsystem parameter\) \(Db2 Installation and Migration\)](#)

Chapter 8. Db2 built-in functions for SQL DI

Db2 introduces the following built-in scalar functions to support SQL DI. You can use these functions to run AI queries on your Db2 user tables and views.

AI_ANALOGY

The AI_ANALOGY function computes an analogy score between two sets of values.

FL 500

➤ AI_ANALOGY (— *source-1* — , — *target-1* — , — *source-2* — , — *target-2* —) ➤

source or *target*:

➤ *expression* — USING MODEL COLUMN *column-name* ➤

The schema is SYSIBM.

The AI_ANALOGY function computes an analogy score using the values returned by the arguments. The arguments to the function specify two pairs, where there is a relationship between *source-1* and *source-2* and a relationship between *target-1* and *target-2*. The interaction between the pairs form an analogy, which can be thought of as a human language analogy: *source-1* is to *target-1* as *source-2* is to *target-2*.

source-1

The expression specifies the first source value for the analogy. The value must not be a binary, LOB, XML, or ROWID data type.

The *column-name* is an identifier in the USING MODEL COLUMN clause, which can be used to specify which machine learning model and column to use for the evaluation of the function.

target-1

The expression specifies the first target value for the analogy. The value must not be a binary, LOB, XML, or ROWID data type.

The *column-name* is an identifier in the USING MODEL COLUMN clause, which can be used to specify which machine learning model and column to use for the evaluation of the function.

source-2

The expression specifies the second source value for the analogy. The value must not be a binary, LOB, XML, or ROWID data type. The values for *source-1* and *source-2* must not be the same (SQLCODE -20580, SQLSTATE 428ID, RC=10).

If the *source-1* and *source-2* model columns are both a SQL DI numeric data type, Db2 may return SQLCODE -20580 even if the two numeric values are different. This happens when the numeric values belong to the same cluster during model training. In this case, SQL DI treats them as the same value (token). See the Model details page of the SQL DI UI for more information how numeric values in a cluster are processed in function arguments.

The *column-name* is an identifier in the USING MODEL COLUMN clause, which can be used to specify which machine learning model and column to use for the evaluation of the function.

target-2

The expression specifies the second target value for the analogy. The value must not be a binary, LOB, XML, or ROWID data type.

The *column-name* is an identifier in the USING MODEL COLUMN clause, which can be used to specify which machine learning model and column to use for the evaluation of the function.

Arguments to AI_ANALOGY must specify a machine learning model and machine learning model columns that are used to evaluate the function. The following rules are used to determine which model and model column is used for each argument:

- If an expression argument is a standalone column reference and no model column is explicitly specified with the USING MODEL COLUMN clause, the standalone column is the model column that is used to evaluate the function, and the model is the model table associated with the Db2 table that the column belongs to.
- If the expression argument is a standalone column reference, but the USING MODEL COLUMN clause specifies a different column, the column specified in the USING MODEL COLUMN clause is the model column, and the model is the model table associated with the Db2 table that the model column belongs to.
- A model column must be determined for at least one of *source-1* and *source-2*. A model column can be explicitly specified in the USING MODEL COLUMN clause, or the expression must be a simple column reference. If a model column can be determined for one of *source-1* or *source-2* but not the other, then the model column that is determined is used for both *source-1* and *source-2*. If a model column is specified for both *source-1* and *source-2*, they must be the same model column.
- A model column must be determined for at least one of *target-1* and *target-2*. The model column can be explicitly specified in the USING MODEL COLUMN clause, or the expression must be a simple column reference. If a model column can be determined for one of *target-1* or *target-2* but not the other, then the model column that is determined is used for both *target-1* and *target-2*. If a model column is determined for both *target-1* and *target-2*, they must be the same model column.
- All model columns specified in the function invocation must refer to columns that belong to the same table or view.
- AI must be enabled for the table or view that the model is associated with and the model must be trained. All model columns must be included in the model.

The model column, either specified as a standalone column reference or with the USING MODEL COLUMN clause, can be a qualified name. The qualifier must not be a synonym name or a correlation name of a table expression. The qualifier must refer to a table name or a view name, or to an alias to a table name or view name.

The result is a double-precision floating point number (FLOAT) that is the analogy score. Larger-valued positive results indicate a better analogy than smaller results. If the expressions in *target-1* and *target-2* return the same value, the result of the function is -1 indicating a poor analogy, unless the expressions in *source-1* and *source-2* return values that are very similar (the similarity score ≥ 0.9).

The result can be null; if any argument is null, the result is the null value. If the arguments to the function contain values that were not seen during model training, the result is the null value.

By default, Db2 returns all results from a query. If you want to limit the result set to just the *source-2* argument, specify the WHERE predicate in your query.

Notes

Configuration requirement

SQL Data Insights must be configured in Db2 to use this function.

AI_ANALOGY must not be specified in a CREATE VIEW statement

AI_ANALOGY must not be specified in a CREATE VIEW statement, and must not be specified in the fullselect of a materialized-query-definition of a CREATE TABLE statement.

AI_ANALOGY is not deterministic

The function reads data from the table associated with the model. The AI_ANALOGY function is not deterministic. Training of the model is also not deterministic. The model may change slightly, even if trained again using the same data, which can cause small differences in the analogy scores produced by the function which can affect the ordering of results.

Examples

The customer with ID '1066_JKSGK' has churned. Given the relationship of that customer to 'YES' in the churn column, find customers with the same relationship to 'NO' in the churn column, in other words, customers unlikely to churn.

```
SELECT AI_ANALOGY('YES' USING MODEL COLUMN CHURN,  
                 '1066_JKSGK' USING MODEL COLUMN CUSTOMERID,  
                 'NO' USING MODEL COLUMN CHURN,  
                 CUSTOMERID),  
       CHURN.*  
FROM CHURN  
ORDER BY 1 DESC  
FETCH FIRST 5 ROWS ONLY
```

The *source-2* argument specifies the column for contract terms. If you want to see just the results from *source-2* and the rows where the CONTRACT is "One year," add the WHERE CONTRACT = 'One year' predicate to your query as shown in the following example:

```
SELECT * FROM  
  (SELECT DISTINCT AI_ANALOGY(  
    'Month-to-month' USING MODEL COLUMN CONTRACT,  
    'Electronic check' USING MODEL COLUMN PAYMENTMETHOD,  
    'One year',  
    PAYMENTMETHOD) AS SIMILARITY,  
    CONTRACT, PAYMENTMETHOD  
   FROM ADMF001.CHURN  
   WHERE CONTRACT = 'One year')  
 WHERE SIMILARITY > 0.0  
 ORDER BY SIMILARITY DESC  
 FETCH FIRST 5 ROWS ONLY;
```

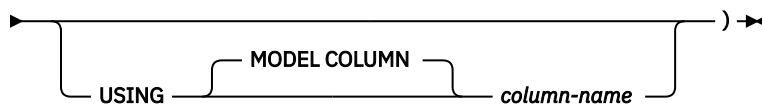
AI_COMMONALITY

The AI_COMMONALITY function computes a similarity score by using the value of the *expression* argument and the centroid value of the model column.

FL 504

The centroid value represents the common behavior of the model column for all the rows in a table. As a result, when a query invokes the AI_COMMONALITY function, the resulting similarity score represents the similarity of the function argument with the common behavior of the model column, computed over all rows in the table, not just the qualified rows.

►► AI_COMMONALITY — (— *expression* —►



The schema is SYSIBM.

The machine learning model that is used to compute the score is determined by the *expression* or the *column-name* specified in the USING MODEL COLUMN clause and the centroid values of the model column. You can use the AI_COMMONALITY function to detect outliers. With -1 as the minimum score, the lower the similarity score is, the further away the argument deviates from the common behavior.

expression

An expression that specifies the value on which the similarity score is computed against the centroid values of the model column. The value returned by *expression* must be a built-in data type (SQLCODE -440, SQLSTATE 42884) that is not binary, LOB, XML, or ROWID (SQLCODE -171, SQLSTATE 42815).

column-name

An identifier that specifies a column to be used as the model column. The identified column determines the machine learning model that is used for evaluating the function.

An argument to the AI_COMMONALITY function must specify a machine learning model and a model column that is used for evaluating the function. The following rules are used to determine the machine learning model and the model column for the argument:

- If the *expression* argument references a standalone column and the USING MODEL COLUMN clause does not specify a model column, the standalone column is used to evaluate the function, and the vector table is used as the model. The vector table is generated from the Db2 table where the standalone column belongs.
- If the *expression* argument references a standalone column, but the USING MODEL COLUMN clause specifies a different column, the column specified in the USING MODEL COLUMN clause is used to evaluate the function, and the vector table is used as the model.

An *expression* other than a standalone column reference must specify a model column by using the USING MODEL COLUMN clause. The model is determined by the table where the model column belongs.

- A model column specified in the function invocation must refer to the columns that belong to the same Db2 table or view (SQLCODE -20579, SQLSTATE 428ID, RC=5).
- AI model training must be enabled for the table with which the model is associated (SQLCODE -20579, SQLSTATE 428ID, RC=1), and the model must be trained (SQLCODE -20579, SQLSTATE 428ID, RC=3). The model column must be included in the model (SQLCODE -20579, SQLSTATE 428ID, RC=2).

A model column, specified either as a standalone column reference or in the USING MODEL COLUMN clause, may be a qualified name. The qualifier must not be a synonym name or a correlation name of a table expression (SQLCODE -20579, SQLSTATE 428ID, RC=6). The qualifier must refer to a table or view, or to an alias of a table or view (SQLCODE -20579, SQLSTATE 428ID, RC=6).

If the model column is a numeric column as indicated during model training, the value of the expression is cast to FLOAT during execution of the function (SQLCODE -420, SQLSTATE 22018, or other SQLCODE/SQLSTATE pairs for cast errors).

The result is a double-precision floating point number (FLOAT) that is the similarity score. The FLOAT is a number between -1.0 and 1.0, where -1.0 represents the minimum common values (outlier) and 1.0 indicates the maximum common values.

The result can be null; if any argument is null, the result is the null value. If the expression evaluates to a value that is not seen during model training, the result is also the null value.

Notes

- You must enable and configure the SQL DI functionality in Db2 to use the AI_COMMONALITY function (SQLCODE -20577, SQLSTATE 0A502, RC=3).
- You cannot specify the AI_COMMONALITY function in a CREATE VIEW statement (SQLCODE -154, SQLSTATE 42909) or in the fullselect of a materialized-query-definition of a CREATE TABLE statement (SQLCODE -20058, SQLSTATE 428EC).
- The AI_COMMONALITY function reads data from the Db2 table that is associated with the model. Neither the function nor the model training is deterministic. The model may change slightly over time even if it is retrained on the same data. This slight change can cause minor differences in the resulting similarity scores and affect the similarity ordering of the results.
- Db2 application compatibility level V13R1M504 is required to use the AI_COMMONALITY function (SQLCODE -20473, SQLSTATE 428HA).
- The AI_COMMONALITY function requires that models be trained after the following z/OS APARs applied:
 - Apply APAR OA64845 on z/OS 3.1 (HZAI310).

- Apply APAR OA64844 on z/OS 2.5 (HZA1250).

The function is not supported on z/OS 2.4.

To use the function on existing models, you must retrain those models after the application of the required APARs. The function returns null if it is executed on a model that was trained without the required APARs applied.

Examples

The following sample SQL statement finds the top five outliers of the model column CUSTOMERID:

```
SELECT AI_COMMONALITY(CUSTOMERID) AS SCORE, C.*
FROM CHURN C
ORDER BY SCORE ASC
FETCH FIRST 5 ROWS ONLY;
```

AI_SIMILARITY

The AI_SIMILARITY function computes a similarity score between two values.

[FL 500](#)

➤ AI_SIMILARITY — (— *expression-1* →

→ USING MODEL COLUMN *column-name* , — *expression-2* →

→ USING MODEL COLUMN *column-name*) ➤

The schema is SYSIBM.

The AI_SIMILARITY function computes a similarity score using the values returned by *expression-1* and *expression-2*. The machine learning model used to compute the score is determined by the columns specified in *expression-1* and *expression-2*, or you can explicitly specify it in the *column-name* in the USING MODEL COLUMN clause.

expression-1

An expression that specifies the first value on which the similarity score is computed. The value returned by *expression-1* must be a built-in data type that is not a binary data type, a LOB data type, XML or ROWID.

expression-2

An expression that specifies the second value on which the similarity score is computed. The value returned by *expression-2* must be a built-in data type that is not a binary data type, a LOB data type, XML or ROWID.

column-name

An identifier that specifies a column to be used as the model column, which determines the machine learning model used to evaluate the function.

Arguments to AI_SIMILARITY must specify a machine learning model and a machine learning model column that is used to evaluate the function. The following rules are used to determine which model and model columns are used for each argument:

- If the expression argument is a standalone column reference and no model column is explicitly specified with the USING MODEL COLUMN clause, the standalone column is the model column that is used to evaluate the function, and the model is the model table associated with the table that the column belongs to.

- If the expression argument is a standalone column reference, but the USING MODEL COLUMN clause specifies a different column, the column specified in the USING MODEL COLUMN clause is the model column, and the model is the model table associated with the table that the model column belongs to.
- Any kind of expression other than a standalone column reference either must specify a model column name using the USING MODEL COLUMN clause, or the model column must be able to be inferred according to these rules:
 - The model column can be explicitly specified by the USING MODEL COLUMN clause. The model is determined by the table that the model column belongs to.
 - The model column of an argument that is not a standalone column-reference and does not have a column specified by the USING MODEL COLUMN clause will be inferred by the model column of the other argument. At least one of the arguments must have a model column.
- All model columns specified in the function invocation must refer to columns that belong to the same table or view.
- AI must be enabled for the table or view that the model is associated with and the model must be trained. All model columns must be included in the model.
- If either expression specifies a model column that was identified during model training as a primary key column, the other expression must specify the same model column or must not specify any model column.

Each model column, either specified as a standalone column reference or with the USING MODEL COLUMN clause, may be a qualified name. The qualifier must not be a synonym name or a correlation name of a table expression. The qualifier must refer to a table name or a view name, or to an alias to a table name or view name.

If the model column is a numeric column as indicated during the training of the model, the value of the expression is cast to DOUBLE during execution of the function. The value of the numeric column is converted into a string token based on the clustering of the value. You can find the minimum values for each cluster in the Model details panel. If the value is outside the range of DOUBLE, the result of the function is a null value.

The result is a double-precision floating point number (FLOAT) that is the similarity score. The result is a number between -1.0 and 1.0, where -1.0 means that the values are least similar, and 1.0 means that they are most similar.

The result can be null; if any argument is null, the result is the null value. If *expression-1* or *expression-2* evaluates to a value that was not seen during model training, and the model column used is a categorical one as indicated during the training of the model, the result is the null value. This does not apply when the model column is a numeric one.

Notes

Configuration requirement

SQL Data Insights must be configured in Db2 to use this function.

AI_SIMILARITY must not be specified in a CREATE VIEW statement

AI_SIMILARITY must not be specified in a CREATE VIEW statement, and must not be specified in the fullselect of a materialized-query-definition of a CREATE TABLE statement.

AI_SIMILARITY is not deterministic

The AI_SIMILARITY function reads data from the table associated with the model. The function is not deterministic. Training of the model is also not deterministic. The model may change slightly, even if trained again using the same data, which can cause small differences in the similarity scores produced by the function. This can affect the similarity ordering of results.

Examples

Find the top five customers by ID most similar to the customer with ID '3668-QPYBK'.

```
SELECT AI_SIMILARITY(CUSTOMERID, '3668-QPYBK'), CHURN.*
FROM CHURN
ORDER BY 1 DESC
FETCH FIRST 5 ROWS ONLY;
```

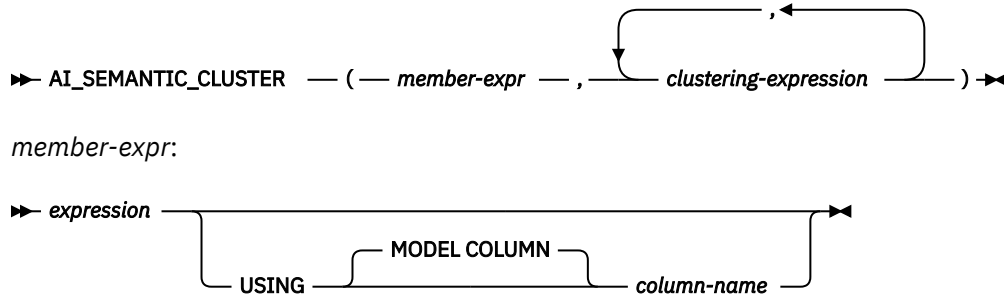
Find the top three payment methods most similar to 'YES' in the CHURN column.

```
SELECT DISTINCT AI_SIMILARITY(PAYMENTMETHOD, 'YES' USING MODEL COLUMN CHURN),
PAYMENTMETHOD
FROM CHURN
ORDER BY 1 DESC
FETCH FIRST 3 ROWS ONLY
```

AI_SEMANTIC_CLUSTER

The AI_SEMANTIC_CLUSTER function computes a semantic clustering score of a member argument against a set of clustering arguments.

[FL 500](#)



The schema is SYSIBM.

The AI_SEMANTIC_CLUSTER function computes a clustering score using the value returned by *member-expr* among the cluster formed by the values returned by *clustering-expression* arguments. The machine learning model and machine learning model column is determined by the column in *member-expr*, or you can specify it in the *column-name* in the USING MODEL COLUMN clause.

member-expr

An expression that specifies the value to be scored against the rest of the cluster. The value must be a built-in data type that is not a binary data type, a LOB data type, XML, or ROWID.

The *member-expr* determines which model and model column is used for the function:

- If the *member-expr* is a standalone column reference and no model column is explicitly specified with the USING MODEL COLUMN clause, the standalone column is the model column that is used to evaluate the function, and the model is the model table associated with the table that the column belongs to.
- The USING MODEL COLUMN clause can be used to specify a column for the *member-expr*. The column named in the USING MODEL COLUMN clause is the model column, and the model is the model table associated with the table that the model column belongs to.
- Db2 must be able to determine a model column from the *member-expr*, either because the expression is a standalone column reference, or because a column is specified in the USING MODEL COLUMN clause.
- AI must be enabled for the table or view that the model is associated with and the model must be trained. All model columns must be included in the model.

The model column, either specified as a standalone column reference or with the USING MODEL COLUMN clause, may be a qualified name. The qualifier must not be a synonym name or a correlation name of a table expression. The qualifier must refer to a table name or a view name, or to an alias to a table name or view name.

If the model column is a numeric column as indicated during the training of the model, the value of the expression and the values of each *clustering-expression* are cast to FLOAT during execution of the function. If the value is outside of the range of FLOAT, the result of the function is a null value.

clustering-expression

The values returned by the set of *clustering-expressions* form a cluster against which the *member-expr* is scored. The values must each be a built-in data type that is not a binary data type, a LOB data type, XML, or ROWID.

The model and model column of the *clustering-expression* argument is inferred to be the same as the model and model column of the *member-expr*.

Up to three *clustering-expression* arguments may be specified for AI_SEMANTIC_CLUSTER.

The result is a double-precision floating point number (FLOAT) between -1.0 and 1.0 that is the semantic clustering score. A larger positive result indicates a better clustering of *member-expr* among the cluster formed by *clustering-expressions* than a lower result.

The result can be null; if any argument is null, the result is the null value. If the arguments to the function contain values that were not seen during model training, and the model column is trained as categorical, the result is the null value.

Notes

Configuration requirement

SQL Data Insights must be configured in Db2 to use this function.

AI_SEMANTIC_CLUSTER must not be specified in a CREATE VIEW statement

AI_SEMANTIC_CLUSTER must not be specified in a CREATE VIEW statement, and must not be specified in the fullselect of a materialized-query-definition of a CREATE TABLE statement.

AI_SEMANTIC_CLUSTER is not deterministic

The function reads data from the table associated with the model. The AI_SEMANTIC_CLUSTER function is not deterministic. Training of the model is also not deterministic. The model may change slightly, even if trained again using the same data, which can cause small differences in the semantic cluster scores produced by the function which can affect the ordering of results.

Example

Customers with IDs '0280_XJGEX', '6467_CHFZW' and '0093_XWZFY' have all churned. If we form a semantic cluster of those three customers, find the top 5 customers that would belong in that cluster.

```
SELECT AI_SEMANTIC_CLUSTER(CUSTOMERID, '0280_XJGEX', '6467_CHFZW', '0093_XWZFY'), CHURN.*
FROM CHURN
ORDER BY 1 DESC
FETCH FIRST 5 ROWS ONLY
```

Chapter 9. Db2 SQL statements for SQL DI

Db2 updates the following SQL statements to support SQL DI. Make sure that you are aware of pertinent restrictions when you use these SQL statements.

CREATE FUNCTION (sourced)

This CREATE FUNCTION statement registers a user-defined function that is based on an existing scalar or aggregate function with a database server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

The privilege set defined below must include at least one of the following:

- The CREATEIN privilege on the schema
- SYSADM or SYSCTRL authority
- System DBADM
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

If the authorization ID that is used to create the function has the installation SYSADM authority or the installation SYSOPR authority and if the current SQLID is set to SYSINSTL, the function is identified as system-defined function.

Additional privileges are required for the source function, and other privileges are also needed if the function uses a table as a parameter, or refers to a distinct type. These privileges are:

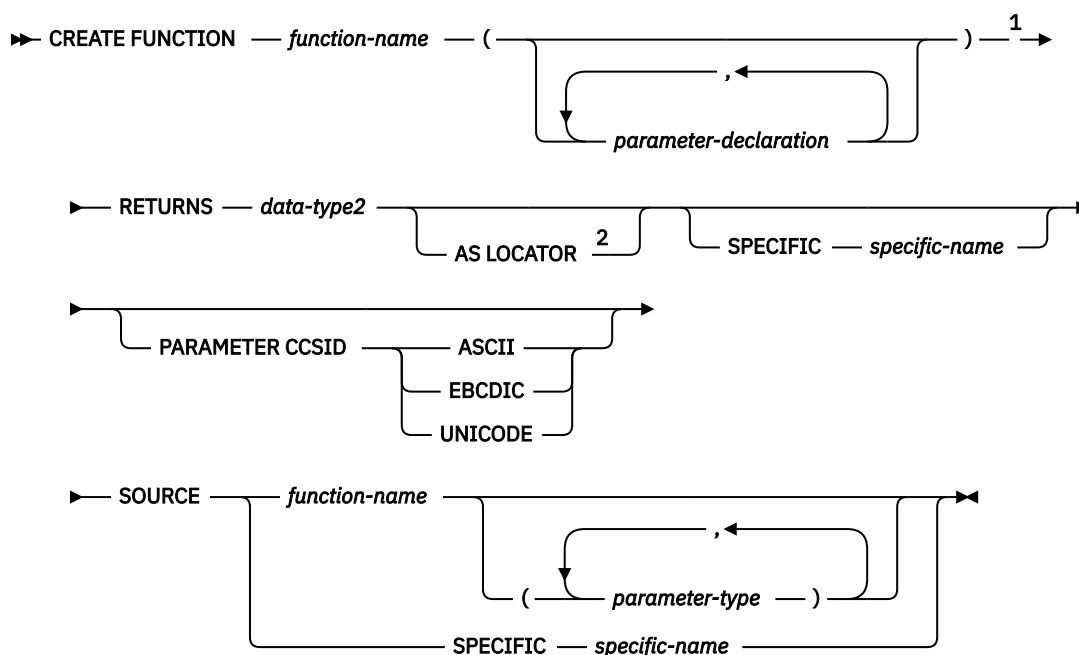
- The EXECUTE privilege for the function that the SOURCE clause references.
- The SELECT privilege on any table that is an input parameter to the function.
- The USAGE privilege on each distinct type that the function references.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the plan or package. If the owner is a role, the implicit schema match does not apply and this role needs to include one of the previously listed conditions.

If the statement is dynamically prepared and is not running in a trusted context for which the ROLE AS OBJECT OWNER clause is specified, the privilege set is the set of privileges that are held by the SQL authorization ID of the process. If the schema name is not the same as the SQL authorization ID of the process, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

Syntax

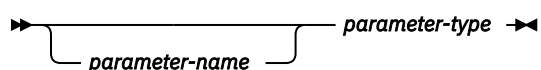


Notes:

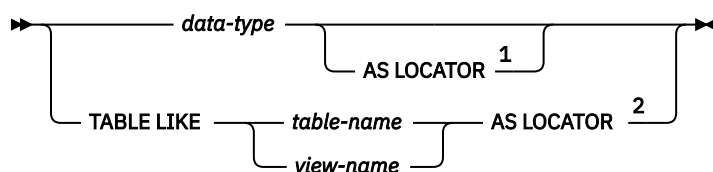
¹ RETURNS, SPECIFIC, and SOURCE can be specified in any order.

² AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

parameter-declaration:



parameter-type:

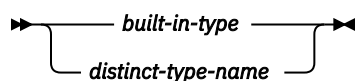


Notes:

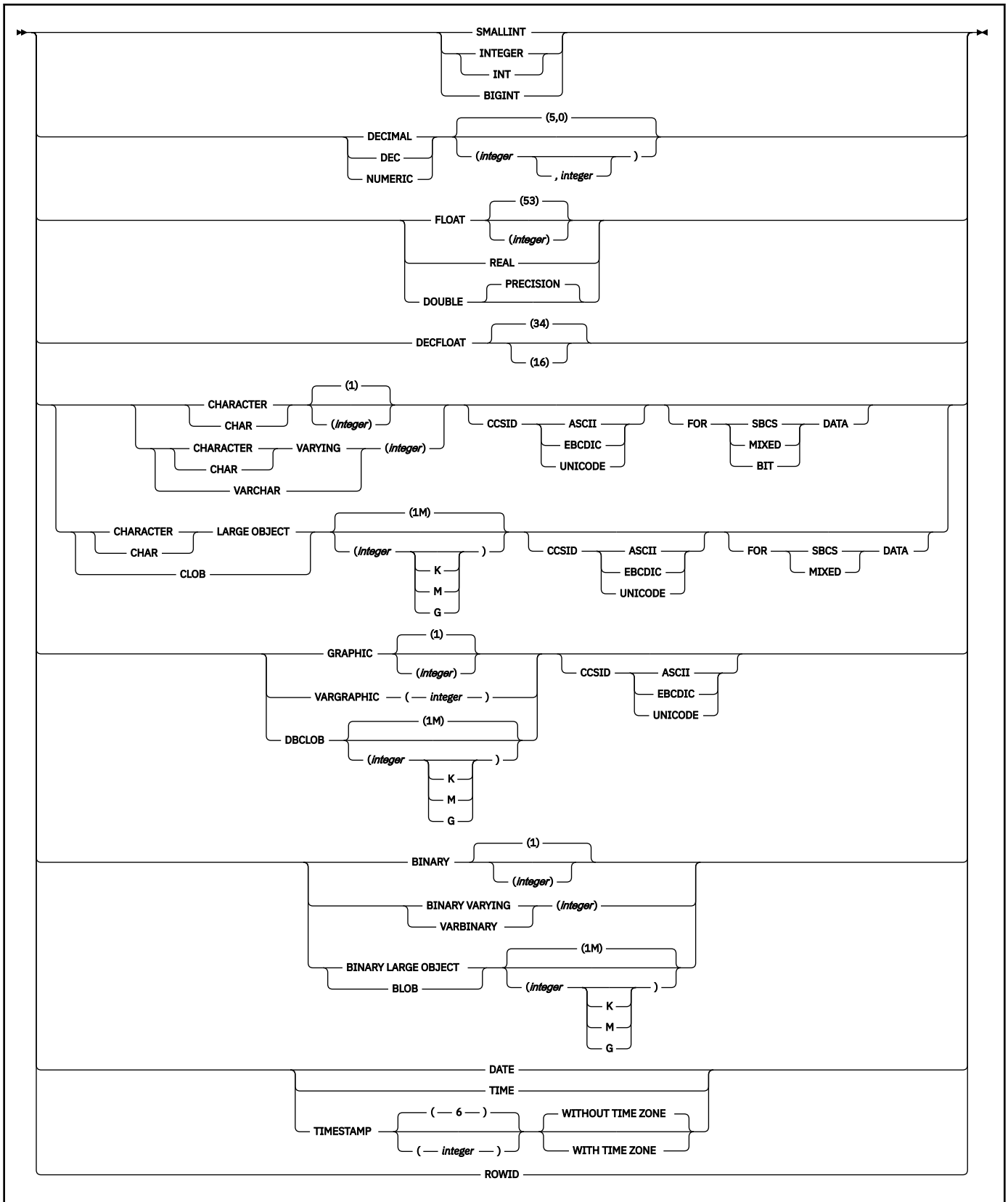
¹ AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

² The TABLE LIKE name AS LOCATOR clause can only be specified for the parameter list of the function that is being defined.

data-type:



built-in-type:



Description

function-name

Names the user-defined function. The name is implicitly or explicitly qualified by a schema name. For more information, see "Choosing the schema and function names" and "Determining the uniqueness of functions in a schema" in [CREATE FUNCTION \(Db2 SQL\)](#).

(parameter-declaration,...)

Specifies the number of input parameters of the function and the data type of each parameter. All of the parameters for a function are input parameters and are nullable. There must be one entry in the list for each parameter that the function expects to receive. Although not required, you can give each parameter a name.

A function can have no parameters. In this case, you must code an empty set of parentheses, for example:

```
CREATE FUNCTION WOOFER()
```

parameter-name

Specifies the name of the input parameter. The name is an SQL identifier, and each name in the parameter list must not be the same as any other name.

data-type

Specifies the data type of the input parameter. The data type can be a built-in data type or a distinct type.

built-in-type

The data type of the input parameter is a built-in data type.

For information on the data types, see [built-in-type](#).

For parameters with a character or graphic data type, the PARAMETER CCSID clause or CCSID clause indicates the encoding scheme of the parameter. If you do not specify either of these clauses, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

distinct-type-name

The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type.

The implicitly or explicitly specified encoding scheme of all of the parameters with a character or graphic string data type must be the same—either all ASCII, all EBCDIC, or all UNICODE.

Although parameters with a character data type have an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the function program can receive character data of any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the function is invoked.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format.

The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

- A distinct type parameter is passed as the source type of the distinct type.

You can specify any built-in data type or distinct type that matches or can be cast to the data type of the corresponding parameter of the source function (the function that is identified in the SOURCE clause). (For information on casting data types, see [Casting between data types \(Db2 SQL\)](#).) Length, precision, or scale attributes do not have to be specified for data types with these attributes. When specifying data types with these attributes, follow these rules:

- An empty set of parentheses can be used to indicate that the length, precision, or scale is the same as the source function.
- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default values are used.

AS LOCATOR

Specifies that a locator to the value of the parameter is passed to the function instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB data type. Passing locators instead of values can result in fewer bytes being passed to the function, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted, nor does it affect the function signature, which is used in function resolution.

TABLE LIKE *table-name* or *view-name* AS LOCATOR

Specifies that the parameter is a transition table. However, when the function is invoked, the actual values in the transition table are not passed to the function. A single value is passed instead. This single value is a locator to the table, which the function uses to access the columns of the transition table. A function with a table parameter can only be invoked from the triggered action of a trigger.

The use of TABLE LIKE provides an implicit definition of the transition table. It specifies that the transition table has the same number of columns as the identified table or view. If a table is specified, the transition table includes columns that are defined as implicitly hidden in the table. The columns have the same data type, length, precision, scale, subtype, and encoding scheme as the identified table or view, as they are described in catalog tables SYSCOLUMNS and SYSTABLESPACE. The number of columns and the attributes of those columns are determined at the time the CREATE FUNCTION statement is processed. Any subsequent changes to the number of columns in the table or the attributes of those columns do not affect the parameters of the function.

table-name or *view-name* must identify a table or view that exists at the current server. A view cannot have columns of length 0. The name must not identify a declared temporary table. The table that is identified can contain XML columns; however, the function cannot reference those XML columns. The name does not have to be the same name as the table that is associated with the transition table for the trigger. An unqualified table or view name is implicitly qualified according to the following rules:

- If the CREATE FUNCTION statement is embedded in a program, the implicit qualifier is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not used, the implicit qualifier is the owner of the plan or package.
- If the CREATE FUNCTION statement is dynamically prepared, the implicit qualifier is the SQL authorization ID in the CURRENT SCHEMA special register.

When the function is invoked, the corresponding columns of the transition table identified by the table locator and the table or view identified in the TABLE LIKE clause must have the same definition. The data type, length, precision, scale, and encoding scheme of these columns must match exactly. The description of the table or view at the time the CREATE FUNCTION statement was executed is used.

Additionally, a character FOR BIT DATA column of the transition table cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is not defined as character FOR BIT DATA. (The definition occurs with the CREATE FUNCTION statement.) Likewise, a character column of the transition table that is not FOR BIT DATA cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is defined as character FOR BIT DATA.

For more information about using table locators, see [Accessing transition tables in a user-defined function or stored procedure \(Db2 Application programming and SQL\)](#).

RETURNS

Identifies the output of the function.

data-type2

Specifies the data type of the output. The output is nullable.

You can specify any built-in data type or distinct type that can be cast from the data type of the result of the source function. (For information on casting data types, see [Casting between data types \(Db2 SQL\)](#).)

AS LOCATOR

Specifies that the function returns a locator to the value rather than the actual value. You can specify AS LOCATOR only if the output from the function has a LOB data type or a distinct type based on a LOB data type.

SPECIFIC *specific-name*

Provides a unique name for the function. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another function that exists at the current server.

The unqualified form of *specific-name* is an SQL identifier. The qualified form is an SQL identifier (the schema name) followed by a period and an SQL identifier.

If you do not specify a schema name, it is the same as the explicit or implicit schema name of the function name (*function-name*). If you specify a schema name, it must be the same as the explicit or implicit schema name of the function name.

If you do not specify the SPECIFIC clause, the default specific name is the name of the function. However, if the function name does not provide a unique specific name or if the function name is a single asterisk, Db2 generates a specific name in the form of:

```
SQLxxxxxxxxxxxxx
```

where 'xxxxxxxxxxxxx' is a string of 12 characters that make the name unique.

The specific name is stored in the SPECIFIC column of the SYSROUTINES catalog table. The specific name can be used to uniquely identify the function in several SQL statements (such as ALTER FUNCTION, COMMENT, DROP, GRANT, and REVOKE) and in Db2 commands (START FUNCTION, STOP FUNCTION, and DISPLAY FUNCTION). However, the function cannot be invoked by its specific name.

PARAMETER CCSID

Indicates whether the encoding scheme for character and graphic string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value specified in the CCSID clauses of the parameter list or RETURNS clause, or in the field DEF ENCODING SCHEME on installation panel DSNTIPF.

This clause provides a convenient way to specify the encoding scheme for character and graphic string parameters. If individual CCSID clauses are specified for individual parameters in addition to this PARAMETER CCSID clause, the value specified in *all* of the CCSID clauses must be the same value that is specified in this clause.

This clause also specifies the encoding scheme to be used for system-generated parameters of the routine such as message tokens and DBINFO.

SOURCE

Specifies that the new function is being defined as a sourced function. A *sourced function* is implemented by another function (the *source function*). The source function must be a scalar or aggregate function that exists at the current server, and it must be one of the following types of functions:

- A function that was defined with a CREATE FUNCTION statement
- A cast function that was generated by a CREATE TYPE statement for a distinct type
- A built-in function

If the source function is not a built-in function, the particular function can be identified by its name, function signature, or specific name.

If the source function is a built-in function, the SOURCE clause must include a function signature for the built-in function.

The source function must not be any of the built-in functions (if a particular syntax is shown, only the indicated form cannot be specified):

- AI_ANALOGY
- AI_COMMONALITY
- AI_SEMANTIC_CLUSTER
- AI_SIMILARITY
- ARRAY_AGG
- ARRAY_DELETE
- ARRAY_FIRST
- ARRAY_LAST
- ARRAY_NEXT
- ARRAY_PRIOR
- CARDINALITY
- CHAR(*datetime-expression*, *second-argument*) where *second-argument* is ISO, USA, EUR, JIS, or LOCAL or if CHAR is specified with OCTETS, CODEUNITS16, or CODEUNITS32.
- CHARACTER_LENGTH
- CLOB if OCTETS, CODEUNITS16, or CODEUNITS32 is specified
- COALESCE if a parameter is an array
- COUNT(*)
- COUNT_BIG(*)
- CUME_DIST
- CUME_DIST (aggregate)
- DBCLOB if OCTETS, CODEUNITS16, or CODEUNITS32 is specified
- DECODE
- DECRYPT_BIT where the second argument is DEFAULT
- DECRYPT_CHAR where the second argument is DEFAULT
- DECRYPT_DB where the second argument is DEFAULT
- DECRYPT_DATAKEY_BIGINT
- DECRYPT_DATAKEY_BIT
- DECRYPT_DATAKEY_CLOB
- DECRYPT_DATAKEY_DBCLOB
- DECRYPT_DATAKEY_DECIMAL
- DECRYPT_DATAKEY_INTEGER
- DECRYPT_DATAKEY_VARCHAR
- DECRYPT_DATAKEY_VARGRAPHIC
- ENCRYPT_DATAKEY
- EXTRACT
- FIRST_VALUE
- GETVARIABLE where the second argument is DEFAULT
- GRAPHIC if OCTETS, CODEUNITS16, or CODEUNITS32 is specified, or if the first argument is numeric
- IFNULL if a parameter is an array

- INSERT if OCTETS, CODEUNITS16, or CODEUNITS32 is specified
- LAG
- LAST_VALUE
- LEAD
- LEFT if OCTETS, CODEUNITS16, or CODEUNITS32 is specified
- LISTAGG
- LOCAL
- LOCATE if OCTETS, CODEUNITS16, or CODEUNITS32 is specified
- MAX
- MAX_CARDINALITY
- MIN
- NTH_VALUE
- NTILE
- NULLIF
- PERCENT_RANK
- PERCENT_RANK (aggregate)
- POSITION
- RATIO_TO_REPORT
- REGEXP_COUNT
- REGEXP_INSTR
- REGEXP_LIKE
- REGEXP_REPLACE
- REGEXP_SUBSTR
- RID
- RIGHT if OCTETS, CODEUNITS16, or CODEUNITS32 is specified
- STRIP where multiple arguments are specified
- SUBSTRING
- TRIM where the first argument is BOTH, B, LEADING, L, TRAILING, T, or the first or second argument is FROM
- TRIM_ARRAY
- VARCHAR if OCTETS, CODEUNITS16, or CODEUNITS32 is specified
- VARGRAPHIC if OCTETS, CODEUNITS16, or CODEUNITS32 is specified, or if the first argument is numeric.
- XMLAGG
- XMLCONCAT
- XMLELEMENT
- XMLFOREST
- XMLNAMESPACES

If you base the sourced function directly or indirectly on an external scalar function, the sourced function inherits the attributes of the external scalar function. This can involve several layers of sourced functions. For example, assume that function A is sourced on function B, which in turn is sourced on function C. Function C is an external scalar function. Functions A and B inherit all of the attributes that are specified on the EXTERNAL clause of the CREATE FUNCTION statement for function C.

function-name

Identifies the function that is to be used as the source function. The source function can be defined with any number of parameters. If more than one function is defined with the specified name in the specified or implicit schema, an error is returned.

If you specify an unqualified *function-name*, Db2 searches the schemas of the SQL path. Db2 selects the first schema that has only one function with this name on which the user has EXECUTE authority. An error is returned if a function is not found or a schema has more than one function with this name.

function-name (parameter-type,...)

Identifies the function that is to be used as the source function by its function signature, which uniquely identifies the function. The *function-name (parameter-type,...)* must identify a function with the specified signature. The specified parameters must match the data types in the corresponding position that were specified when the function was created. Db2 uses the number of data types and the logical concatenation of the data types to identify the specific function instance. Synonyms for data types are considered a match.

If the function was defined with a table parameter (the LIKE TABLE *name* AS LOCATOR clause was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), the function signature cannot be used to uniquely identify the function. Instead, use one of the other syntax variations to identify the function with its function name, if unique, or its specific name.

If *function-name()* is specified, the identified function must have zero parameters.

function-name

Identifies the function name of the source function. If you specify an unqualified name, Db2 searches the schemas of the SQL path. Otherwise, Db2 searches for the function in the specified schema.

parameter-type,...

Identifies the parameters of the function.

If an unqualified distinct type name is specified, Db2 searches the SQL path to resolve the schema name for the distinct type.

Empty parentheses are allowed for some data types that are specified in this context. For data types that have a length, precision, or scale attribute, use one of the following specifications:

- Empty parentheses indicate that Db2 ignores the attribute when determining whether the data types match. For example, DEC() is considered a match for a parameter of a function that is defined with a data type of DEC(7,2). However, FLOAT cannot be specified with empty parentheses because its parameter value indicates a specific data type (REAL or DOUBLE).
- If a specific value for a length, precision, or scale attribute is specified, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. If the data type is FLOAT, the precision does not need to exactly match the value that was specified because matching is based on the data type (REAL or DOUBLE).
- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default attributes of the data type are implied. The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement.

If you omit the FOR *subtype* DATA clause or the CCSID clause for data types with a subtype or encoding scheme attribute, Db2 is to ignore the attribute when determining whether the data types match. An exception to ignoring the attribute is FOR BIT DATA. A character FOR BIT DATA parameter of the new function cannot correspond to a parameter of the source function that is not defined as character FOR BIT DATA. Likewise, a character parameter of the new function that is not FOR BIT DATA cannot correspond to a parameter of the source function that is defined as character FOR BIT DATA.

The number of input parameters in the function that is being created must be the same as the number of parameters in the source function. If the data type of each input parameter is

not the same as or castable to the corresponding parameter of the source function, an error occurs. The data type of the final result of the source function must match or be castable to the result of the sourced function.

AS LOCATOR

Specifies that the function is defined to receive a locator for this parameter. If AS LOCATOR is specified, the data type must be a LOB or distinct type that is based on a LOB.

SPECIFIC *specific-name*

Identifies the function to be used as the source function by its specific name.

If you specify an unqualified *specific-name*, Db2 searches the SQL path to locate the schema. Db2 selects the first schema that contains a function with this specific name for which the user has EXECUTE authority. Db2 returns an error if it cannot find a function with the specific name in one of the schemas in the SQL path.

If you specify a qualified *specific-name*, Db2 searches the named schema for the function. Db2 returns an error if it cannot find a function with the specific name.

Notes

Considerations for all types of user-defined functions:

For considerations that apply to all types of user-defined functions, see [CREATE FUNCTION \(Db2 SQL\)](#).

Owner privileges for sourced functions:

For sourced functions, the owner is authorized to execute the function (EXECUTE privilege) in the following cases:

- If the underlying function is a user-defined function, and the owner is authorized with the grant option to execute the underlying function, the privilege on the new function includes the grant option. Otherwise, the owner can execute the new function but cannot grant others the privilege to do so.
- If the underlying function is a built-in function, the owner is authorized with the grant option to execute the underlying built-in function and the privilege on the new function includes the grant option.

For more information, see [GRANT \(function or procedure privileges\) \(Db2 SQL\)](#). For more information about ownership of the object, see [Authorization, privileges, permissions, masks, and object ownership \(Db2 SQL\)](#).

Rules for creating sourced functions:

Assume that the function that is being created is named NEWF and the source function is named SOURCEF. Consider the following rules when creating a sourced function:

- The unqualified names of the sourced function and source function can be different (NEWF and SOURCEF).
- The number of input parameters for NEWF and SOURCEF must be the same.
- When specifying the input parameters and output for NEWF, you can specify a value for the precision, scale, subtype, or encoding scheme for a data type with any of these attributes or use empty parentheses.

Empty parentheses, such as VARCHAR(), indicate that the value of the attribute is the same as the attribute for the corresponding parameter of SOURCEF, or that is determined by data type promotion. If you specify any values for the attributes, Db2 checks the values against the corresponding input parameters and returned output of SOURCEF as described next.

- When the CREATE FUNCTION statement is executed, Db2 checks the input parameters of NEWF against those of SOURCEF. The data type of each input parameter of NEWF function must be either the same as, or promotable to, the data type of the corresponding parameter of SOURCEF. (For information on the promotion of data types, see [Casting between data types \(Db2 SQL\)](#).)

This checking does not guarantee that an error will not occur when NEWF is invoked. For example, an argument that matches the data type and length or precision attributes of a NEWF parameter might not be promotable if the corresponding SOURCEF parameter has a shorter length or less precision. In general, do not define the parameters of a sourced function with length or precision attributes that are greater than the attributes of the corresponding parameters of the source function.

- When the CREATE FUNCTION statement is executed, Db2 checks the data type identified in the RETURNS clause of NEWF against the data type that SOURCEF returns. The data type that SOURCEF returns must be either the same as, or promotable to, the RETURNS data type of NEWF.

This checking does not guarantee that an error will not occur when NEWF is invoked. For example, the value of a result that matches the data type and length or precision attributes of those specified for the SOURCEF result might not be promotable if the RETURNS data type of NEWF has a shorter length or less precision. Consider the possible effects of defining the RETURNS data type of a sourced function with length or precision attributes that are less than the attributes defined for the data type returned by source function.

Secure functions:

The sourced user-defined function inherits the SECURED or NOT SECURED attribute from the source function in which only the topmost user-defined function is considered. If the topmost user-defined function is secure, any nested user-defined functions are also considered secure. Db2 does not validate whether those nested user-defined functions are secure. If those nested functions can access sensitive data, the security administrator needs to ensure that those functions are allowed to access sensitive data and should ensure that a change control audit procedure has been established for all changes to those functions.

If the sourced function is defined with the VERIFY_GROUP_FOR_USER or VERIFY_ROLE_FOR_USER function as its source, the sourced function must specify only two input parameters.

Examples

Example 1

Assume that you created a distinct type HATSIZE, which you based on the built-in data type INTEGER. You want to have an AVG function to compute the average hat size of different departments. Create a sourced function that is based on built-in function AVG.

```
CREATE FUNCTION AVE (HATSIZE) RETURNS HATSIZE
SOURCE SYSIBM.AVG (INTEGER);
```

When you created distinct type HATSIZE, two cast functions were generated, which allow HATSIZE to be cast to INTEGER for the argument and INTEGER to be cast to HATSIZE for the result of the function.

Example 2

After Smith registered the external scalar function CENTER in his schema, you decide that you want to use this function, but you want it to accept two INTEGER arguments instead of one INTEGER argument and one FLOAT argument. Create a sourced function that is based on CENTER.

```
CREATE FUNCTION MYCENTER (INTEGER, INTEGER)
RETURNS FLOAT
SOURCE SMITH.CENTER (INTEGER, FLOAT);
```

Related concepts

[Sourced functions \(Db2 Application programming and SQL\)](#)

[Naming conventions \(Db2 SQL\)](#)

Related tasks

[Creating a user-defined function \(Db2 Application programming and SQL\)](#)

CREATE FUNCTION (inlined SQL scalar)

The CREATE FUNCTION (inlined SQL scalar) statement defines an SQL scalar function at the current server and specifies an SQL procedural language RETURN statement for the body of the function. The function returns a single value each time it is invoked.

A package is not created for an inlined SQL scalar function. The function is not invoked as part of a query; instead, the *expression* in the RETURN statement of the function is copied (inlined) into the query itself.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

The privilege set defined below must include at least one of the following:

- The CREATEIN privilege on the schema
- SYSADM or SYSCTRL authority
- System DBADM
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

If the authorization ID that is used to create the function has the installation SYSADM authority or the installation SYSOPR authority and if the current SQLID is set to SYSINSTL, the function is identified as system-defined function.

If a user-defined type is referenced (as the data type of a parameter), the privilege set must also include at least one of the following:

- Ownership of the user-defined type
- The USAGE privilege on the user-defined type
- SYSADM authority

At least one of the following additional privileges is required if the SECURED option is specified:

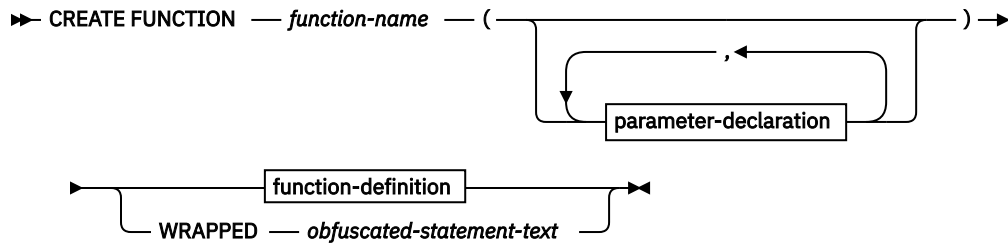
- SECADM authority
- CREATE_SECURE_OBJECT privilege

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the plan or package. If the owner is a role, the implicit schema match does not apply and this role needs to include one of the previously listed conditions.

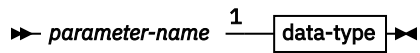
If the statement is dynamically prepared and is not running in a trusted context for which the ROLE AS OBJECT OWNER clause is specified, the privilege set is the set of privileges that are held by the SQL authorization ID of the process. If the schema name is not the same as the SQL authorization ID of the process, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

Syntax



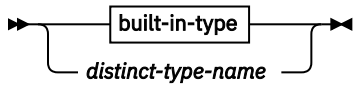
parameter-declaration:



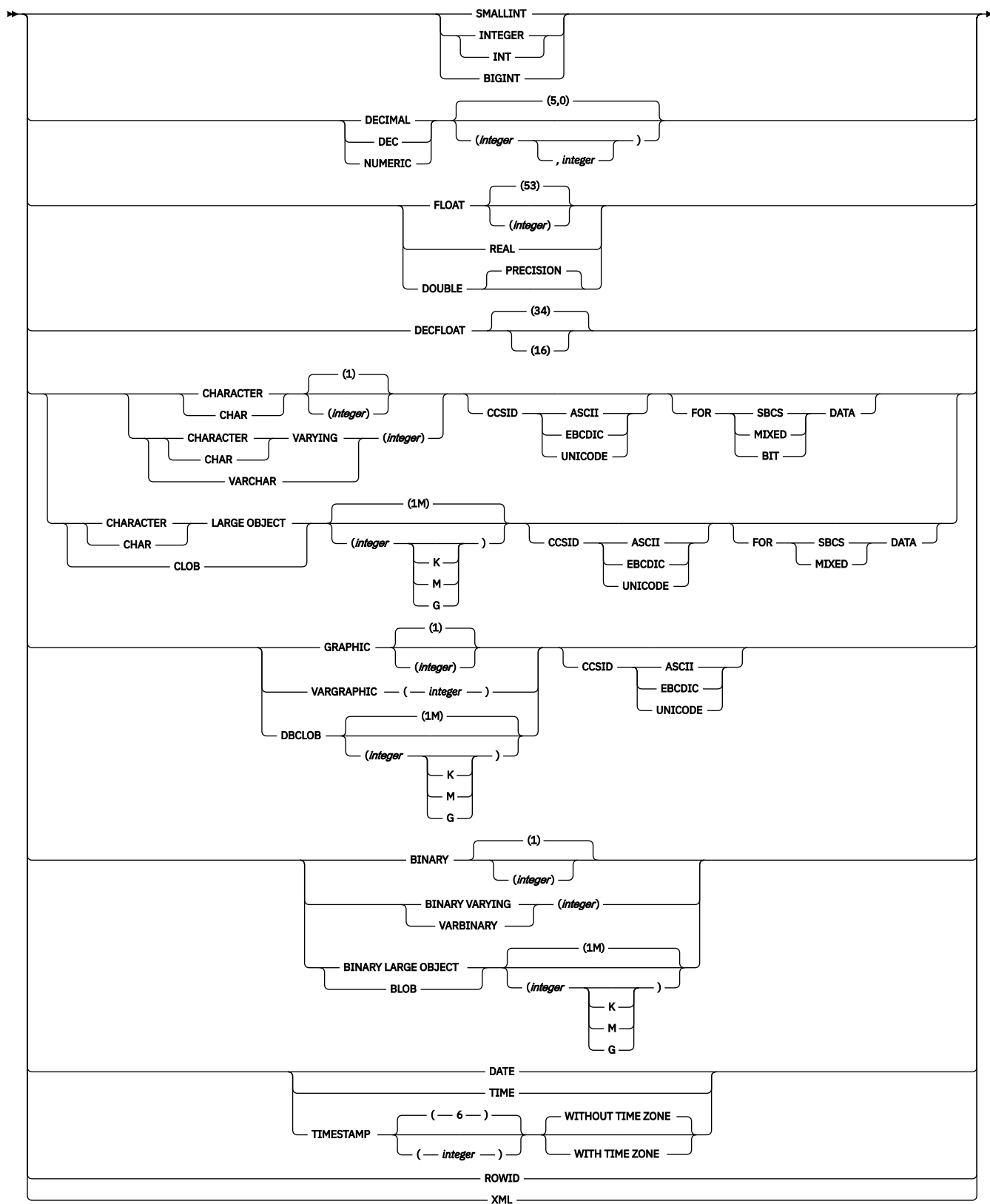
Notes:

¹ Note that the `parameter-name` is required for SQL functions.

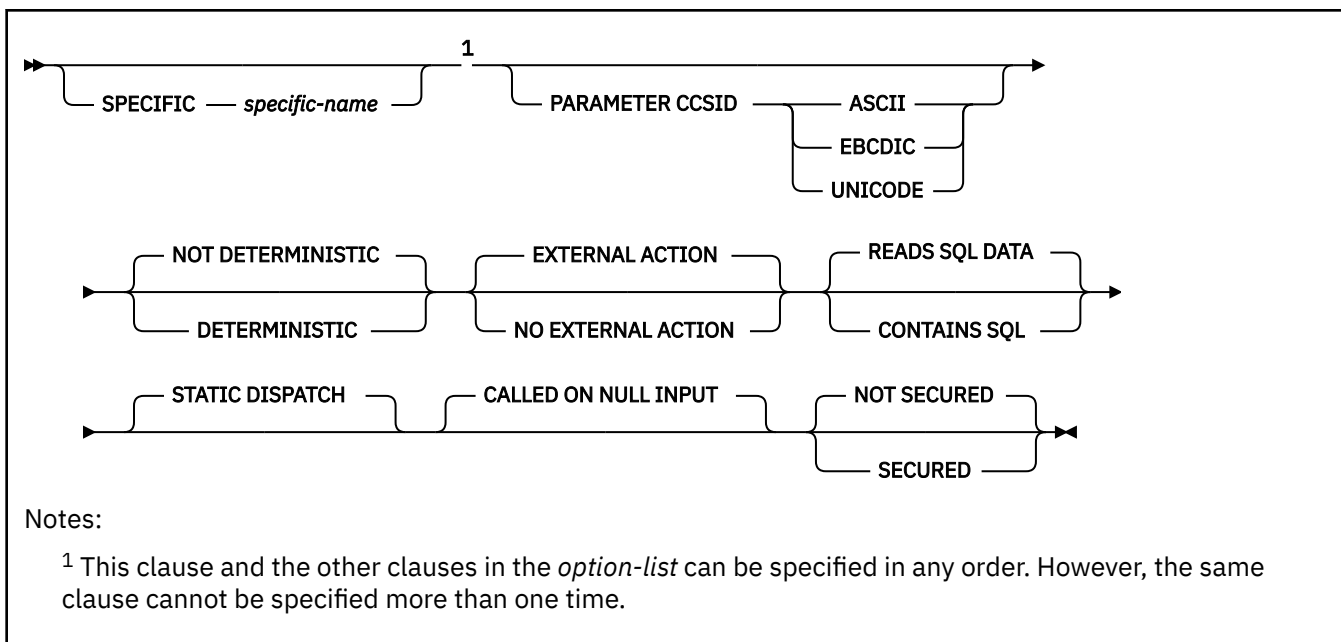
data-type:



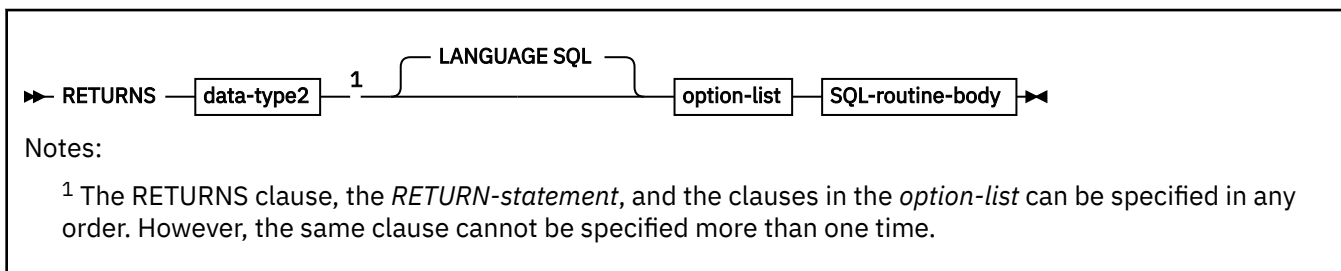
built-in-type:



option-list:



function-definition



SQL-routine-body

RETURN statement

Description

function-name

Names the user-defined function. The name is implicitly or explicitly qualified by a schema name. For more information, see "Choosing the schema and function names" and "Determining the uniqueness of functions in a schema" in [CREATE FUNCTION \(Db2 SQL\)](#).

(parameter-declaration,...)

Specifies the number of input parameters of the function and the name and data type of each parameter. Each *parameter-declaration* specifies an input parameter for the function. A function can have zero or more input parameters. There must be one entry in the list for each parameter that the function expects to receive. All of the parameters for a function are input parameters and are nullable. If the function has more than 30 parameters, only the first 30 parameters are used to determine if the function is unique.

parameter-name

Specifies the name of the input parameter. The name is an SQL identifier, and each name in the parameter list must not be the same as any other name.

data-type

Specifies the data type of the input parameter. The data type can be a built-in data type or a user-defined type.

built-in-type

The data type of the input parameter is a built-in data type.

For information on the data types, see "built-in-type" in [CREATE TABLE \(Db2 SQL\)](#).

For parameters with a character or graphic data type, the PARAMETER CCSID clause or CCSID clause indicates the encoding scheme of the parameter. If you do not specify either of these clauses, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

distinct-type-name

The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type. The distinct type must not be based on a LOB data type.

If you specify the name of the distinct type without a schema name, Db2 resolves the distinct type by searching the schemas in the SQL path.

The implicitly or explicitly specified encoding scheme of all of the parameters with a character or graphic string data type must be the same—either all ASCII, all EBCDIC, or all UNICODE.

Although parameters with a character data type have an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the function program can receive character data of any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the function is invoked. An error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format.

The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

- A distinct type parameter is passed as the source type of the distinct type.

RETURNS

Identifies the output of the function.

data-type2

Specifies the data type of the output. The output is nullable.

The same considerations that apply to the data type of input parameter, as described under "data-type" in ["data-type" on page 65](#), apply to the data type of the output of the function.

LANGUAGE SQL

Specifies that the function is written exclusively in SQL.

SPECIFIC specific-name

Specifies a unique name for the function. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another function that exists at the current server.

The unqualified form of *specific-name* is an SQL identifier. The qualified form is an SQL identifier (the schema name) followed by a period and an SQL identifier.

If you do not specify a schema name, it is the same as the explicit or implicit schema name of the function name (*function-name*). If you specify a schema name, it must be the same as the explicit or implicit schema name of the function name.

If you do not specify the **SPECIFIC** clause, the default specific name is the name of the function. However, if the function name does not provide a unique specific name or if the function name is a single asterisk, Db2 generates a specific name in the form of:

```
SQLxxxxxxxxxxxx
```

where 'xxxxxxxxxxxx' is a string of 12 characters that make the name unique.

The specific name is stored in the **SPECIFIC** column of the **SYSROUTINES** catalog table. The specific name can be used to uniquely identify the function in several SQL statements (such as **ALTER FUNCTION**, **COMMENT**, **DROP**, **GRANT**, and **REVOKE**) and must be used in Db2 commands (**START FUNCTION**, **STOP FUNCTION**, and **DISPLAY FUNCTION**). However, the function cannot be invoked by its specific name.

PARAMETER CCSID

Indicates whether the encoding scheme for character and graphic string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value specified in the **CCSID** clauses of the parameter list or **RETURNS** clause, or in the field **DEF ENCODING SCHEME** on installation panel **DSNTIPF**.

This clause provides a convenient way to specify the encoding scheme for character and graphic string parameters. If individual **CCSID** clauses are specified for individual parameters in addition to this **PARAMETER CCSID** clause, the value specified in *all* of the **CCSID** clauses must be the same value that is specified in this clause.

This clause also specifies the encoding scheme to be used for system-generated parameters of the routine such as message tokens and **DBINFO**.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the function returns the same results each time that the function is invoked with the same input arguments.

NOT DETERMINISTIC

The function might not return the same result each time that the function is invoked with the same input arguments. The function depends on some state values that affect the results. Db2 uses this information to disable the merging of views and table expressions when processing **SELECT** and SQL data change statements that refer to this function. An example of a function that is not deterministic is one that generates random numbers.

NOT DETERMINISTIC must be specified explicitly or implicitly if the function program accesses a special register or invokes another function that is not deterministic. **NOT DETERMINISTIC** is the default.

DETERMINISTIC

The function always returns the same result function each time that the function is invoked with the same input arguments. An example of a deterministic function is a function that calculates the square root of the input. Db2 uses this information to enable the merging of views and table expressions for **SELECT** and SQL data change statements that refer to this function. **DETERMINISTIC** is not the default. If applicable, specify **DETERMINISTIC** to prevent non-optimal access paths from being chosen for SQL statements that refer to this function.

Db2 does not verify that the function program is consistent with the specification of **DETERMINISTIC** or **NOT DETERMINISTIC**.

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the function takes an action that changes the state of an object that Db2 does not manage. An example of an external action is sending a message or writing a record to a file.

EXTERNAL ACTION

The function can take an action that changes the state of an object that Db2 does not manage.

Some SQL statements that invoke functions with external actions can result in incorrect results if parallel tasks execute the function. For example, if the function sends a note for each initial call to

it, one note is sent for each parallel task instead of once for the function. Specify the `DISALLOW PARALLEL` clause for functions that do not work correctly with parallelism.

If you specify `EXTERNAL ACTION`, then Db2:

- Materializes the views and table expressions in `SELECT` and SQL data change statements that refer to the function. This materialization can adversely affect the access paths that are chosen for the SQL statements that refer to this function. Do not specify `EXTERNAL ACTION` if the function does not have an external action.
- Does not move the function from one task control block (TCB) to another between `FETCH` operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared `WITH HOLD`.

The only changes to resources made outside of Db2 that are under the control of commit and rollback operations are those changes made under RRS control.

`EXTERNAL ACTION` must be specified implicitly or explicitly specified if the SQL routine body invokes a function that is defined with `EXTERNAL ACTION`. `EXTERNAL ACTION` is the default.

NO EXTERNAL ACTION

The function does not take any action that changes the state of an object that Db2 does not manage. Db2 uses this information to enable the merging of views and table expressions for `SELECT` and SQL data change statements that refer to this function. If applicable, specify `NO EXTERNAL ACTION` to prevent non-optimal access paths from being chosen for SQL statements that refer to this function.

Although the scope of global variables are beyond the scope of the routine, global variables can be set in the routine body when `NO EXTERNAL ACTION` is specified.

Db2 does not verify that the function program is consistent with the specification of `EXTERNAL ACTION` or `NO EXTERNAL ACTION`.

READS SQL DATA or CONTAINS SQL

Specifies the classification of SQL statements and nested routines that this routine can execute or invoke. The database manager verifies that the SQL statements issued by the function, and all routines locally invoked by the routine, are consistent with this specification; the verification is not performed when nested remote routines are invoked. For the classification of each statement, see [SQL statement data access classification for routines \(Db2 SQL\)](#).

READS SQL DATA

Specifies that the function can execute statements with a data access classification of `READS SQL DATA`, `CONTAINS SQL`, or `NO SQL`. The function cannot execute SQL statements that modify data.

`READS SQL DATA` is the default.

CONTAINS SQL

Specifies that the function can execute only SQL statements with a data access classification of `CONTAINS SQL` or `NO SQL`. The function cannot execute SQL statements that read or modify data.

STATIC DISPATCH

At function resolution time, Db2 chooses a function based on the static (or declared) types of the function parameters. `STATIC DISPATCH` is the default.

CALLED ON NULL INPUT

Specifies that the function is to be invoked if any, or if all, of the argument values are null. Specifying `CALLED ON NULL INPUT` means that the body of the function must be coded to test for null argument values.

`CALLED ON NULL INPUT` is the default.

NOT SECURED or SECURED

Specifies if the function is considered secure for row access control and column access control. The `SECURED` or `NOT SECURED` option applies to all future versions of the function.

NOT SECURED

Specifies that the function is not considered secure for row access control and column access control.

NOT SECURED is the default.

When the function is invoked, the arguments of the function must not reference a column for which a column mask is enabled when the table is using active column access control.

SECURED

Specifies that the function is considered secure for row access control and column access control.

The function must be secure when it is referenced in a row permission or a column mask.

SQL-routine-body

Specifies a single RETURN statement. For more information, see [RETURN statement \(Db2 SQL\)](#).

If the RETURN statement includes a scalar fullselect, Db2 attempts to define a compiled function. For more information, see [CREATE FUNCTION \(compiled SQL scalar\) \(Db2 SQL\)](#).

To determine what type of SQL scalar function is created, refer to the INLINE column of the SYSIBM.SYSROUTINES catalog table.

WRAPPED obfuscated-statement-text

Specifies the encoded definition of the function. A CREATE FUNCTION statement can be encoded using the WRAP scalar function.

WRAPPED must not be specified on a static CREATE statement.

Notes**Considerations for all types of user-defined functions:**

For considerations that apply to all types of user-defined functions, see [CREATE FUNCTION \(Db2 SQL\)](#).

Types of SQL scalar functions:

If the syntax of the CREATE FUNCTION statement conforms to the syntax diagrams and descriptions for CREATE FUNCTION (inlined SQL scalar), Db2 defines an inlined function, and a package is not created. When an inlined SQL scalar function is invoked, the *expression* in the RETURN statement of the function is copied (inlined) into the query itself; the function is not invoked. The attributes of an inlined SQL scalar function are described in [CREATE FUNCTION \(inlined SQL scalar\) \(Db2 SQL\)](#).

Otherwise, Db2 attempts to define a compiled function with an associated package. For example, if the RETURN statement contains a scalar fullselect, Db2 attempts to define a compiled function. The attributes of a compiled SQL scalar function are described in [CREATE FUNCTION \(compiled SQL scalar\) \(Db2 SQL\)](#).

To determine what type of SQL scalar function is created, refer to the INLINE column of the SYSIBM.SYSROUTINES catalog table. In the INLINE column, a value of Y indicates that the function is an inlined function, and a value of N indicates that the function is a compiled function.

Considerations for functions defined with MODIFIES SQL DATA:

If a function is specified in a subselect, and the function is defined as MODIFIES SQL DATA, the number of times the function is invoked is invoked will vary depending on the access plan used.

Self-referencing function:

The body of an SQL function (that is, the *expression* or NULL in the RETURN statement in the body of the CREATE FUNCTION statement) cannot contain a recursive invocation of itself or to another function that invokes it, because such a function would not exist to be referenced.

Dependent objects:

An SQL routine is dependent on objects that are referenced in the routine body.

Obfuscated statements:

A CREATE FUNCTION statement can be executed in obfuscated form. In an obfuscated statement, only the function name, parameters, and the WRAPPED keyword are readable. The rest of the

statement is encoded in such a way that it is not readable but can be decoded by a database server that supports obfuscated statements. The WRAP scalar function produces obfuscated statements. Any debug options that are specified when the function is created from an obfuscated statement are ignored.

Resolution of object names:

Db2 resolves object names inside the body of the function according to the rules in [Resolution of unqualified object names \(Db2 SQL\)](#) and the type of the object. The name resolution occurs when the function is created.

Referencing date and time special registers:

If an SQL function contains multiple references to any of the date or time special registers, all references return the same value. In addition, this value is the same value that is returned by the retrieving value of the special register in the statement that invoked the function.

Self-referencing function:

The body of an SQL function (that is, the *expression* or NULL in the RETURN clause of the CREATE FUNCTION (inlined SQL scalar) statement) cannot contain a recursive invocation of itself or to another function that invokes it, because such a function would not exist to be referenced.

Dependent objects:

An SQL routine is dependent on objects that are referenced in the routine body.

Alternative syntax and synonyms:

To provide compatibility with previously releases of Db2 or other products in the Db2 family, Db2 supports the following alternative syntax:

- VARIANT as a synonym for NOT DETERMINISTIC
- NOT VARIANT as a synonym for DETERMINISTIC
- NULL CALL as a synonym for CALLED ON NULL INPUT
- TIMEZONE can be specified as an alternative to TIME ZONE.

For an inlined SQL scalar function, the RETURNS clause and the clauses in the *option-list* can be specified in any order.

Restriction on use of AI Scalar Functions:

Scalar functions AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, and AI_SIMILARITY cannot be used in an inlined function. The AI functions must refer to model columns, which cannot be specified in an inlined function definition. If the AI function appears inside of a scalar fullselect in the RETURN statement of the function, then the function is created as a compiled SQL scalar function as described above, and the use of the AI function is allowed.

Examples

Example 1: Define a scalar function that returns the tangent of a value using existing SIN and COS built-in functions:

```
CREATE FUNCTION TAN (X DOUBLE)
  RETURNS DOUBLE
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN SIN(X)/COS(X);
```

CREATE FUNCTION (SQL table)

The CREATE FUNCTION (SQL table) statement creates an SQL table function at the current server. The function returns a set of rows.

Invocation

This statement can only be dynamically prepared only if dynamic rules run behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

The privilege set that is defined below must include at least one of the following privileges or authorities:

- The CREATEIN privilege on the schema
- SYSADM authority
- SYSCTRL authority
- System DBADM
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

If the authorization ID that is used to create the function has the installation SYSADM authority or the installation SYSOPR authority and if the current SQLID is set to SYSINSTL, the function is identified as system-defined function.

If a distinct type is referenced (i.e. as the data type of a parameter or SQL variable), the privilege set must also include at least one of the following:

- Ownership of the distinct type
- The USAGE privilege on the distinct type
- SYSADM authority
- SYSDBADM authority

If the function uses a table as a parameter, the privilege set must also include at least one of the following:

- Ownership of the table
- The SELECT privilege on the table
- SYSADM authority

At least one of the following additional privileges is required if the SECURED option is specified

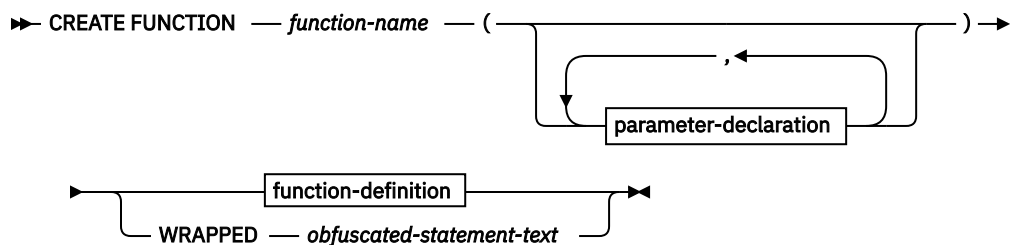
- SECADM authority
- CREATE_SECURE_OBJECT privilege

Privilege set: If the statement is embedded in an application program, the privilege set is the set of privileges that are held by the owner of the plan or package. If the owner is a role, matching of the implicit schema name does not apply and the role must include one of the previously listed privileges or authorities.

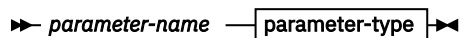
If the statement is dynamically prepared and is not running in a trusted context for which the ROLE AS OBJECT OWNER clause is specified, the privilege set is the set of privileges that are held by the SQL authorization ID of the process. If the schema name is not the same as the SQL authorization ID of the process, one of the following conditions must be met:

- The privilege set includes SYSADM authority
- The privilege set includes SYSCTRL authority
- The SQL authorization ID of the process has the CREATEIN privilege on the schema

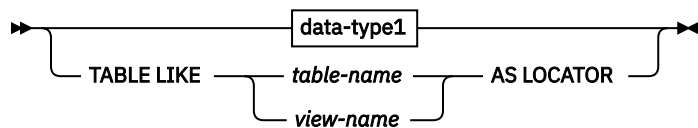
Syntax



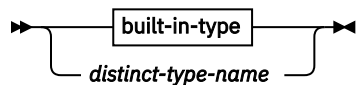
parameter-declaration:



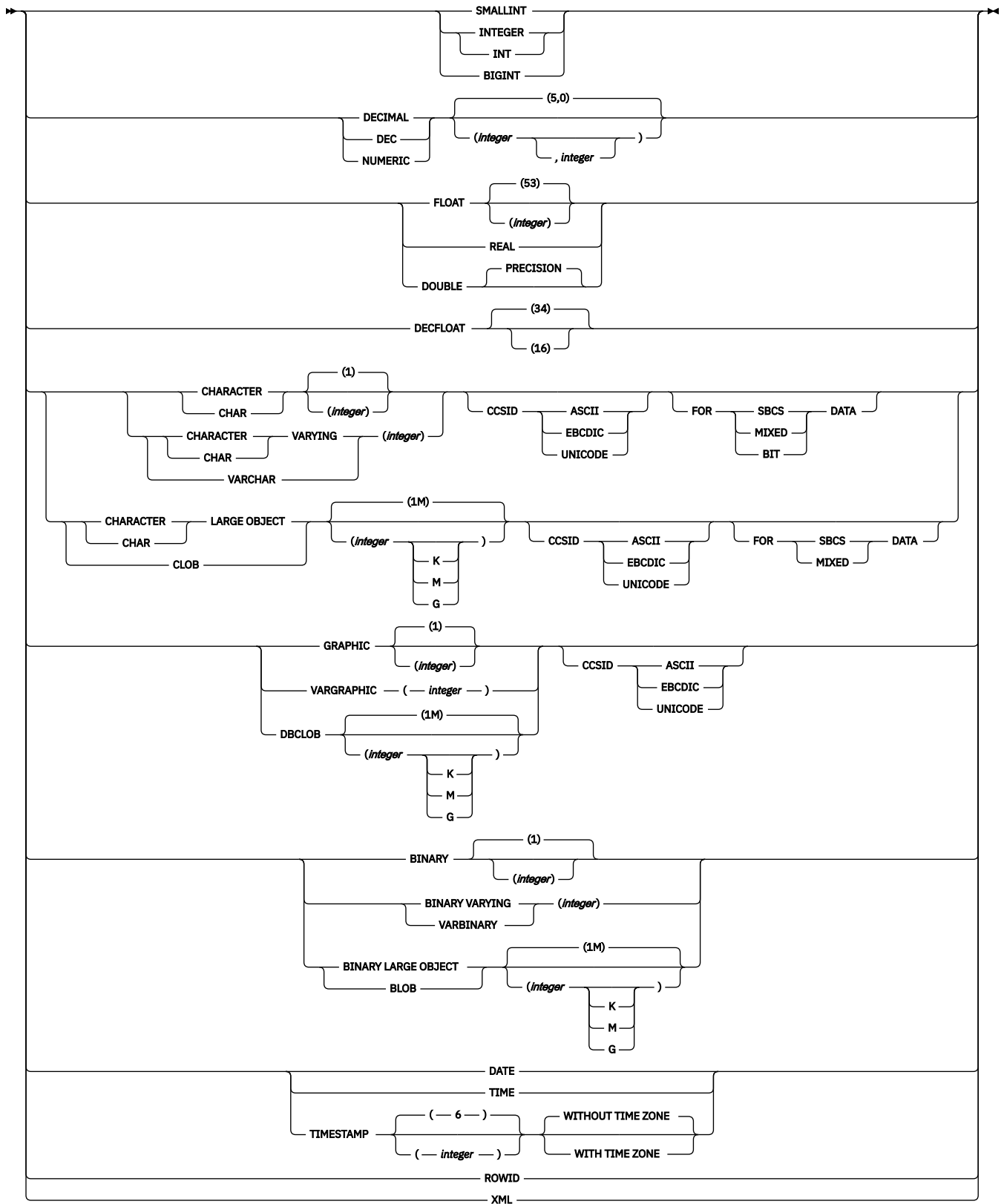
parameter-type:



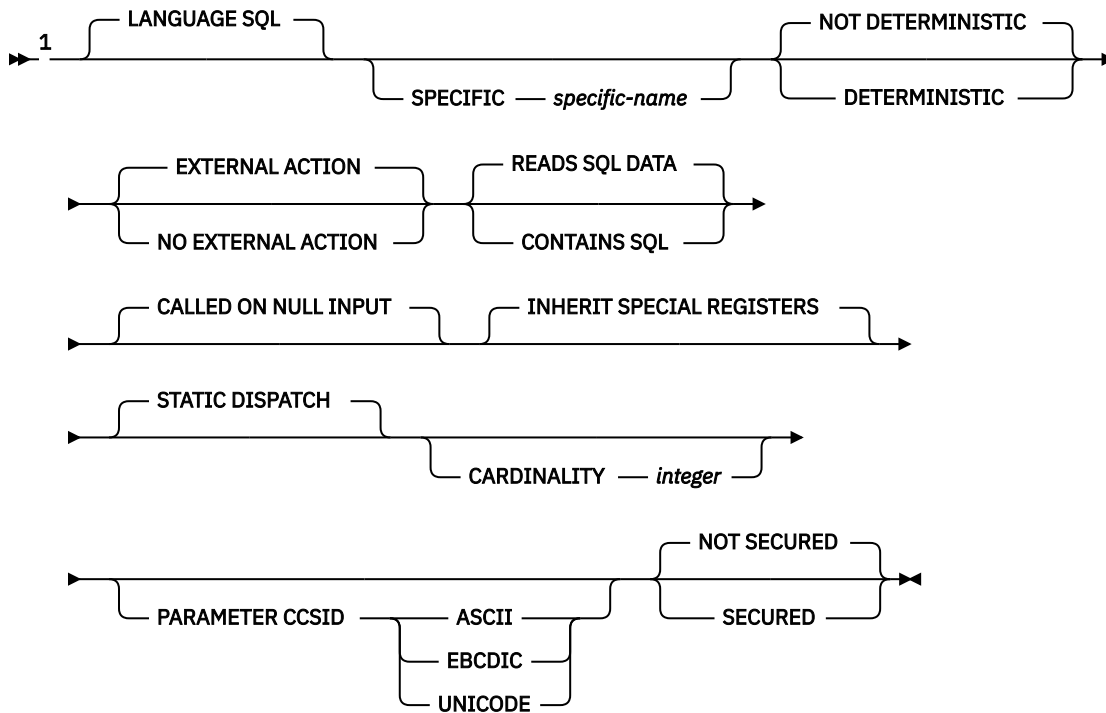
data-type1, data-type2:



built-in-type:



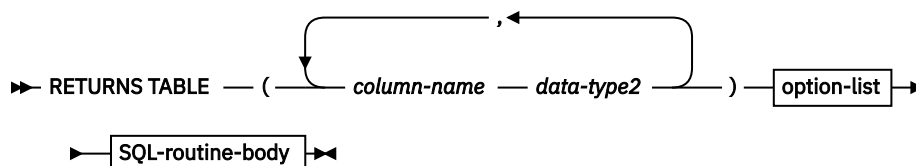
option-list:



Notes:

¹ The options in the *option-list* can be specified in any order. However, the same clause cannot be specified more than one time.

function-definition



SQL-routine-body:



Description

function-name

Names the user-defined function. The name is implicitly or explicitly qualified by a schema name. The combination of the name, the schema name, the number of parameters, and the data type of each parameter (without regard to any length, precision, scale, subtype, or encoding scheme attribute of the data type) must not identify a user-defined function that exists at the current server. For more information, see "Choosing the schema and function names" and "Determining the uniqueness of functions in a schema" in [CREATE FUNCTION \(Db2 SQL\)](#).

(parameter-declaration,...)

Identifies the number of input parameters of the function, and specifies the name and data type of each parameter. All of the parameters for a function are input parameters and are nullable. There must be one entry in the list for each parameter that the function expects to receive.

parameter-name

Specifies the name of the input parameter. Each name in the parameter list must not be the same as any other name.

data-type1

Specifies the data type of the input parameter. The data type can be a built-in data type or a distinct type.

built-in-type

The data type of the parameter is a built-in data type.

For more information on the data types, including the subtype of character data types (the FOR subtype DATA clause), see [built-in types](#). However, the varying length string data types have different maximum lengths for this statement than for the CREATE TABLE statement. The maximum lengths for parameters (and SQL variables) are as follows:

- VARCHAR or VARBINARY: 32704
- VARGRAPHIC: 16352

For parameters with a character or graphic data type, the PARAMETER CCSID clause or the CCSID clause indicates the encoding scheme of the parameter. If you do not specify either of the CCSID clauses, the encoding scheme is the value of the DEF ENCODING SCHEME field on installation panel DSNTIPF.

Although an input parameter with a character data type has an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the value that is actually passed in the input parameter can have any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the function is invoked. With ASCII or EBCDIC, an error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

distinct-type-name

The data type of the parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type for the distinct type.

TABLE LIKE *table-name* AS LOCATOR

Specifies that the parameter is a transition table. However, when the function is invoked, the actual values in the transition table are not passed to the function. A single value is passed instead. This value is a locator for the table, which the function uses to access the columns of the transition table. The table that is identified can contain XML columns; however, the function cannot reference those XML columns.

A function with a table parameter can only be invoked from the triggered action of a trigger.

RETURNS TABLE

Specifies that the output of the function is a table. The RETURN statement in an SQL table function must return a table result. The parentheses that follow the RETURNS TABLE keyword delimit a list of name and data type pairs of the columns of the output table. All columns of the output table are nullable.

column-name

Specifies the name of the column. The name cannot be qualified, and must be unique within the RETURNS TABLE clause for the function.

data-type2

Specifies the data type and attributes of the column of the output table.

For SQL table functions, the result table of the function might include multiple encoding schemes – similar to what a view definition can include.

LANGUAGE SQL

Specifies that the function is written exclusively in SQL.

SPECIFIC *specific-name*

Specifies a unique name for the function.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the function returns the same results each time that the function is invoked with the same input arguments. Db2 does not verify that the function program is consistent with the specification of NOT DETERMINISTIC or DETERMINISTIC.

NOT DETERMINISTIC

Specifies that the function might not return the same result table each time that the function is invoked with the same input arguments, even when the referenced data in the database has not changed. The function depends on some state values that might affect the results. Db2 disables the merging of views and table expressions when processing SELECT and SQL data change operations that refer to a function that is defined with this option. An example of a table function that is not deterministic is one which references special registers, other functions that are not deterministic, or a sequence in a way that affects the table function's result table. NOT DETERMINISTIC is the default.

DETERMINISTIC

Specifies that the function always returns the same result table each time that the function is invoked with the same input arguments (provided that the referenced data in the database has not changed). Db2 enables the merging of SQL table functions that are defined with this option.

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the function contains an external action. Db2 does not verify that the function program is consistent with the specification of EXTERNAL ACTION or NO EXTERNAL ACTION.

EXTERNAL ACTION

The function performs some external action (outside the scope of the function program). Thus, the function must be invoked with each successive function invocation. EXTERNAL ACTION must be specified if the function invokes another function that has external actions. EXTERNAL ACTION is the default.

NO EXTERNAL ACTION

The function does not perform any external action. It need not be called with each successive function invocation. Functions that are defined with NO EXTERNAL ACTION might perform better than functions that are defined with EXTERNAL ACTION because the function might not be invoked for each successive function invocation.

Although the scope of global variables are beyond the scope of the routine, global variables can be set in the routine body when NO EXTERNAL ACTION is specified.

READS SQL DATA or CONTAINS SQL

Specifies the classification of SQL statements and nested routines that this routine can execute or invoke. The database manager verifies that the SQL statements issued by the function, and all routines locally invoked by the routine, are consistent with this specification; the verification is not performed when nested remote routines are invoked. For the classification of each statement, see SQL statement data access classification for routines (Db2 SQL).

READS SQL DATA

Specifies that the function can execute statements with a data access indication of READS SQL DATA or CONTAINS SQL. The function cannot execute SQL statements that modify data.

READS SQL DATA is the default.

CONTAINS SQL

Specifies that the function can execute only SQL statements with a data access indication of CONTAINS SQL. The function cannot execute statements that read or modify data.

CALLED ON NULL INPUT

Specifies that the function is called regardless of whether any of the input arguments are null, making the function responsible for testing for null argument values. The function can return an empty table, depending on its logic.

CALLED ON NULL INPUT is the default.

INHERIT SPECIAL REGISTERS

Specifies that existing values of special registers are inherited upon entry to the function. **INHERIT SPECIAL REGISTERS** is the default.

STATIC DISPATCH

At function resolution time, Db2 chooses a function based on the static (or declared) types of the function parameters. **STATIC DISPATCH** is the default.

CARDINALITY integer

Specifies an estimate of the expected number of rows that the function returns. The number is used for optimization purposes. The value of *integer* must be between 0 and 2147483647.

If you do not specify **CARDINALITY**, Db2 assumes a finite value. The finite value is the same value that Db2 assumes for tables for which the RUNSTATS utility has not gathered statistics.

If a function has an infinite cardinality (the function never returns the end-of-table condition and always returns a row), a query that requires the end-of-table condition to work correctly will need to be interrupted.

PARAMETER CCSID

Specifies the encoding scheme for character and graphic string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value that is specified in the CCSID clauses of the parameter list or RETURNS clause, or in the DEF ENCODING SCHEME field on installation panel DSNTIPF. This clause provides a convenient way to specify the encoding scheme for character and graphic string parameters. If individual CCSID clauses are specified for individual parameters in addition to this **PARAMETER CCSID** clause, the value specified in all of the CCSID clauses must be the same value that is specified in this clause. This clause also specifies the encoding scheme that is used for system-generated parameters of the routine such as message tokens and DBINFO.

NOT SECURED or SECURED

Specifies if the function is considered secure for row access control and column access control. The **SECURED** or **NOT SECURED** option applies to all future versions of the function.

NOT SECURED

Specifies that the function is not considered secure for row access control and column access control.

NOT SECURED is the default.

When the function is invoked, the arguments of the function must not reference a column for which a column mask is enabled when the table is using active column access control.

SECURED

Specifies that the function is considered secure for row access control and column access control.

The function must be secure when it is referenced in a row permission or a column mask.

SQL-routine-body***RETURN-statement***

Specifies the return value of the function. A RETURN statement must be specified for an SQL table function.

WRAPPED obfuscated-statement-text

Specifies the encoded definition of the function. A CREATE FUNCTION statement can be encoded using the WRAP scalar function.

WRAPPED must not be specified on a static CREATE statement.

ATOMIC

ATOMIC indicates that an unhandled exception condition within the RETURN statement causes the statement to be rolled back.

Notes

Considerations for all types of user-defined functions:

For considerations that apply to all types of user-defined functions, see [CREATE FUNCTION \(Db2 SQL\)](#).

Self-referencing function:

The body of an SQL function (that is, the *expression* or NULL in the RETURN statement in the body of the CREATE FUNCTION statement) cannot contain a recursive invocation of itself or to another function that invokes it, because such a function would not exist to be referenced.

Dependent objects:

An SQL routine is dependent on objects that are referenced in the routine body.

Obfuscated statements:

A CREATE FUNCTION statement can be executed in obfuscated form. In an obfuscated statement, only the function name, parameters, and the WRAPPED keyword are readable. The rest of the statement is encoded in such a way that it is not readable but can be decoded by a database server that supports obfuscated statements. The WRAP scalar function produces obfuscated statements. Any debug options that are specified when the function is created from an obfuscated statement are ignored.

Resolution of object names:

Db2 resolves object names inside the body of the function according to the rules in [Resolution of unqualified object names \(Db2 SQL\)](#) and the type of the object. The name resolution occurs when the function is created.

Referencing date and time special registers:

If an SQL function contains multiple references to any of the date or time special registers, all references return the same value. In addition, this value is the same value that is returned by the retrieving value of the special register in the statement that invoked the function.

Considerations for column names longer than 30 bytes

If a length of a new column name is greater than 30 Unicode bytes, truncation occurs in the SQLNAME field of the SQLDA when the column is described in an application. A column name in UTF8, and its equivalent in the system EBCDIC CCSID, must be 128 bytes or less. For more information about long column names, see [Column names longer than 30 bytes \(Db2 SQL\)](#).

Considerations for columns that are defined with a field procedure:

The body of an SQL table function must not reference a column that is defined with a field procedure, and the RETURNS clause of an SQL table function must not reference a column that is defined with a field procedure. An SQL table function must not be invoked with an expression that is derived from a column that is defined with a field procedure.

Restrictions involving pending definition changes:

The body of an SQL table function must not reference a table that has pending definition changes.

Alternative syntax and synonyms:

To provide compatibility with previously releases of Db2 or other products in the Db2 family, Db2 supports the following alternative syntax:

- VARIANT as a synonym for NOT DETERMINISTIC
- NOT VARIANT as a synonym for DETERMINISTIC
- NULL CALL as a synonym for CALLED ON NULL INPUT

Restrictions involving AI functions:

The body of a SQL table function must not reference AI functions: AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.

Examples

Example 1

Define a table function, JTABLE, to return a table with 3 columns:

```
CREATE FUNCTION JTABLE (COLD_VALUE CHAR(9), T2_FLAG CHAR(1))
  RETURNS TABLE (COLA INT, COLB INT, COLC INT)
  LANGUAGE SQL
  SPECIFIC DEPTINFO
  NOT DETERMINISTIC
  READS SQL DATA
  RETURN
    SELECT A.COLA, B.COLB, B.COLC
      FROM TABLE1 AS A
      LEFT OUTER JOIN
      TABLE2 AS B
      ON A.COL1 = B.COL1 AND T2_FLAG = 'Y'
     WHERE A.COLD = COLD_VALUE;
```

Example 2

Define a table function that returns the employees in a specified department number. The function simply returns the employees for the requested department:

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
  RETURNS TABLE (EMPNO CHAR(6), LASTNAME VARCHAR(15), FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  READS SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN
    SELECT EMPNO, LASTNAME, FIRSTNAME
      FROM YEMP
     WHERE YEMP.WORKDEPT = DEPTEMPLOYEES.DEPTNO;
```

Related concepts

[SQL table functions \(Db2 Application programming and SQL\)](#)

[Naming conventions \(Db2 SQL\)](#)

Related tasks

[Creating a user-defined function \(Db2 Application programming and SQL\)](#)

CREATE INDEX

The CREATE INDEX statement creates a partitioning index or a secondary index and an index space at the current server. The columns included in the key of the index are columns of a table at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

The privilege set that is defined below must include at least one of the following:

- The INDEX privilege on the table
- Ownership of the table
- DBADM authority for the database that contains the table
- SYSADM or SYSCTRL authority
- System DBADM
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

If the database is implicitly created, the database privileges must be on the implicit database or on DSNOB04.

If the index is created using an expression, the EXECUTE privilege is required on any user-defined function that is invoked in the index expression.

Additional privileges might be required, as explained in the description of the BUFFERPOOL and USING STOGROUP clauses.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the plan or package. If the specified index name includes a qualifier that is not the same as this owner, the privilege set must include SYSADM or SYSCTRL authority, or DBADM or DBCTRL authority for the database.

If ROLE AS OBJECT OWNER is in effect, the schema qualifier must be the same as the role, unless the role has the CREATEIN privilege on the schema, SYSADM authority, or SYSCTRL authority.

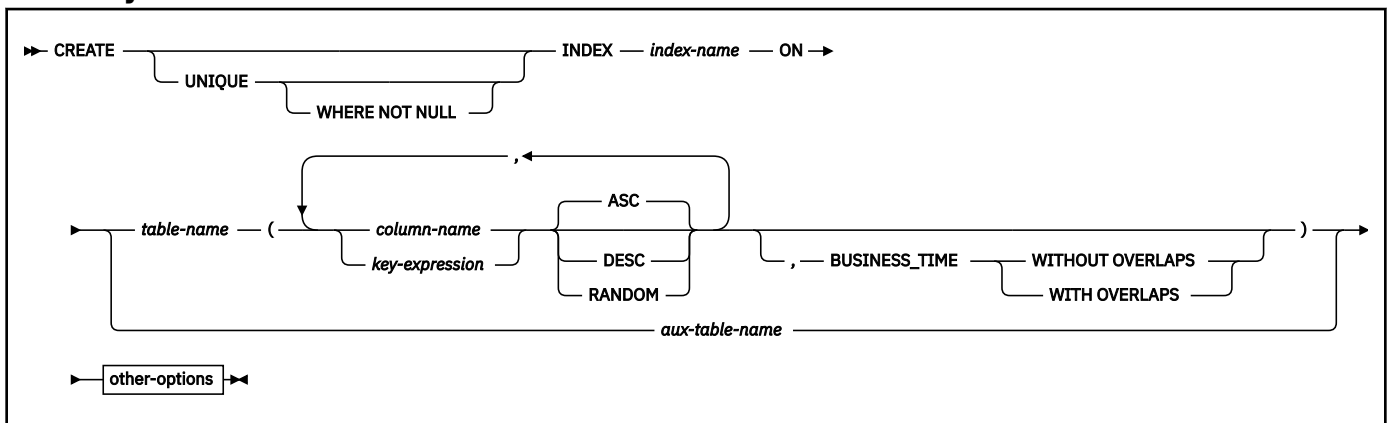
If ROLE AS OBJECT OWNER is not in effect, one of the following rules applies:

- If the privilege set lacks the CREATEIN privilege on the schema, SYSADM authority, or SYSCTRL authority, the schema qualifier (implicit or explicit) must be the same as one of the authorization ids of the process.
- If the privilege set includes SYSADM authority or SYSCTRL authority, the schema qualifier can be any valid schema name.

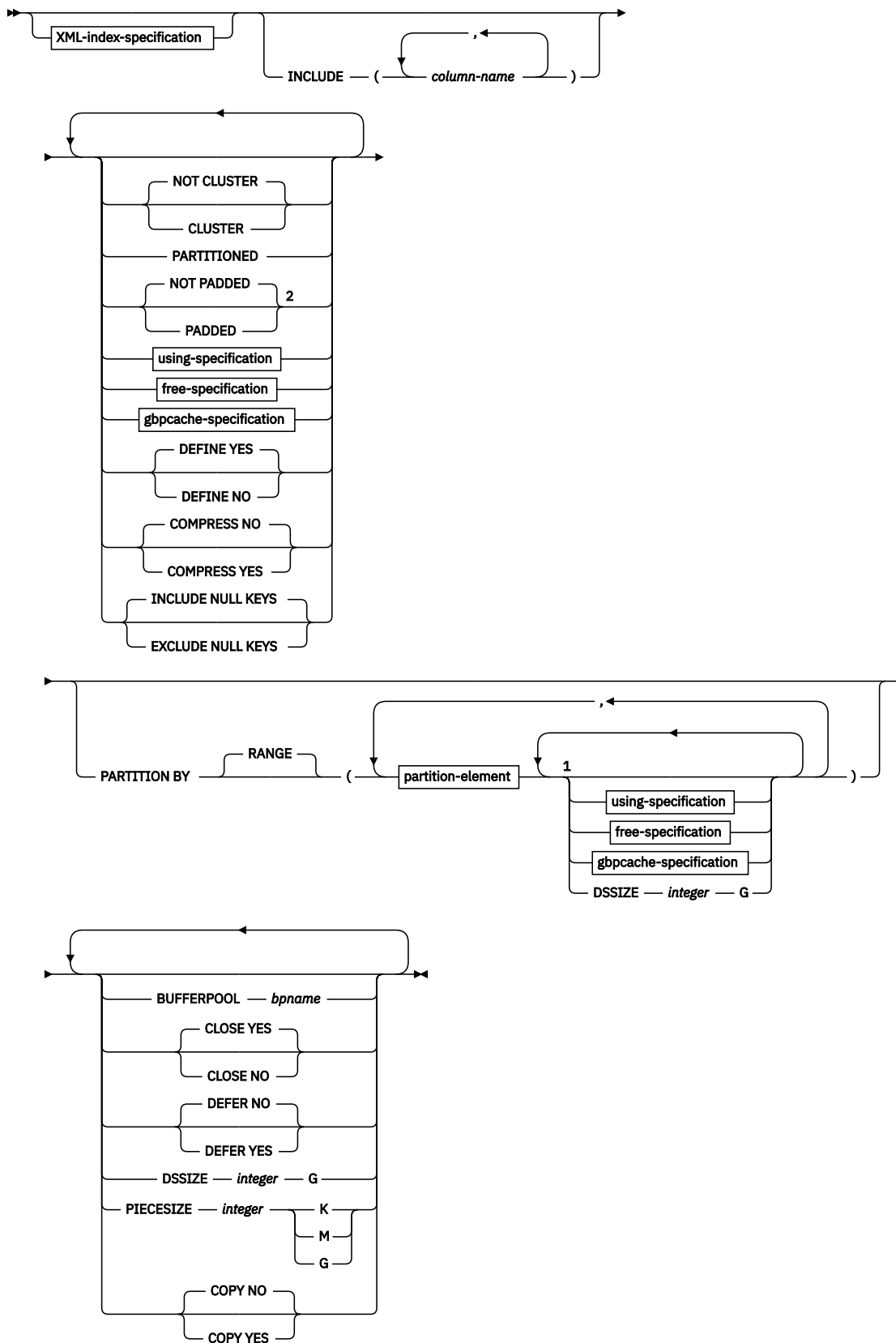
If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process unless the process is within a trusted context and the ROLE AS OBJECT OWNER clause is specified. In that case, the privilege set is the set of privileges that are held by the role that is associated with the primary authorization ID of the process. However, if the specified index name includes a qualifier that is not the same as this authorization ID, the following rules apply:

- If the privilege set includes SYSADM or SYSCTRL authority (or DBADM authority for the database, or DBCTRL authority for the database when creating a table), the schema qualifier can be any valid schema name.
- If the privilege set lacks SYSADM or SYSCTRL authority (or DBADM authority for the database, or DBCTRL authority for the database when creating a table), the schema qualifier is valid only if it is the same as one of the authorization IDs of the process and the privilege set that are held by that authorization ID includes all privileges needed to create the index. This is an exception to the rule that the privilege set is the privileges that are held by the SQL authorization ID of the process.

Syntax



other-options:

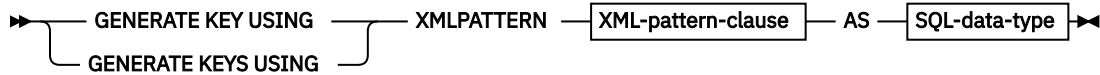


Notes:

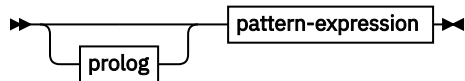
¹ The same clause must not be specified more than one time.

² The value of field PAD INDEXES BY DEFAULT (on installation panel DSNTIPE) determines the default. When the value is NO, NOT PADDED is the default. When the value is YES, PADDED is the default. For more information, see the description of the PADDED or NOT PADDED options.

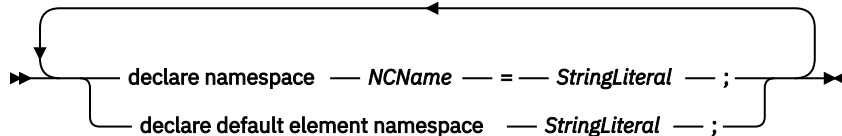
XML-index-specification:



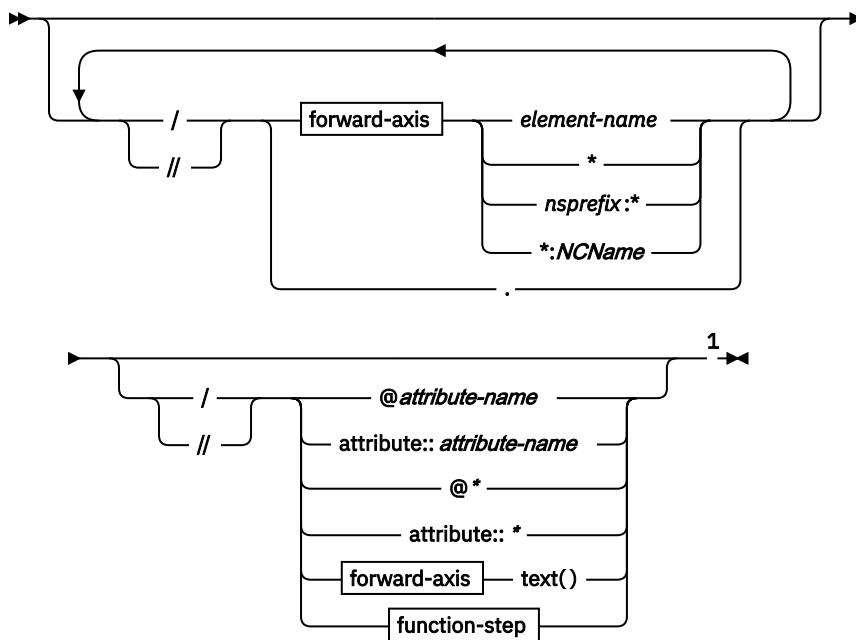
XML-pattern-clause:



prolog:



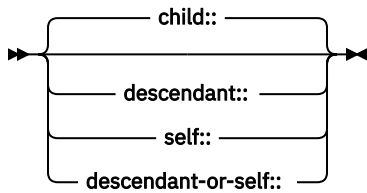
pattern-expression:



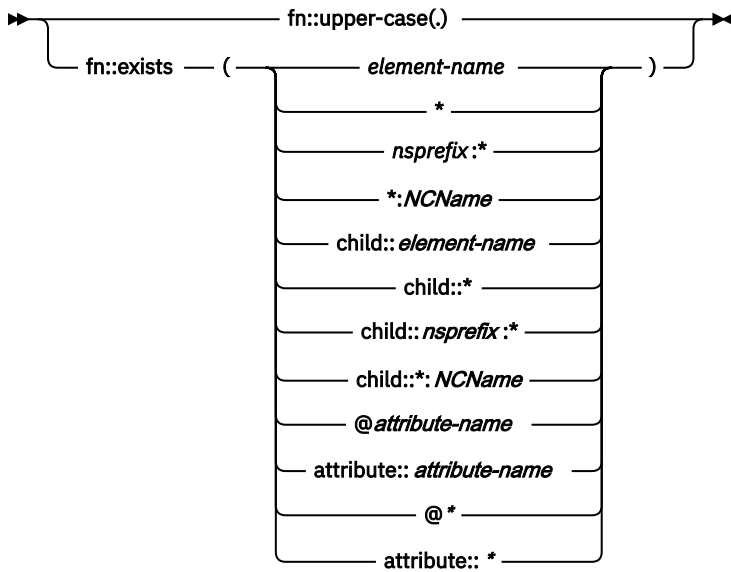
Notes:

¹ *pattern-expression* cannot be an empty string.

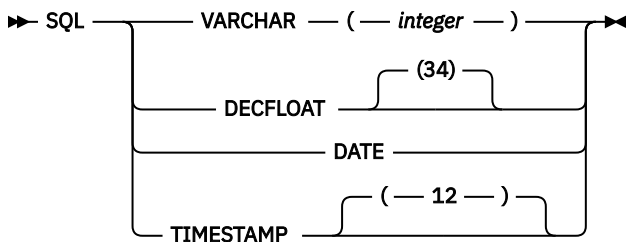
forward-axis:



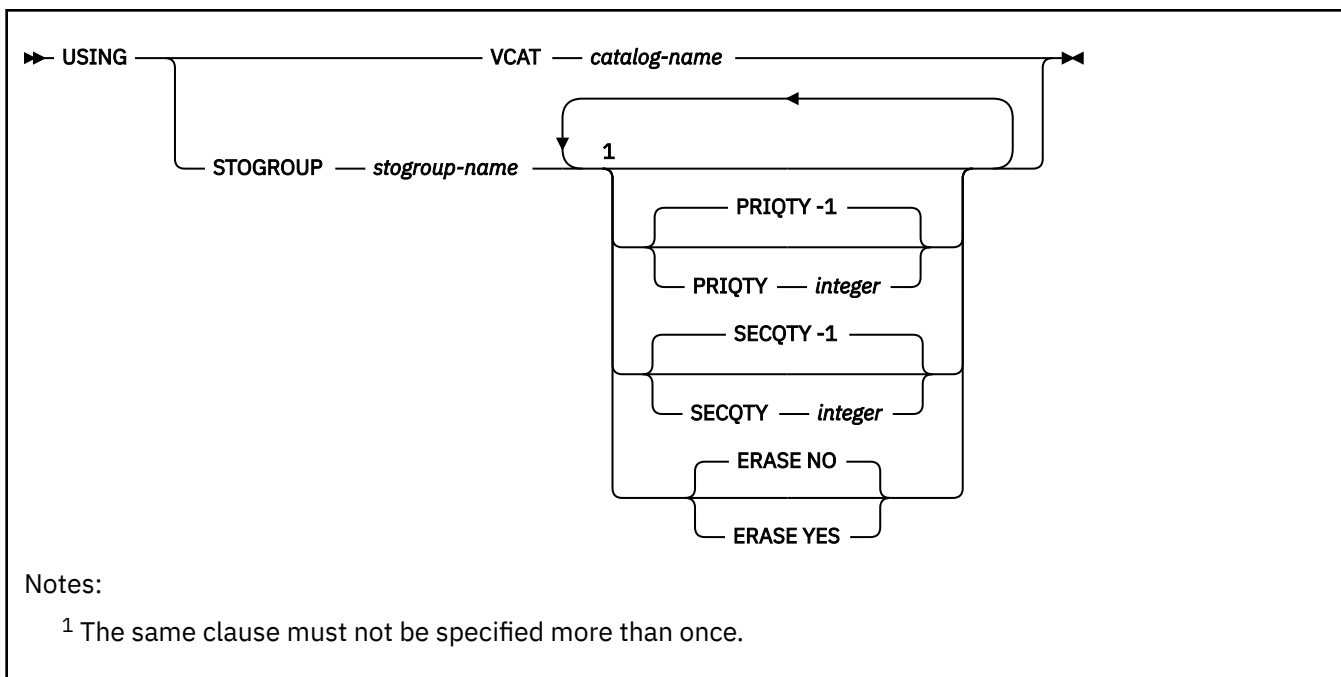
function-step:



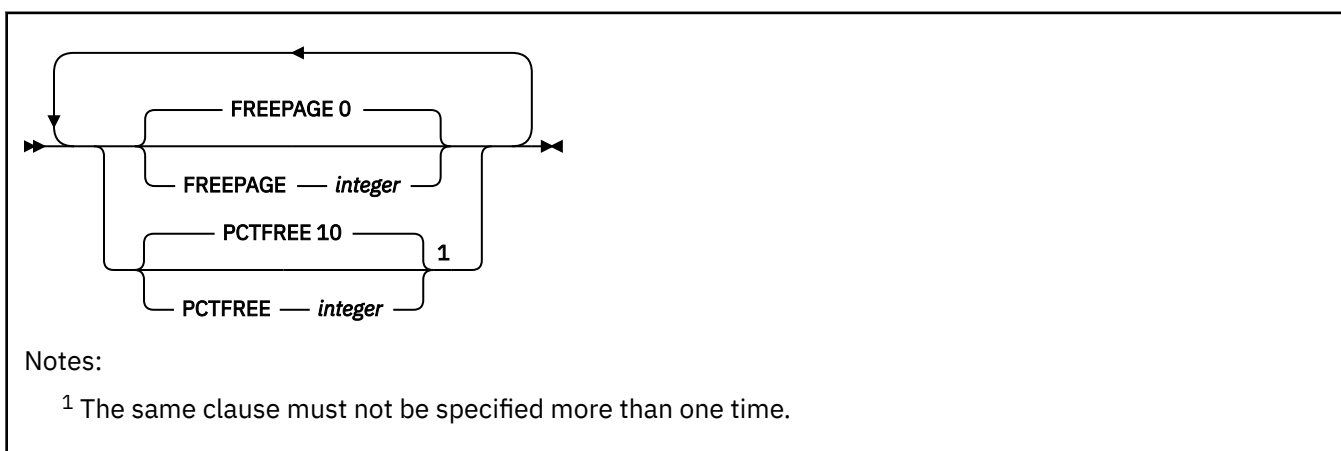
SQL-data-type:



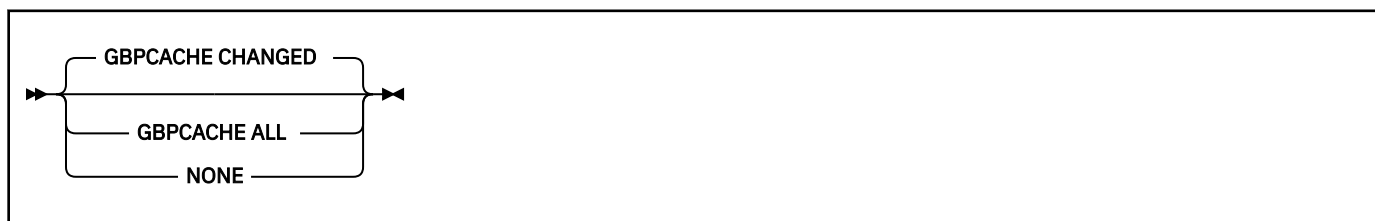
using-specification:



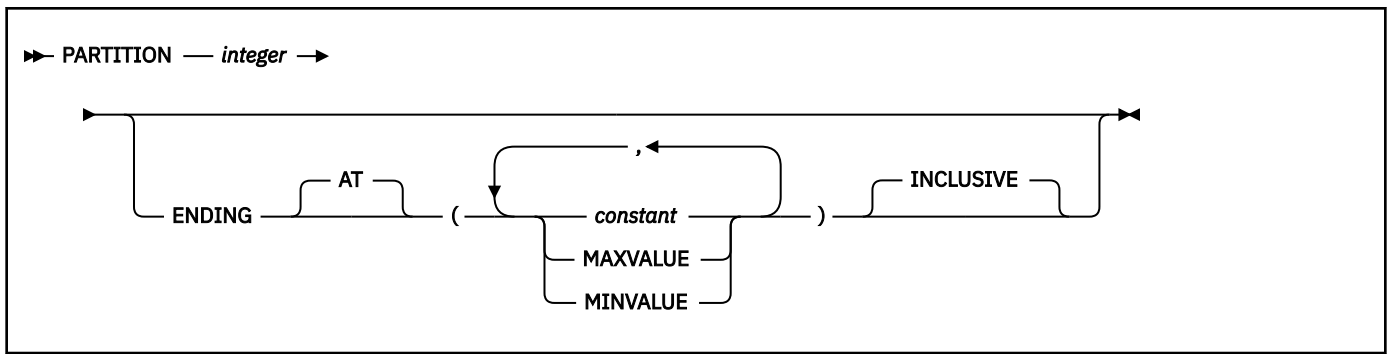
free-specification:



gbpcache-specification:



partition-element:



Description

UNIQUE

Prevents the table from containing two or more rows with the same value of the index key. When UNIQUE is used, all null values for a column are considered equal. For example, if the key is a single column that can contain null values, that column can contain only one null value. The constraint is enforced when rows of the table are updated or new rows are inserted.

The constraint is also checked during the execution of the CREATE INDEX statement. If the table already contains rows with duplicate key values, the index is not created.

UNIQUE WHERE NOT NULL

Prevents the table from containing two or more rows with the same value of the index key where all null values for a column are not considered equal. Multiple null values are allowed. Otherwise, this is identical to UNIQUE.

INDEX *index-name*

Names the index. The name must not identify an index that exists at the current server, or is listed in the SYSIBM.SYSPENDINGOBJECTS catalog table, or is in an accelerator-only table.

The associated index space also has a name. That name appears as a qualifier in the names of data sets defined for the index. If the data sets are managed by the user, the name is the same as the second (or only) part of *index-name*. If this identifier consists of more than eight characters, only the first eight are used. The name of the index space must be unique among the names of the index spaces and table spaces of the database for the identified table. If the data sets are defined by Db2, Db2 derives a unique name.

If the index is an index on a declared temporary table, the qualifier, if explicitly specified, must be SESSION. If the index name is unqualified, Db2 uses SESSION as the implicit qualifier.

For more information, see [Index names and guidelines \(Db2 Administration Guide\)](#).

ON *table-name* or *aux-table-name*

Identifies the table on which the index is created. The name can identify a base table, a materialized query table, a declared temporary table, or an auxiliary table.

table-name

Identifies the base table, materialized query table, or declared temporary table on which the index is created. The name must identify a table that exists at the current server. (The name of a declared temporary table must be qualified with SESSION.)

The name must not identify a clone table. The name must not identify a created temporary table or a table that is implicitly created for an XML column. If the index that is being created is for XML values, the table can contain an XML column, otherwise, the table must not contain an XML column. The name must not identify a catalog table or declared temporary table if the index is created using expressions. The name must not identify an accelerator-only table or a directory table.

If the table has enforced row or column access controls, the row permissions and column masks are not applied during key generation.

column-name,...

Specifies the columns of the index key.

Each *column-name* must identify a column of the table. Do not specify more than 64 columns or the same column more than one time. Do not qualify *column-name*.

Do not specify a column for *column-name* that is defined as follows:

- a LOB column (or a column with a distinct type that is based on a LOB data type)
- a BINARY or VARBINARY column (or a column with a distinct type that is based on a BINARY or VARBINARY data type) when the PARTITION BY RANGE clause is also specified
- a VARBINARY column (or a column with a distinct type that is based on a VARBINARY data type) when the PADDED clause is also specified
- a row change timestamp column when the PARTITION BY RANGE or PARTITIONED clause is also specified.
- a timestamp with time zone column (or a column with a distinct type that is based on the timestamp with time zone data type) when the PARTITION or PARTITION BY RANGE clause is also specified.

A column with an XML type can only be specified if the XMLPATTERN clause is also specified. If the XMLPATTERN clause is specified, only one column can be identified and the column must be an XML type. The resulting index is an XML index.

If the table is an EBCDIC table with Unicode columns, character and graphic columns that are specified for the index key must be all EBCDIC or all Unicode.

The sum of the length attributes of the columns must not be greater than the following limits, where *n* is the number of columns that can contain null values, *m* is the number of varying-length columns, and *d* is the number of DECFLOAT columns in the key:

- $2000 - n$ for a padded, nonpartitioning index
- $2000 - n - 2m - 3d$ for a nonpadded, nonpartitioning index
- $255 - n$ for a partitioning index (padded or nonpadded)
- $255 - n - 2m - 3d$ for a nonpadded, partitioning index

key-expression

Specifies an expression that returns a scalar value. An index with a key that includes one or more expressions consisting of more than just a column name is an *expression-based index*. *key-expression* cannot be specified with the GENERATE KEY USING clause or the INCLUDE clause. *key-expression* has the following restrictions:

- Each *key-expression* must contain at least one reference to a column of *table-name*.

All references to columns of *table-name* must be unqualified. Referenced columns cannot include any FIELDPROCs or a SECURITY LABEL. Referenced columns cannot be implicitly hidden (that is, defined with the IMPLICITLY HIDDEN attribute).

- *key-expression* must not include any of the following:
 - A subquery
 - An aggregate function
 - A function that is not deterministic function
 - A function that has an external action
 - A user-defined function
 - The VERIFY_GROUP_FOR_USER or VERIFY_ROLE_FOR_USER functions
 - A sequence reference
 - A host variable
 - A parameter marker

- A global variable
 - A special register
 - An expression for which implicit time zone value apply (or example, cast a timestamp to a timestamp with time zone)
 - A CASE expression
 - An OLAP specification
 - An AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY function.
- If *key-expression* invokes a cast function, the privilege set must implicitly include EXECUTE authority on the generated cast functions for the distinct type.
 - If *key-expression* invokes the LOWER or UPPER functions, the input *string-expression* cannot be FOR BIT DATA, and the function invocation must contain the *locale-name* argument.
 - If *key-expression* invokes the TRANSLATE function, the function invocation must contain the *to-string* argument.
 - *key-expression* must not invoke a built-in function with an argument that references a LOB column, unless the function is SUBSTR or JSON_VAL.
 - If *key-expression* invokes the SUBSTR function, an argument to the function that references a LOB column can reference only the inline portion of the LOB column.
 - If *key-expression* invokes the JSON_VAL function and the first argument is a LOB column, the column must be defined as an inline LOB.
 - If *key-expression* invokes the JSON_VAL function, the function invocation must meet the following conditions:
 - The invocation of the JSON_VAL function must be the outermost expression for *key-expression*.
 - If the first argument is a column, that column must be contained in a table in a partition-by-growth table space.
 - The third argument must end with the string ':na', to indicate that the first argument does not contain a JSON array.
 - If *key-expression* invokes the JSON_VAL built-in function, the CREATE INDEX statement must not reference any LOB columns other than the LOB column that is the argument to the JSON_VAL function. Such a CREATE INDEX statement can refer only to a single LOB column.
 - The same expression cannot be used more than one time in the same index.
 - The data type of the result of the expression cannot be a LOB, XML, DECFLOAT, or array value. However, the data type of an intermediate result can be a LOB or DECFLOAT value (or a distinct type that is based on one of these data types), but not an XML value. For an index with a DECFLOAT intermediate result, the rounding mode that was in effect when the index was created should also be in effect when the index is used.
 - If a Unicode column in an EBCDIC table is referenced in a *key-expression*, the encoding scheme of the index keys must either be all Unicode or all EBCDIC. Otherwise, the encoding scheme of the result of a *key-expression* must be the same encoding scheme as the table.

The maximum length of the text string of each *key-expression* is 4000 bytes after conversion to UTF-8. The maximum number of *key-expression* in an extended index is 64.

ASC

Puts the index entries in ascending order by the column. ASC cannot be specified with the GENERATE KEY USING clause.

ASC is the default.

DESC

Puts the index entries in descending order by the column. DESC cannot be specified with the GENERATE KEY USING clause or if the ON clause contains *key-expression*.

RANDOM

Index entries are put in a random order by the column. RANDOM cannot be specified in the following cases:

- A varying length column is part of the index key and the index is defined with the NOT PADDED option
- A column of the index key is defined as TIMESTAMP WITH TIME ZONE or DECFLOAT
- The index is an XML index. An XML index is defined with the GENERATE KEY USING clause
- The index is part of the partitioning key
- The index is an expression-based index

BUSINESS_TIME

Specifies that the columns of the BUSINESS_TIME period are automatically added to the end of the index key in the following order:

- The end column of the BUSINESS_TIME period in ascending order
- The start column of the BUSINESS_TIME period in ascending order

BUSINESS_TIME can be specified as the last item in the list. The list must also include at least one *column-name* or *key-expression*. When BUSINESS_TIME is specified, the columns of the BUSINESS_TIME period must not be specified as *column-name* or a *key-expression*, or as columns in the partitioning key.

WITH OR WITHOUT OVERLAPS

Indicates whether multiple rows may exist with the same values for the non-period columns and expressions of the index key for a row, with overlapping time periods.

WITH OVERLAPS

Indicates that multiple rows may exist with the same values for the non-period columns and expressions of the index key for a row, with overlapping time periods. The BUSINESS_TIME WITH OVERLAPS clause is intended for use in defining an index for the foreign key of a temporal referential constraint.

BUSINESS_TIME WITH OVERLAPS must not be specified when UNIQUE is specified for the index definition.

BUSINESS_TIME WITHOUT OVERLAPS must not be specified if the table is defined with a partitioning key that includes any columns of the BUSINESS_TIME period.

WITHOUT OVERLAPS

Indicates that the values for the non-period columns and expressions of the index key for a row must be unique with respect to the time represented by the BUSINESS_TIME period for the row. Db2 enforces that multiple rows do not exist with the same key values for the columns or expressions of the index, with overlapping time periods. The BUSINESS_TIME WITHOUT OVERLAPS clause is intended for use in defining a unique index to enforce a primary key or unique constraint.

BUSINESS_TIME WITHOUT OVERLAPS can only be specified for an index that is defined as UNIQUE.

aux-table-name

Identifies the auxiliary table on which the index is created. The name must identify an auxiliary table that exists at the current server. If the auxiliary table already has an index, do not create another one. An auxiliary table can only have one index.

Do not specify any columns for the index key. The key value is implicitly defined as a unique 19 byte value that is system generated.

If qualified, *table-name* or *aux-table-name* can be a two-part or three-part name. If a three-part name is used, the first part must match the value of the field Db2 LOCATION NAME of installation panel DSNTIPR at the current server. (If the current server is not the local Db2, this name is not necessarily

the name in the CURRENT SERVER special register.) Whether the name is two-part or three-part, the authorization ID that qualifies the name is the owner of the index.

The table space that contains the named table must be available to Db2 so that its data sets can be opened. If the table space is EA-enabled, the data sets for the index must be defined to belong to a DFSMS data class that has the extended format and addressability attributes.

GENERATE KEY USING

Along with XMLPATTERN, GENERATE KEY USING is required to generate an XML index.

XMLPATTERN

When an XML column is indexed, only parts of the documents will be indexed. To identify those parts, a path expression that follows the XMLPATTERN clause is specified. Only values of those element, attribute, or text nodes which match the specified pattern are indexed. An XML pattern can be specified using an optional namespace declaration where namespace prefixes are mapped to namespace URIs and by providing a path expression. The path expression is similar to a path expression in XQuery except that the paths that are specified for the XML index can support child axis, self-or-descendant axis, wildcard expressions, or attribute only. The maximum length of an XML pattern text is 4000 bytes after being converted to UTF-8. For more information about XQuery, see [Overview of pureXML \(Db2 Programming for XML\)](#).

prolog

To use qualified names in the *pattern-expression*, namespace prefixes need to be declared. A default namespace can also be declared for use with unqualified names.

declare namespace *NCName=StringLiteral*

The namespace prefix, *NCName*, is mapped to a namespace URI that is identified in *StringLiteral*. Multiple namespaces can be declared, but each namespace prefix must be unique within the list of namespace declarations. *NCName* is an XML name as defined by the XML 1.0 standard. *NCName* cannot include a colon character. The namespace URI cannot be `http://www.w3.org/XML/1998/namespace` or `http://w3.org/2000/xmlns/`.

declare default element namespace *StringLiteral*

Specifies the default namespace URI for unqualified names of elements and types. *StringLiteral* is a namespace URI. If no default element namespace is declared, unqualified names of element and types are in no namespace. Only one default namespace can be declared.

pattern-expression

Pattern-expression is used to identify those nodes in an XML document that are indexed. *Pattern-expression* cannot be an empty or invalid string, and the XQuery expression cannot be nested more than 50 levels. *pattern-expression* cannot be an XQuery updating expression.

/ (forward slash)

Separates path expression steps.

// (double forward slash)

Abbreviated syntax for `/descendant-or-self::node()`

.(dot)

Abbreviated syntax for `/self::node()`

child::

Specifies children of the context node. `child::` is the default if no forward axis is specified.

descendant::

Specifies the descendants of the context node.

self::

Specifies the current context node.

descendant-or-self::

Specifies the context node and the descendents of the context node.

element-name

Identifies an element in an XML document. *element-name* is an XML QName that can have one of the following forms:

nsprefix:NCName

nsprefix explicitly specifies a namespace prefix that must be declared.

NCName

An unqualified XML name that uses the default namespace.

*** (an asterisk)**

Indicates any element name. If * is prefixed by *attribute::* or @, * indicates any attribute name.

nsprefix:*

Indicates any NCName within the specified namespace.

****:NCName***

Indicates a specific XML name in any of the currently declared namespaces.

***attribute::* or @**

Specifies attributes of the context node.

attribute-name

Identifies an attribute in an XML document. *attribute-name* is an XML QName that can have one of the following forms:

nsprefix:NCName

nsprefix explicitly specifies a namespace prefix that must be declared.

NCName

An unqualified XML name that uses the default namespace.

text()

Matches any text node.

fn:upper-case()

Specifies an element node or an attribute node that identifies the key value for the index for each node that is specified by the context step (the part of *pattern-expression* that is specified prior to *fn:upper-case*).

The context step of *fn:upper-case()* must specify an element node or an attribute node. The argument of *fn:upper-case()* must be a self step. The key values of an XML value index must be specified as the SQL data type VARCHAR. The length of the VARCHAR value can be any value that is allowed in Db2.

fn:exists()

Specifies an element node that identifies the key value for the index for each node that is specified by the context step (the part of *pattern-expression* that is specified prior to *fn:exists*).

The context step of *fn:exists()* must specify an element node. The argument of *fn:exists()* must be either a single step of a child element node or an attribute node. The name test part can be a wildcard character for either the namespace prefix or NCName. The key values of an XML value index for an XPath expression that ends with *fn:exists()* must be specified as the SQL data type VARCHAR(1). The key value will be "T" or "F". "T" implies that *fn:exists()* evaluates to true and "F" implies that *fn:exists()* evaluates to false.

AS SQL data-type

Specifies that indexed values are stored as an instance of the specified SQL data type. Casting to the specified data type can result in a loss of precision of the values. For example, a loss of precision can occur when an XML integer value is cast to the SQL data type DECFLOAT. If the cast causes a loss of precision, the result will be rounded to the approximate value when it is stored in the index. The cast result cannot be outside of the range that is supported by the SQL data type. If the value cannot be cast to the specified data type, the document is still inserted into the table, but the index entry for that value is not created. No error or warning code is returned.

If the index is unique, the uniqueness is enforced on the value after it is cast to the specified type. Because rounding can occur during the cast to the SQL data type, if a value is cast to the same key value as a document that the table already contains, Db2 will return duplicate key errors at insert time, or fail to create the index.

VARCHAR (*integer*)

The length *integer* is a value in the range 1 - 1000 bytes. If VARCHAR is specified with a length, the specified length is treated as a constraint. If documents are inserted into a table (or exist in the table at create index time) that have nodes with values that are longer than the specified length, the insert or index creation will fail.

DECFLOAT

DECFLOAT can be specified to index numeric values. For the cast to succeed, the string must be a valid XML numeric type. Otherwise the value will be ignored and no insert to the index will occur. The result of the cast cannot be outside of the range that DECFLOAT can represent. Because the XML Schema data type for numeric values allows greater precision than the SQL data type, the result might be rounded to fit into the SQL data type. The DECFLOAT values that are stored in the index are the normalized numeric values.

DATE

The SQL DATE data type values will be normalized to UTC (Coordinated Universal Time) before being stored in the index. For invalid xs:date values, the value will be ignored without being inserted into the index. The XML schema data type for DATE allows for greater precision than the SQL data type. If an out-of-range value is encountered, an error is returned.

TIMESTAMP (12)

The SQL TIMESTAMP data type values will be normalized to UTC (Coordinated Universal Time) before being stored in the index. If the value that is specified in the document does not specify the time zone, Db2 will use the implicit time zone to normalize the value to UTC. For invalid xs:dateTime values, the value will be ignored without being inserted into the index. The XML schema data type for timestamps allows for greater precision than the SQL data type. If an out-of-range value is encountered, an error is returned. Only a precision of 12 fractional digits is allowed for an SQL TIMESTAMP index key.

INCLUDE (*column-name*)

Specifies additional columns to append to the set of index key columns of a unique index. Any column that is specified using INCLUDE *column-name* is not used to enforce uniqueness. The included columns might improve performance for some queries using index only access.

The UNIQUE clause must be specified when INCLUDE is specified. Columns that are specified in the INCLUDE clause count towards the limits for the number of columns and the limits on the sum of the length attributes of the columns that are specified in the index. The total number of columns for the index cannot exceed 64.

column-name must be distinct from the columns that are used to enforce uniqueness and from other columns specified in the INCLUDE clause. *column-name* must be unqualified, must identify a column of the specified table, and must not be one of the existing columns of the index. *column-name* must not identify a LOB or DECFLOAT column (or a distinct type that is based on one of those types).

The INCLUDE clause cannot be specified for the following types of indexes:

- A non-unique index
- A partitioning index when index-controlled partitioning is used
- An auxiliary index
- An XML index
- An extended index
- An expression-based index

Columns in the INCLUDE list that are defined as character or graphic string data types must be defined with the same encoding scheme as other key columns with character or graphic string data types.

CLUSTER or NOT CLUSTER

Specifies whether the index is the clustering index for the table. This clause must not be specified for an index on an auxiliary table, or on a table that is defined to use hash organization.

CLUSTER

The index is to be used as the clustering index of the table. CLUSTER cannot be specified if XMLPATTERN or *key-expression* is specified.

NOT CLUSTER

The index is not to be used as the clustering index of the table.

PARTITIONED

Specifies that the index is data partitioned (that is, partitioned according to the partitioning scheme of the underlying data). A partitioned index can be created only on a partitioned table space, not on a partition-by-growth table space. PARTITIONED cannot be specified if XMLPATTERN is specified. The types of partitioned indexes are partitioning and secondary.

An index is considered a partitioning index if the specified index key columns match or comprise a superset of the columns specified in the partitioning key, are in the same order, and have the same ascending or descending attributes.

If PARTITION BY was not specified when the table was created, the CREATE INDEX statement must have the ENDING AT clause specified to define a partitioning index and use index-controlled partitioning. This index is created as a partitioned index even if the PARTITIONED keyword is not specified. When a partitioning index is created, if both the PARTITIONED and ENDING AT keywords are omitted, the index will be non-partitioned. If PARTITIONED is specified, the USING specification with PRIQTY and SECQTY specifications are optional. If these space parameters are not specified, default values are used.

A secondary index is any index defined on a partitioned table space that does not meet the definition of the partitioning index. For partitioned secondary indexes (data-partitioned secondary indexes), the ENDING AT clause is not allowed because the partitioning scheme of the index is predetermined by that of the underlying data. UNIQUE and UNIQUE WHERE NOT NULL are allowed only if the columns in the index are a superset of the partitioning columns. All of the index columns must be specified in a *table-name(column-name)* clause, and not in an INCLUDE clause. If a partitioned secondary index is created on a table that uses index-controlled partitioning, the table is converted to use table-controlled partitioning.

Index-controlled partitioning cannot be used if the PREVENT_NEW_IXCTRL_PART subsystem parameter is set to YES.

For more information, see [PREVENT_NEW_IXCTRL_PART](#) in macro DSN6SPRM (Db2 Installation and Migration).

NOT PADDED or PADDED

Specifies how varying-length string columns are to be stored in the index. If the index contains no varying-length columns, this option is ignored, and a warning message is returned. Indexes that do not have varying-length string columns are always created as physically padded indexes.

NOT PADDED

Specifies that varying-length string columns are not to be padded to their maximum length in the index. The length information for a varying-length column is stored with the key.

NOT PADDED is ignored and has no effect if the index is being created on an auxiliary table. Indexes on auxiliary tables are always padded.

PADDED

Specifies that varying-length string columns within the index are always padded with the default pad character to their maximum length. PADDED cannot be specified if XMLPATTERN is specified. PADDED cannot be specified for indexes that are defined on VARBINARY columns.

When the index contains at least one varying-length column, the default for the option depends on the value of field PAD INDEXES BY DEFAULT on installation panel DSNTIPE:

- When the value of this field is NO, new indexes are not padded unless PADDED is specified.
- When the value of this field is YES, new indexes are padded unless NOT PADDED is specified.

USING (for non-partitioned indexes)

For non-partitioned indexes, the USING clause indicates whether the data sets for the index are to be managed by the user or managed by Db2. If Db2 definition is specified, the clause also gives space allocation parameters (PRIQTY and SECQTY) and an erase rule (ERASE).

If you omit USING, the data sets Db2 manages on volumes listed in the default storage group of the database that is associated with the table. The default storage group for the database must exist. With no USING clause, PRIQTY, SECQTY, and ERASE assume their default values.

VCAT *catalog-name*

Specifies that the first data set for the index is managed by the user, and that following data sets, if needed, are also managed by the user.

The data sets are VSAM linear data sets cataloged in the integrated catalog facility catalog that *catalog-name* identifies. For more information about *catalog-name* values, see [Naming conventions \(Db2 SQL\)](#).

More than one Db2 subsystem can share the integrated catalog facility catalogs with the current server. To avoid the chance of those subsystems attempting to assign the same name to different data sets, specify a *catalog-name* value that is not used by the other Db2 subsystems.

Do not specify VCAT in any of the following circumstances:

- For an index on a declared temporary table.
- If the table space is partition-by-growth, and the table space is not part of the Db2 catalog.

STOGROUP *stogroup-name*

Specifies that Db2 will define and manage the data sets for the index. Each data set will be defined on a volume listed in the identified storage group. The values specified (or the defaults) for PRIQTY and SECQTY determine the primary and secondary allocations for the data set. If $PRIQTY + 118 \times SECQTY$ is 2 gigabytes or greater, more than one data set could eventually be used, but only the first is defined during execution of this statement.

To use USING STOGROUP, the privilege set must include one of the following, except when creating an index on a declared global temporary table if *stogroup-name* matches the default storage group of the work file database:

- SYSADM authority
- SYSCTRL authority
- The USE privilege for that storage group

Moreover, *stogroup-name* must identify a storage group that exists at the current server and includes in its description at least one volume serial number. The description can indicate that the choice of volumes will be left to Storage Management Subsystem (SMS). Each volume specified in the storage group must be accessible to z/OS for dynamic allocation of the data set, and all these volumes must be of the same device type.

The integrated catalog facility catalog used for the storage group must **not** contain an entry for the first data set of the index. If the catalog is password protected, the description of the storage group must include a valid password.

The storage group supplies the data set name. The first level qualifier is also the name of, or an alias for, the integrated catalog facility catalog on which the data set is to be cataloged. The naming convention for the data set is the same as if the data set is managed by the user.

PRIQTY *integer*

Specifies the minimum primary space allocation for a Db2-managed data set. *integer* must be a positive integer, or -1. When you specify PRIQTY with a positive integer value, the primary space allocation is at least *n* kilobytes, where *n* is:

12

If *integer* is greater than 0 and less than 12.

integer

If *integer* is between 12 and 4194304.

2097152

If both of the following conditions are true:

- *integer* is greater than 2097152.
- The index is a non-partitioned index on a table space that is not defined with the LARGE or DSSIZE attribute.

4194304

If *integer* is greater than 4194304.

If you do not specify PRIQTY, or you specify a PRIQTY value of -1, Db2 uses a default value for the primary space allocation. For information on how Db2 determines the default value, see .

If you specify PRIQTY, and do not specify a value of -1, Db2 specifies the primary space allocation to access method services using the smallest multiple of 4KB not less than *n*. The allocated space can be greater than the amount of space requested by Db2. For example, it could be the smallest number of tracks that will accommodate the space requested. To more closely estimate the actual amount of storage, see [DEFINE CLUSTER command \(DFSMS Access Method Services for Catalogs\)](#).

When determining a suitable value for PRIQTY, be aware that two of the pages of the primary space could be used by Db2 for purposes other than storing index entries.

SECQTY *integer*

Specifies the minimum secondary space allocation for a Db2-managed data set. *integer* must be a positive integer, 0, or -1. If you do not specify SECQTY, or specify a SECQTY value of -1, Db2 uses a formula to determine a value. For information on the actual value that is used for secondary space allocation, whether you specify a value or not, see "Rules for primary and secondary space allocation" in [CREATE TABLESPACE \(Db2 SQL\)](#).

If you specify SECQTY, and do not specify a value of -1, Db2 specifies the secondary space allocation to access method services using the smallest multiple of 4KB not less than *integer*. The allocated space can be greater than the amount of space requested by Db2. For example, it could be the smallest number of tracks that will accommodate the space requested. To more closely estimate the actual amount of storage, see [DEFINE CLUSTER command \(DFSMS Access Method Services for Catalogs\)](#).

ERASE

Indicates whether the Db2-managed data sets are to be erased when they are deleted during the execution of a utility or an SQL statement that drops the index.

NO

Does not erase the data sets. Operations involving data set deletion will perform better than ERASE YES. However, the data is still accessible, though not through Db2. This is the default.

YES

Erases the data sets. As a security measure, Db2 overwrites all data in the data sets with zeros before they are deleted.

USING (partitioned indexes)

If the index is partitioned, there is a PARTITION clause for each partition. Within a PARTITION clause, a USING clause is optional. If a USING clause is present, it applies to that partition in the same way that a USING clause for a secondary index applies to the entire index.

When a USING specification is absent from a PARTITION clause, the USING clause parameters for the partition depend on whether a USING clause is specified before the PARTITION clauses.

- If the USING clause is specified, it applies to every PARTITION clause that does not include a USING clause.
- If the USING clause is not specified, the following defaults apply to the partition:

- Data sets are managed by Db2.
- The default storage group for the database is used. If the USING clause for the index space is omitted, the default storage group for database must exist.
- Default values of -1 are used for both PRIQTY and SECQTY.
- A value of NO is used for ERASE.

VCAT *catalog-name*

Specifies a user-managed data set with a name that starts with the specified catalog name. The identified integrated catalog facility catalog must already contain an entry for the *n*th data set of the index, where *n* is the partition number.

The data sets are VSAM linear data sets cataloged in the integrated catalog facility catalog that *catalog-name* identifies. For more information about *catalog-name* values, see [Naming conventions \(Db2 SQL\)](#).

More than one Db2 subsystem can share the integrated catalog facility catalogs with the current server. To avoid the chance of those subsystems attempting to assign the same name to different data sets, specify a *catalog-name* value that is not used by the other Db2 subsystems.

Db2 assumes one and only one data set for each partition.

STOGROUP *stogroup-name*

If USING STOGROUP is used, explicitly or by default, for a partition *n*, Db2 defines the data set for the partition during the execution of the CREATE INDEX statement, using space from the named storage group. The privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for that storage group. The integrated catalog facility catalog used for the storage group must NOT contain an entry for the *n*th data set of the index.

stogroup-name must identify a storage group that exists at the current server and the privilege set must include one of the following privileges or authorities, except when creating an index on a declare global temporary table if *stogroup-name* matches the default storage group of the work file database:

- SYSADM authority
- SYSCTRL authority
- USE privilege for the storage group

If you omit PRIQTY, SECQTY, or ERASE from a USING STOGROUP clause for some partition, their values are given by the next USING STOGROUP clause that governs that partition: either a USING clause that is not in any PARTITION clause, or a default USING clause. Db2 assumes one and only one data set for each partition.

FREEPAGE *integer*

Specifies how often to leave a page of free space when index entries are created as the result of executing a Db2 utility or when creating an index for a table with existing rows. One free page is left for every *integer* pages. The value of *integer* can range 0 - 255. The default is 0, leaving no free pages.

Do not specify FREEPAGE for an index on a declared temporary table.

PCTFREE *integer*

Determines the percentage of free space to leave in each nonleaf page and leaf page when entries are added to the index or index partition as the result of executing a Db2 utility or when creating an index for a table with existing rows. The first entry in a page is loaded without restriction. When additional entries are placed in a nonleaf or leaf page, the percentage of free space is at least as great as *integer*.

The value of *integer* can range from 0 to 99, however, if a value greater than 10 is specified, only 10 percent of free space will be left in nonleaf pages. The default is 10.

Do not specify PCTFREE for an index on a declared temporary table.

If the index is partitioned , the values of FREEPAGE and PCTFREE for a particular partition are given by the first of these choices that applies:

- The values of FREEPAGE and PCTFREE given in the PARTITION clause for that partition. Do not use more than one *free-specification* in any PARTITION clause.
- The values given in a *free-specification* that is not in any PARTITION clause.
- The default values FREEPAGE 0 and PCTFREE 10.

GBPCACHE

In a data sharing environment, specifies what index pages are written to the group buffer pool. In a non-data-sharing environment, the option is ignored unless the index is on a declared temporary table. Do not specify GBPCACHE for an index on a declared temporary table in either environment (data sharing or non-data-sharing).

CHANGED

Specifies that updated pages are written to the group buffer pool, when there is inter-Db2 R/W interest on the index or partition. When there is no inter-Db2 R/W interest, the group buffer pool is not used. Inter-Db2 R/W interest exists when more than one member in the data sharing group has the index or partition open, and at least one member has it open for update. GBPCACHE CHANGED is the default.

If the index is in a group buffer pool that is defined as GBPCACHE(NO), CHANGED is ignored and no pages are written to the group buffer pool.

ALL

Indicates that pages are written to the group buffer pool as they are read in from DASD.

Exception: In the case of a single updating Db2 subsystem when no other Db2 subsystems have any interest in the page set, no pages are written to the group buffer pool.

If the index is in a group buffer pool that is defined as GBPCACHE(NO), ALL is ignored and no pages are written to the group buffer pool.

NONE

Indicates that no pages are written to the group buffer pool. Db2 uses the group buffer pool only for cross-invalidation.

If the index is partitioned, the value of GBPCACHE for a particular partition is given by the first of these choices that applies:

1. The value of GBPCACHE given in the PARTITION clause for that partition. Do not use more than one *gbpcache-specification* in any PARTITION clause.
2. The value given in a *gbpcache-specification* that is not in any PARTITION clause.
3. GBPCACHE CHANGED is the default value.

DEFINE

Specifies when the underlying data sets for the index are physically created. The SPACE column in catalog table SYSINDEXPART is used to record the status of the data sets (undefined or allocated). If the DEFINE keyword is not specified, the define attribute is inherited from the current state of the base table space.

YES

The data sets are created when the index is created (the CREATE INDEX statement is executed).

NO

The data sets are not created until data is inserted into the index.

DEFINE NO is applicable only for Db2-managed data sets (USING STOGROUP is specified). Use DEFINE NO especially when performance of the CREATE INDEX statement is important or DASD resource is constrained.

Do not use DEFINE NO on an index if you use a program outside of Db2 to propagate data into a table on which that index is defined. If you use DEFINE NO on an index of a table and data is then propagated into the table from a program that is outside of Db2, the index space data sets are

allocated, but the Db2 catalog will not reflect this fact. As a result, Db2 treats the data sets for the index space as if they have not yet been allocated. The resulting inconsistency causes Db2 to deny application programs access to the data until the inconsistency is resolved.

DEFINE NO is ignored for user-managed data sets (USING VCAT is specified). DEFINE NO is also ignored if the index is being created on a table that is not empty.

Do not specify DEFINE NO if the index is created on a base table that is involved in a clone relationship.

Do not specify DEFINE NO for an index on a declared temporary table.

COMPRESS NO or COMPRESS YES

Specifies whether compression for index data will be used. If the index is partitioned, the clause will apply to all partitions.

COMPRESS NO

Specifies that no index compression will be used.

COMPRESS NO is the default.

COMPRESS YES

Specifies that index compression will be used. The buffer pool that is used to create the index must be 8K, 16K, or 32K in size. The physical page size on disk will be 4K. The index compression will take place immediately.

Index compression is recommended for applications that do sequential insert operations with few or no delete operations. Random inserts and deletes can adversely effect compression. Index compress is also recommended for applications where the indexes are created primarily for scan operations.

INCLUDE NULL KEYS or EXCLUDE NULL KEYS

Specifies whether an index entry will be created when every key column contains the NULL value.

INCLUDE NULL KEYS

Specifies that an index entry will be created when every key column contains the NULL value.

INCLUDE NULL KEYS is the default.

EXCLUDE NULL KEYS

Specifies that no index entry will be created when every key column contains the NULL value. If any key column is not null the index entry will be created.

EXCLUDE NULL KEYS must not be specified with the following:

- UNIQUE
- BUSINESS_TIME WITHOUT OVERLAPS
- *XML-index-specification*
- *key-expression*
- INCLUDE (*column-name*)

EXCLUDE NULL KEYS must also not be specified if any of the columns that are identified by *column-name* are defined as NOT NULL, or if the index is defined as a partitioning index for use with index-controlled partitioning.

PARTITION BY RANGE

Specifies the partitioning index for the table, which determines the partitioning scheme for the data in the table.

PARTITION BY RANGE should only be specified if the table space is partitioned and the partitioning schema has not already been established.

PARTITION BY RANGE must not be specified if the index is an extended index, is defined with the BUSINESS_TIME WITHOUT OVERLAPS, or if the table is in a universal table space (ranged-partitioned or partition-by-growth table space).

partition-element

Specifies the range for each partition.

PARTITION *integer*

A PARTITION clause specifies the highest value of the index key in one partition of a partitioning index. In this context, highest means highest in the sorting sequences of the index columns. In a column defined as *ascending* (ASC), highest and lowest have their usual meanings. In a column defined as *descending* (DESC), the lowest actual value is highest in the sorting sequence.

If you use CLUSTER, and the table is contained in a partitioned table space, you must use exactly one PARTITION clause for each partition (defined with Numparts on CREATE TABLESPACE). If there are *p* partitions, the value of *integer* must range from 1 through *p*.

The length of the highest value of a partition (also called the limit key) is the same as the length of the partitioning index.

ENDING AT(*constant*, MAXVALUE, or MINVALUE...)

Specifies that this is the partitioning index and indicates how the data will be partitioned. The table space is marked complete after this partitioning index is created. You must use at least one value (*constant*, MAXVALUE, or MINVALUE) after ENDING AT in each PARTITION clause. You can use as many as there are columns in the key. The concatenation of all the values is the highest value of the key in the corresponding partition of the index unless the VALUES statement was already specified when the table or previous index was created.

constant

Specifies a constant value with a data type that must conform to the rules for assigning that value to the column. If a string constant is longer or shorter than required by the length attribute of its column, the constant is either truncated or padded on the right to the required length. If the column is ascending, the padding character is X'FF'. If the column is descending, the padding character is X'00'. The precision and scale of a decimal constant must not be greater than the precision and scale of its corresponding column. A hexadecimal string constant (GX) cannot be specified.

MAXVALUE

Specifies a value greater than the maximum value for the limit key of a partition boundary (that is, all X'FF' regardless of whether the column is ascending or descending). If all of the columns in the partitioning key are ascending, a constant or the MINVALUE clause cannot be specified following MAXVALUE. After MAXVALUE is specified, all subsequent columns must be MAXVALUE.

MINVALUE

Specifies a value that is smaller than the minimum value for the limit key of a partition boundary (that is, all X'00' regardless of whether the column is ascending or descending). If all of the columns in the partitioning key are descending, a constant or the MAXVALUE clause cannot be specified following MAXVALUE. After MINVALUE is specified, all subsequent columns must be MINVALUE.

The key values are subject to the following rules:

- The first value corresponds to the first column of the key, the second value to the second column, and so on. Using fewer values than there are columns in the key has the same effect as using the highest or lowest values for the omitted columns, depending on whether they are ascending or descending.
- If a key includes a ROWID column or a column with a distinct type that is based on a ROWID data type, 17 bytes of the constant that is specified for the corresponding ROWID column are considered.
- The highest value of the key in any partition must be lower than the highest value of the key in the next partition.
- If the concatenation of all the values exceeds 255 bytes, only the first 255 bytes are considered.

- The highest value of the key in the last partition depends on how the table space is defined. For table spaces that are created without the LARGE or DSSIZE options, the values that you specify after VALUES are not enforced. The highest value of the key that can be placed in the table is the highest possible value of the key.

For large partitioned table space, the values you specify are enforced. The value specified for the last partition is the highest value of the key that can be placed in the table. Any key values greater than the value that is specified for the last partition are out of range.

ENDING AT can be specified only if the ENDING AT clause was not specified on a previous CREATE or ALTER TABLE statement for the underlying table.

INCLUSIVE

Specifies that the specified range values are included in the data partition.

BUFFERPOOL *bpname*

Identifies the buffer pool that is to be used for the index. The privilege set must include SYSADM or SYSCTRL authority or the USE privilege for the buffer pool, except when creating an index on a declared global temporary table and *bpname* matches the default index buffer pool of the work file database.

The *bpname* must identify an activated 4KB, 8KB, 16KB, or 32KB buffer pool.

A buffer pool with a smaller size should be chosen for indexes with random insert patterns. A buffer pool with a larger size should be chosen for indexes with sequential insert patterns.

For more details about *bpname*, see [Naming conventions \(Db2 SQL\)](#). For a description of active and inactive buffer pools, see [Controlling Db2 databases and buffer pools \(Db2 Administration Guide\)](#).

CLOSE

Specifies whether or not the data set is eligible to be closed when the index is not being used and the limit on the number of open data sets is reached.

YES

Eligible for closing. This is the default unless the index is on a declared temporary table.

NO

Not eligible for closing.

If the limit on the number of open data sets is reached and there are no page sets that specify CLOSE YES to close, page sets that specify CLOSE NO will be closed.

For an index on a declared temporary table, Db2 uses CLOSE NO regardless of the value specified.

DEFER

Indicates whether the index is built during the execution of the CREATE INDEX statement. Regardless of the option specified, the description of the index and its index space is added to the catalog. If the table is determined to be empty and DEFER YES is specified, the index is neither built nor placed in a rebuild-pending status. For more information about using DEFER, see [Index names and guidelines \(Db2 Administration Guide\)](#). Do not specify DEFER for an index on a declared temporary table or an auxiliary table.

NO

The index is built. This is the default.

YES

The index is not built. If the table is populated, the index is placed in a rebuild-pending status and a warning message is issued; the index must be rebuilt by the REBUILD INDEX utility.

DSSIZE *integer G*

Specifies the maximum size for each partition of a partitioned index. Any integer 1 - 1024 can be specified (for example, 1G – 1024G). This keyword is not valid on nonpartitioned secondary indexes. You can only specify DSSIZE on CREATE INDEX if the index is on a table space with relative page numbers.

To specify a value greater than 4G, the data sets for the table space must be associated with a DFSMS data class that has been specified with extended format and extended addressability.

If the index is a partitioned index using relative page numbering, the value of DSSIZE for a particular partition is given by the first of these choices that applies:

- The value of DSSIZE given in the PARTITION clause for that partition.
- The value given by a DSSIZE keyword that is not in any PARTITION clause.
- The default value is inherited from the base table space.

PIECESIZE integer

Specifies the maximum addressability of each data set for a non-partitioned index. The subsequent keyword K, M, or G, indicates the units of the value that is specified in *integer*. *integer* can be separated from K, M, or G by 0 or more spaces.

K

Indicates that the *integer* value is to be multiplied by 1024 to specify the maximum data set size in bytes. *integer* must be a power of two between 1 and 268435456.

M

Indicates that the *integer* value is to be multiplied by 1048576 to specify the maximum data set size in bytes. *integer* must be a power of two between 1 and 262144.

G

Indicates that the *integer* value is to be multiplied by 1073741824 to specify the maximum data set size in bytes. *integer* must be a power of two between 1 and 256.

The following table shows the valid values for the data set size, which depend on the size of the table space.

<i>Table 2. Valid values of PIECESIZE clause</i>			
K units	M units	G units	Size attribute of table space
256K			
512 K			
1024 K	1 M		
2048 K	2 M		
4096 K	4 M		
8192 K	8 M		
16384 K	16 M		
32768 K	32 M		
65536 K	64 M		
131072 K	128 M		
262144 K	256 M		
524288 K	512 M		
1048576 K	1024 M	1 G	
2097152 K	2048 M	2 G	
4194304 K	4096 M	4 G	LARGE, DSSIZE 4 G (or greater)
8388608 K	8192 M	8 G	DSSIZE 8 G (or greater)
16777216 K	16384 M	16 G	DSSIZE 16 G (or greater)
33554432 K	32768 M	32 G	DSSIZE 32 G (or greater)
67108864 K	65536 M	64 G	DSSIZE 64 G (or greater)

Table 2. Valid values of PIECESIZE clause (continued)

K units	M units	G units	Size attribute of table space
134217728 K	131072 M	128 G	DSSIZE 128 G (or greater)
268435456 K	262144 M	256 G	DSSIZE 256 G

PIECESIZE has no effect on primary and secondary space allocation as it is only a specification of the maximum amount of data that a data set can hold and not the actual allocation of storage.

If you change the PIECESIZE value with the ALTER INDEX statement, the index is put into REBUILD-pending status.

See the following for additional information:

- [Number of pieces and maximum piece size for non-partitioned indexes and data-partitioned secondary indexes](#)
- [Choosing a value for PIECESIZE](#)

COPY

Indicates whether the COPY utility is allowed for the index. Do not specify COPY for an index on a declared temporary table.

NO

Does not allow full image or concurrent copies or the use of the RECOVER utility on the index. NO is the default.

YES

Allows full image or concurrent copies and the use of the RECOVER utility on the index.

Notes

Owner privileges:

The owner of the table has all table privileges (see GRANT (table or view privileges) (Db2 SQL)) with the ability to grant these privileges to others. For more information about ownership of the object, see [Authorization, privileges, permissions, masks, and object ownership \(Db2 SQL\)](#).

Effects of the DEFER clause:

If DEFER NO is implicitly or explicitly specified, the CREATE INDEX statement cannot be executed while a Db2 utility has control of the table space that contains the identified table.

If the identified table already contains data and if the index build is not deferred, CREATE INDEX creates the index entries for it. If the table does not yet contain data, CREATE INDEX creates a description of the index; the index entries are created when data is inserted into the table.

Errors evaluating the expressions for an index:

Errors that occur during the evaluation of an expression for an index are returned when the expression is evaluated. This can occur on an SQL data change statement, SELECT from an SQL data change statement, or the REBUILD INDEX utility. For example, the evaluation of the expression `10 / column_1` returns an error if the value in `column_1` is 0. The error is returned during CREATE INDEX processing if the table is not empty and contains a row with a value of zero in `column_1`, otherwise the error is returned during the processing of the insert or update operation when a row with a value of zero in `column_1` is inserted or updated.

Result length of expressions that return a string type:

If the result data type of *key-expression* is a string type and the result length cannot be calculated at bind time, the length is set to the maximum allowable length of that data type or the largest length that Db2 can estimate. In this case, the CREATE INDEX statement can fail because the total key length might exceed the limit of an index key.

For example, the result length of the expression `REPEAT('A', CEIL(1.1))` is VARCHAR(32767) and the result length of the expression `SUBSTR(DESCRIPTION, 1, INTEGER(1.2))` is the length of the DESCRIPTION column. Therefore, a CREATE INDEX statement that uses any of these expressions

as a *key-expression* might not be created because the total key length might exceed the limit of an index key.

Use of ASC or DESC on key columns:

There are no restrictions on the use of ASC or DESC for the columns of a parent key or foreign key. An index on a foreign key does not have to have the same ascending and descending attributes as the index of the corresponding parent key.

EBCDIC, ASCII, and UNICODE encoding schemes for an index:

In general, an index has the same encoding scheme as its associated table. However, if an index on an EBCDIC table consists of only Unicode columns, the encoding scheme of the index is Unicode.

Maximum partition size of a partitioned index

The size of a partitioned index depends on whether the corresponding partitioned table space is created with or without the LARGE or DSSIZE keywords, and on the number of partitions.

The following table provides information about partitioned indexes on table spaces that are created without the LARGE or DSSIZE keywords and with 64 or fewer partitions.

Table 3. Maximum number of pieces and the default size of a partitioned index on a partitioned table space that is created without the LARGE or DSSIZE clauses and with a Numparts value of less than or equal to 64

Definition of partitioned table space (non-large)	Maximum number of pieces for a partitioned index	Default size of a partitioned index, per data set
NUMPARTS <= 16	16	4G
NUMPARTS >= 17 but NUMPARTS <= 32	32	2G
NUMPARTS >= 33	64	1G

The following table shows information about partitioned indexes on table spaces that are created with the LARGE or DSSIZE keywords and with more than 64 partitions.

Table 4. Maximum number of pieces and the default partitioned index size for a partitioned table space that is created with the LARGE or DSSIZE clauses or with a Numparts value of greater than 64

Definition of partitioned table space (large)	Maximum number of pieces for a partitioned index	Default index piece size for a partitioned index
One or more of the following conditions are true: <ul style="list-style-type: none"> • LARGE clause - specified • NUMPARTS greater than 64 but less than 256 	Maximum number of partitions in the partitioned table space	4G
One or more of the following conditions are true: <ul style="list-style-type: none"> • DSSIZE clause - specified • NUMPARTS greater than or equal to 256 	Maximum number of partitions in the partitioned table space	$\text{MIN}(\text{table space DSSIZE}, 2^{32} / (\text{Maximum number of partitions in the table space}) * \text{index page size})$

To calculate the maximum data set size for a partitioned index, you need to first calculate the maximum number of partitions in the table space by using the following formula:

$\text{MIN}(4096, 2^{32} / (\text{table space DSSIZE} / \text{table space page size}))$

After you calculate the maximum number of partitions in the table space, you can calculate the maximum data set size for a partitioned index with the following formula, using the number of partitions that you calculated above:

$$\text{MIN}(\text{table space DSSIZE}, 2^{32} / (\text{Maximum number of partitions in the table space}) * \text{index page size})$$

For an index that is defined with COMPRESS YES, *index page size* is always 4096 (4KB).

For example, suppose that a table space and an index on that table space have the following characteristics:

- DSSIZE: 64 GB
- Page size: 32 KB
- Index page size: 4 KB
- Maximum number of partitions: 2048

Given those characteristics, you can begin by calculating the maximum number of partitions in the table space:

$$\text{MIN}(4096, 2^{32} / (64\text{GB} / 32\text{KB})) = \mathbf{2048}$$

You can then use the value of 2048 to calculate the maximum data set size for the partitioned index:

$$\begin{aligned} &\text{MIN}(64 \text{ GB}, 2^{32} / \mathbf{2048} * 4\text{KB}) \\ &= \text{MIN}(64\text{GB}, 8\text{GB}) \\ &= 8\text{GB} \end{aligned}$$

Number of pieces and maximum piece size for non-partitioned indexes

The largest amount of data that an index can hold is the maximum number of pieces for the index times the maximum amount of data that a piece can hold.

For a non-partitioned index, the maximum amount of data that an index can hold is defined by using the PIECESIZE parameter.

The default piece size for an index is as follows:

- 2 GB (PIECESIZE 2 G) for indexes of table spaces created without the LARGE or DSSIZE option
- 4 GB (PIECESIZE 4 G) for indexes of table spaces created with the LARGE or DSSIZE option
- 4 GB (PIECESIZE 4 G) for auxiliary indexes

The following tables list the maximum number of pieces and the default index piece size for various table spaces.

Table 5. Maximum number of pieces and the default index piece size for a partitioned table space that is created without the LARGE or DSSIZE clauses and has a Numpart value of less than or equal to 64

Definition of partitioned table space (non-large), Numpart value	Maximum number of pieces in a non-partitioned index	Default index piece size for a non-partitioned index
NUMPARTS <= 16	32	2G
NUMPARTS >= 17 but NUMPARTS <= 32	32	2G
NUMPARTS >= 33	32	2G

Table 6. Maximum number of pieces and the default index piece size for a partitioned table space that is created with the LARGE or DSSIZE clauses or has a Numparts value of greater than or equal to 65

Definition of partitioned table space (large)	Maximum number of pieces for a non-partitioned index	Default index piece size for a non-partitioned index
<ul style="list-style-type: none"> • LARGE clause - specified • DSSIZE clause - not specified 	$\text{MIN}(4096, 2^{32}/(x/y))$ - see “1” on page 104	4G
<ul style="list-style-type: none"> • LARGE clause - not specified • DSSIZE clause - not specified • Numparts clause - greater than 64 but less than 256 	$\text{MIN}(4096, 2^{32}/(x/y))$ - see “1” on page 104	4G
<ul style="list-style-type: none"> • LARGE clause - not specified • DSSIZE clause - specified or Numparts clause - greater than or equal to 256 	$\text{MIN}(4096, 2^{32}/(x/y))$ - see “1” on page 104	4G

Note:

1. For a non-partitioned index, the formula $\text{MIN}(4096, 2^{32} / (x / y))$, determines the maximum number of pieces for the non-partitioned index, where x and y have the following values:
 - x is the piece size of the index (stored in the PIECESIZE column of the SYSIBM.SYSINDEXES catalog table)
 - y is the page size of the index (stored in the PGSIZE column of the SYSIBM.SYSINDEXES catalog table)

Table 7. Maximum number of pieces and the default index piece size for a non-partitioned table space

Type of non-partitioned table space	Maximum number of pieces	Default index piece size
non-segmented table space	32	2G
segmented table space	32	2G
LOB, auxiliary, or XML table space	32	4G

Choosing a value for PIECESIZE:

To choose a value for PIECESIZE, divide the size of the non-partitioned index by the number of data sets that you want. For example, to ensure that you have five data sets for the non-partitioned index, and your index is 10MB (and not likely to grow much), specify PIECESIZE 2 M. If your non-partitioned index is likely to grow, choose a larger value.

Remember that 32 data sets is the limit if the underlying table space is not defined as LARGE or with a DSSIZE parameter and that the limit is 4096 for objects with greater than 254 parts. For a non-partitioned index on a table space that is defined as LARGE or with a DSSIZE parameter, the maximum is $\text{MIN}(4096, 2^{32} / (\text{index piece size}/\text{index page size}))$.

Keep the PIECESIZE value in mind when you are choosing values for primary and secondary quantities. Ideally, the value of your primary quantity plus the secondary quantities should be evenly divisible into PIECESIZE.

Dropping an index:

Partitioning indexes can be dropped. If the table space is using index-controlled partitioning, the table space is converted to table-controlled partitioning. Secondary indexes that are not indexes on

auxiliary tables can be dropped simply by dropping the indexes. An empty index on an auxiliary table can be explicitly dropped; a populated index can be dropped only by dropping other objects. For details, see "Dropping an index on a base table and auxiliary table" in [DROP \(Db2 SQL\)](#).

If the index is a unique index that enforces a primary key, unique key, or referential constraint, the constraint must be dropped before the index is dropped. See [DROP \(Db2 SQL\)](#).

Unique indexes and enforcement of UNIQUE or PRIMARY KEY specifications for a table:

A table requires a unique index (that is not defined as UNIQUE WHERE NOT NULL) if you use the UNIQUE or PRIMARY KEY clause in the CREATE or ALTER TABLE statements, or if there is a ROWID column that is defined as GENERATED BY DEFAULT. Db2 implicitly creates those unique indexes if the table space is explicitly created and the CREATE or ALTER TABLE statement is processed by the schema processor or if the table space is implicitly created; otherwise, you must explicitly create them. If any of the unique indexes that must be explicitly defined do not exist, the definition of the table is incomplete, and the following rules apply:

- Let K denote a key for which a required unique index does not exist and let n denote the number of unique indexes that remain to be created before the definition of the table is complete. (For a new table that has no indexes, K is its primary key or any of the keys defined in the CREATE or ALTER TABLE statement as UNIQUE and n is the number of such keys. After the definition of a table is complete, an index cannot be dropped if it is enforcing a primary key or unique key.)
- The creation of the unique index reduces n by one if the index key is identical to K . The keys are identical only if they have the same columns in the same order.
- If n is now zero, the creation of the index completes the definition of the table.
- If K is a primary key, the description of the index indicates that it is a primary index. If K is not a primary key, the description of the index indicates that it enforces the uniqueness of a key defined as UNIQUE in the CREATE or ALTER TABLE statement.

A unique index cannot be created on a materialized query table.

Unique indexes and XML columns:

If the index is an XML index on a unique XML column, the uniqueness applies to values of the specified pattern across all documents of that column, and the uniqueness is enforced on the value after the value is cast to the specified SQL data type. Because the data type conversion might result in a loss of precision and normalization, multiple values that appear unique in the XML document might still result in duplicate errors. If the index is defined using an expression, the uniqueness is enforced against the values that are stored in the index, not against the original values of the columns. The WHERE NOT NULL specification is ignored with a warning if XMLPATTERN is also specified, and the index is treated as if UNIQUE had been specified.

Defining an XML index using an XPath pattern-expression that includes functions:

An XPath *pattern-expression* that includes functions (including fn:exists() or fn:upper-case()) will have two parts. The first part is referred to as the *context step* and specifies the XPath of the element node or attribute node for which an index entry will be created (the element or attributes NodeID will be included in the index). The context step follows the same syntax as the XPath *pattern-expression* for an XML index, except that for fn:exists() it has to specify an element node, and for fn:upper-case() it has to specify an element node or an attribute node.

The second part is referred to as the *function expression step* and specifies the fn:exists() or fn:upper-case() XPath function. The function expression step is the right-most part of an XPath *pattern-expression*. For each node specified by the context step, the function expression step specifies the key value for the index. For example, in the XPath *pattern-expression* /purchaseOrder/items/item/fn:exists(shipDate), the context step is /purchaseOrder/items/item, and the function expression step is fn:exists(shipDate).

Use of PARTITIONED keyword:

When a partitioned index is created and no additional keywords are specified, the index is non-partitioned. If the keyword PARTITIONED is specified, the index is partitioned. If PARTITION BY RANGE is specified, the index is both data-partitioned and key-partitioned because it is defined on the partitioning columns of the table. Any index on a partitioned table space that does not meet the definition of a partitioning index is a secondary index. When a secondary index is created and

no additional keywords are specified, the secondary index is non-partitioned (NPSI). If the keyword **PARTITIONED** is specified, the index is a data-partitioned secondary index (DPSI).

Creating a partitioning index for a table created without partition boundaries:

When a table is created without specifying partition boundaries using the **ENDING AT** clause, the table is incomplete until a partitioning index is created. The first index that is created for a table must specify both the **PARTITION** and the **ENDING AT** clauses.

When the **PARTITION** clause is specified while creating an index, either the **PARTITIONED** clause, or the **ENDING AT** clause must also be specified.

Considerations for tables that are involved in a clone relationship:

If an index is created on a base table that is involved in a clone relationship, an index with the same name is also created on the clone table. The index on the clone table will be placed in rebuild-pending status unless the clone table is empty when the index is created.

Considerations for tables that contain a row change timestamp column:

To create an index that refers to a row change timestamp column in the table, values must already exist in the column for all rows. Values are stored in row change timestamp columns whenever a row is inserted or updated in the table. If the row change timestamp column is added to an existing table that contains rows, the values for the row change timestamp column is not materialized and stored at the time of the **ALTER TABLE** statement. Values are materialized for these rows when they are updated, or when a **REORG** or a **LOAD REPLACE** utility is run on the table or table space.

Restriction on table spaces when there are pending changes to the definition:

A **CREATE INDEX** statement is not allowed if there are pending changes to the definition of the table space or to any objects in the table space. In addition, an index that references an expression cannot be created on a table where the inline length of a LOB column has been changed and the table space has not been reorganized.

Effects of **DEFINE NO and **INCLUDE NULL KEYS** or **EXCLUDE NULL KEYS**:**

When **INCLUDE NULL KEYS** is specified (implicitly or explicitly) with **DEFINE NO** and the table that is being indexed is populated, a warning is returned, the index is created, and the data set is defined. When **EXCLUDE NULL KEYS** is specified, it is possible that the data set will not be defined if the all rows for the table that is being indexed contain the **NULL** value for all key columns. The index will be empty after the **CREATE INDEX** statement. However, if **DEFINE NO** is specified with **EXCLUDE NULL KEYS** a warning is returned.

Creating indexes on Db2 catalog tables:

For details on creating indexes on catalog tables, see [SQL statements allowed on the catalog \(Db2 SQL\)](#).

EA-enabled index data sets:

If an index is created for an EA-enabled table space, the data sets for the index must be set up to belong to a DFSMS data class that has the extended format and extended addressability attributes.

Alternative syntax and synonyms:

To provide compatibility with previous releases of Db2 or other products in the Db2 family, Db2 supports the following keywords when creating a partitioned index:

- **PART *integer* VALUES** as an alternative syntax for **PARTITION *integer* ENDING**. The **PARTITION BY RANGE** keyword that precedes the *partition-element* clause is optional.

Although these keywords are supported as alternatives, they are not the preferred syntax.

User-defined indexes on catalog tables:

If you issue **CREATE INDEX** for an index on a catalog table, and you specify the **USING** clause, Db2 ignores that clause. Instead, Db2 defines and manages the index data sets. The data sets are defined in the same SMS environment that is used for the catalog data sets with default space attributes.

Temporal referential constraints:

An index is required for the foreign key of a temporal referential constraint. The index must be defined in one of the following ways:

- Specify the **BUSINESS_TIME WITH OVERLAPS** clause after the columns and key expressions.

- Specify the end column of the BUSINESS_TIME period, followed by the begin column of the BUSINESS_TIME period as the last two keys of the index. ASC must be used for each of these columns.

When a temporal referential constraint is defined for a table, the first index that is created that meets the criteria for an index on the foreign key, is recorded as a dependency for the constraint. An index used for the foreign key of a temporal referential constraint cannot be dropped. A column cannot be added to an index used for a temporal referential constraint.

Examples

Example 1

Create a unique index, named DSN8D10.XDEPT1, on table DSN8D10.DEPT. Index entries are to be in ascending order by the single column DEPTNO. Db2 is to define the data sets for the index, using storage group DSN8G130. Each data set should hold 1 megabyte of data at most. Use 512 kilobytes as the primary space allocation for each data set and 64 kilobytes as the secondary space allocation. These specifications enable each data set to be extended up to 8 times before a new data set is used— $512\text{KB} + (8 \times 64\text{KB}) = 1024\text{KB}$. Make the index padded.

The data sets can be closed when no one is using the index and do not need to be erased if the index is dropped.

```
CREATE UNIQUE INDEX DSN8D10.XDEPT1
ON DSN8D10.DEPT
(DEPTNO ASC)
PADDED
USING STOGROUP DSN8G130
PRIQTY 512
SECQTY 64
ERASE NO
BUFFERPOOL BP1
CLOSE YES
PIECESIZE 1 M;
```

For the above example, the underlying data sets for the index will be created immediately, which is the default (DEFINE YES). Assuming that table DSN8D10.DEPT is empty, if you wanted to defer the creation of the data sets until data is first inserted into the index, you would specify DEFINE NO instead of accepting the default behavior. Specifying PADDED ensures that the varying-length character string columns in the index are padded with blanks.

Example 2

Create a cluster index, named XEMP2, on table EMP in database DSN8D10. Put the entries in ascending order by column EMPNO. Let Db2 define the data sets for each partition using storage group DSN8G130. Make the primary space allocation be 36 kilobytes, and allow Db2 to use the default value for SECQTY, which for this example is 12 kilobytes (3 times 4KB). If the index is dropped, the data sets need not be erased.

There are to be 4 partitions, with index entries divided among them as follows:

- Partition 1: entries up to H99
- Partition 2: entries above H99 up to P99
- Partition 3: entries above P99 up to Z99
- Partition 4: entries above Z99

Associate the index with buffer pool BP1 and allow the data sets to be closed when no one is using the index. Enable the use of the COPY utility for full image or concurrent copies and the RECOVER utility.

```
CREATE INDEX DSN8D10.XEMP2
ON DSN8D10.EMP
(EMPNO ASC)
USING STOGROUP DSN8G130
PRIQTY 36
```

```

ERASE NO
CLUSTER
PARTITION BY RANGE
(PARTITION 1 ENDING AT('H99'),
 PARTITION 2 ENDING AT('P99'),
 PARTITION 3 ENDING AT('Z99'),
 PARTITION 4 ENDING AT('999'))
BUFFERPOOL BP1
CLOSE YES
COPY YES;

```

Example 3

Create a secondary index, named DSN8D10.XDEPT1, on table DSN8D10.DEPT. Put the entries in ascending order by column DEPTNO. Assume that the data sets are managed by the user with catalog name DSNCAT and each data set is to hold 1GB of data, at most, before the next data set is used.

```

CREATE UNIQUE INDEX DSN8D10.XDEPT1
ON DSN8D10.DEPT
(DEPTNO ASC)
USING VCAT DSNCAT
PIECESIZE 1048576 K;

```

Example 4

Assume that a column named EMP_PHOTO with a data type of BLOB(110K) was added to the sample employee table for each employee's photo and auxiliary table EMP_PHOTO_ATAB was created in LOB table space DSN8D13A.PHOTOLTS to store the BLOB data for the column. Create an index named XPHOTO on the auxiliary table. The data sets are to be user-managed with catalog name DSNCAT.

```

CREATE UNIQUE INDEX DSN8D10.XPHOTO
ON DSN8D10.EMP_PHOTO_ATAB
USING VCAT DSNCAT
COPY YES;

```

In this example, no columns are specified for the key because auxiliary indexes have implicitly generated keys.

Related concepts

[Implementing Db2 indexes \(Db2 Administration Guide\)](#)

[Naming conventions \(Db2 SQL\)](#)

CREATE MASK

The CREATE MASK statement creates a column mask at the current server. A *column mask* is used for column access control and specifies the value that should be returned for a specified column.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

The privilege set that is defined below must include the following authority:

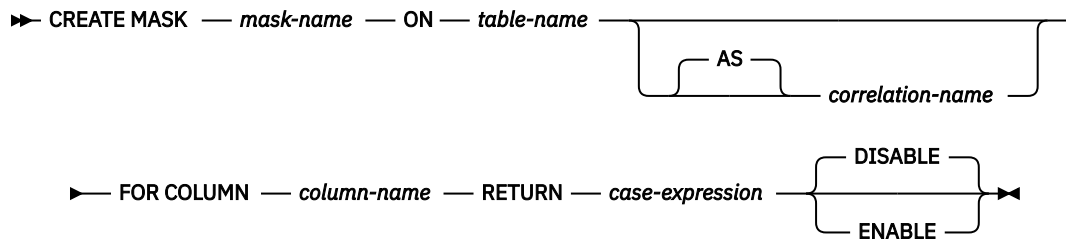
SECADM authority

SECADM authority can create a column mask in any schema. Additional privileges are not needed to reference other objects in the mask definition. For example, the SELECT privilege is not needed query a table, and the EXECUTE privilege is not needed to invoke a user-defined function.

Privilege set: If the statement is embedded in an application program, the privilege set is the set of privileges that are held by the owner of the package. If the statement is dynamically prepared, the privilege set is the set of privileges that are held by the SQL authorization ID of the process. However,

if the process is running in a trusted context that is defined with the `ROLE AS OBJECT OWNER AND QUALIFIER` clause, the privilege set is the set of privileges that are held by the role that is in effect.

Syntax



Description

mask-name

Specifies the names the column mask. The name, including the implicit or explicit qualifier, must not identify a column mask or a row permission that already exists at the current server.

ON table-name

Identifies the table for which the column mask is created. The name must identify a table that exists at the current server. It must not identify any of the following objects:

- An auxiliary table
- A created or declared temporary table
- A view
- A catalog table
- An alias
- A synonym
- A materialized query table or table that is directly or indirectly referenced in the definition of a materialized query table
- A table that was implicitly created for an XML column
- A table that contains a period
- A history table
- An accelerator-only table
- An archive-enabled table
- An archive table

correlation-name

Specifies a correlation name that can be used within *case-expression* to designate the table. For information about *correlation-name*, see [Correlation names \(Db2 SQL\)](#).

FOR COLUMN column-name

Identifies the column to which the mask applies. *column-name* must be an unqualified name that identifies a column of the specified table. A mask must not already exist for the column. The column must not be:

- a LOB column or a distinct type column that is based on a LOB
- an XML column
- defined with a `FIELDPROC`

RETURN case-expression

Specifies a CASE expression that determines the value that is returned for the column. The result of the CASE expression is returned in place of the column value in a row. The result data type,

null attribute, data length, subtype, encoding scheme, and CCSID of the CASE expression must be identical to those attributes of the column that is specified by *column-name*. If the data type of *column-name* is a user-defined data type, the result data type of the CASE expression must be the same user-defined type. The CASE expression must not reference any of the following objects:

- A remote object
- The table for which the column mask is being defined
- A created global temporary table or a declared global temporary table
- An auxiliary table
- A table that was implicitly created for an XML column
- A column that is defined with a FIELDPROC
- A LOB column or a distinct type column that is based on a LOB
- An XML column
- A select list notation *** or *name . ** in the SELECT clause
- A table function
- A collection-derived table (UNNEST)
- A user-defined function that is defined as not secure
- A function that is not deterministic or that has an external action or is defined with the MODIFIES SQL DATA option
- An aggregate function, unless it is specified in a subquery
- A built-in table function
- An XMLTABLE table function
- An XMLEXISTS predicate
- An OLAP specification
- A ROW CHANGE expression
- A sequence reference
- A host variable, SQL variable, SQL parameter, or trigger transition variable
- A parameter marker
- A table reference that contains a period specification
- A view that includes any of the preceding restrictions in its definition
- An accelerator-only table
- An AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY function.

The encoding scheme of the table is used to evaluate the CASE expression. Tables and language elements that require multiple encoding scheme evaluation, other than EBCDIC tables with Unicode columns, must not be referenced in the CASE expression. See [Determining the encoding scheme and CCSID of a string \(Introduction to Db2 for z/OS\)](#) for language elements that require multiple evaluation.

If the CASE expression references tables for which row or column access control is active, access controls for those tables are not cascaded.

DISABLE or ENABLE

Specifies that the column mask is to be enabled or disabled for column access control.

DISABLE

Specifies that the column mask is to be disabled for column access control. The column mask will remain disabled regardless of whether column access control is activated for the table.

DISABLE is the default.

ENABLE

Specifies that the column mask is to be enabled for column access control. If column access control is not currently active for the table, the column mask will become enabled when column access control is activated for the table. If column access control is currently active for the table, the column mask becomes enabled immediately and all packages and statements in the dynamic statement cache that reference the table are invalidated. For more information, see [Changes that invalidate packages \(Db2 Application programming and SQL\)](#).

Notes

How column masks affect queries:

The application of enabled column masks does not interfere with the operations of other clauses within the statement such as the WHERE, GROUP BY, HAVING, SELECT DISTINCT, or ORDER BY. The rows that are returned in the final result table remain the same, except that the values in the resulting rows might have been masked by the column masks. As such, if the masked column also appears in an ORDER BY clause with a *sort-key* expression, the order is based on the original values of the column and the masked values in the final result table might not reflect that order. Similarly, the masked values might not reflect the uniqueness enforced by a SELECT DISTINCT statement. If the masked column is embedded in an expression, the result of the expression might become different because the column mask is applied on the column before the expression evaluation can take place. For example, a column mask on column SSN might change the result of the aggregate function COUNT(DISTINCT SSN) because the DISTINCT operation is performed on the unmasked values.

Conflicts between the definition of a column mask and SQL:

A column mask is created as a stand alone object, without knowing all of the contexts in which it might be used. To mask the value of a column in the final result table, the definition of the column mask is merged into a query by Db2. When the definition of the column mask is brought into the context of the statement, it might conflict with certain SQL semantics in the statement. Therefore, in some situations, the combination of the statement and the application of the column mask can return an error. When this happens, either the statement needs to be modified or the column mask must be dropped or re-created with a different definition. See [ALTER TABLE \(Db2 SQL\)](#) for those situations in which a bind time error might be issued for the statement.

Column masks and null columns:

If the column is not nullable, the definition of its column mask will not, most likely, consider a null value for the column. After the column access control is activated for the table, if the table is the null-padded table in an outer join, the value of the column in the final result table might be a null. To ensure that the column mask can mask a null value, if the table is the null-padded table in an outer join, Db2 will add "WHEN *target-column* IS NULL THEN NULL" as the first WHEN clause to the column mask definition. This forces a null value to always be masked as a null value. For a nullable column, this removes the ability to mask a null value as something else. Example 5 shows this added WHEN clause.

Column mask values for SQL data change statements

When columns are used to derive new values for an INSERT, UPDATE, MERGE, or a SET *transition-variable* assignment statement, the original values of the column, not the masked values, are used to derive the new values. If the columns have column masks, those column masks are applied to ensure that the evaluation of the access control rules at run time masks the column to itself, not to a constant or an expression. This is to ensure that the masked values are the same as the original column values. If a column mask does not mask the column to itself, the existing row is not updated or the new row is not inserted and an error is returned at run time. The rules that are used to apply column masks in order to derive the new values follow the same rules for the final result table of a query.

Column masks that are created before column access control is activated:

The CREATE MASK statement is an independent statement that can be used to create a column access control mask before column access control is activated for a table. The only requirement is that the table and the columns exist before the mask is created. Multiple column masks can be created for a table but a column can have one mask only.

The definition of a mask is stored in the Db2 catalog. Dependency on the table for which the mask is being created and dependencies on other objects referenced in the definition are recorded. No package or dynamic cached statement is invalidated. A column mask can be created as enabled or disabled for column access control. An enabled column mask does not take effect until the ALTER TABLE statement with the ACTIVATE COLUMN ACCESS CONTROL clause is used to activate column access control for the table. SECADM authority is required to issue such an ALTER TABLE statement. A disabled column mask remains ineffective even when column access control is activated for the table. The ALTER MASK statement can be used to alter between ENABLE and DISABLE.

After column access control is activated for a table, when the table is referenced in a data manipulation statement, all enabled column masks that have been created for the table are implicitly applied by Db2 to mask the values returned for the columns referenced in the final result table of the queries or to determine the new values used in the data change statements.

Tip: To avoid multiple invalidations of packages and dynamic cached statements that reference the table, creating column masks before activating column access control for a table .

Column masks that are created after column access control is activated:

The enabled column masks become effective as soon as they are committed. All the packages and dynamic cached statements that reference the table are invalidated. Thereafter, when the table is referenced in a data manipulation statement, all enabled column masks are implicitly applied by Db2 to the statement. Any disabled column mask remains ineffective even when column access control is activated for the table.

No cascaded effect when column or row access control enforced tables are referenced in column mask definitions:

A column mask definition may reference tables and columns that are currently enforced by row or column access control. Access control from those tables and columns are ignored when the table for which the column mask is being created is referenced in a data manipulation statement.

Multiple column masks and row permissions sharing the same environment variables:

Multiple column masks and row permissions can be created for a table. They must use the same set of environment variables. The set of environment variables is determined when the first column mask or row permission is created for the table.

The catalog table SYSENVIRONMENT contains the list of environment variables. The following table shows which environment variable must be the same among the multiple column masks and row permissions.

Table 8. Environment Variables in SYSIBM.SYSENVIRONMENT

Environment variables shown as SYSENVIRONMENT columns	Description	Static create statement	Dynamic create statement	Must be the same among multiple column masks and row permissions?
ENVID	Internal identifier of the environment	Assigned by Db2	Assigned by Db2	Yes
CURRENT_SCHEMA	The qualifier used to qualify unqualified objects such as tables, views. etc.	Package owner	Value of CURRENT_SCHEMA special register	Yes
PATHSCHEMAS	The schema path used to qualify unqualified object such as user-defined functions and CAST functions for user-defined data types.	PATH bind option	Value of CURRENT_PATH special register	Yes

Table 8. Environment Variables in SYSIBM.SYSENVIRONMENT (continued)

Environment variables shown as SYSENVIRONMENT columns	Description	Static create statement	Dynamic create statement	Must be the same among multiple column masks and row permissions?
APPLICATION_ENCODING_CCSID	The CCSID of the application environment	ENCODING bind option	CURRENT APPLICATION ENCODING SCHEME special register	Yes
ORIGINAL_ENCODING_CCSID	The original CCSID of the statement text string	CCSID(n) pre-compiler option or EBCDIC CCSID on DSNTIPF installation panel	CCSID based on DEF ENCODING SCHEME on DSNTIPF installation panel	Yes
DECIMAL_POINT	The decimal point indicator	COMMA or PERIOD precompiler option or DECIMAL POINT IS on DSNTIPF installation panel	DECIMAL POINT IS on DSNTIPF installation panel	Yes
MIN_DIVIDE_SCALE	The minimum divide scale	MINIMUM DIVIDE SCALE on DSNTIP4 installation panel	MINIMUM DIVIDE SCALE on DSNTIP4 installation panel	Yes ^v
STRING_DELIMITER	The string delimiter that is used in COBOL string constants	APOST precompiler option or STRING DELIMITER on DSNTIPF installation panel	STRING DELIMITER on DSNTIPF installation panel	No
SQL_STRING_DELIMITER	The SQL string delimiter that is used in constants	APOSTSQL pre-compiler option or SQL STRING DELIMITER on DSNTIPF installation panel	SQL STRING DELIMITER on DSNTIPF installation panel	Yes
MIXED_DATA	Uses mixed DBCS data	MIXED DATA on DSNTIPF installation panel	MIXED DATA on DSNTIPF installation panel	Yes
DECIMAL_ARITHMETIC	The rules that are to be used for CURRENT PRECISION and when both operands in a decimal operation have a precision of 15 or less.	DEC(15) or DEC(31) precompiler option or DECIMAL ARITHMETIC on DSNTIP4 installation panel	DECIMAL ARITHMETIC on DSNTIP4 installation panel	Yes
DATE_FORMAT	The date format	DATE pre-compiler option or DATE FORMAT on DSNTIP4 installation panel	DATE FORMAT on DSNTIP4 installation panel	Yes

Table 8. Environment Variables in SYSIBM.SYSENVIRONMENT (continued)

Environment variables shown as SYSENVIRONMENT columns	Description	Static create statement	Dynamic create statement	Must be the same among multiple column masks and row permissions?
TIME_FORMAT	The time format	TIME pre-compiler option or TIME FORMAT on DSNTIP4 installation panel	TIME FORMAT on DSNTIP4 installation panel	Yes
FLOAT_FORMAT	The floating point format	FLOAT (S390 IEEE) pre-compiler option or default of FLOAT S390	Default of FLOAT S390	No
HOST_LANGUAGE	The host language	HOST pre-compiler option or LANGUAGE DEFAULT on DSNTIPF installation panel	LANGUAGE DEFAULT on DSNTIPF installation panel	No
CHARSET	The character set	CCSID(n) pre-compiler option or EBCDIC CCSID on DSNTIPF installation panel	EBCDIC CCSID on DSNTIPF installation panel	No
FOLD	FOLD is only applicable when HOST_LANGUAGE is C or CPP. Otherwise FOLD is blank.	HOST(C(FOL D) precompiler option or default of NO FOLD	default of NO FOLD	No
ROUNDING	The rounding mode that is used when arithmetic and casting operations are performed on DECFLOAT data.	ROUNDING bind option	CURRENT DECFLOAT ROUNDING MODE special register	Yes

Note: In a data sharing environment, if a separate DSNHDECP module is provided for each member of the group, the DSNHDECP settings for each environment variable should be the same in all members of the data sharing group, otherwise an error might be issued when multiple column masks or row permissions are created.

Ordinary SQL identifiers specified in a static CREATE MASK statement in a COBOL application:

If the CREATE MASK statement is a static statement in a COBOL application, the ordinary SQL identifiers used in the column mask definition must not follow the rules for naming COBOL words. They must follow the rules for naming SQL identifiers (Db2 SQL). For example, the COBOL word 1ST-TIME is not allowed as an ordinary SQL identifier in a column mask definition; change it to FIRST_TIME or put it in the delimiters.

Encoding scheme and CCSIDs of the data manipulation statement after column masks are applied:

The encoding scheme and CCSIDs of the data manipulation statement are not affected by the column masks that are implicitly applied by Db2 for the column access control. For a target table or a referenced table that is not an EBCDIC table with Unicode columns, the column mask definition is evaluated using its table's encoding scheme and CCSIDs. For a target table or a referenced table that

is an EBCDIC table with Unicode columns, the column mask definition is evaluated using the rules for multiple encoding schemes.

Consideration for Db2 limits:

If the data manipulation statement already approaches some Db2 limits in the statement, it should be noted that the more enabled column masks and enabled row permissions are created, the more likely they would impact some limits. For example, they may cause the statement to exceed the maximum total length (32600 bytes) of columns of a query operation requiring sort and evaluating aggregate functions (MULTIPLE DISTINCT and GROUP BY). This is because the enabled column mask and enabled row permission definitions are implicitly merged into the statement when the table is referenced in a data manipulation statement. See "Limits in Db2 for z/OS" in SQL Reference for the limits of a statement.

Restrictions involving pending definition changes:

CREATE MASK is not allowed if the mask is defined on a table or references a table that has pending definition changes.

Examples

In the following examples, the data type of column SSN is VARCHAR(11).

Example 1

After column access control is activated for table EMPLOYEE, Paul from the payroll department can see the social security number of the employee whose employee number is 123456. Mary who is a manager can see the last four characters only of the social security number. Peter who is neither cannot see the social security number.

```
CREATE MASK SSN_MASK ON EMPLOYEE
FOR COLUMN SSN RETURN
CASE
    WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER, 'PAYROLL') = 1)
        THEN SSN
    WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER, 'MGR') = 1)
        THEN 'XXX-XX-' || SUBSTR(SSN,8,4)
    ELSE NULL
END
ENABLE;

COMMIT;

ALTER TABLE EMPLOYEE
ACTIVATE COLUMN ACCESS CONTROL;

COMMIT;

SELECT SSN FROM EMPLOYEE
WHERE EMPNO = 123456;
```

Example 2

In the SELECT statement, column SSN is embedded in an expression that is the same as the expression used in the column mask SSN_MASK. After column access control is activated for table EMPLOYEE, the column mask SSN_MASK is applied to column SSN in the SELECT statement. For this particular expression, the SELECT statement produces the same result as before column access control is activated for all users. The user can replace the expression in the SELECT statement with column SSN to avoid the same expression gets evaluated twice.

```
CREATE MASK SSN_MASK ON EMPLOYEE
FOR COLUMN SSN RETURN
CASE
    WHEN (1 = 1)
        THEN 'XXX-XX-' || SUBSTR(SSN,8,4)
    ELSE NULL
END
ENABLE;

COMMIT;

ALTER TABLE EMPLOYEE
```

```

    ACTIVATE COLUMN ACCESS CONTROL;

COMMIT;

SELECT 'XXX-XX-' || SUBSTR(SSN,8,4) FROM EMPLOYEE
WHERE EMPNO = 123456;

```

Example 3

A state government conducted a survey for the library usage of the households in each city. Fifty households in each city were sampled in the survey. Each household was given an option, opt-in or opt-out, whether to show their usage in any reports generated from the result of the survey.

A SELECT statement is used to generate a report to show the average hours used by households in each city. Column mask CITY_MASK is created to mask the city name based on the opt-in or opt-out information chosen by the sampled households. However, after column access control is activated for table LIBRARY_USAGE, the SELECT statement receives a bind time error. This is because column mask CITY_MASK references another column LIBRARY_OPT and LIBRARY_OPT does not identify a grouping column.

```

CREATE MASK CITY_MASK ON LIBRARY_USAGE
FOR COLUMN CITY RETURN
CASE
    WHEN (LIBRARY_OPT = 'OPT-IN')
    THEN CITY
    ELSE ' '
END
ENABLE;

COMMIT;

ALTER TABLE LIBRARY_USAGE
    ACTIVATE COLUMN ACCESS CONTROL;

COMMIT;

SELECT CITY, AVG(LIBRARY_TIME) FROM LIBRARY_USAGE
GROUP BY CITY;

```

Example 4

Employee with EMPNO 123456 earns bonus \$8000 and salary \$80000 in May. When the manager retrieves his salary, the manager receives his salary, not the null value. This is because of no cascaded effect when column mask SALARY_MASK references column BONUS for which column mask BONUS_MASK is defined.

```

CREATE MASK SALARY_MASK ON EMPLOYEE
FOR COLUMN SALARY RETURN
CASE
    WHEN (BONUS < 10000)
    THEN SALARY
    ELSE NULL
END
ENABLE;

COMMIT;

CREATE MASK BONUS_MASK ON EMPLOYEE
FOR COLUMN BONUS RETURN
CASE
    WHEN (BONUS > 5000)
    THEN NULL
    ELSE BONUS
END
ENABLE;

COMMIT;

ALTER TABLE EMPLOYEE
    ACTIVATE COLUMN ACCESS CONTROL;

COMMIT;

SELECT SALARY FROM EMPLOYEE
WHERE EMPNO = 123456;

```

Example 5

This example shows Db2 adds "WHEN target-column IS NULL THEN NULL" as the first WHEN clause to the column mask definition then merges the column mask definition into the statement.

```
CREATE EMPLOYEE (EMPID INT,
                 DEPTID CHAR(8),
                 SALARY DEC(9,2) NOT NULL,
                 BONUS DEC(9,2));

CREATE MASK SALARY_MASK ON EMPLOYEE
  FOR COLUMN SALARY RETURN
  CASE
    WHEN SALARY < 10000
    THEN CAST(SALARY*2 AS DEC(9,2))
    ELSE COALESCE(CAST(SALARY/2 AS DEC(9,2)), BONUS)
  END
  ENABLE;

COMMIT;

CREATE MASK BONUS_MASK ON EMPLOYEE
  FOR COLUMN BONUS RETURN
  CASE
    WHEN BONUS > 1000
    THEN BONUS
    ELSE NULL
  END
  ENABLE;

COMMIT;

ALTER TABLE EMPLOYEE
  ACTIVATE COLUMN ACCESS CONTROL;

COMMIT;

SELECT SALARY FROM DEPT
  LEFT JOIN EMPLOYEE ON DEPTNO = DEPTID;

/* When SALARY_MASK is merged into the above statement,
 * 'WHEN SALARY IS NULL THEN NULL' is added as the
 * first WHEN clause, as follows:
 */

SELECT CASE WHEN SALARY IS NULL THEN NULL
           WHEN SALARY < 10000 THEN CAST(SALARY*2 AS DEC(9,2))
           ELSE COALESCE(CAST(SALARY/2 AS DEC(9,2)), BONUS)
  END SALARY
  FROM DEPT
  LEFT JOIN EMPLOYEE ON DEPTNO = DEPTID;
```

Related concepts

[Column mask \(Managing Security\)](#)

[Naming conventions \(Db2 SQL\)](#)

Related tasks

[Creating column masks \(Managing Security\)](#)

CREATE PERMISSION

The CREATE PERMISSION statement creates a row permission for row access control at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

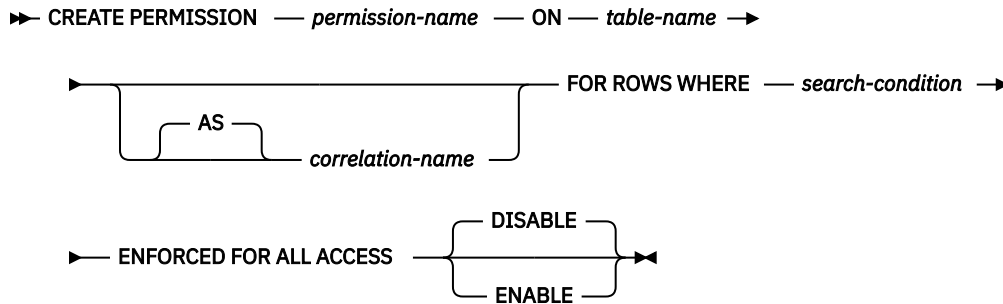
The privilege set that is defined below must include the following authority:

- SECADM authority

SECADM authority can create a row permission in any schema. Additional privileges are not needed to reference other objects in the permission definition. For example, the SELECT privilege is not needed to retrieve from a table, and the EXECUTE privilege is not needed to invoke a user-defined function.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the package. If the statement is dynamically prepared, the privilege set is the set of privileges that are held by the SQL authorization ID of the process. However, if it is running in a trusted context defined with the ROLE AS OBJECT OWNER AND QUALIFIER clause, the privilege set is the set of privileges that are held by the role in effect.

Syntax



Description

permission-name

Names the row permission for row access control. The name, including the implicit or explicit qualifier, must not identify a row permission or a column mask that already exists at the current server

ON *table-name*

Identifies the table on which the row permission is created. The name must identify a table that exists at the current server. It must not identify any of the following objects:

- An auxiliary table
- A created or declared temporary table
- A view
- A catalog table
- An alias
- A synonym
- A materialized query table or table that is directly or indirectly referenced in the definition of a materialized query table
- A table that was implicitly created for an XML column
- A table that contains a period
- A history table
- An accelerator-only table
- An archive-enabled table
- An archive table
- A table that has a security label column.

correlation-name

Can be used within *search-condition* to designate the table. For the explanation of *correlation-name*, see [Correlation names \(Db2 SQL\)](#).

FOR ROWS WHERE

Indicates that a row permission is created. A row permission specifies a search condition under which rows of the table can be accessed.

search-condition

Specifies a condition that can be true, false, or unknown for a row of the table. *search-condition* follows the same rules that are used by the search condition in a WHERE clause of a subselect. In addition, the search condition must not reference any of the following objects:

- A remote object
- The table for which the row permission is being defined
- A table that has a security label column
- A created global temporary table or a declared global temporary table
- An auxiliary table
- A table that was implicitly created for an XML column
- A collection-derived table (UNNEST)
- A table function
- A host variable, SQL variable, SQL parameter, or trigger transition variable
- A user-defined function that is defined as not secure
- A function that is not deterministic or that has an external action or is defined with the MODIFIES SQL DATA option
- A parameter marker
- A column that is defined with a FIELDPROC
- A LOB column or a distinct type column that is based on a LOB
- An XML column
- An XMLEXISTS predicate
- An OLAP specification
- A ROW CHANGE expression
- A sequence reference
- A select list notation *** or *name . ** in the SELECT clause
- A table reference that contains a period specification
- A view that includes any of the preceding restrictions in its definition
- An AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY function.

The encoding scheme of the table is used to evaluate the *search-condition*. Tables and language elements that require multiple encoding scheme evaluation, other than EBCDIC tables with Unicode columns, must not be referenced in the *search-condition*. See [Determining the encoding scheme and CCSID of a string \(Introduction to Db2 for z/OS\)](#) for those language elements.

If the *search-condition* references tables for which row or column access control is activated, access control from those tables is not cascaded.

ENFORCED FOR ALL ACCESS

Specifies that the row permission applies to all references of the table. If row access control is activated for the table, when the table is referenced in a data manipulation statement, Db2 implicitly applies the row permission to control the access of the table. If the reference of the table is for a fetch operation such as SELECT, the application of the row permission determines what set of rows can be retrieved by the user who requested the fetch operation. If the reference of the table is for a data change operation such as INSERT, the application of the row permission determines whether all rows to be changed are insertable or updatable by the user who requested the data change operation.

DISABLE or ENABLE

Specifies that the row permission is to be enabled or disabled for row access control.

DISABLE

Specifies that the row permission is to be disabled for row access control. The row permission will remain ineffective regardless the row access control is activated for the table or not.

DISABLE is the default.

ENABLE

Specifies that the row permission is to be enabled for row access control. If row access control is not currently activated for the table, the row permission will become effective when row access control is activated for the table. If row access control is currently activated for the table, the row permission becomes effective immediately and all packages and dynamic cached statements that reference the table are invalidated. For more information, see [Changes that invalidate packages \(Db2 Application programming and SQL\)](#).

Notes

How row permission are applied and how they affect certain statements:

See the ALTER TABLE statement with the ACTIVATE ROW ACCESS CONTROL clause for information on how to activate row access control and how row permissions are applied. See the description of subselect for information on how the application of row permissions affects the fetch operation. See the data change statements for information on how the application of row permissions affects the data change operation.

Row permissions that are created before row access control is activated for a table:

The CREATE PERMISSION statement is an independent statement that can be used to create a row permission before row access control is activated for a table. The only requirement is that the table and the columns exist before the permission is created. Multiple row permissions can be created for a table.

The definition of the row permission is stored in the Db2 catalog. Dependency on the table for which the permission is being created and dependencies on other objects referenced in the definition are recorded. No package or dynamic cached statement is invalidated. A row permission can be created as enabled or disabled for row access control. An enabled row permission does not take effect until the ALTER TABLE statement with the ACTIVATE ROW ACCESS CONTROL clause is used to activate row access control for the table. A disabled row permission remains ineffective even when row access control is activated for the table. The ALTER PERMISSION statement can be used to alter between ENABLE and DISABLE.

After row access control is activated for a table, when the table is referenced in a data manipulation statement, all enabled row permissions that are defined for the table are implicitly applied by Db2 to control access to the table.

Tip: Create row permissions before activating row access control for a table to avoid multiple invalidations of packages and dynamic cached statements that reference the table.

Row permissions that are created after row access control is activated for a table:

An enabled row permission becomes effective as soon as it is committed. All the packages and dynamic cached statements that reference the table are invalidated. Thereafter, when the table is referenced in a data manipulation statement, all enabled row permissions are implicitly applied to the statement. Any disabled row permission remains ineffective even when row access control is activated for the table.

No cascaded effect when row or column access control enforced tables are referenced in row permission definitions:

A row permission definition may reference tables and columns that are currently enforced by row or column access control. Access control from those tables are ignored when the table for which the row permission is being created is referenced in a data manipulation statement.

Multiple column masks and row permissions sharing the same environment variables:

Multiple column masks and row permissions can be created for a table. They must use the same set of environment variables. The set of environment variables is determined when the first column mask or row permission is created for the table.

The catalog table SYSENVIRONMENT contains the list of environment variables. The following table shows which environment variable must be the same among the multiple column masks and row permissions.

Table 9. Environment Variables in SYSIBM.SYSENVIRONMENT

Environment variables shown as SYSENVIRONMENT columns	Description	Static create statement	Dynamic create statement	Must be the same among multiple column masks and row permissions?
ENVID	Internal identifier of the environment	Assigned by Db2	Assigned by Db2	Yes
CURRENT_SCHEMA	The qualifier used to qualify unqualified objects such as tables, views. etc.	Package owner	Value of CURRENT_SCHEMA special register	Yes
PATHSCHEMAS	The schema path used to qualify unqualified object such as user-defined functions and CAST functions for user-defined data types.	PATH bind option	Value of CURRENT_PATH special register	Yes
APPLICATION_ENCODING_CCSID	The CCSID of the application environment	ENCODING bind option	CURRENT APPLICATION ENCODING SCHEME special register	Yes
ORIGINAL_ENCODING_CCSID	The original CCSID of the statement text string	CCSID(n) pre-compiler option or EBCDIC CCSID on DSNTIPF installation panel	CCSID based on DEF ENCODING SCHEME on DSNTIPF installation panel	Yes
DECIMAL_POINT	The decimal point indicator	COMMA or PERIOD precompiler option or DECIMAL POINT IS on DSNTIPF installation panel	DECIMAL POINT IS on DSNTIPF installation panel	Yes
MIN_DIVIDE_SCALE	The minimum divide scale	MINIMUM DIVIDE SCALE on DSNTIP4 installation panel	MINIMUM DIVIDE SCALE on DSNTIP4 installation panel	Yes
STRING_DELIMITER	The string delimiter that is used in COBOL string constants	APOST precompiler option or STRING DELIMITER on DSNTIPF installation panel	STRING DELIMITER on DSNTIPF installation panel	No
SQL_STRING_DELIMITER	The SQL string delimiter that is used in constants	APOSTSQL pre-compiler option or SQL STRING DELIMITER on DSNTIPF installation panel	SQL STRING DELIMITER on DSNTIPF installation panel	Yes

Table 9. Environment Variables in SYSIBM.SYSENVIRONMENT (continued)

Environment variables shown as SYSENVIRONMENT columns	Description	Static create statement	Dynamic create statement	Must be the same among multiple column masks and row permissions?
MIXED_DATA	Uses mixed DBCS data	MIXED DATA on DSNTIPF installation panel	MIXED DATA on DSNTIPF installation panel	Yes
DECIMAL_ARITHMETIC	The rules that are to be used for CURRENT PRECISION and when both operands in a decimal operation have a precision of 15 or less.	DEC(15) or DEC(31) precompiler option or DECIMAL ARITHMETIC on DSNTIP4 installation panel	DECIMAL ARITHMETIC on DSNTIP4 installation panel	Yes
DATE_FORMAT	The date format	DATE pre-compiler option or DATE FORMAT on DSNTIP4 installation panel	DATE FORMAT on DSNTIP4 installation panel	Yes
TIME_FORMAT	The time format	TIME pre-compiler option or TIME FORMAT on DSNTIP4 installation panel	TIME FORMAT on DSNTIP4 installation panel	Yes
FLOAT_FORMAT	The floating point format	FLOAT (S390 IEEE) pre-compiler option or default of FLOAT S390	Default of FLOAT S390	No
HOST_LANGUAGE	The host language	HOST pre-compiler option or LANGUAGE DEFAULT on DSNTIPF installation panel	LANGUAGE DEFAULT on DSNTIPF installation panel	No
CHARSET	The character set	CCSID(n) pre-compiler option or EBCDIC CCSID on DSNTIPF installation panel	EBCDIC CCSID on DSNTIPF installation panel	No
FOLD	FOLD is only applicable when HOST_LANGUAGE is C or CPP. Otherwise FOLD is blank.	HOST(C(FOLD)) precompiler option or default of NO FOLD	default of NO FOLD	No

Table 9. Environment Variables in SYSIBM.SYSENVIRONMENT (continued)

Environment variables shown as SYSENVIRONMENT columns	Description	Static create statement	Dynamic create statement	Must be the same among multiple column masks and row permissions?
ROUNDING	The rounding mode that is used when arithmetic and casting operations are performed on DECFLOAT data.	ROUNDING bind option	CURRENT DECFLOAT ROUNDING MODE special register	Yes

Note: In a data sharing environment, if a separate DSNHDECP module is provided for each member of the group, the DSNHDECP settings for each environment variable should be the same in all members of the data sharing group, otherwise an error might be issued when multiple column masks or row permissions are created.

Ordinary SQL identifiers specified in a static CREATE PERMISSION statement in a COBOL application:

If the CREATE PERMISSION statement is a static statement in a COBOL application, the ordinary SQL identifiers used in the row permission definition must not follow the rules for naming COBOL words (DSNH20474, reason code 14). They must follow the rules for naming SQL identifiers as described in the topic “SQL identifiers” in Db2 SQL Reference. For example, the COBOL word 1ST-TIME is not allowed as an ordinary SQL identifier in a row permission definition; change it to FIRST_TIME or put it in the delimiters.

Encoding scheme and CCSIDs of the data manipulation statement after row permissions are applied:

The encoding scheme and CCSIDs of the data manipulation statement are not affected by the row permissions that are implicitly applied by Db2 for the row access control. For a target table or a referenced table that is not an EBCDIC table with Unicode columns, the row permission definition is evaluated using its table's encoding scheme and CCSIDs. For a target table or a referenced table that is an EBCDIC table with Unicode columns, the row permission definition is evaluated using the rules for multiple encoding schemes.

Consideration for Db2 limits:

If the data manipulation statement already approaches some Db2 limits in the statement, it should be noted that the more enabled row permissions and enabled column masks are created, the more likely they would impact some limits. For example, they may cause the statement to exceed the maximum total length (32600 bytes) of columns of a query operation requiring sort and evaluating aggregate functions (MULTIPLE DISTINCT and GROUP BY). This is because the enabled column mask and enabled row permission definitions are implicitly merged into the statement when the table is referenced in a data manipulation statement. See "Limits in Db2 for z/OS" in SQL Reference for the limits of a statement.

Restrictions involving pending definition changes:

CREATE PERMISSION is not allowed if the permission is defined on a table or references a table that has pending definition changes.

Examples

Example 1

Secure user-defined function ACCOUNTING_UDF in row permission SALARY_ROW_ACCESS processes the sensitive data in column SALARY. After row access control is activated for table EMPLOYEE, Accountant Paul retrieves the salary of employee with EMPNO 123456 who is making \$100,000 a year. Paul may or may not see the row depending on the output value from user-defined function ACCOUNTING_UDF.

```

CREATE PERMISSION SALARY_ROW_ACCESS ON EMPLOYEE
  FOR ROWS WHERE VERIFY_GROUP_FOR_USER(SESSION_USER, 'MGR', 'ACCOUNTING') = 1
              AND
              ACCOUNTING_UDF(SALARY) < 120000
  ENFORCED FOR ALL ACCESS
  ENABLE;

COMMIT;

ALTER TABLE EMPLOYEE
  ACTIVATE ROW ACCESS CONTROL;

COMMIT;

SELECT SALARY FROM EMPLOYEE
  WHERE EMPNO = 123456;

```

Example 2

The tellers in a bank can only access customers from their branch. All tellers have secondary authorization ID TELLER. The customer service representatives are allowed to access all customers of the bank. All customer service representatives have secondary authorization ID CSR. A row permission is created for each group of personnel in the bank accordingly to the access rule defined by SECADM authority. After row access control is activated for table CUSTOMER, in the SELECT statement the search conditions of both row permissions are merged into the statement and they are combined with the logic OR operator to control the set of rows accessible by each group.

```

CREATE PERMISSION TELLER_ROW_ACCESS ON CUSTOMER
  FOR ROWS WHERE VERIFY_GROUP_FOR_USER(SESSION_USER, 'TELLER') = 1
              AND
              BRANCH = (SELECT HOME_BRANCH FROM INTERNAL_INFO
                        WHERE EMP_ID = SESSION_USER)
  ENFORCED FOR ALL ACCESS
  ENABLE;

COMMIT;

CREATE PERMISSION CSR_ROW_ACCESS ON CUSTOMER
  FOR ROWS WHERE VERIFY_GROUP_FOR_USER(SESSION_USER, 'CSR') = 1
  ENFORCED FOR ALL ACCESS
  ENABLE;

COMMIT;

ALTER TABLE CUSTOMER
  ACTIVATE ROW ACCESS CONTROL;

COMMIT;

SELECT * FROM CUSTOMER;

```

Related concepts

[Row permission \(Managing Security\)](#)

[Naming conventions \(Db2 SQL\)](#)

Related tasks

[Creating row permissions \(Managing Security\)](#)

CREATE TABLE

The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns. The definition can include other attributes of the table, such as its primary key and its table space.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

The privilege set that is defined below must include at least one of the following:

- The CREATETAB privilege for the database explicitly specified by the IN clause.

If the IN clause is not specified, the CREATETAB privilege on database DSNDDB04 is required.

- DBADM, DBCTRL, or DBMAINT authority for the database explicitly specified by the IN clause. If the IN clause is not specified, DBADM, DBCTRL, or DBMAINT authority for database DSNDDB04 is required.
- SYSADM or SYSCTRL authority
- System DBADM
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

If the table space is created implicitly, the privilege set that is defined below must include at least one of the following:

- The CREATETS privilege for the database explicitly specified by the IN clause.

If the IN clause is not specified, the CREATETS privilege on database DSNDDB04 is required.

- DBADM, DBCTRL, or DBMAINT authority for the database explicitly specified by the IN clause. If the IN clause is not specified, DBADM, DBCTRL, or DBMAINT authority for database DSNDDB04 is required.
- SYSADM or SYSCTRL authority
- Installation SYSOPR authority (when the current SQLID of the process is set to SYSINSTL)

The privilege set must also have the USE privilege for the following objects:

- For the table space if one is specified in the IN clause
- For the default buffer pool and default storage group of the database if a database is specified in the IN clause

If you specify a table space name, you must also have the SYSADM or SYSCTRL authority or the DBADM authority for the database.

For tables that are created in an implicit database, the database authority must be held on DSNDDB04.

Additional privileges might be required in the following conditions:

- The clause IN, LIKE or FOREIGN KEY is specified.
- The data type of a column is a distinct type.
- The table space is implicitly created.
- A *fullselect* is specified.
- A column is defined as a security label column.

Privilege set: See the description of the appropriate clauses for details about these privileges.

If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the package.

If the application is bound in a trusted context with the ROLE AS OBJECT OWNER clause specified:

- A role is the owner of the table that is being created
- The privilege set is the set of privileges that are held by that role
- The schema qualifier (implicit or explicit) must be the same as the role, unless the role has the CREATEIN privilege on the schema, or SYSADM, SYSCTRL, or system DBADM authority

Otherwise, an authorization ID is the owner of the package, and the following rules apply:

- If the privilege set lacks the CREATEIN privilege on the schema, SYSADM authority, SYSCTRL authority, and System DBADM authority, the schema qualifier (implicit or explicit) must be the same as the authorization ID of the owner of the package.

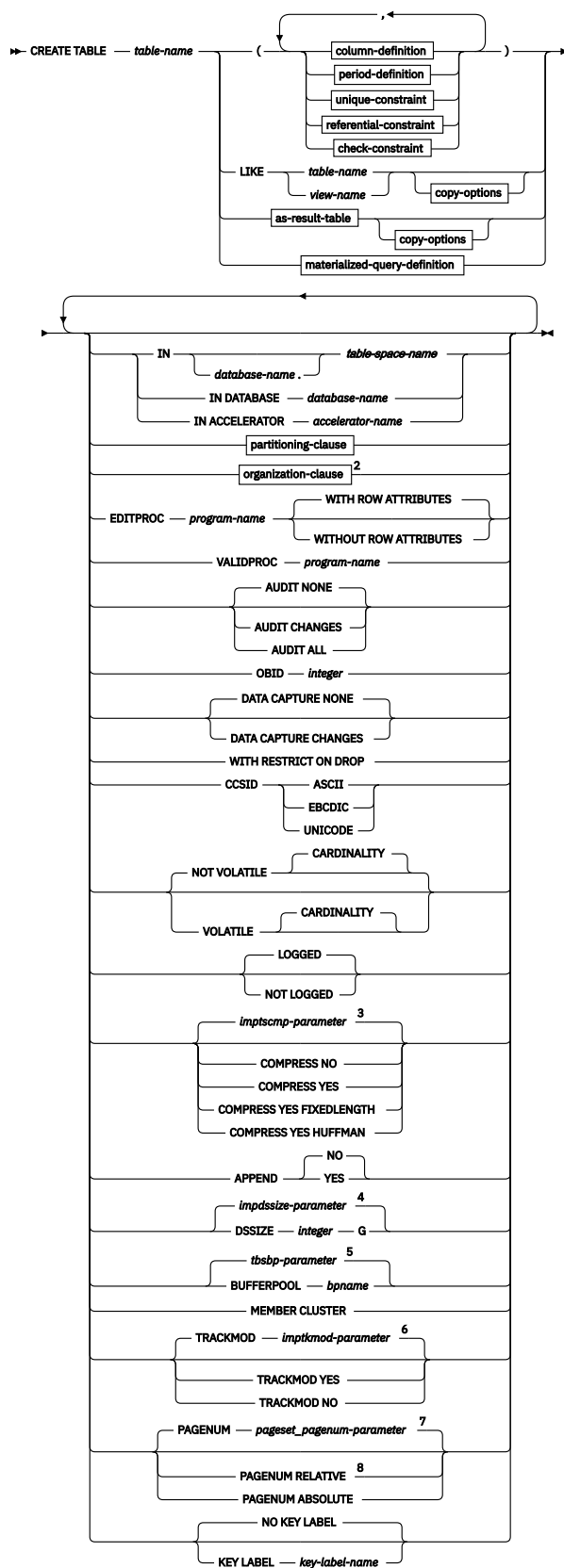
- If the privilege set lacks SYSADM authority, SYSCTRL authority, and System DBADM authority, and the table is explicitly qualified, the authorization ID that is the same as the schema name must have all the necessary privileges to create the table, and that authorization ID is the owner of the table. Otherwise, the authorization ID of the owner of the package must have all the necessary privileges to create the table, and that authorization ID is the owner of the table.
- If the privilege set includes SYSADM authority, SYSCTRL authority, or system DBADM authority, the schema qualifier (implicit or explicit) can be any schema name. However, if the table is explicitly qualified, the authorization ID that is the same as the schema name is the owner of the table. Otherwise, the authorization ID of the owner of the package is the owner of the table.
- If the privilege set includes DBADM authority and DBCTRL authority for the database, the schema qualifier (implicit or explicit) can be any schema name. However, if the table is explicitly qualified, the authorization ID that is the same as the schema name is the owner of the table. Otherwise, the authorization ID of the owner of the package is the owner of the table.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process unless the process is within a trusted context and the `ROLE AS OBJECT OWNER` clause is in effect. When `ROLE AS OBJECT OWNER` is in effect, the privileges set is the privileges that are held by the role that is associated with the primary authorization ID of the process, and the owner of the table is that role. The schema qualifier (implicit or explicit) must be the same as that role, unless the role has `CREATEIN` privilege on the schema, or SYSADM authority, SYSCTRL authority, or System DBADM authority.

For the case where the SQL authorization ID of the process holds the privileges, the following rules apply:

- If the privilege set lacks `CREATEIN` privilege on the schema, SYSADM authority, SYSCTRL authority, and System DBADM authority, the schema qualifier must be the same as one of the authorization IDs of the process.
- If the privilege set lacks SYSADM authority, SYSCTRL authority, and System DBADM authority, and the table is explicitly qualified, then the authorization ID that is the same as the schema name must have all the necessary privileges to create the table, and that authorization ID is the owner of the table. Otherwise, the SQL authorization ID of the process must include all privileges that are needed to create the table, and that authorization ID is the owner of the table.
- If the privilege set includes SYSADM authority, SYSCTRL authority, or System DBADM authority, the schema qualifier can be any schema name. However, if the table is explicitly qualified, then the authorization ID that is the same as the schema name is the owner of the table. Otherwise, the SQL authorization ID of the process is the owner of the table.

Syntax



Notes:

¹ The same clause must not be specified more than once.

² Hash-organized tables are deprecated. Beginning in Db2 12, packages bound with APPLCOMPAT(V12R1M504) or higher cannot create hash-organized tables or alter existing tables to use hash-organization. Existing hash organized tables remain supported, but they are likely to be unsupported in the future.

³ The IMPTSCMP subsystem parameter specifies the default value. See [USE DATA COMPRESSION field \(IMPTSCMP subsystem parameter\)](#) (Db2 Installation and Migration).

⁴ The IMPDSSIZE subsystem parameter specifies the default value. See [IMPDSSIZE in macro DSN6SYSP](#) (Db2 Installation and Migration).

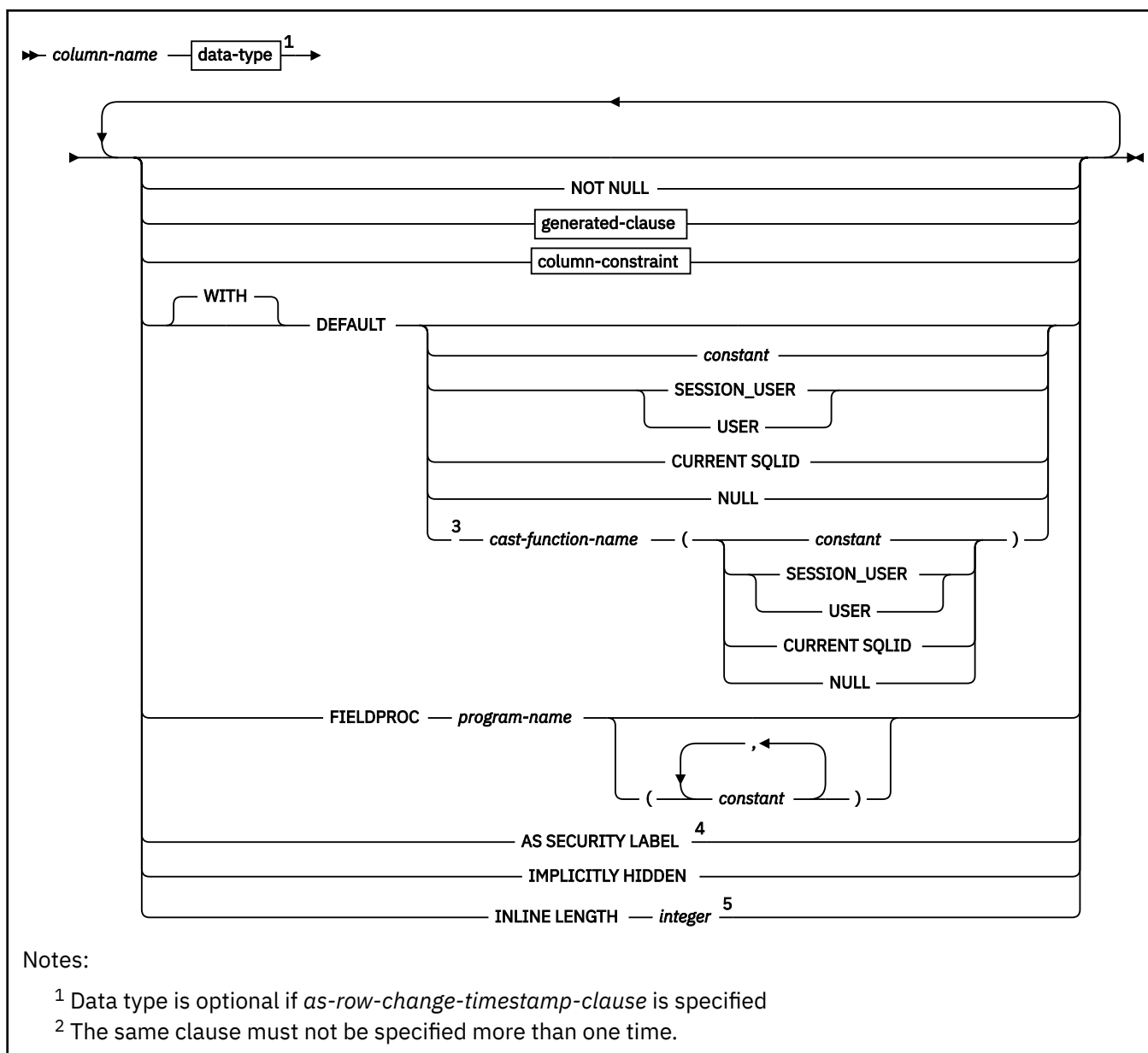
⁵ The TBSBPOOL, TBSBP8K, TBSBP16K, or TBSBP32K subsystem parameter determines the default value. See [DSNTIP2: Buffer pool sizes panel 2](#) (Db2 Installation and Migration).

⁶ The IMPTKMOD subsystem parameter specifies the default value. See [IMPTKMOD in macro DSN6SYSP](#) (Db2 Installation and Migration).

⁷ See [PAGE SET PAGE NUMBERING field \(PAGESET_PAGENUM subsystem parameter\)](#) (Db2 Installation and Migration).

⁸ PAGENUM RELATIVE is allowed only if a partitioning clause is specified.

column-definition:

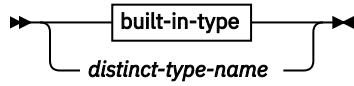


³ The *cast-function-name* form of the DEFAULT value can only be used with a column that is defined as a distinct type.

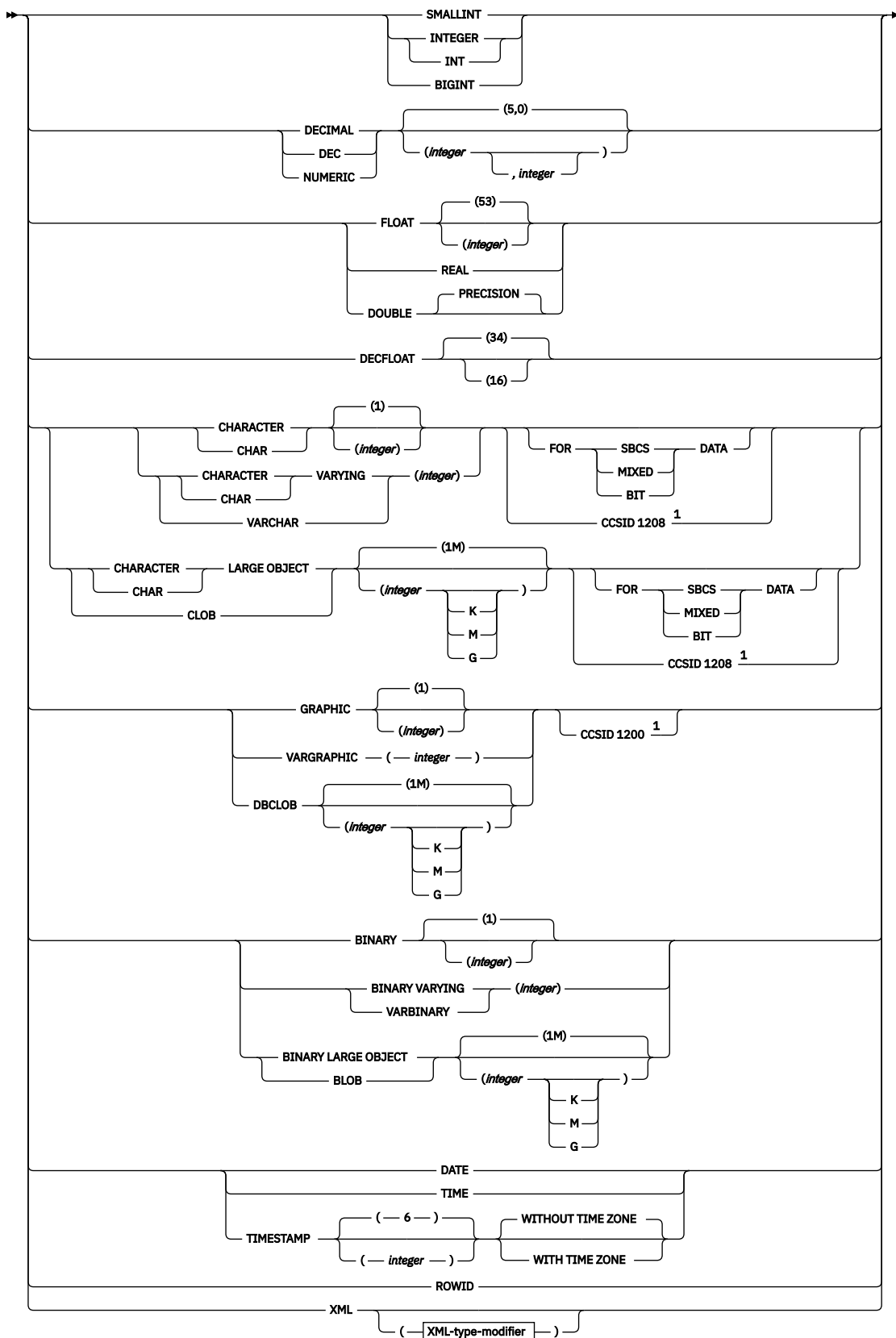
⁴ AS SECURITY LABEL can be specified only for a CHAR(8) data type and requires that the NOT NULL and WITH DEFAULT clauses be specified.

⁵ INLINE LENGTH only applies to a column with a LOB data type or a distinct type that is based on a LOB data type.

data-type:



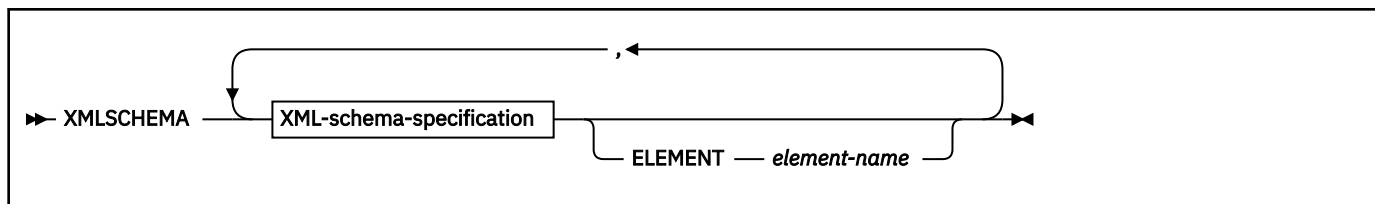
built-in-type:



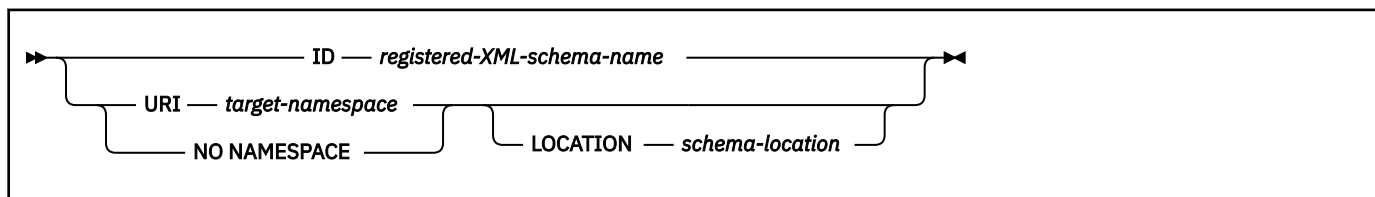
Notes:

¹ The CCSID clause must only be specified for a character string or graphic string column in an EBCDIC table. The CCSID clause must not be specified with *non-deterministic-expression*.

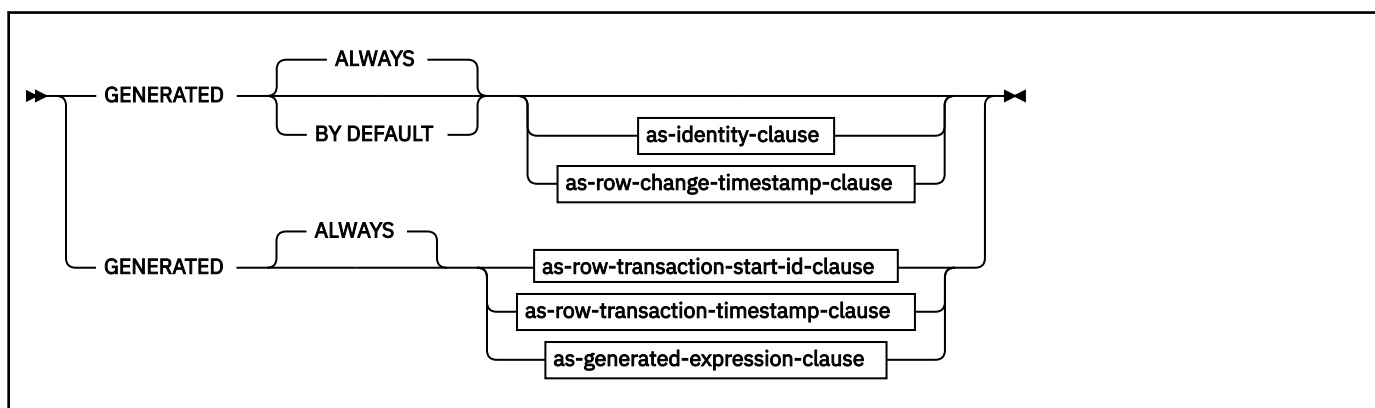
XML-type-modifier:



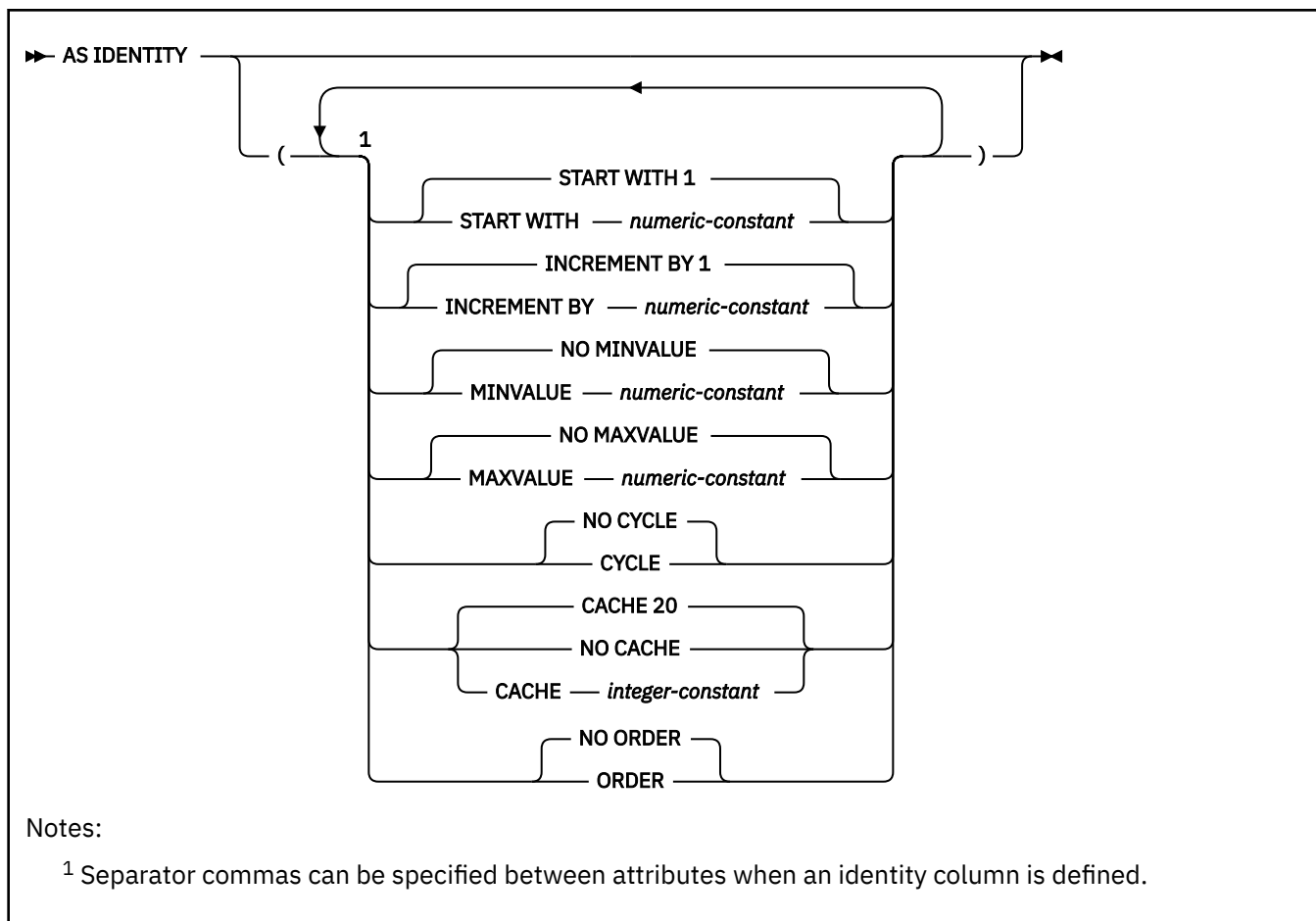
XML-schema-specification:



generated-clause:



as-identity-clause:



as-row-change-timestamp-clause:

➤ FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP ➤

as-row-transaction-start-id-clause:

➤ AS TRANSACTION START ID ➤

as-row-transaction-timestamp-clause:

➤ AS ROW ➤

BEGIN

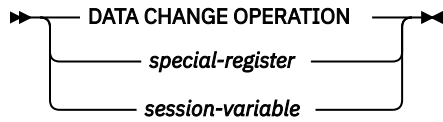
START

END

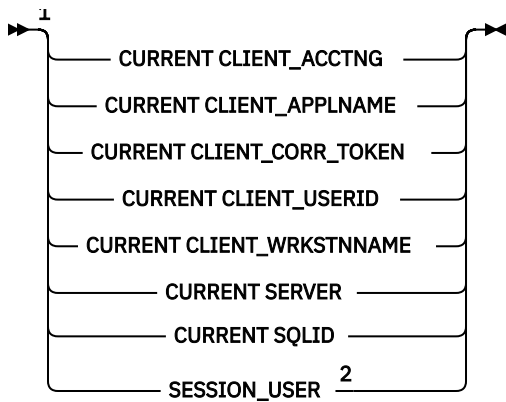
as-generated-expression-clause:

➤ AS — (— non-deterministic-expression —) ➤

non-deterministic-expression:



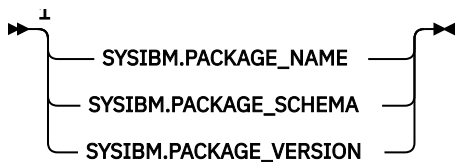
special-register:



Notes:

- ¹ This definition of special register is specific to this context, as part of *non-deterministic-expression*.
- ² USER can be specified as a synonym for SESSION_USER.

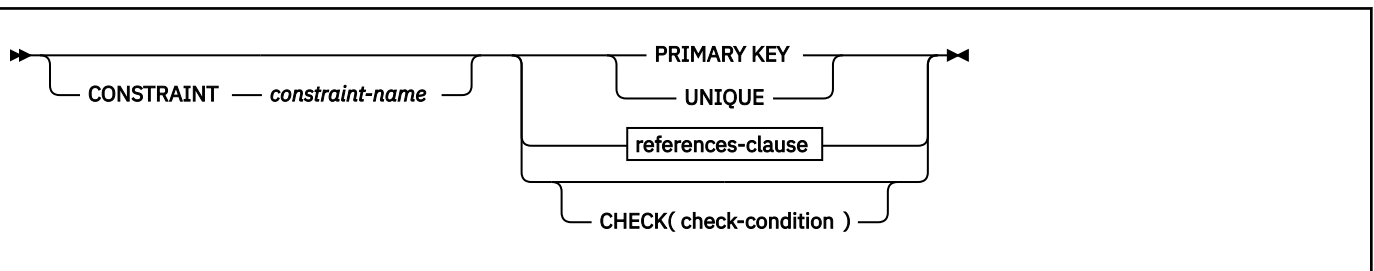
session-variable:



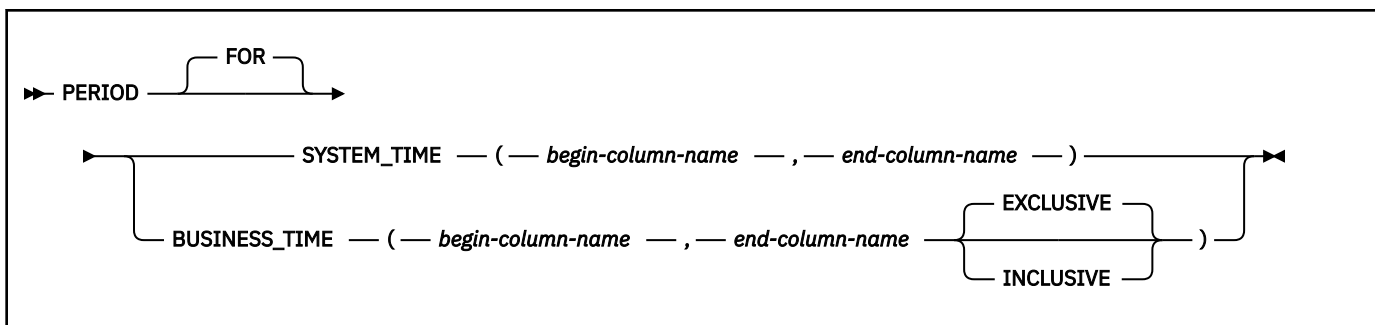
Notes:

- ¹ This definition of session variable is specific to this context, as part of *non-deterministic-expression*.

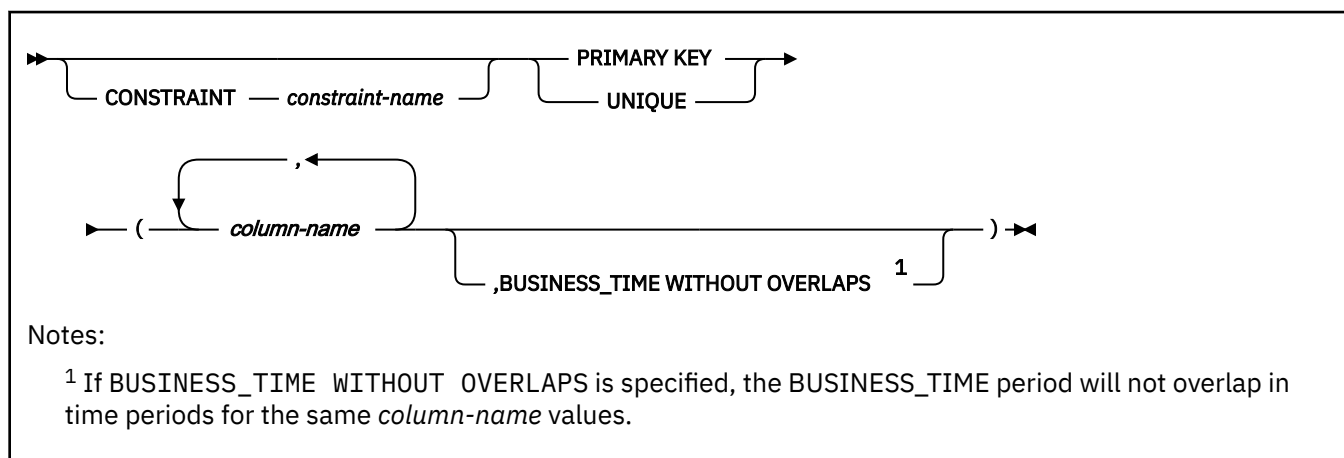
column-constraint:



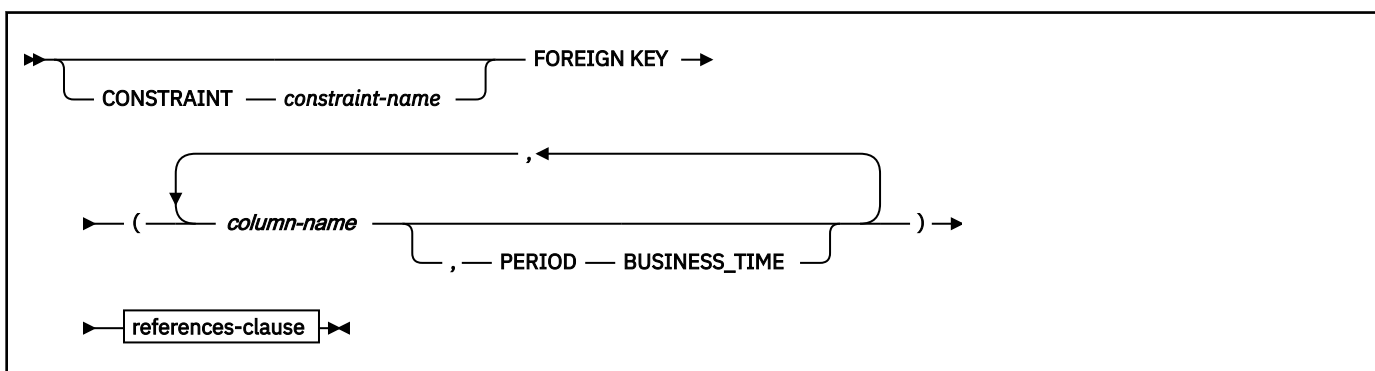
period-definition:



unique-constraint:

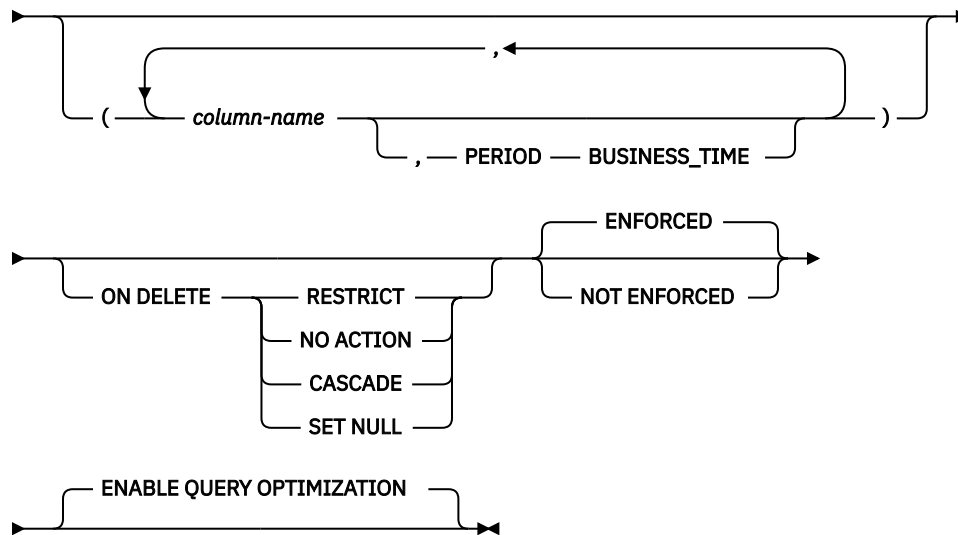


referential-constraint:



references-clause:

➤➤ REFERENCES — *parent-table-name* ➔



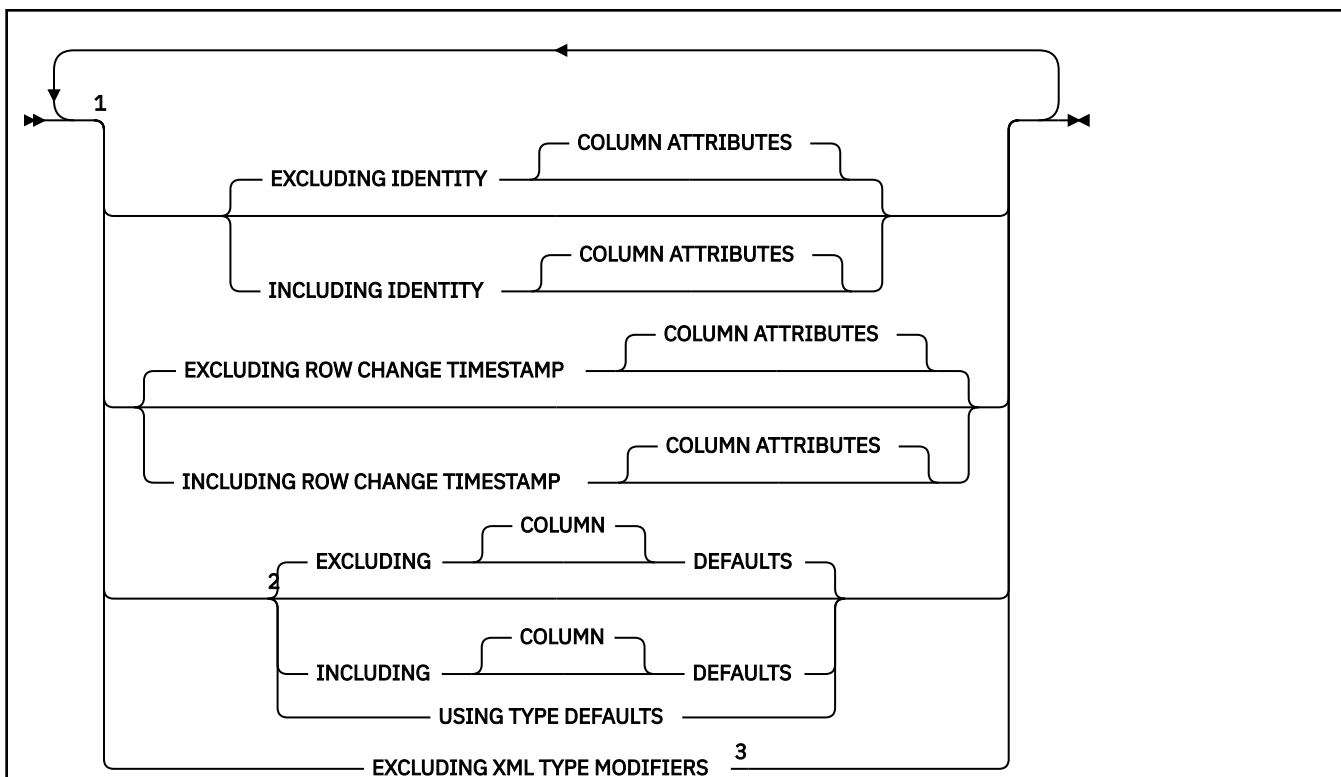
check-constraint:

➤➤ CONSTRAINT — *constraint-name* — CHECK — (*check-condition*) ➤➤

as-result-table:

➤➤ AS — (— *fullselect* —) — WITH NO DATA ➤➤

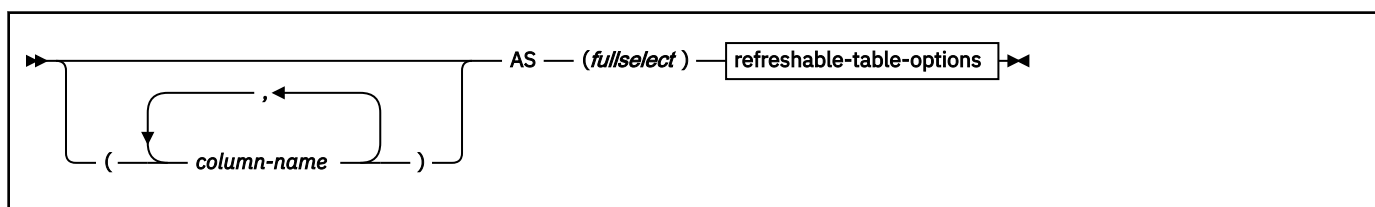
copy-options:



Notes:

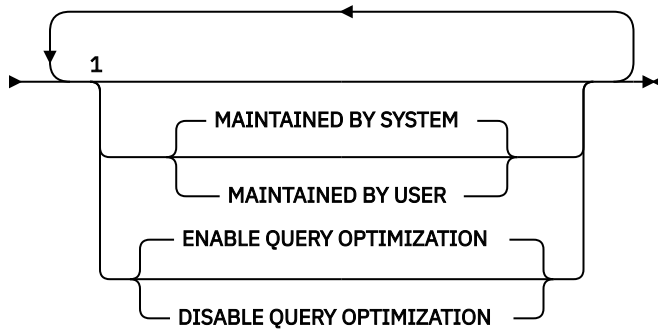
- ¹ These clauses can be specified in any order and must not be specified more than one time.
- ² EXCLUDING COLUMN DEFAULTS, INCLUDING COLUMN DEFAULTS, and USING TYPE DEFAULTS must not be specified with the LIKE clause.
- ³ EXCLUDING XML TYPE MODIFIERS must be specified with the LIKE clause if the identified table has an XML type modifier and none of the XML columns of the new table has an XML type modifier. EXCLUDING XML TYPE MODIFIERS is not supported when a view is identified in a LIKE clause and the view contains XML columns.

materialized-query-definition



refreshable-table-options:

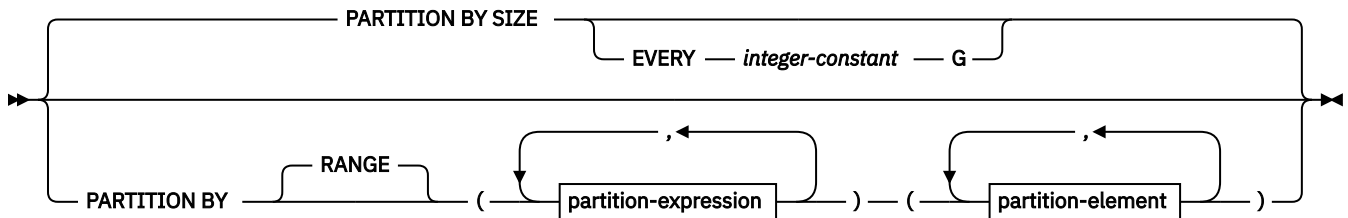
➤ DATA INITIALLY DEFERRED — REFRESH DEFERRED →



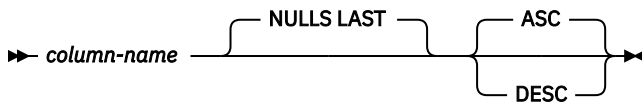
Notes:

¹ The same clause must not be specified more than one time.

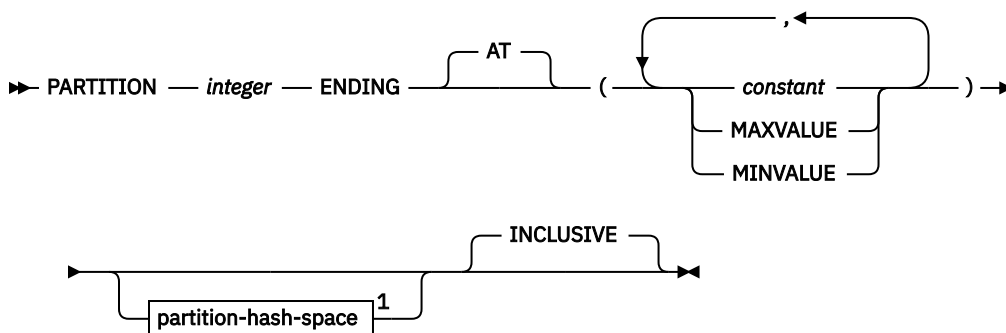
partitioning-clause:



partition-expression:



partition-element:



Notes:

¹ Hash-organized tables are deprecated. Beginning in Db2 12, packages bound with APPLCOMPAT(V12R1M504) or higher cannot create hash-organized tables or alter existing tables to use hash-organization. Existing hash organized tables remain supported, but they are likely to be unsupported in the future.

Description

table-name

Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, alias, or synonym that exists at the current server or a table that exists in the SYSIBM.SYSPENDINGOBJECTS catalog table. The unqualified name must not be the same as an existing synonym.

If the name is qualified, the name can be a two-part or three-part name. If a three-part name is used, the first part must match the value of field Db2LOCATION NAME on installation panel DSNTIPR at the current server. (If the current server is not the local Db2, this name is not necessarily the name in the CURRENT SERVER special register.)

For more information, see [Guidelines for table names \(Db2 Administration Guide\)](#).

KEY LABEL *key-label-name* or NO KEY LABEL

Specifies whether key label is specified at the table level for encryption. The *table-name* must identify a table that resides in a universal table space, or a partitioned (non-UTS) table space. If you specify a *table-space-name* using the IN clause, a subsequent REORG of the table space is required for the key label value to take effect.

KEY LABEL *key-label-name*

Specifies the default key label that is used to encrypt all the table spaces and index spaces associated with the table. This includes base table spaces, auxiliary table spaces, XML table spaces, index spaces, and clone table spaces, regardless of whether they are explicitly or implicitly created. Users must set the key label for archive or history tables independently.

The data set must be Db2-managed for all the table spaces and index spaces associated with the table.

The *table-name* must not identify one of the following:

- An accelerator-only table.
- An auxiliary table.

The key label must be defined in ICSF and not refer to an archived key for decryption operations only. Db2 address space RACF user ID or group must be permitted access to the key label in RACF.

The key label can be inherited or overridden when the data set is allocated. For details about the order of precedence, see Notes®.

NO KEY LABEL

Indicates that there is no key label specified at the table level for encryption.

column-definition

column-name

Names a column of the table. For a dependent table, up to 749 columns can be named. For a table that is not a dependent, this number is 750. Do not qualify *column-name* and do not use the same name for more than one column of the table.

built-in-type

Specifies the data type of the column as one of the following built-in data types, and for character string data types, specifies the subtype. For more information about defining a table with a LOB column (CLOB, BLOB, or DBCLOB), see [Creating a table with LOB columns](#).

If IN ACCELERATOR is specified, not all data types are supported. For example, DECFLOAT, LOB, ROWID, TIMESTAMP WITH TIME ZONE, and XML are not supported. The *IBM Db2 Analytics Accelerator for z/OS: Stored Procedures Reference* has a complete list of supported data types.

SMALLINT

For a small integer.

INTEGER or INT

For a large integer.

BIGINT

For a big integer.

DECIMAL(*integer*,*integer*) or DEC(*integer*,*integer*)**DECIMAL(*integer*) or DEC(*integer*)****DECIMAL or DEC**

For a decimal number. The first integer is the precision of the number. That is, the total number of digits, which can range from 1 to 31. The second integer is the scale of the number. That is, the number of digits to the right of the decimal point, which can range from 0 to the precision of the number.

You can use DECIMAL(*p*) for DECIMAL(*p*,0) and DECIMAL for DECIMAL(5,0).

You can also use the word NUMERIC instead of DECIMAL. For example, NUMERIC(8) is equivalent to DECIMAL(8). Unlike DECIMAL, NUMERIC has no allowable abbreviation.

DECFLOAT(*integer*)

For a decimal floating-point number. The value of *integer* must be either 16 or 34 and represents the number of significant digits that can be stored. If *integer* is omitted, the DECFLOAT column will be capable of representing 34 significant digits.

FLOAT(*integer*)**FLOAT**

For a floating-point number. If *integer* is between 1 and 21 inclusive, the format is single precision floating-point. If the integer is between 22 and 53 inclusive, the format is double precision floating-point.

You can use DOUBLE PRECISION or FLOAT for FLOAT(53).

REAL

For single precision floating-point.

DOUBLE or DOUBLE PRECISION

For double precision floating-point

CHARACTER(*integer*) or CHAR(*integer*)**CHARACTER or CHAR**

For a fixed-length character string of length *integer*, which can range 1 - 255. If the length specification is omitted, a length of 1 character is assumed.

CCSID 1208

Specifies that the column is a Unicode column encoded in UTF-8. This clause must not be specified for an ASCII or Unicode table.

VARCHAR(*integer*), CHAR VARYING(*integer*), or CHARACTER VARYING(*integer*)

For a varying-length character string of maximum length *integer*, which can range from 1 to the maximum record size minus 10 bytes. See [Table 14 on page 182](#) to determine the maximum record size.

CCSID 1208

Specifies that the column is a Unicode column encoded in UTF-8. This clause must not be specified for an ASCII or Unicode table.

FOR *subtype* DATA

Specifies a subtype for a character string column, which is a column with a data type of CHAR, VARCHAR, or CLOB. Do not use the FOR *subtype* DATA clause with columns of any other data type (including any distinct type). *subtype* can be one of the following:

SBCS

Column holds single-byte data.

MIXED

Column holds mixed data. Do not specify MIXED if the value of field MIXED DATA on installation panel DSNTIPF is NO unless the CCSID UNICODE clause is also specified, or the table is being created in a Unicode table space or database.

BIT

Column holds BIT data. Do not specify BIT for a CLOB column.

Only character strings are valid when subtype is BIT.

If you do not specify the FOR clause, the column is defined with a default subtype. For ASCII or EBCDIC data:

- The default is SBCS when the value of field MIXED DATA on installation panel DSNTIPF is NO.
- The default is MIXED when the value is YES.

For Unicode data, the default subtype is MIXED.

A security label column is always considered SBCS data, regardless of the encoding scheme of the table.

CLOB(*integer* [K|M|G]), CHAR LARGE OBJECT(*integer* [K|M|G]), or CHARACTER LARGE OBJECT(*integer* [K|M|G])
CLOB, CHAR LARGE OBJECT, or CHARACTER LARGE OBJECT

For a character large object (CLOB) string of the specified maximum length in bytes. The maximum length must be in the range of 1 - 2147483647. A CLOB column has a varying-length. It cannot be referenced in certain contexts regardless of its maximum length. For more information, see [Restrictions using LOBs \(Db2 SQL\)](#).

When *integer* is not specified, the default length is 1M. The maximum value that can be specified for *integer* depends on whether a units indicator is also specified as shown in the following list.

integer

The maximum value for *integer* is 2147483647. The maximum length of the string is *integer*.

***integer* K**

The maximum value for *integer* is 2097152. The maximum length is 1024 times *integer*.

***integer* M**

The maximum value for *integer* is 2048. The maximum length is 1,048,576 times *integer*.

***integer* G**

The maximum value for *integer* is 2. The maximum length is 1,073,741,824 times *integer*.

If you specify a value that evaluates to 2 gigabytes (2,147,483,648), Db2 uses a value that is one byte less, or 2147483647.

CCSID 1208

Specifies that the column is a Unicode column encoded in UTF-8. This clause must not be specified for an ASCII or Unicode table.

GRAPHIC(*integer*)
GRAPHIC

For a fixed-length graphic string of length *integer*, which can range 1 - 127. If the length specification is omitted, a length of 1 character is assumed.

CCSID 1200

Specifies that the column is a Unicode column encoded in UTF-16. This clause must not be specified for an ASCII or Unicode table.

VARGRAPHIC(*integer*)

For a varying-length graphic string of maximum length *integer*, which must range from 1 to $n/2$, where n is the maximum row size minus 2 bytes.

CCSID 1200

Specifies that the column is a Unicode column encoded in UTF-16. This clause must not be specified for an ASCII or Unicode table.

DBCLOB(*integer* [K|M|G])
DBCLOB

For a double-byte character large object (DBCLOB) string of the specified maximum length in double-byte characters. The maximum length must be in the range of 1 - 1,073,741,823. A

DBCLOB column has a varying-length. It cannot be referenced in certain contexts regardless of its maximum length. For more information, see [Restrictions using LOBs \(Db2 SQL\)](#).

When *integer* is not specified, the default length is 1M. The meaning of *integer* K|M|G is similar to CLOB. The difference is that the number specified is the number of double-byte characters.

CCSID 1200

Specifies that the column is a Unicode column encoded in UTF-16. This clause must not be specified for an ASCII or Unicode table.

BINARY(*integer*)

A fixed-length binary string of length *integer*. The *integer* can range from 1 through 255. If the length specification is omitted, a length of 1 byte is assumed.

BINARY VARYING(*integer*) or VARBINARY(*integer*)

A varying-length binary string of maximum length *integer*, which can range from 1 through 32704. The length is limited by the page size of the table space.

BLOB (*integer* [K|M|G] or BINARY LARGE OBJECT(*integer* [K|M|G])

BLOB or BINARY LARGE OBJECT

For a binary large object (BLOB) string of the specified maximum length in bytes. The maximum length must be in the range of 1 through 2147483647. A BLOB column has a varying-length. It cannot be referenced in certain contexts regardless of its maximum length. For more information, see [Restrictions using LOBs \(Db2 SQL\)](#).

When *integer* is not specified, the default length is 1M. The meaning of *integer* K|M|G is the same as for CLOB.

DATE

For a date.

TIME

For a time.

TIMESTAMP(*integer*) WITHOUT TIME ZONE

For a timestamp. *integer* specifies the optional timestamp precision attribute and must be in the range from 0 to 12. The timestamp precision denotes the number of fractional second digits that are included in the timestamp. The default is 6.

TIMESTAMP(*integer*) WITH TIME ZONE

For a timestamp with time zone. *integer* specifies the optional timestamp precision attribute and must be in the range from 0 to 12. The timestamp precision denotes the number of fractional second digits that are included in the timestamp. The default is 6.

ROWID

For a row ID type.

A table can contain at most two ROWID columns. If it contains two, one column is implicitly generated by Db2 and the other column is explicitly defined as a ROWID without the IMPLICITLY HIDDEN attribute. The values in a ROWID column are unique for every row in the table and cannot be updated. You must specify NOT NULL with ROWID.

XML

For an XML document. Only well-formed XML documents can be inserted into an XML column.

If the XML column is the first XML column that you create for the table, a BIGINT DOCID column is implicitly created and is used to store a unique document identifier for the XML columns of a row.

XMLSCHEMA

Specifies one or more XML schemas that are used to validate the XML value. The same XML schema can not be specified more than one time.

If the XML value has already been validated, for example, the XML value is the result of the DSN_XMLVALIDATE function or from an XML column with a type modifier, and the XML schema against which the XML value is validated is one of the schemas specified in the *XML-type-modifier*, Db2 accepts the XML value without revalidation.

XML-schema-specification

Specifies one XML schema. The XML schema can be identified by using either the *registered-XML-schema-name* or the schema's target namespace followed by an optional schema location. Any XML schema that is referenced in this clause must be registered in the XML schema repository prior to use.

ID registered-XML-schema-name

Identifies an XML schema by using its *registered-XML-schema-name*. The name must uniquely identify an existing XML schema in the XML schema repository at the current server. If no XML schema by this name exists, an error is returned.

The schema qualifier must be SYSXSR.

URI target-namespace

Specifies the target namespace URI of the XML schema. The value for the target-namespace URI is a character string constant which is not empty. The URI must be the target namespace of a registered XML schema and, if no LOCATION clause is specified, it must uniquely identify the registered XML schema.

NO NAMESPACE

Specifies that the XML schema has no target namespace. There must be a registered XML schema that has no target namespace. If no LOCATION clause is specified, there must be only one such registered XML schema.

LOCATION schema-location

Specifies the XML schema location URI of the XML schema. The value of *schema-location* is a character string constant that is not empty. The schema location URI, combined with the target namespace URI, must identify a registered XML schema.

ELEMENT element-name

Specifies the name of the global element declaration. *element-name* must match the local name of the root element node in the instance XML document. The namespace name of the root element node must be the same as the target namespace URI.

distinct-type-name

Specifies the data type of the column is a distinct type (a user-defined data type). The length, precision, and scale of the column are respectively the length, precision, and scale of the source type of the distinct type. The privilege set must implicitly or explicitly include the USAGE privilege on the distinct type.

The encoding scheme of the distinct type must be the same as the encoding scheme of the table. The subtype for the distinct type, if it has the attribute, is the subtype with which the distinct type was created.

If the column is to be used in the definition of the foreign key of a referential constraint, the data type of the corresponding column of the parent key must have the same distinct type.

NOT NULL

Prevents the column from containing null values. Omission of NOT NULL implies that the column can contain null values.

column-constraint

The *column-constraint* of a *column-definition* provides a shorthand method of defining a constraint composed of a single column. Thus, if a *column-constraint* is specified in the definition of column C, the effect is the same as if that constraint were specified as a *unique-constraint*, *referential-constraint*, or *check-constraint* in which C is the only identified column.

CONSTRAINT constraint-name

Names the constraint. If a constraint name is not specified, a unique constraint name is generated. If the name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

PRIMARY KEY

Provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

The NOT NULL clause must be specified with this clause. PRIMARY KEY cannot be specified more than one time in a column definition, and must not be specified if the UNIQUE clause is specified in the definition. This clause must also not be specified if the definition is for one of the following types of columns:

- a LOB column
- a ROWID column
- a distinct type column that is based on a LOB or ROWID data type
- an XML column
- a row change timestamp column
- a column in an accelerator-only table

The table is marked as unavailable until its primary index is explicitly created unless the CREATE TABLE statement is processed by the schema processor or the table space that contains the table is implicitly created. In that case, Db2 implicitly creates an index to enforce the uniqueness of the primary key and the table definition is considered complete. (For more information about implicitly created indexes, see [Implicitly created indexes](#).)

UNIQUE

Provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

The NOT NULL clause must be specified with this clause. UNIQUE cannot be specified more than one time in a column definition and must not be specified if the PRIMARY KEY clause is specified in the column definition or if the definition is for one of the following types of columns:

- a LOB column
- a ROWID column
- a distinct type column that is based on a LOB or ROWID data type
- an XML column
- a row change timestamp column
- a column in an accelerator-only table

The table is marked as unavailable until all the required indexes are explicitly created unless the CREATE TABLE statement is processed by the schema processor or the table space that contains the table is implicitly created. In that case, Db2 implicitly creates the indexes that are required for the unique keys and the table definition is considered complete. (For more information about implicitly created indexes, see [Implicitly created indexes](#).)

references-clause

The *references-clause* of a *column-definition* provides a shorthand method of defining a foreign key composed of a single column. Thus, if *references-clause* is specified in the definition of column C, the effect is the same as if the *references-clause* were specified as part of a FOREIGN KEY clause in which C is the only identified column.

Do not specify *references-clause* in the definition of the following types of columns because these types of columns cannot be a foreign key:

- a LOB column
- a ROWID column
- a DECFLOAT column
- a distinct type column that is based on a LOB, ROWID, or DECFLOAT data type

- an XML column
- a row change timestamp column
- a security label column

CHECK (*check-condition*)

CHECK (*check-condition*) provides a shorthand method of defining a check constraint that applies to a single column. For conformance with the SQL standard, if CHECK is specified in the column definition of column C, no columns other than C should be referenced in the check condition of the check constraint. The effect is the same as if the check condition were specified as a separate clause.

DEFAULT

Specifies the default value that is assigned to the column in the absence of a value specified on an insert or update operation or LOAD. DEFAULT must not be specified more than one time in the same *column-definition*. Do not specify DEFAULT for the following types of columns because Db2 generates default values:

- An identity column (a column that is defined AS IDENTITY)
- A ROWID column (or a distinct type that is based on a ROWID)
- A row change timestamp column
- A row-begin column
- A row-end column
- A *transaction-start-id* column
- An XML column

If IN ACCELERATOR is specified, do not specify DEFAULT for a column.

Do not specify a value after the DEFAULT keyword for a security label column. Db2 provides the default value for a security label column.

If a value is not specified after DEFAULT, the default value depends on the data type of the column, as follows:

Data Type

Default Value

Numeric

0

Big integer

0

Fixed-length character string

Blanks

Fixed-length graphic string

Blanks

Fixed-length binary string

Hexadecimal zeros

Varying-length string

A string of length 0

Inline BLOB

Hexadecimal zeros

Inline CLOB

Blanks

Inline DBCLOB

Blanks

Date

CURRENT DATE

Time

CURRENT TIME

TIMESTAMP(*integer*) WITHOUT TIME ZONE

CURRENT TIMESTAMP(*p*) WITHOUT TIME ZONE where *p* is the corresponding timestamp precision.

TIMESTAMP(*integer*) WITH TIME ZONE

CURRENT TIMESTAMP(*p*) WITH TIME ZONE where *p* is the corresponding timestamp precision.

If the column is defined as timestamp with time zone the default value must include a time zone.

Distinct type

The default of the source data type

A default value other than the one that is listed above can be specified in one of the following forms:

- WITH DEFAULT for a default value of an empty string
- DEFAULT NULL for a default value of null

Omission of NOT NULL and DEFAULT from a *column-definition*, for a column other than an identity column, is an implicit specification of DEFAULT NULL. For an identity column, it is an implicit specification of NOT NULL, and Db2 generates default values.

constant

Specifies a constant as the default value for the column. The value of the constant must conform to the rules for assigning that value to the column.

A character or graphic string constant must be short enough so that its UTF-8 representation requires no more than 1536. A hexadecimal graphic string constant (GX) cannot be specified.

In addition, the length of the constant value cannot be greater than the INLINE LENGTH attribute for LOB columns.

SESSION_USER or USER

Specifies the value of the SESSION_USER (USER) special register at the time of an SQL data change statement or LOAD as the default value for the column. If SESSION_USER is specified, the data type of the column must be a character string with a length attribute greater than or equal to 8 characters when the value is expressed in CCSID 37. If the data type of the column is an inline CLOB, the INLINE LENGTH attribute must be greater than or equal to 8 characters when the value is expressed as CCSID 37.

CURRENT SQLID

Specifies the value of the SQL authorization ID of the process at the time of an insert or update operation or LOAD as the default value for the column. If CURRENT SQLID is specified, the data type of the column must be a character string with a length attribute greater than or equal to the length attribute of the CURRENT SQLID special register. If the data type of the column is an inline CLOB, the INLINE LENGTH attribute must be greater than or equal to the length attribute of the CURRENT SQLID special register.

NULL

Specifies null as the default value for the column. If NOT NULL is specified, DEFAULT NULL must not be specified with the same *column-definition*.

cast-function-name

The name of the cast function that matches the name of the distinct type for the column. A cast function can only be specified if the data type of the column is a distinct type.

The schema name of the cast function, whether it is explicitly specified or implicitly resolved through function resolution, must be the same as the explicitly or implicitly specified schema name of the distinct type.

constant

Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type. The length of the constant cannot be greater than the INLINE LENGTH attribute for LOB columns.

SESSION_USER or USER

Specifies the value of the SESSION_USER (USER) special register at the time a row is inserted as the default for the column. The source type of the distinct type of the column must be a CHAR, VARCHAR, or inline CLOB with a length attribute (inline length attribute for CLOB) that is greater than or equal to the length attribute of the SESSION_USER special register.

CURRENT SQLID

Specifies the value of the CURRENT SQLID special register at the time a row is inserted as the default for the column. The source type of the distinct type of the column must be a CHAR, VARCHAR, or inline CLOB with a length attribute (or inline length attribute for CLOB) that is greater than or equal to the length attribute of the CURRENT SQLID special register.

NULL

Specifies the NULL value as the argument.

In a given column definition:

- DEFAULT and FIELDPROC cannot both be specified.
- NOT NULL and DEFAULT NULL cannot both be specified.

Table 10 on page 146 summarizes the effect of specifying the various combinations of the NOT NULL and DEFAULT clauses on the CREATE TABLE statement *column-description* clause.

Table 10. Effect of specifying combinations of the NOT NULL and DEFAULT clauses

If NOT NULL is:	And DEFAULT is:	The effect is:
Specified ¹	Omitted	An error occurs if a value is not provided for the column on an insert or update operation or LOAD.
	Specified without an operand	The system defined nonnull default value is used.
	<i>constant</i>	The specified constant is used as the default value.
	SESSION_USER	The value of the SESSION_USER special register at the time of an insert or update operation or LOAD is used as the default value.
	CURRENT SQLID	The SQL authorization ID of the process at the time of an insert or update operation or LOAD is used as the default value.
	NULL	An error occurs during the execution of CREATE TABLE.

Table 10. Effect of specifying combinations of the NOT NULL and DEFAULT clauses (continued)

If NOT NULL is:	And DEFAULT is:	The effect is:
Omitted	Omitted	Equivalent to an implicit specification of DEFAULT NULL.
	Specified without an operand	The system defined nonnull default value is used.
	<i>constant</i>	The specified constant is used as the default value.
	SESSION_USER	The value of the SESSION_USER special register at execution time is used as the default value.
	CURRENT SQLID	The SQL authorization ID of the process is used as the default value.
	NULL	Null is used as the default value.

Note: The table does not apply to a column with a ROWID data type or to an identity column.

GENERATED

Specifies that Db2 generates values for the column. GENERATED must be specified if the column is to be considered one of the following types of columns:

- An identity column
- A row change timestamp column.
- A ROWID column
- A row-begin column
- A row-end column
- A *transaction-start-id* column
- A generated expression column

GENERATED must only be specified for these types of columns. GENERATED must not be specified with *default-clause* in a column definition.

GENERATED must not be specified if the column definition references global variables.

ALWAYS

Specifies that Db2 always generates a value for the column when a row is inserted or updated and a default value must be generated. ALWAYS is the default and recommended value.

BY DEFAULT

Specifies that Db2 will generate a value for the column when a row is inserted or updated and a default value must be generated, unless an explicit value is specified.

For a row change timestamp column, Db2 inserts or updates a specified value but does not verify that the value is unique for the column unless the row change timestamp column has a unique constraint or a unique index that specifies only the row change timestamp column.

For a ROWID column, Db2 uses a specified value only if it is a valid row ID value that was previously generated by Db2 and the column has a unique, single-column index. Until this index is created on the ROWID column, the SQL insert or update operation and the LOAD utility cannot be used to add rows to the table. If the table space is explicitly created and the value of the CURRENT RULES special register is 'STD' when the CREATE TABLE statement is processed, or if the table space is implicitly created, Db2 implicitly creates the index on the ROWID column. The name of this index is 'I' followed by the first ten characters of the column name followed by seven randomly generated characters. If the column name is less than ten characters, Db2 adds

underscore characters to the end of the name until it has ten characters. The implicitly created index has the COPY NO attribute.

For an identity column, Db2 inserts a specified value but does not verify that it is a unique value for the column unless the identity column has a unique, single-column index.

BY DEFAULT is the recommended value only when you are using data propagation.

FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

Specifies that the column is a timestamp column for the table. Db2 generates a value for the column for each row as the row is inserted, and for any row in which any column is updated. The value that is generated for a row change timestamp column is a timestamp that corresponds to the insert or update time of the row. If multiple rows are inserted or updated with a single statement, the value for the row change timestamp column might be different for each row.

A table can only have one row change timestamp column.

If *data-type* is specified, it must be `TIMESTAMP WITHOUT TIME ZONE` with a precision of 6.

A row change timestamp column cannot have a `DEFAULT` clause. `NOT NULL` must be specified for a row change timestamp column.

AS TRANSACTION START ID

Specifies that the value is assigned by Db2 whenever a row is inserted into the table or any column in the row is updated. Db2 assigns a unique timestamp value per transaction or the null value. The null value is assigned to the transaction-start-ID column if the column is nullable. Otherwise, the value is generated using the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to a row-begin column or transaction-start-ID column in the table, or when a row in a system-period temporal table is deleted. If multiple rows are inserted or updated within a single SQL transaction, the values for the transaction-start-ID column are the same for all the rows and are unique from the values that are generated for the column for another transaction.

A transaction-start-ID column is required for a system-period temporal table.

A table can have only one transaction-start-ID column. If a data type is not specified, the column is defined as `TIMESTAMP(12) WITHOUT TIME ZONE`. If a data type is specified, it must be `TIMESTAMP(12) WITHOUT TIME ZONE` or `TIMESTAMP(12) WITH TIME ZONE`. If the column is defined as `TIMESTAMP WITH TIME ZONE`, the values are stored in UTC, with a time zone of `+00:00`. A transaction-start-ID column cannot have a `DEFAULT` clause. A transaction-start-ID column is not updatable.

A value for a transaction-start-ID column is composed of a `TIMESTAMP(9)` value that is unique per transaction per data sharing member followed by 3 digits that indicate the data sharing member number.

AS ROW BEGIN

Specifies that a timestamp value is assigned to the column whenever a row is inserted or any column in the row is updated. If the value of the `SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME` built-in global variable at the time of the insert or update is null, the value is generated using a reading of the time-of-day clock during execution of the first data change statement in the unit of work that requires a value to be assigned to a row-begin column or transaction-start-ID column in a table, or a row in a system-period temporal table is deleted. Otherwise, the row-begin column is assigned the value of the `SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME` built-in global variable at the time of the insert or update.

A *row-begin* column is intended to be used for a system-period temporal table.

A table can have only one column defined as `AS ROW BEGIN`. If a data type is specified, it must be `TIMESTAMP(12) WITHOUT TIME ZONE` or `TIMESTAMP(12) WITH TIME ZONE`. If the column is defined as `TIMESTAMP(12) WITH TIME ZONE`, the values are stored in UTC, with a time zone of `+00:00`. If no data type is specified, the column is defined as `TIMESTAMP(12) WITHOUT TIME`

ZONE. A column defined as a row-begin column cannot have a DEFAULT clause, and must be defined as NOT NULL.

A row-begin column is not updatable.

A value for a row-begin column is composed of a TIMESTAMP(9) value that is unique per transaction per data sharing member followed by 3 digits that indicate the data sharing member number.

AS ROW END

Specifies that a value for the data type of the column is assigned by Db2 whenever a row is inserted or any column in the row is updated. The value that is assigned for a TIMESTAMP WITHOUT TIME ZONE column is the TIMESTAMP value '9999-12-30-00.00.00.000000000000'. The value that is assigned for a TIMESTAMP WITH TIME ZONE column is the TIMESTAMP value '9999-12-30-00.00.00.000000000000 +00:00'.

A row-end column is required as the second column of a SYSTEM_TIME period.

A table can have only one row-end column. If a data type is not specified, the column is defined as TIMESTAMP(12) WITHOUT TIME ZONE. If a data type is specified, it must be TIMESTAMP(12) WITHOUT TIME ZONE or TIMESTAMP(12) WITH TIME ZONE. If the column is defined as TIMESTAMP WITH TIME ZONE, the values are stored in UTC, with a time zone of +00:00. A row-end column cannot have a DEFAULT clause and must be defined as NOT NULL. A row-end column is not updatable.

as-generated-expression-clause

Specifies that values for the column are generated by Db2. The generated value is assigned to the column whenever a row is inserted, or any column in the row is updated.

DATA CHANGE OPERATION

Specifies that the database manager generates one of the following values, depending on the specified expression:

- I**
Insert operation.
- U**
Update operation.
- D**
Delete operation.

A table can have only one DATA CHANGE OPERATION column. The column must be defined as CHAR(1). The column cannot have a DEFAULT clause and must not be defined as NOT NULL.

The column is a non-deterministic generated expression column.

Do not specify any of the following clauses for the column:

- CCSID 1200
- CCSID 1208
- FIELDPROC

special-register

Specifies the value of the special register. The column is to contain the value of the special register at the time of the data change statement that assigns the value to the column. If multiple rows are inserted or updated with a single SQL statement, the value for the column is the same for all of the rows.

special-register must be one of the following special registers, and the column must use the required data type.

Table 11. Possible special register values for non-deterministic generated expression columns

Special register	Data type for the column
CURRENT CLIENT_ACCTNG	VARCHAR(255)
CURRENT CLIENT_APPLNAME	VARCHAR(255)
CURRENT CLIENT_CORR_TOKEN	VARCHAR(255)
CURRENT CLIENT_USERID	VARCHAR(255)
CURRENT CLIENT_WRKSTNNAME	VARCHAR(255)
CURRENT SERVER	CHAR(16)
CURRENT SQLID	VARCHAR(<i>n</i>) where $n \geq 8$
SESSION_USER or USER	VARCHAR(128)

The column cannot have a DEFAULT clause and must not be defined as NOT NULL.

The column is a non-deterministic generated expression column.

Do not specify any of the following clauses for the column:

- CCSID 1200
- CCSID 1208
- FIELDPROC

For more information, see [Special registers \(Db2 SQL\)](#).

session-variable

Specifies the value of a built-in session variable. The fully qualified name of the session variable must be specified. The value of the session variable is obtained from the GETVARIABLE function at the time of the data change operation that assigns the value to the column. If multiple rows are changed with a single SQL statement, the value for the column is the same for all of the rows.

session-variable must be one of the following session variables, and the column must use the required data type.

Table 12. Possible session variable values for non-deterministic generated expression columns

Session variable	Data type for the column
SYSIBM.PACKAGE_NAME	VARCHAR(128)
SYSIBM.PACKAGE_SCHEMA	VARCHAR(128)
SYSIBM.PACKAGE_VERSION	VARCHAR(122)

The column cannot have a DEFAULT clause and must not be defined as NOT NULL.

The column is a non-deterministic generated expression column.

Do not specify any of the following clauses for the column:

- CCSID 1200
- CCSID 1208
- FIELDPROC

For more information, see [Built-in session variables \(Db2 SQL\)](#).

AS IDENTITY

Specifies that the column is an identity column for the table. A table can have only one identity column. AS IDENTITY can be specified only if the data type for the column is an exact numeric

type with a scale of zero (SMALLINT, INTEGER, BIGINT, DECIMAL with a scale of zero, or a distinct type based on one of these types).

An identity column is implicitly NOT NULL. An identity column cannot have a WITH DEFAULT clause.

Defining a column AS IDENTITY does not necessarily ensure the uniqueness of the values. To ensure uniqueness of the values, define a unique, single-column index on the identity column.

If IN ACCELERATOR is specified, AS IDENTITY must not be specified.

START WITH *numeric-constant*

Specifies the first value that is generated for the identity column. The value can be any positive or negative value that could be assigned to the column without non-zero digits existing to the right of the decimal point.

If a value is not explicitly specified when the identity column is defined, the default is the MINVALUE for an ascending identity column and the MAXVALUE for a descending identity column. This value is not necessarily the value that would be cycled to after reaching the maximum or minimum value for the identity column. The range used for cycles is defined by MINVALUE and MAXVALUE. MAXVALUE and MINVALUE do not constrain the numeric-constant value. That is, the START WITH clause can be used to start the generation of values outside the range that is used for cycles. However, the next generated value after the specified START WITH value is MINVALUE for an ascending identity column or MAXVALUE for a descending identity column.

INCREMENT BY *numeric-constant*

Specifies the interval between consecutive values of the identity column. The value can be any positive or negative value (including 0) that does not exceed the value of a large integer constant, and could be assigned to the column without any non-zero digits existing to the right of the decimal point.

If this value is negative, the values for the identity column descend. If this value is 0 or positive, the values for the identity column ascend. The default is 1.

MINVALUE or NO MINVALUE

Specifies the minimum value at which a descending identity column either cycles or stops generating values or an ascending identity column cycles to after reaching the maximum value.

NO MINVALUE

Specifies that the minimum end point of the range of values for the identity column has not been set. In such a case, the default value for MINVALUE becomes one of the following:

- For an ascending identity column, the value is the START WITH value or 1 if START WITH is not specified.
- For a descending identity column, the value is the minimum value of the data type of the column.

The default is NO MINVALUE.

MINVALUE *numeric-constant*

Specifies the numeric constant that is the minimum value that is generated for this identity column. This value can be any positive or negative value that could be assigned to this column without non-zero digits existing to the right of the decimal point. The value must be less than or equal to the maximum value.

MAXVALUE or NO MAXVALUE

Specifies the maximum value at which an ascending identity column either cycles or stops generating values or a descending identity column cycles to after reaching the minimum value.

NO MAXVALUE

Specifies that the minimum end point of the range of values for the identity column has not been set. In such a case, the default value for MAXVALUE becomes one of the following:

- For an ascending identity column, the value is the maximum value of the data type associated with the column.
- For a descending identity column, the value is the START WITH value -1 if START WITH is not specified.

The default is NO MAXVALUE.

MAXVALUE *numeric-constant*

Specifies the numeric constant that is the maximum value that is generated for this identity column. This value can be any positive or negative value that could be assigned to this column without non-zero digits existing to the right of the decimal point. The value must be greater than or equal to the minimum value.

CYCLE or NO CYCLE

Specifies whether this identity column should continue to generate values after reaching either its maximum or minimum value. The default is NO CYCLE.

NO CYCLE

Specifies that values will not be generated for the identity column after the maximum or minimum value has been reached.

CYCLE

Specifies that values continue to be generated for the identity column after the maximum or minimum value has been reached. If this option is used, after an ascending identity column reaches the maximum value, it generates its minimum value. After a descending identity column reaches its minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, duplicate values can be generated by Db2 for an identity column. However, if a unique index exists on the identity column and a non-unique value is generated for it, an error occurs.

CACHE *integer-constant* or NO CACHE

Specifies whether to keep some preallocated values in memory. Preallocating and storing values in the cache improves the performance of inserting rows into a table. The default is CACHE 20.

In a non-data sharing environment, if the system is shut down (either normally or through a system failure), all cached sequence values that have not been used in committed statements are lost (that is, they will never be used). The value specified for the CACHE option is the maximum number of sequence values that could be lost when the system is shut down.

In a data sharing environment, you can use the CACHE and NO ORDER options to allow multiple Db2 members to cache sequence values simultaneously.

NO CACHE

Specifies that values for the identity column and sequences are not preallocated and stored in the cache, ensuring that values will not be lost in the case of a system failure. In this case, every request for a new value for the identity column or sequence results in synchronous I/O.

In a data sharing environment, use NO CACHE if you need to guarantee that the identity column and sequence values are generated in the order in which they are requested.

CACHE *integer-constant*

Specifies the maximum number of values of the identity column sequence that Db2 can preallocate and keep in memory.

During a Db2 shutdown, all cached identity column values and sequence values that are yet to be assigned will be lost and will not be used. Therefore, the value that is specified for CACHE also represents the maximum number of identity column values and sequence values that will be lost during a Db2 shutdown.

The minimum value is 2.

In a data sharing environment, you can use the CACHE and NO ORDER options to allow multiple Db2 members to cache sequence values simultaneously.

ORDER or NO ORDER

Specifies whether the identity column values must be generated in order of request. The default is NO ORDER.

In a non-data sharing environment, there is no guarantee that values are assigned in order across the entire server unless NO CACHE is also specified. ORDER applies only to a single-application process.

In a data sharing environment, if ORDER is specified, NO CACHE is implicitly set, even if CACHE integer-constant is specified.

NO ORDER

Specifies that the values do not need to be generated in order of request.

ORDER

Specifies that the values are generated in order of request. Specifying ORDER might disable the caching of values. ORDER applies only to a single-application process.

In a data sharing environment, if the CACHE and NO ORDER options are in effect, multiple caches can be active simultaneously, and the requests for identity values from different Db2 members might not result in the assignment of values in strict numeric order. For example, if members DB2A and DB2B are using the identity column, and DB2A gets the cache values 1 to 20 and DB2B gets the cache values 21 to 40, the actual order of values assigned would be 1,21,2 if DB2A requested a value first, then DB2B requested, and then DB2A again requested. Therefore, to guarantee that identity values are generated in strict numeric order among multiple Db2 members using the same identity column, specify the ORDER option.

FIELDPROC *program-name*

Designates *program-name* as the field procedure exit routine for the column. A field procedure can be specified only for a column with a length attribute that is not greater than 255 bytes. FIELDPROC can only be specified for columns that are a built-in character string or graphic string data types. The column must not be one of the following:

- a LOB column
- a security label column
- a row change timestamp column
- a column with the TIMESTAMP WITH TIME ZONE data type
- a Unicode column in an EBCDIC table
- a column in an accelerator-only table

The field procedure encodes and decodes column values: before a value is inserted in the column, it is passed to the field procedure for encoding. Before a value from the column is used by a program, it is passed to the field procedure for decoding. A field procedure could be used, for example, to alter the sorting sequence of values entered in the column.

The field procedure is also invoked during the processing of the CREATE TABLE statement. When so invoked, the procedure provides Db2 with the column's *field description*. The field description defines the data characteristics of the encoded values. By contrast, the information you supply for the column in the CREATE TABLE statement defines the data characteristics of the decoded values.

For more information, see:

[Field procedures \(Db2 Administration Guide\)](#)

[Character and graphic string comparisons \(Db2 SQL\)](#)

constant

Is a parameter that is passed to the field procedure when it is invoked. A parameter list is optional. The *n*th parameter specified in the FIELDPROC clause on CREATE TABLE corresponds to the *n*th

parameter of the specified field procedure. The maximum length of the parameter list is 254 bytes, including commas but excluding insignificant blanks and the delimiting parentheses.

If you omit FIELDPROC, the column has no field procedure.

AS SECURITY LABEL

Specifies that the column will contain security label values. This also indicates that the table is defined with multilevel security with row level granularity. A table can have only one security label column. A security label column cannot be defined for an accelerator-only table. To define a table with a security label column, the primary authorization ID of the statement must have a valid security label, and the RACF SECLABEL class must be active. In addition, the following conditions are also required:

- The data type of the column must be CHAR(8).
- The subtype of the column must be SBCS.
- The column must be defined with the NOT NULL and WITH DEFAULT clauses.
- The column must be an EBCDIC column.
- The WITH DEFAULT clause must not specify a default value (Db2 determines the default value)
- No field procedures, check constraints, or referential constraints are defined on the column.
- No edit procedure for the table can be defined with row attribute sensitivity.

For information about using multilevel security, see [Multilevel security \(Managing Security\)](#).

IMPLICITLY HIDDEN

Specifies that the column is not visible in the result for SQL statements unless you explicitly refer to the column by name. For example, assuming that the table T1 includes a column that is defined with the IMPLICITLY HIDDEN clause, the result of a SELECT * would not include the implicitly hidden column. However, the result of a SELECT statement that explicitly refers to the name of the implicitly hidden column would include that column in the result table.

IMPLICITLY HIDDEN must not be specified for all columns of a table. If IN ACCELERATOR is specified, IMPLICITLY HIDDEN must not be specified.

INLINE LENGTH *integer*

Specifies the maximum length of the inline portion of a LOB column value. The inline portion is the portion that is stored in the base table space. INLINE LENGTH cannot be specified if the column is not a LOB column (or a distinct type that is based on a LOB), if the table is not in a universal table space, or if the table is an accelerator-only table.

For BLOB and CLOB columns, *integer* specifies the maximum number of bytes that are stored in the base table space for the column. *integer* must be between 0 and 32680 (inclusive) for a BLOB or CLOB column.

For a DBCLOB column, *integer* specifies the maximum number of double-byte characters that are stored in the table space for the column. *integer* must be between 0 and 16340 (inclusive) for a DBCLOB column.

If INLINE LENGTH is specified, the value of *integer* cannot be greater than the maximum length of the LOB column.

If the INLINE LENGTH clause is not specified, the maximum length of the LOB column depends on the following conditions:

- If a distinct type is not used or the distinct type that is used has been created without the INLINE LENGTH attribute, the LOB column will use the value of the LOB INLINE LENGTH parameter on installation panel DSNTIPD as the default inline length when the value of LOB INLINE LENGTH does not exceed the maximum length of the LOB column. If the value of LOB INLINE LENGTH exceeds the maximum length of the LOB column, the maximum length is the inline length of this LOB column.
- If a distinct type that has been created with the INLINE LENGTH attribute is used, the LOB column inherits the inline length from the distinct type.

Regardless of how the length is determined, the inline length of the LOB cannot be greater than its maximum length.

period-definition

PERIOD FOR

Defines a period for the table. *begin-column-name* must not be the same as *end-column-name*. The data type, length, precision, and scale for *begin-column-name* must be the same as for *end-column-name*.

If IN ACCELERATOR is specified, PERIOD must not be specified.

SYSTEM_TIME (*begin-column-name*, *end-column-name*)

Defines a system period with the name SYSTEM_TIME. There must not be a column in the table with the name SYSTEM_TIME. A table can have only one SYSTEM_TIME period. *begin-column-name* must be defined as AS ROW BEGIN and *end-column-name* must be defined as AS ROW END.

BUSINESS_TIME (*begin-column-name*, *end-column-name*)

Defines an application period with the name BUSINESS_TIME. There must not be a column in the table with the name BUSINESS_TIME. A table can have only one BUSINESS_TIME period. *begin-column-name* and *end-column-name* must be defined as DATE or TIMESTAMP(6) WITHOUT TIME ZONE, and the columns must be defined as NOT NULL. *begin-column-name* and *end-column-name* must not identify a column that is defined with a GENERATED clause.

An implicit check constraint is generated to ensure the relationship of the value of *end-column-name* to the value of *begin-column-name* as follows:

- For an inclusive-exclusive BUSINESS_TIME period, the value of *end-column-name* is greater than the value of *begin-column-name*.
- For an inclusive-inclusive BUSINESS_TIME period, the value of *end-column-name* is greater than or equal to the value of *begin-column-name*.

The name of the implicitly created check constraint is DB2_GENERATED_CHECK_CONSTRAINT_FOR_BUSINESS_TIME, and that name must not be defined as the name of an existing check constraint.

begin-column-name

Identifies the column that records the beginning of the period of time in which a row is valid. The name must identify a column that exists in the table and must not be the same as a column that is used in the definition of another period for the table. *begin-column-name* must not be the same as *end-column-name*. The data type and precision for *begin-column-name* must be the same as for *end-column-name*.

For a SYSTEM_TIME period, *begin-column-name* must be defined as AS ROW BEGIN.

For a BUSINESS_TIME period, the column must not be defined with a GENERATED clause.

end-column-name

Identifies the column that records the end of the period of time in which a row is valid. In the history table that is associated with a system-period temporal table, the history table column that corresponds to *end-column-name* in the system-period temporal table is set to reflect the deletion of the row. The name must identify a column that exists in the table and must not be the same as a column that is used in the definition of another period for the table.

For a SYSTEM_TIME period, *end-column-name* must be defined as AS ROW END.

For a BUSINESS_TIME period, the column must not be defined with a GENERATED clause.

EXCLUSIVE

Specifies that the value of the end column is not included in the period. The BUSINESS_TIME period is defined as inclusive-exclusive.

INCLUSIVE

Specifies that the value of the end column is included in the period. The BUSINESS_TIME period is defined as inclusive-inclusive.

unique-constraint

CONSTRAINT *constraint-name*

Names the constraint. If a constraint name is not specified, a unique constraint name is generated. If a name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

PRIMARY KEY(*column-name*,...)

Defines a primary key composed of the identified columns. The clause must not be specified more than one time and the same column must not be identified more than one time. The identified columns must be defined as NOT NULL. Each *column-name* must be an unqualified name that identifies a column of the table except for the following types of columns:

- a LOB column
- a ROWID column
- a distinct type column that is based on a LOB or ROWID data type
- an XML column
- a row change timestamp column
- a column in an accelerator-only table

All character and graphic string columns in the key must use the same encoding scheme.

The number of identified columns must not exceed 64. In addition, the sum of the length attributes of the columns must not be greater than $2000 - n - 2m - 3d$, where m is the number of varying-length columns and d is the number of DECFLOAT columns in the key.

The table is marked as unavailable until its primary index is explicitly created unless the table space is explicitly created and the CREATE TABLE statement is processed by the schema processor, or the table space is implicitly created. In that case, Db2 implicitly creates an index to enforce the uniqueness of the primary key and the table definition is considered complete. (For more information about implicitly created indexes, see [Implicitly created indexes](#).)

BUSINESS_TIME WITHOUT OVERLAPS can be specified as the last item in the list. If BUSINESS_TIME WITHOUT OVERLAPS is specified, the list must include at least one *column-name* or *key-expression*. When WITHOUT OVERLAPS is specified, the values for the rest of the specified keys are unique with respect to the time for the BUSINESS_TIME period. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the columns of the BUSINESS_TIME period must not be specified as part of the constraint. The specification of BUSINESS_TIME WITHOUT OVERLAPS adds the following to the constraint:

- The end column of the BUSINESS_TIME period in ascending order
- The begin column of the BUSINESS_TIME period in ascending order

UNIQUE(*column-name*,...)

Defines a unique key composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of the table. Each identified column must be defined as NOT NULL. The same column must not be identified more than one time. The following types of columns cannot be identified:

- a LOB column
- a ROWID column
- a distinct type column that is based on a LOB or ROWID data type
- an XML column
- a row change timestamp column
- a column in an accelerator-only table

The number of identified columns must not exceed 64. In addition, the sum of the length attributes of the columns must not be greater than $2000 - n - 2m - 3d$, where m is the number of varying-length columns and d is the number of DECFLOAT columns in the key.

All character and graphic string columns in the key must use the same encoding scheme.

A unique key is a duplicate if it is the same as the primary key or a previously defined unique key. The specification of a duplicate unique key is ignored with a warning.

The table is marked as unavailable until all the required indexes are explicitly created unless the table space is explicitly created and the CREATE TABLE statement is processed by the schema processor, or the table space is implicitly created. In these cases, Db2 implicitly creates the indexes that are required for the unique keys and the table definition is considered complete. (For more information about implicitly created indexes, see [Implicitly created indexes](#).)

BUSINESS_TIME WITHOUT OVERLAPS can be specified as the last item in the list. If BUSINESS_TIME WITHOUT OVERLAPS is specified, the list must include at least one *column-name* or *key-expression*. When WITHOUT OVERLAPS is specified, the values for the rest of the specified keys are unique with respect to the time for the BUSINESS_TIME period. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the columns of the BUSINESS_TIME period must not be specified as part of the constraint. The specification of BUSINESS_TIME WITHOUT OVERLAPS adds the following to the constraint:

- The end column of the BUSINESS_TIME period in ascending order
- The begin column of the BUSINESS_TIME period in ascending order

referential-constraint

CONSTRAINT *constraint-name*

Names the referential constraint. If a constraint name is not specified, a unique constraint name is generated. If a name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

FOREIGN KEY (*column-name*,...) references-clause

Each specification of the FOREIGN KEY clause defines a referential constraint. The table being created is the child table for the referential constraint.

The foreign key of the referential constraint is composed of the identified columns, and the columns of the BUSINESS_TIME period if the clause PERIOD BUSINESS_TIME is specified. Each *column-name* must be an unqualified name that identifies a column of the table. The same column must not be identified more than one time. If PERIOD BUSINESS_TIME is specified, the columns of the BUSINESS_TIME period must not be specified as part of the constraint. The column cannot be any of the following types of columns:

- a LOB column
- a ROWID column
- a DECFLOAT column
- an XML column
- a row change timestamp column
- a security label column
- a column in an accelerator-only table

The number of identified columns, and the columns of the BUSINESS_TIME period if the clause PERIOD BUSINESS_TIME is specified, must not exceed 64, including columns of the BUSINESS_TIME period if PERIOD BUSINESS_TIME is specified. The sum of the column length attributes must not exceed 255 minus the number of columns that allow null values. The referential constraint is a duplicate if the FOREIGN KEY and parent table are the same as the FOREIGN KEY and parent table of a previously defined referential constraint. The specification of a duplicate referential constraint is ignored with a warning. An exception is that a duplicate referential constraint is not allowed if the definition of the constraint includes the PERIOD BUSINESS_TIME clause.

REFERENCES *parent-table-name* (*column-name*,...)

The table name that is specified after REFERENCES is the parent table for the referential constraint. *parent-table-name* must identify a table that exists at the current server⁴. The table name must not identify one of the following tables:

- A catalog table

- A directory table
- A declared global temporary table
- A history table
- An archive table

In the following discussion, let T2 denote an identified table and let T1 denote the table that you are creating (T1 and T2 cannot be the same table⁴).

T2 must have a unique index. The privilege set must include the ALTER or REFERENCES privilege on the parent table, or the REFERENCES privilege on the columns of the nominated parent key, including the columns of the BUSINESS_TIME period if the PERIOD BUSINESS_TIME clause is specified..

The parent key of the referential constraint is composed of the identified columns, or columns of the BUSINESS_TIME period if PERIOD BUSINESS_TIME is specified. Each *column-name* must be an unqualified name that identifies a column of T2. The same column must not be identified more than one time. If PERIOD BUSINESS_TIME is specified, the columns of the BUSINESS_TIME period must not be specified as part of the constraint. The identified column cannot be any of the following types of columns:

- a LOB column
- a ROWID column
- a DECFLOAT column
- an XML column
- a row change timestamp column
- a security label column

The list of column names in the parent key must match the list of column names in a primary key or unique key in the parent table T2. The column names must be specified in the same order as in the primary key or unique key. If PERIOD BUSINESS_TIME was specified for the primary key or unique key of the parent table T2, then PERIOD BUSINESS_TIME must also be specified for the foreign key clause for T1. If any of the referenced columns in T2 has a non-numeric data type, T2 and T1 must use the same encoding scheme, unless T2 is a Unicode table, and T1 is an EBCDIC table with Unicode key columns. In that case, for each character or graphic string column in T1, the CCSID must be the same as the corresponding column in T2.

If a list of column names is not specified, T2 must have a primary key. Omission of a list of column names is an implicit specification of the columns of the primary key for T2.

The specified foreign key must have the same number of columns as the parent key of T2 and, except for their names, default values, null attributes and check constraints, the description of the *n*th column of the foreign key must be identical to the description of the *n*th column of the nominated parent key. If the foreign key includes a column defined as a distinct type, the corresponding column of the nominated parent key must be the same distinct type. If a column of the foreign key has a field procedure, the corresponding column of the nominated parent key must have the same field procedure and an identical field description. A *field description* is a description of the encoded value as it is stored in the database for a column that has been defined to have an associated field procedure.

If PERIOD BUSINESS_TIME is specified in the FOREIGN KEY clause, then PERIOD BUSINESS_TIME must also be specified in the REFERENCES clause. If PERIOD BUSINESS_TIME is not specified in the FOREIGN KEY clause, then PERIOD BUSINESS_TIME must also not be specified in the REFERENCES clause.

If the PERIOD BUSINESS_TIME clause is specified, T2 must not be defined as part of a referential cycle. T1 and T2 must not be the same table, and T1 must not be a descendent, directly or indirectly, of another table that is a descendent of T2.

⁴ This restriction is relaxed when the statement is processed by the schema processor and the other table is created within the same CREATE SCHEMA.

The table space that contains T1 must be available to Db2. If T1 is populated, its table space is placed in a check pending status. A table in a segmented table space is populated if the table is not empty. A table in a table space that is not segmented is considered populated if the table space has ever contained any records.

The referential constraint that is specified by a FOREIGN KEY clause defines a relationship in which T2 is the parent and T1 is the dependent. A description of the referential constraint is recorded in the catalog.

PERIOD BUSINESS_TIME

Specifies that the BUSINESS_TIME period is considered part of the referential constraint. When PERIOD BUSINESS_TIME is specified, the values for the rest of the specified columns are unique with respect to the specified point of time.

PERIOD BUSINESS_TIME can be specified as the last key expression. If PERIOD BUSINESS_TIME is not the last key expression, an error is returned. If PERIOD BUSINESS_TIME is specified, the columns of the BUSINESS_TIME period must not be specified as part of the constraint.

When PERIOD BUSINESS_TIME is specified, the following columns are implicitly added to the end of the constraint:

- The end column of the BUSINESS_TIME period.
- The start column of the BUSINESS_TIME period.

The PERIOD BUSINESS_TIME clause specifies that there must not be a row in the child table for which the period of time represented by the BUSINESS_TIME period values for that row is not contained in the BUSINESS_TIME period of a corresponding row in the parent table. Furthermore, it is not necessary that there be exactly one corresponding row in the parent table where the BUSINESS_TIME period contains the BUSINESS_TIME period of the child row. As long as the BUSINESS_TIME period of a row in the child table is contained in the union of the BUSINESS_TIME periods of two or more contiguous matching rows in the parent table, the referential constraint is considered satisfied.

When the FOREIGN KEY clause specifies the PERIOD BUSINESS_TIME clause, the following conditions apply:

- The corresponding REFERENCES clause must also specify the PERIOD BUSINESS_TIME clause.
- A unique index with the BUSINESS_TIME WITHOUT OVERLAPS clause must be defined on the table. The table is marked as unavailable until the index is created.
- A unique index must be defined on the parent table with the BUSINESS_TIME WITHOUT OVERLAPS clause.

ON DELETE RESTRICT must be, implicitly or explicitly, specified when PERIOD BUSINESS_TIME is also specified.

ON DELETE

The delete rule of the relationship is determined by the ON DELETE clause. For more on the concepts used here, see [Referential constraints \(Introduction to Db2 for z/OS\)](#).

SET NULL must not be specified unless some column of the foreign key allows null values. The default value for the rule depends on the value of the CURRENT RULES special register when the CREATE TABLE statement is processed. If the value of the register is 'Db2', the delete rule defaults to RESTRICT; if the value is 'STD', the delete rule defaults to NO ACTION.

The delete rule applies when a row of T2 is the object of a DELETE or propagated delete operation and that row has dependents in T1. Let *p* denote such a row of T2. Then:

- If RESTRICT or NO ACTION is specified, an error occurs and no rows are deleted.
- If CASCADE is specified, the delete operation is propagated to the dependents of *p* in T1.
- If SET NULL is specified, each nullable column of the foreign key of each dependent of *p* in T1 is set to null.

Let T3 denote a table identified in another FOREIGN KEY clause (if any) of the CREATE TABLE statement. The delete rules of the relationships involving T2 and T3 must be the same and must not be SET NULL if:

- T2 and T3 are the same table.
- T2 is a descendent of T3 and the deletion of rows from T3 cascades to T2.
- T2 and T3 are both descendents of the same table and the deletion of rows from that table cascades to both T2 and T3.

ENFORCED or NOT ENFORCED

Indicates whether or not the referential constraint is enforced by Db2 during normal operations, such as insert, update, or delete.

ENFORCED

Specifies that the referential constraint is enforced by the Db2 during normal operations (such as insert, update, or delete) and that it is guaranteed to be correct. This is the default.

NOT ENFORCED

Specifies that the referential constraint is not enforced by Db2 during normal operations, such as insert, update, or delete. This option should only be used when the data that is stored in the table is verified to conform to the constraint by some other method than relying on the database manager.

ENABLE QUERY OPTIMIZATION

Specifies that the constraint can be used for query optimization. Db2 uses the information in query optimization using materialized query tables with the assumption that the constraint is correct. This is the default.

check-constraint

CONSTRAINT *constraint-name*

Names the check constraint. The constraint name must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

If *constraint-name* is not specified, a unique constraint name is derived from the name of the first column in the *check-condition* specified in the definition of the check constraint.

CHECK (*check-condition*)

Defines a check constraint. At any time, the *check-condition* must be true or unknown for every row of the table. A *check-condition* can evaluate to unknown if a column that is an operand of the predicate is null. A *check-condition* that evaluates to unknown does not violate the check constraint. A *check-condition* is a search condition, with the following restrictions:

- It can refer only to columns of table *table-name*.
- The columns cannot be the following types of columns:
 - LOB columns
 - ROWID columns
 - DECFLOAT columns
 - distinct type columns that are based on LOB, ROWID, and DECFLOAT data types
 - XML columns
 - security label columns
 - columns in an accelerator-only table
- It can be up to 3800 bytes long, not including redundant blanks.
- It must not contain any of the following:
 - Subselects
 - Built-in or user-defined functions
 - CAST specifications

- Cast functions other than those created when the distinct type was created
- Host variables
- Parameter markers
- Special registers
- Global variables
- Columns that include a field procedure
- CASE expressions
- ROW CHANGE expressions
- Row-value expressions
- DISTINCT predicates
- GX constants (hexadecimal graphic string constants)
- Sequence references
- OLAP specifications
- It must not result in CCSID conversion.
- If a *check-condition* refers to a LOB column (including a distinct type that is based on a LOB), the reference must occur within a LIKE predicate.
- The AND and OR logical operators can be used between predicates. The NOT logical operator cannot be used with the following predicates: NOT BETWEEN, NOT IN, NOT LIKE, or IS NOT NULL.
- The first operand of every predicate must be the column name of a column in the table.
- The second operand in the *check-condition* must be either a constant or the name of a column in the table.
 - If the second operand of a predicate is a constant, and if the constant is:
 - A floating-point number, then the column data type must be floating point.
 - A decimal number, then the column data type must be either floating point or decimal.
 - An integer number, then the column data type must not be a small integer.
 - A small integer number, then the column data type must be small integer.
 - A decimal constant, then its precision must not be larger than the precision of the column.
 - If the second operand of a predicate is a column, then both columns of the predicate must have:
 - The same data type.
 - Identical descriptions with the exception that the specification of the NOT NULL and DEFAULT clauses for the columns can be different, and that string columns with the same data type can have different length attributes.

LIKE

table-name or view-name

Specifies that the columns of the table have exactly the same name and description as the columns of the identified table or view.

The name that is specified after LIKE must identify a table or view that exists at the current server or a declared temporary table. A view cannot contain columns of length 0.

LIKE must not reference an accelerator-only table or be used with the IN ACCELERATOR clause.

The privilege set must implicitly or explicitly include the SELECT privilege on the identified table or view. If the identified table or view contains a column with a distinct type, the USAGE privilege on the distinct type is also needed. An identified table must not be an auxiliary table or a clone table. An identified view must not include a column that is an explicitly defined ROWID column (including a distinct type that is based on a ROWID), an identity column, or a row change timestamp column.

The use of LIKE is an implicit definition of n columns, where n is the number of columns in the identified table (including implicitly hidden columns) or view. A column of the new table that corresponds to an implicitly hidden column in the existing table will also be defined as implicitly hidden. The implicit definition includes all attributes of the n columns as they are described in SYSCOLUMNS with the following exceptions:

- When a table is identified in the LIKE clause and a column in the table has a field procedure, the corresponding column of the new table has the same field procedure and the field description. However, the field procedure is not invoked during the execution of the CREATE TABLE statement. When a view is identified in the LIKE clause, none of the columns of the new table will have a field procedure. This is true even in the case that a column of a base table underlying the view has a field procedure defined.
- When a table is identified in the LIKE clause and a column in the table is an identity column, the corresponding column of the new table inherits only the data type of the identity column; none of the identity attributes of the column are inherited unless the INCLUDING IDENTITY clause is specified.
- When a table is identified in the LIKE clause and a column in the table is a security label column, the corresponding column of the new table inherits only the data type of the security label column; none of the security label attributes of the column are inherited.
- When a table that contains a ROWID column is identified in the LIKE clause, the corresponding column of the new table inherits the ROWID column, regardless of whether the column has the IMPLICITLY HIDDEN attribute.
- When a table is identified in the LIKE clause and the table contains a row change timestamp column, a transaction-start-ID column, a row-begin column, or a row-end column, the corresponding column of the new table inherits only the data type of the original column. The new column is not considered a generated column.
- When a table is identified in the LIKE clause and a column in the table is a generated expression column, the corresponding column of the new table inherits only the data type of the original column. The new column is not considered a generated column.
- When a table is identified in the LIKE clause and the table contains an inline LOB column, the corresponding columns of the new table will inherit the inline attribute if the table is in an universal table space. Otherwise, the inline attribute of the table identified in the LIKE clause is ignored.
- When a view is identified in the LIKE clause, the default value that is associated with the corresponding column of the new table depends on the column of the underlying base table for the view. If the column of the base table does not have a default, the new column does not have a default. If the column of the base table has a default, the default of the new column is:
 - Null if the column of the underlying base table allows nulls.
 - The default for the data type of the underlying base table if the underlying base table does not allow nulls.

The above defaults are chosen regardless of the current default of the base table column. The existence of an INSTEAD OF trigger does not affect the inheritance of default values.

- When a table that uses table-controlled partitioning is identified in the LIKE clause, the new table does not inherit partitioning scheme of that table. You can add these partition boundaries by specifying ALTER TABLE with the ADD PARTITION BY RANGE clause.
- The CCSID of the column is determined by the implicit or explicit CCSID clause. For more information, see the CCSID clause.

An exception is a Unicode column in an EBCDIC table, which inherits the CCSID of the column in the existing table.

- When a table is identified in the LIKE clause and the table includes a period definition, the new table does not inherit the period. definition.
- When the table that is identified in the LIKE clause is a system-period temporal table, the new table is not a system-period temporal table.

- When the table that is identified in the LIKE clause has row access controls or column access controls activated, the new table does not inherit the row access controls or the column access controls.

The implicit definition does not include any other attributes of the identified table or view. For example, the new table does not have a primary key or foreign key. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

copy-options

copy-options

Specifies whether identity column attributes, row change timestamp attributes, and column defaults are inherited from the definition of the source of the result table.

EXCLUDING IDENTITY COLUMN ATTRIBUTES or INCLUDING IDENTITY COLUMN ATTRIBUTES

Specifies whether identity column attributes are inherited from the definition of the source of the result table.

EXCLUDING IDENTITY COLUMN ATTRIBUTES

Specifies that identity column attributes are not inherited from the definition of the source of the result table. This is the default.

INCLUDING IDENTITY COLUMN ATTRIBUTES

Specifies that, if available, identity column attributes (such as START WITH, INCREMENT BY, and CACHE values) are inherited from the definition of the source table. These attributes can be inherited if the element of the corresponding column in the table, view, or *fullselect* is the name of a column of a table or the name of a column of a view that directly or indirectly maps to the column name of a base table with the identity attribute. In other cases, the columns of the new temporary table do not inherit the identity attributes. The columns of the new table do not inherit the identity attributes in the following cases:

- The select list of the *fullselect* includes multiple instances of an identity column name (that is, selecting the same column more than one time).
- The select list of the *fullselect* includes multiple identity columns (that is, it involves a join).
- The identity column is included in an expression in the select list.
- The *fullselect* includes a set operation.

EXCLUDING ROW CHANGE TIMESTAMP COLUMN ATTRIBUTES or INCLUDING ROW CHANGE TIMESTAMP COLUMN ATTRIBUTES

Specifies whether row change timestamp column attributes are inherited from the definition of the source of the result table.

EXCLUDING ROW CHANGE TIMESTAMP COLUMN ATTRIBUTES

Specifies that row change timestamp column attributes are not inherited from the source result table definition. This is the default.

INCLUDING ROW CHANGE TIMESTAMP COLUMN ATTRIBUTES

Specifies that, if available, row change timestamp column attributes are inherited from the definition of the source table. These attributes can be inherited if the element of the corresponding column in the table, view, or *fullselect* is the name of a column of a table or the name of a column of a view that directly or indirectly maps to the column name of a base table defined as a row change timestamp column. In other cases, the columns of the new temporary table do not inherit the row change timestamp column attributes. The columns of the new table do not inherit the row change timestamp attributes in the following cases:

- The select list of the *fullselect* includes multiple instances of a row change timestamp column name (that is, selecting the same column more than one time).
- The select list of the *fullselect* includes multiple row change timestamp column names (that is, it involves a join).
- The row change timestamp column is included in an expression in the select list.

- The *fullselect* includes a set operation (such as union).

EXCLUDING COLUMN DEFAULTS, INCLUDING COLUMN DEFAULTS, or USING TYPE DEFAULTS

Specifies whether column defaults are inherited from the source result table definition.

EXCLUDING COLUMN DEFAULTS, INCLUDING COLUMN DEFAULTS, and USING TYPE DEFAULTS must not be specified if the LIKE clause is specified.

EXCLUDING COLUMN DEFAULTS

Specifies that the column defaults are not inherited from the definition of the source table. The default values of the column of the new table are either null or there are no default values. If the column can be null, the default is the null value. If the column cannot be null, there is no default value, and an error occurs if a value is not provided for a column on an insert or update operation, or LOAD for the new table.

INCLUDING COLUMN DEFAULTS

Specifies that column defaults for each updatable column of the definition of the source table are inherited. Columns that are not updatable do not have a default defined in the corresponding column of the created table. The existence of an INSTEAD OF trigger for a view does not affect the inheritance of default values.

USING TYPE DEFAULTS

Specifies that the default values for the table depend on data type of the columns that result from *fullselect*, as follows:

Data type

Default value

Numeric

0

Fixed-length character string

Blanks

Fixed-length graphic string

Blanks

Fixed-length binary string

Hexadecimal zeros

Varying-length string

A string of length 0

Fixed-length char or fixed-length graphic

A string of blanks

Fixed-length binary

Hexadecimal zeros

Date

CURRENT DATE

Time

CURRENT TIME

Timestamp(*integer*) without time zone

CURRENT TIMESTAMP(*p*) WITHOUT TIME ZONE where *p* is the corresponding timestamp precision.

Timestamp(*integer*) with time zone

CURRENT TIMESTAMP(*p*) WITH TIME ZONE where *p* is the corresponding timestamp precision.

as-result-table

as-result-table

Specifies that the column definitions of the table are based on the result of the *fullselect*.

column-name

Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the *fullselect*. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the *fullselect*.

A list of column names must be specified if the result table of the *fullselect* has duplicate column names or an unnamed column. An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause.

AS (*fullselect*)

Specifies that the table definition is based on the column definitions from the result of the *fullselect*. The use of AS (*fullselect*) is an implicit definition of *n* columns for the table, where *n* is the number of columns that would result from the *fullselect*. The columns of the new table are defined by the columns that result from the *fullselect*. Every select list element must have a unique name. The AS clause can be used in the *select-clause* to provide unique names.

The implicit definition includes the column name, data type, length, precision, scale, and nullability characteristic of each of the result columns of *fullselect*. The length of each column must not be 0. Other column attributes, such as DEFAULT and IDENTITY, are not inherited from the *fullselect*. A column of the new table that corresponds to an implicitly hidden column of a base table referenced in the *fullselect* is not considered hidden in the new table. The generated column attributes are not inherited from the *fullselect*. That is, a new column of the table is not considered as a generated column. A FIELDPROC is inherited for a column if the corresponding select item of the *fullselect* is a column that can be mapped to a column of a base table or a view. The new table contains a security label column if only one table in the *fullselect* contains a security label column and the primary authorization ID of the statement has a valid security label.

The outermost SELECT list of the *fullselect* must not reference data that is encoded with different encoding schemes. An exception is that the outermost SELECT list can contain a mixture of EBCDIC and Unicode columns. In this case, the new table is an EBCDIC table with one or more Unicode columns.

The implicit definition does not include any other attributes of the identified table or view. For example, the new table does not have a primary key or foreign key. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

If IN ACCELERATOR is specified, AS (*fullselect*) cannot be specified.

The owner of the table being created must have the SELECT privilege on the tables or views referenced in the *fullselect*, or the privilege set must include SYSADM or DBADM authority for the database in which the tables of the *fullselect* reside. Having SELECT privilege means that the owner has at least one of the following authorizations.

- Ownership of the tables or views referenced in the *fullselect*
- The SELECT privilege on the tables and views referenced in the *fullselect*
- SYSADM authority
- DBADM authority for the database in which the tables of the *fullselect* reside

Additional privileges might be necessary for accessing other objects that are referenced in the *fullselect*.

The *fullselect* must not:

- Result in a column having a ROWID, BLOB, CLOB, DBCLOB, or XML data type or a distinct type based on these data types.
- Include multiple security label columns.
- Include a PREVIOUS VALUE or a NEXT VALUE expression.
- Refer to host variables or include parameter markers.
- Include an SQL data change statement in the FROM clause.

- In the outermost SELECT, reference a combination of ASCII and EBCDIC data, or a combination of ASCII and Unicode data.
- Result in a column that is an array.
- Reference a remote object.
- Reference an accelerator-only table.

WITH NO DATA

Specifies that the query is used only to define the attributes of the new table. The table is not populated using the results of the *fullselect* and the REFRESH TABLE statement cannot be used.

If the tables that are specified in the *fullselect* use row access controls or column access controls, the row access controls and the column access controls are not defined for the new table.

materialized-query-definition

materialized-query-definition

Specifies that the column definitions of the materialized query table are based on the result of a *fullselect*. If *materialized-query-table-options* are specified, the REFRESH TABLE statement can be used to populate the table with the results of the *fullselect*.

column-name

Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the *fullselect*. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the *fullselect*.

A list of column names must be specified if the result table of the *fullselect* has duplicate column names or an unnamed column. An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS (*fullselect*)

Specifies that the table definition is based on the column definitions from the result of the *fullselect*. The use of AS (*fullselect*) is an implicit definition of *n* columns for the table, where *n* is the number of columns that would result from the *fullselect*. The columns of the new table are defined by the columns that result from the *fullselect*. Every select list element must have a unique name. The AS clause can be used in the *select-clause* to provide unique names.

The implicit definition includes the column name, data type, length, precision, scale, and nullability characteristic of each of the result columns of *fullselect*. The length of each column must not be a 0. Other column attributes, such as DEFAULT, IDENTITY, and unique constraints, are not inherited from the *fullselect*. A column of the new table that corresponds to an implicitly hidden column of a base table referenced in the *fullselect* is not considered hidden in the new table. The generated column attributes are not inherited from the *fullselect*. That is, the new column of the materialized query table is not considered as a generated column. A FIELDPROC is inherited for a column if the corresponding select item of the *fullselect* is a column that can be directly mapped to a column of a base table or a view in the FROM clause of the *fullselect*. The materialized query table contains a security label column if only one table in the *fullselect* contains a security label column and the primary authorization ID of the statement has a valid security label.

The outermost SELECT list of the *fullselect* can include result columns that are defined as EBCDIC columns and result columns that are defined as Unicode columns. In this case, the materialized query table is an EBCDIC table with one or more Unicode columns.

Authorization for creating materialized query tables

The owner of the table being created must have the SELECT privilege on the tables or views referenced in the *fullselect*, or the privilege set must include SYSADM or DBADM authority for the database in which the tables of the *fullselect* reside. Having SELECT privilege means that the owner has at least one of the following authorizations:

- Ownership of the tables or views referenced in the *fullselect*

- The SELECT privilege on the tables and views referenced in the fullselect
- SYSADM authority
- DBADM authority for the database in which the tables of the fullselect reside

Additional privileges might be necessary for accessing other objects that are referenced in the fullselect.

The rules for establishing the qualifiers for names used in the fullselect are the same as the rules used to establish the qualifiers for *table-name*.

The following restrictions apply when creating materialized query tables. When *fullselect* does not satisfy the restrictions, an error occurs:

- The length of each result column of the fullselect must not be 0.
- The fullselect cannot contain a column of a LOB or XML data type.
- No more than one table in the fullselect can contain a security label column.
- The fullselect must not contain a period specification.
- The outermost SELECT list must not reference data that is encoded with a combination of ASCII and EBCDIC CCSID sets, or a combination of ASCII and Unicode CCSID sets.
- The object that is specified in the FROM clause of the fullselect cannot be a view with columns of length 0.
- The fullselect cannot contain a reference to a created global temporary table, a declared global temporary table, an accelerator-only table, a directory table, or another materialized query table.
- If IN ACCELERATOR is specified, *materialized-query-definition* cannot be specified.
- The fullselect cannot directly or indirectly reference a base table that has been activated for the row or column access control or a base table for which a row permission or a column mask has been defined.
- The fullselect must not refer to host variables or include parameter markers.
- The fullselect must not refer to global variables.
- The fullselect must not include the following built-in functions: LISTAGG, PERCENTILE_CONT, or PERCENTILE_DISC.
- The fullselect must not include the following built-in functions: AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.

Additional restrictions when ENABLE QUERY OPTIMIZATION is in effect:

- The fullselect must be a subselect.
- The subselect cannot include the following:
 - A special register
 - A scalar fullselect
 - A row change timestamp column
 - A ROW CHANGE expression
 - An expression for which implicit time zone values apply (for example, cast a timestamp to a timestamp with time zone)
 - The RAND built-in function
 - The RID built-in function
 - A user-defined scalar or table function that is not deterministic or that has external actions
 - Any predicates that include a subquery
 - A *row-value-expression* in a predicate
 - A join using the INNER JOIN syntax, or an outer join

- A lateral correlation
- A nested table expression or view that requires temporary materialization
- A direct or indirect reference to a table that uses activated row or column access controls, or a table for which row or column access controls have been defined.
- A FETCH FIRST clause
- A reference to a global variable
- A collection-derived table (UNNEST)
- A GROUPING SETS or *super-groups* clause
- If a table with a security label is referenced, the security label column must be referenced in the outer select list of the subselect.
- If the subselect references a view, the fullselect in the view definition must satisfy all other restrictions.

refreshable-table-options

Specifies the options for a refreshable materialized query table. The ORDER BY clause is allowed, but it is used only by REFRESH. The ORDER BY clause can improve the locality of reference of data in the materialized query table.

DATA INITIALLY DEFERRED

Specifies that the data is not inserted into the materialized query table when it is created. Use the REFRESH TABLE statement to populate the materialized query table, or use the INSERT statement to insert data into a user-maintained materialized query table.

REFRESH DEFERRED

Specifies that the data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time when the REFRESH TABLE statement is processed or when it was last updated for a user-maintained materialized query table.

MAINTAINED BY SYSTEM or MAINTAINED BY USER

Specifies how the data in the materialized query table is maintained.

MAINTAINED BY SYSTEM

Specifies that the materialized query table is maintained by the system. Only the REFRESH statement is allowed on the table. This is the default.

MAINTAINED BY USER

Specifies that the materialized query table is maintained by the user, who can use the LOAD utility, an SQL data change statement, a SELECT from data change statement, or REFRESH TABLE SQL statements on the table.

ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION

Specifies whether this materialized query table can be used for optimization.

ENABLE QUERY OPTIMIZATION

Specifies that the materialized query table can be used for query optimization. If the fullselect specified does not satisfy the restrictions for query optimization, an error occurs.

ENABLE QUERY OPTIMIZATION is the default.

The fullselect must not contain a period specification.

DISABLE QUERY OPTIMIZATION

Specifies that the materialized query table cannot be used for query optimization. The table can still be queried directly.

IN

IN *database-name.table-space-name* or IN DATABASE *database-name*

Identifies the database and table space in which the table is created. Both forms are optional.

If you specify *database-name* and *table-space-name*, the database must be described in the catalog on the current server. The database must not be DSNDB06 or a work file database. The table space must belong to the database that you specify.

If you specify *database-name* but not *table-space-name*, a table space is implicitly created in *database-name*. The name of the table space is derived from the name of the table. The buffer pool that is used is the default buffer pool for user data that is specified on installation panel DSN TIP1.

If you specify a table space but not a database, the database that contains the table space is used.

If you do not specify the IN clause, a database is implicitly created with the name DSNxxxxx, where xxxxx is a five-digit number. A table space is also implicitly created.

If you specify *table-space-name*, the table space cannot be one of the following table spaces:

- A table space that was created implicitly
- A partitioned table space that already contains a table
- A LOB table space
- An XML table space
- A non-UTS table space

If you specify a partitioned table space, you cannot load or use the table until its partitioned scheme is created.

You cannot specify a name in the format of an implicitly created database name, which is DSNxxxxx, where xxxxx is a five-digit number..

If you specify *table-space-name*, but you do not specify *database-name*, or you do not specify the IN clause, users who have the authority to create table spaces or tables in database DSNDB04 have authority to create tables and table spaces in the implicitly created database.

If you do not specify *table-space-name*, the privilege set must have: SYSADM or SYSCTRL authority; DBADM, DBCTRL, or DBMAINT authority for the database; or the CREATETS privilege for the database. You must also have the USE privilege for the default buffer pool in the database and default storage group.

For implicitly created table spaces, Db2 selects the buffer pool as described in [Implicitly defined table spaces](#) (Db2 Administration Guide).

IN ACCELERATOR *accelerator-name*

Specifies that the table is an accelerator-only table. *accelerator-name* identifies the accelerator in which the table will be defined.

You can specify an alias (logical name) for *accelerator-name*. For more information, see [Using an alias for an accelerator](#) (Db2 Performance). To create a high availability accelerator-only table, specify a location alias that represents multiple accelerators to define the table in all accelerators that are associated with the location alias.

If you specify an accelerator-only table, the table and the data of the table exists only in the accelerator, not in Db2. However, the table and column definition of the accelerator-only table are contained in Db2 catalog tables.

partitioning-clause block

PARTITION BY SIZE or PARTITION BY RANGE

Specifies the partitioning scheme for the table. For more information, see [Partitioning data in Db2 tables](#) (Db2 Administration Guide).

PARTITION BY SIZE

Specifies that the table is created in a partition-by-growth table space. If the IN clause specifies a *table-space-name*, it must identify a partition-by-growth table space. If the IN clause does not specify an existing table space name and the PARTITION BY clause is not specified, PARTITION BY SIZE is the default.

If IN ACCELERATOR is specified, PARTITION BY SIZE must not be specified.

EVERY *integer* G

Specifies that the table is to be partitioned by growth, every *integer* G bytes. *integer* must not be greater than 256. If the IN clause identifies a table space, *integer* must be the same as the DSSIZE value that is in effect for the table space that will contain the table.

PARTITION BY RANGE

Specifies the range partitioning scheme for the table (the columns that are used to partition the data). When this clause is specified, the table space is complete, and it is not necessary to create a partitioned index on the table. If this clause is used, the ENDING AT clause cannot be used on a subsequent CREATE INDEX statement for this table.

PARTITION BY RANGE must not be specified for a table that is created in a partition-by-growth table space. If IN ACCELERATOR is specified, PARTITION BY RANGE must not be specified.

partition-expression

Specifies the key data over which the range is defined to determine the target data partition of the data.

column-name

Specifies the columns of the key. Each *column-name* must identify a column of the table. Do not specify more than 64 columns or the same column more than one time. The sum of length attributes of the columns must not be greater than 255 - *n*, where *n* is the number of columns that can contain null values. Do not specify a qualified column name.

A timestamp with time zone column (or a column with a distinct type that is based on the timestamp with time zone data type) can only be specified as the last column in a partitioning key.

Do not specify a column for *column-name* if the column is defined as follows:

- a LOB column (or a column with a distinct type that is based on a LOB data type)
- a BINARY column (or a column with a distinct type that is based on a BINARY data type)
- a VARBINARY column (or a column with a distinct type that is based on a VARBINARY data type)
- a DECFLOAT column (or a column with a distinct type that is based on a DECFLOAT data type)
- an XML column

All character and graphic string columns in the key must be defined with the same encoding scheme.

NULLS LAST

Specifies that null values are treated as positive infinity for purposes of comparison.

ASC

Puts the entries in ascending order by the column. ASC is the default.

DESC

Puts the entries in descending order by the column.

partition-element

Specifies ranges for a data partitioning key and the table space where rows of the table in the range will be stored.

PARTITION *integer*

integer is the physical number of a partition in the table space. A PARTITION clause must be specified for every partition of the table space. In this context, highest means highest in the sorting sequences of the columns. In a column defined as ascending (ASC), highest and lowest have their usual meanings. In a column defined as descending (DESC), the lowest actual value is highest in the sorting sequence.

ENDING AT (*constant*, MAXVALUE, or MINVALUE, ...)

Defines the limit key for a partition boundary. Specify at least one value (*constant*, MAXVALUE, or MINVALUE) after ENDING AT in each PARTITION clause. You can use as many values as there are columns in the key. The concatenation of all values is the highest value of the key for ascending and the lowest for descending.

constant

Specifies a constant value with a data type that must conform to the rules for assigning that value to the column. If a string constant is longer or shorter than required by the length attribute of its column, the constant is either truncated or padded on the right to the required length. If the column is ascending, the padding character is X'FF'. If the column is descending, the padding character is X'00'. The precision and scale of a decimal constant must not be greater than the precision and scale of its corresponding column. A hexadecimal string constant (GX) cannot be specified.

MAXVALUE

Specifies a value greater than the maximum value for the limit key of a partition boundary (that is, all X'FF' regardless of whether the column is ascending or descending). If all of the columns in the partitioning key are ascending, a constant or the MINVALUE clause cannot be specified following MAXVALUE. After MAXVALUE is specified, all subsequent columns must be MAXVALUE.

MINVALUE

Specifies a value that is smaller than the minimum value for the limit key of a partition boundary (that is, all X'00' regardless of whether the column is ascending or descending). If all of the columns in the partitioning key are descending, a constant or the MAXVALUE clause cannot be specified following MINVALUE. After MINVALUE is specified, all subsequent columns must be MINVALUE.

The key values are subject to the following rules:

- The first value corresponds to the first column of the key, the second value to the second column, and so on. Using fewer values than there are columns in the key has the same effect as using the highest or lowest values for the omitted columns, depending on whether they are ascending or descending.
- The highest value of the key in any partition must be lower than the highest value of the key in the next partition for ascending cases.
- The values specified for the last partition are enforced. The value specified for the last partition is the highest value of the key that can be placed in the table. Any key values greater than the value specified for the last partition are out of range.
- If the concatenation of all the values exceeds 255 bytes, only the first 255 bytes are considered.
- If a key includes a ROWID column or a column with a distinct type that is based on a ROWID data type, 17 bytes of the constant that is specified for the corresponding ROWID column are considered.
- If a null value is specified for the partitioning key and the key is ascending, an error is returned unless MAXVALUE is specified. If the key is descending, an error is returned unless MINVALUE is specified.

partition-hash-space

See [partition-hash-space](#).

INCLUSIVE

Specifies that the specified range values are included in the data partition.

organization-clause

See [organization-clause](#).

Other options

EDITPROC *program-name*

Identifies the user-written code that implements the edit procedure for the table. The edit procedure must exist at the current server. The procedure is invoked during the execution of an SQL data change statement or LOAD and all row retrieval operations on the table.

An edit routine receives an entire table row, and can transform that row in any way. Also, it receives a transformed row and must change the row back to its original form.

For information on writing an EDITPROC exit routine, see [Edit procedures \(Db2 Administration Guide\)](#).

WITH ROW ATTRIBUTES

Specifies that the edit procedure parameter list contains an address for the description of a row. WITH ROW ATTRIBUTES must not be specified for a table with an identity, LOB, XML, ROWID, or SECURITY LABEL column. WITH ROW ATTRIBUTES is the default. When WITH ROW ATTRIBUTES is specified, the column names in the table must not be longer than 18 EBCDIC SBCS characters in length.

WITHOUT ROW ATTRIBUTES

Specifies that the description of the row is not provided to the edit procedure. On entry to the edit procedure, the address for the row description in the parameter list contains a value of zero.

VALIDPROC *program-name*

Designates *program-name* as the validation exit routine for the table. Writing a validation exit routine is described in [Validation routines \(Db2 Administration Guide\)](#).

The validation routine can inhibit a load, insert, update, or delete operation on any row of the table: before the operation takes place, the procedure is passed the row. The values that are represented by any LOB or XML columns in the table are not passed to the validation routine. On an insert or update operation, if the table has a security label column and the user does not have write-down privilege, the user's security label value is passed to the validation routine as the value of the column. After examining the row, the procedure returns a value that indicates whether the operation should proceed. A typical use is to impose restrictions on the values that can appear in various columns. If IN ACCELERATOR is specified, VALIDPROC must not be specified.

A table can have only one validation procedure at a time. In an ALTER TABLE statement, you can designate a replacement procedure or discontinue the use of a validation procedure.

If you omit VALIDPROC, the table has no validation routine.

AUDIT

Identifies the types of access to this table that causes auditing to be performed. For information about audit trace classes, see [Types of Db2 traces \(Db2 Performance\)](#) and [-START TRACE \(Db2\) \(Db2 Commands\)](#).

If a materialized query table is refreshed with the REFRESH TABLE statement, the auditing also occurs during the REFRESH TABLE operation. AUDIT works as usual for LOAD and SQL data change operations on a user-maintained materialized query table.

NONE

Specifies that no auditing is to be done when this table is accessed. This is the default.

CHANGES

Specifies that auditing is to be done when the table is accessed during the first insert, update, or delete operation. However, the auditing is done only if the appropriate audit trace class is active.

ALL

Specifies that auditing is to be done when the table is accessed during the first operation of any kind performed by a utility or application process. However, the auditing is done only if the appropriate audit trace class is active and the access is not performed with COPY, RECOVER, REPAIR, LOAD with a dummy input data set, or any stand-alone utility.

If the table is subsequently altered with an ALTER TABLE statement, the ALTER TABLE statement is audited for successful and failed attempts in the following cases, if the appropriate audit trace class is active:

- AUDIT attribute is changed to NONE, CHANGES, or ALL on an audited or non-audited table.
- AUDIT CHANGES or AUDIT ALL is in effect.

If IN ACCELERATOR is specified, AUDIT NONE, CHANGES, and ALL must not be specified.

OBID integer

Identifies the OBID to be used for this table. An OBID is the identifier for an object's internal descriptor. The integer must be greater than 1 and must not identify an existing or previously used OBID of the database. If you omit OBID, Db2 generates a value.

The following statement retrieves the value of OBID:

```
SELECT OBID
FROM SYSIBM.SYSTABLES
WHERE CREATOR = 'ccc' AND NAME = 'nnn';
```

Here, *nnn* is the table name and *ccc* is the creator of the table.

DATA CAPTURE

Specifies whether the logging of the following actions on the table includes additional information to support data replication processing:

- SQL data change operations
- Adding columns (using the ADD COLUMN clause)
- Changing columns (using the ALTER COLUMN clause)

For more information, see [Altering a table to capture changed data \(Db2 Administration Guide\)](#).

If a materialized query table is refreshed with the REFRESH TABLE statement, the logging of the augmented information occurs during the REFRESH TABLE operation. DATA CAPTURE works as usual for insert, update, and delete operations on a user-maintained materialized query table.

A table with data that is stored only in an accelerator-only table cannot be defined with this attribute.

NONE

Do not record additional information to the log. This is the default.

CHANGES

Write additional data about SQL updates to the log. Information about the values that are represented by any LOB or XML columns is not available. Do not specify DATA CAPTURE CHANGES for tables that reside in table spaces that specify NOT LOGGED.

WITH RESTRICT ON DROP

Indicates that the table can be dropped only by using REPAIR DBD DROP. In addition, the database and table space that contain the table can be dropped only by using REPAIR DBD DROP.

The WITH RESTRICT ON DROP clause can be removed using the ALTER TABLE statement with the DROP RESTRICT ON DROP clause. After the WITH RESTRICT ON DROP clause is removed from the definition of the table, the table, the database, and the containing table space can be dropped using the DROP statement.

CCSID encoding-scheme

Specifies the encoding scheme for string data stored in the table. If the IN clause is specified with a table space, the value must agree with the encoding scheme that is already in use for the specified table space. The specific CCSIDs for SBCS, mixed, and graphic data are determined by the table space or database specified in the IN clause. If the IN clause is not specified, the value specified is used for the table being created as well as for the table space that Db2 implicitly creates. The specific CCSIDs for SBCS, mixed, and graphic data are determined by the default CCSIDs for the server for the specified encoding scheme. The valid values are ASCII, EBCDIC, and UNICODE.

If IN ACCELERATOR is specified, a Unicode column cannot be defined in an EBCDIC table and a column cannot be defined as ASCII mixed or graphic. *IBM Db2 Analytics Accelerator for z/OS: Stored Procedures Reference* contains a complete description of encoding schemes allowed in an accelerator.

If the CCSID clause is not specified, the encoding scheme for the table depends on the IN clause:

- If the IN clause is specified, the encoding scheme already in use for the table space or database specified in the IN clause is used.
- If the IN clause is not specified, the encoding scheme of the new table is the same as the scheme for the table that is specified in the LIKE clause.

If CCSID EBCDIC is explicitly or implicitly specified, and any columns in the table are defined with the CCSID 1208 or CCSID 1200 clause, CCSID EBCDIC represents the default encoding scheme for character or graphic columns that do not include the CCSID 1208 or CCSID 1200 clause.

If the CCSID clause is specified for a materialized query table:

- If the encoding scheme in the CCSID clause is ASCII or Unicode, or if the encoding scheme in the CCSID clause is EBCDIC and the result table of the fullselect contains no Unicode columns, the encoding scheme specified in the clause must be the same as the scheme for the result CCSID of the fullselect. The CCSID must also be the same as the CCSID of the table space for the table being created.
- If the encoding scheme in the CCSID clause is EBCDIC, and the result table of the fullselect contains Unicode columns, the encoding scheme of the table space for the table must be EBCDIC.

VOLATILE or NOT VOLATILE

Specifies how Db2 chooses to access the table.

VOLATILE

Specifies that Db2 uses index access to the table whenever possible for SQL operations. However, be aware that list prefetch and certain other optimization techniques might be disabled when VOLATILE is used.

One instance in which you might want to use VOLATILE is for a table whose size can vary greatly. If statistics are taken when the table is empty or has only a few rows, those statistics might not be appropriate when the table has many rows.

Another instance in which you might want to use VOLATILE is for a table that contains groups of rows, as defined by the primary key on the table. All but the last column of the primary key of such a table indicate the group to which a given row belongs. The last column of the primary key is the sequence number indicating the order in which the rows are to be read from the group. VOLATILE maximizes concurrency of operations on rows within each group, since rows are usually accessed in the same order for each operation. If IN ACCELERATOR is specified, VOLATILE must not be specified. For this usage, the primary index must be the only index that is defined on the table, and list prefetch is disabled to ensure the sequence in which the rows are locked.

NOT VOLATILE

Specifies that SQL access to this table should be based on the current statistics. NOT VOLATILE is the default.

CARDINALITY

An optional keyword that currently has no effect, but that is provided for Db2 family compatibility.

LOGGED or NOT LOGGED

Specifies whether changes that are made to the data in an implicitly created table space are recorded in the log. This parameter applies to an implicitly created table space and to all indexes of this table. XML table spaces and indexes associated with the XML table spaces inherit the logging attribute from the associated base table space. Auxiliary indexes also inherit the logging attribute from the associated base table space.

Do not specify LOGGED or NOT LOGGED if the table space name is specified by using the IN *table-space-name* clause or if the IN ACCELERATOR clause is specified.

LOGGED

Specifies that changes that are made to the data in an implicitly created table space are recorded in the log.

LOGGED is the default.

NOT LOGGED

Specifies that changes that are made to data in an implicitly created table space are not recorded in the log.

NOT LOGGED prevents undo and redo information from being recorded in the log. However, control information for an implicitly created table space will continue to be recorded in the log.

COMPRESS YES or COMPRESS NO

Specifies whether data compression applies to the rows of any implicitly created table space. The IMPTSCMP subsystem parameter specifies the default value. See USE DATA COMPRESSION field (IMPTSCMP subsystem parameter) (Db2 Installation and Migration).

If the IN *table-space-name* clause or the IN ACCELERATOR clause is specified, COMPRESS must not be specified.

YES

Specifies that data compression applies to the rows of the implicitly created table space. The rows are not compressed until the LOAD or REORG utility is run on the table in the implicitly created table space, or the total row data size reaches the compression data threshold while an insert operation is performed.

If a keyword for the compression algorithm is not specified, the default compression algorithm is used. The data compression algorithm is determined by the TS_COMPRESSION_TYPE subsystem parameter.

If a keyword for the compression algorithm is specified:

- LOB table spaces that are implicitly created for LOB columns in this table are defined as if COMPRESS YES is specified without a compression algorithm. LOB compression is managed by zEnterprise data compression (zEDC) hardware if available.
- XML table spaces that are implicitly created for XML columns in this table inherit the COMPRESS attribute.

FIXEDLENGTH

Specifies the fixed-length data compression algorithm.

HUFFMAN

Specifies the Huffman data compression algorithm. See Using Huffman compression to compress your data (Db2 Performance) for requirements to enable Huffman compression.

NO

Specifies that data compression is not used for the rows of the implicitly created table space. Inserted and updated rows are not subject to data compression.

APPEND NO or APPEND YES

Specifies whether append processing is used for the table. The APPEND clause must not be specified for a table that is created in a work file table space.

NO

Specifies that append processing is not used for the table. For insert and LOAD operations, Db2 will attempt to place data rows in a well clustered manner with respect to the value in the row's cluster key column.

NO is the default.

YES

Specifies that data rows are to be placed into the table by disregarding the clustering during insert and LOAD operations.

DSSIZE *integer G*

Specifies the maximum size for an implicitly created partition-by-growth or partition-by-range table space. This value is only applied to an implicitly created base table space, not to any associated implicitly created XML or LOB table spaces.

Do not specify DSSIZE *integer G* if any of the following conditions are true:

- The table space name is specified by using the IN *table-space-name* clause.
- The PARTITION BY clause includes the EVERY *integer-constant G* clause.
- The statement contains an accelerator-only table.

The IMPDSSIZE subsystem parameter specifies the default value. See [IMPDSSIZE in macro DSN6SYSP \(Db2 Installation and Migration\)](#).

For more detailed information about the DSSIZE clause, refer to [CREATE TABLESPACE \(Db2 SQL\)](#).

BUFFERPOOL *bpname*

Specifies the buffer pool to use for an implicitly created table space and determines the page size of the table space. For 4KB, 8KB, 16KB and 32KB page buffer pools, the page sizes are 4 KB, 8 KB, 16 KB, and 32 KB, respectively.

bpname must identify an activated buffer pool. The privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege on the buffer pool.

Do not specify BUFFERPOOL *bpname* if the table space name is specified by using the IN *table-space-name* clause or the IN ACCELERATOR clause is specified.

If you do not specify the BUFFERPOOL clause, Db2 selects the buffer pool as described in [Implicitly defined table spaces \(Db2 Administration Guide\)](#).

Refer to [Naming conventions \(Db2 SQL\)](#) for more information about *bpname*.

MEMBER CLUSTER

Specifies that data that is inserted by an insert operation is not clustered by the implicit clustering index (the first index) or the explicit clustering index. Db2 places the data in an implicitly created table space based on available space.

Do not specify MEMBER CLUSTER if the table space name is specified by using the IN *table-space-name* clause or if IN ACCELERATOR clause is specified.

TRACKMOD YES or TRACKMOD NO

Specifies whether Db2 tracks modified pages in the space map pages of an implicitly created table space. The IMPTKMOD subsystem parameter specifies the default value. See [IMPTKMOD in macro DSN6SYSP \(Db2 Installation and Migration\)](#).

Do not specify TRACKMOD YES or TRACKMOD NO if the table space name is specified by using the IN *table-space-name* clause or if using the IN ACCELERATOR clause.

TRACKMOD YES

Changed pages are tracked in the space map pages to help improve performance of incremental image copies.

TRACKMOD NO

Changed pages are not tracked in the space map pages. Db2 uses the LRSN value in each page to determine whether a page has been changed.

PAGENUM

Identifies the type of page numbering that is used when you create a partition-by-range table space. This value is applied to an implicitly created base table space. The PAGESET_PAGENUM subsystem parameter specifies the default PAGENUM value. See [PAGE SET PAGE NUMBERING field \(PAGESET_PAGENUM subsystem parameter\) \(Db2 Installation and Migration\)](#).

RELATIVE

Indicates that internal page numbering is kept as a 4-byte value without a partition number. The page number is a relative page from the start of the partition, and the partition number is kept only in the header page.

ABSOLUTE

Indicates that internal page numbering is kept as a 4-byte value that includes a partition number and page number. Distinguishing which bits represent the partition and which represent the page number requires a shift value. The shift value is LOG base 2 (DSSIZE/(page size)).

Notes

Owner privileges

The owner of the table has all table privileges (see [GRANT \(table or view privileges\) \(Db2 SQL\)](#)) with the ability to grant these privileges to others. For more information about ownership of the object, see [Authorization, privileges, permissions, masks, and object ownership \(Db2 SQL\)](#).

Table design

Designing tables is part of the process of database design. For more information, see [Db2 database design \(Introduction to Db2 for z/OS\)](#).

Considerations for column names longer than 30 bytes

If a length of a new column name is greater than 30 Unicode bytes, truncation occurs in the SQLNAME field of the SQLDA when the column is described in an application. A column name in UTF8, and its equivalent in the system EBCDIC CCSID, must be 128 bytes or less. For more information about long column names, see [Column names longer than 30 bytes \(Db2 SQL\)](#).

If the IN DATABASE clause is specified without a table space name

If you specify IN DATABASE (either explicitly or by default), but do not specify a table space, a table space is implicitly created in the specified database. The name of the table space is derived from the table name. The qualifier of the table space is the same as the qualifier of the table. The owner of the table space is SYSIBM.

For more information, see [Implicitly defined table spaces \(Db2 Administration Guide\)](#).

If the IN clause is not specified

If you do not specify the IN clause, the Db2 implicitly creates a table space as described previously, but the Db2 also chooses a database. Db2 creates a name in the form of DSNnnnnn, where nnnnn is between 00001 and the maximum value of the sequence SYSIBM.DSNSEQ_IMPLICITDB, which has a default of 10000, inclusive. The owner of the database is SYSIBM.

- If DSNnnnnn already exists and is an implicitly created database, the Db2 subsystem creates the table in that database.
- If DSNnnnnn does not exist, the Db2 subsystem creates a database with the name DSNnnnnn.

If DSNnnnnn cannot be created because of a deadlock, timeout, or resource unavailable condition, the Db2 subsystem increments nnnnn by one and tries the resultant database name. If the Db2 subsystem reaches the maximum value of the sequence SYSIBM.DSNSEQ_IMPLICITDB, and the corresponding database name is not available, the Db2 subsystem sets nnnnn to 00001 and tries the resultant database name. If the Db2 subsystem attempts to create the table a number of times that is equal to the maximum value of the sequence SYSIBM.DSNSEQ_IMPLICITDB without success, an error occurs.

System objects for implicitly created table spaces

If a table space is implicitly created, all of the following required system objects are also implicitly created:

- The enforcing primary key index
- The enforcing unique key index
- Any necessary LOB table spaces, auxiliary table spaces, and auxiliary indexes
- The ROWID index (if the ROWID column is defined as GENERATED BY DEFAULT)

When Db2 implicitly creates a base table space for a table with LOB columns that can have inline LOBs, Db2 creates the base table space in reordered row format, regardless of the value of the RRF subsystem parameter.

The attributes of an implicitly created table space can be changed by using the [ALTER TABLESPACE \(Db2 SQL\)](#) statement.

Creating a table in a segmented (non-UTS) table space (deprecated)

A table cannot be created in a segmented table space if any of the following conditions are true:

- The effective application compatibility of the CREATE TABLE statement is V12R1M504 or higher.
- The available space in the data set is less than the segment size specified for the table space, and
- The data set cannot be extended.

Deprecated function: Non-UTS table spaces for base tables are deprecated. CREATE TABLESPACE statements that run at application compatibility level V12R1M504 or higher always create a partition-by-growth or partition-by-range table space, and CREATE TABLE statements that specify a non-UTS table space (including existing multi-table segmented table spaces) return an error. However, you can use a lower application compatibility level to create table spaces of the deprecated types if needed, such as for recovery situations. For instructions, see [Creating non-UTS table spaces \(deprecated\)](#) (Db2 Administration Guide).

Creating a table with graphic and mixed data columns

You cannot create an ASCII or EBCDIC table with a GRAPHIC, VARGRAPHIC, or DBCLOB column or a CHAR, VARCHAR, or CLOB column defined as FOR MIXED DATA when the setting for installation option MIXED DATA is NO, unless the table is EBCDIC, and the columns are Unicode.

Creating a table with distinct type columns based on LOB, ROWID, and DECFLOAT columns

Because a distinct type is subject to the same restrictions as its source type, all the syntactic rules that apply to LOB columns (CLOB, DBCLOB, and BLOB), ROWID columns, and DECFLOAT columns apply to distinct type columns that are based on LOBs, row IDs, and DECFLOATs. For example, a table cannot have both an explicitly defined ROWID column and a column with a distinct type that is based on a row ID.

Tables with inline LOB columns

If the page size is exceeded for a table in a universal table space, Db2 recalculates the record size using 0 as the inline length for LOB columns that do not specify the INLINE LENGTH clause. A record size of 0 is used in the recalculation even if the LOB_INLINE_LENGTH subsystem parameter value is greater than 0. After the recalculation, if the page size is still exceeded, the CREATE TABLE statement returns an error.

You cannot create a table with an inline LOB column in a table space that has basic row format.

Creating a table with LOB columns

A table with a LOB column (CLOB, DBCLOB, or BLOB) must also have a ROWID column, one or more auxiliary tables, and indexes on the auxiliary tables. In many situations, Db2 can implicitly create the required objects for you. For more information, see [“ROWID columns for tables with LOB columns”](#) on page 178 and [“Auxiliary tables and indexes for LOB columns”](#) on page 179.

ROWID columns for tables with LOB columns

When you create the table without explicitly defining a ROWID column, Db2 implicitly generates a ROWID column for you. This column is called an *implicitly hidden ROWID column*. The implicitly hidden ROWID column has the following attributes:

- Db2 creates the column with a name of DB2_GENERATED_ROWID_FOR_LOBS nn .
Db2 appends nn only if the column name already exists in the table, replacing nn with 00 and incrementing by 1 until the name is unique within the row.
- Defines the column as GENERATED ALWAYS.
- Appends the implicitly hidden ROWID column to the end of the row after all the other explicitly defined columns.

For example, assume that Db2 generated an implicitly hidden ROWID column named DB2_GENERATED_ROWID_FOR_LOBS for table MYTABLE. The result table for a SELECT * statement for table MYTABLE would not contain that ROWID column. However, the result table for SELECT COL1, DB2_GENERATED_ROWID_FOR_LOBS would include the implicitly hidden ROWID column.

If the MIXED DATA subsystem parameter is set to YES, and a lowercase or mixed case hexadecimal constant is specified as the default value for a LOB column, the CREATE TABLE statement returns an error.

Auxiliary tables and indexes for LOB columns

The definition of a table that contains a LOB column is marked incomplete until an auxiliary table is created in a LOB table space for each LOB column in the base table and an index is created on each auxiliary table. The auxiliary table stores the actual values of a LOB column. For each LOB column in a partitioned table space, one auxiliary table and related index must be defined for each partition of the base table space.

Db2 sometimes implicitly creates the LOB table space, auxiliary table, and index on the auxiliary table for each LOB column in a table or partition. For more information, see [LOB table space implicit creation \(Db2 Administration Guide\)](#).

If Db2 does not implicitly create the LOB table spaces, auxiliary tables, and indexes on the auxiliary tables, you must create these objects by issuing CREATE TABLESPACE, CREATE AUXILIARY TABLE, and CREATE INDEX statements.

Creating a table with an XML column

If the table has XML columns, the underlying XML table space is implicitly created with the same PAGENUM attribute as the base table space. The DSSIZE is inherited from the base table space for a base table in the partition-by-growth (PBG) table space.

The following table shows the DSSIZE for an implicitly created XML table space for a base table in a partition-by-range (PBR) or range-partitioned (non-UTS) table space. For partition-by-range (PBR) table spaces with relative page numbering, Db2 also rounds the DSSIZE up to the nearest power of two before using the following table.

<i>Table 13. Default DSSIZE for XML table spaces, given the base table space DSSIZE and buffer-pool page size</i>				
Base table space DSSIZE	4KB base page size	8KB base page size	16KB base page size	32KB base page size
1-4 GB	4G B	4 GB	4 GB	4 GB
5-8 GB	32 GB	16 GB	16 GB	16 GB
9-16 GB	64 GB	32 GB	16 GB	16 GB
17-32 GB	64 GB	64 GB	32 GB	16 GB
33-64 GB	64 GB	64 GB	64 GB	32 GB
65-128 GB	256 GB	256 GB	128 GB	64 GB
129-256 GB	256 GB	256 GB	256 GB	128 GB
257-512 GB	512 GB	512 GB	512 GB	256 GB
513-1024 GB	1024 GB	1024 GB	1024 GB	512 GB

For more information, see [XML table space implicit creation \(Db2 Administration Guide\)](#).

Naming convention for implicitly created XML objects

Implicitly created XML table spaces names will be Xyyyynnnn, where yyy is derived from the first three bytes of the base table name (if the name is shorter than 3, yyy is padded with X). nnnn is a numeric string that will start at 0000 and be incremented by 1 until a unique number is found.

Implicitly created XML table names will be Xyyyyyyyyyyyyyyyynnnn, where yyyyyyyyyyyyyyyyyy is the first 18 UTF-8 bytes of the base table name or of the entire name if it is less than 18. nnnn will only be appended if the name already exists in the table. If the name already exists, nnnn will be replaced with 000 and will be incremented by 1 until the name is unique.

Implicitly created document ID index names will be I_DOCIDyyyyyyyyyyyyyyyyyyyynnn, where yyyyyyyyyyyyyyyyyy is the first 18 UTF-8 bytes of the base table name or the entire name if it is less than 18. nnn will only be appended if the index already exists in the table. If the index already exists, nnn will be replaced with 000 and will be incremented by 1 until the name is unique.

Implicitly created node ID index names will be I_NODEIDyyyyyyyyyyyyyyyyyyyynnn, where yyyyyyyyyyyyyyyyyy is the first 18 UTF-8 bytes of the XML table name or the entire name if it is less than 18. nnn will only be appended if the index already exists in the table. If the index already exists, nnn will be replaced with 000 and will be incremented by 1 until the name is unique.

Creating a table with an identity column

When a table has an identity column, Db2 can automatically generate sequential numeric values for the column as rows are inserted into the table. Thus, identity columns are ideal for primary keys. Identity columns and ROWID columns are similar in that both types of columns contain values that Db2 generates. ROWID columns are used in large object (LOB) table spaces and can be useful in direct-row access. ROWID columns contain values of the ROWID data type, which returns a 40-byte VARCHAR value that is not regularly ascending or descending. ROWID data values are therefore not well suited to many application uses, such as generating employee numbers or product numbers. For data that is not LOB data and that does not require direct-row access, identity columns are usually a better approach, because identity columns contain existing numeric data types and can be used in a wide variety of uses for which ROWID values would not be suitable.

When a table is recovered to a point-in-time, it is possible that a large gap in the generated values for the identity column might result. For example, assume a table has an identity column that has an incremental value of 1 and that the last generated value at time T1 was 100 and Db2 subsequently generates values up to 1000. Now, assume that the table space is recovered back to time T1. The generated value of the identity column for the next row that is inserted after the recovery completes will be 1001, leaving a gap from 100 to 1001 in the values of the identity column.

If you want to ensure that an identity column has unique values, create a unique index on the column.

Creating a table with a LONG VARCHAR or LONG VARGRAPHIC column

Although the syntax LONG VARCHAR and LONG VARGRAPHIC is allowed for compatibility with previous releases of Db2, its use is not encouraged. VARCHAR(*integer*) and VARGRAPHIC(*integer*) is the recommended syntax, because after the CREATE TABLE statement is processed, Db2 considers a LONG VARCHAR column to be VARCHAR and a LONG VARGRAPHIC column to be VARGRAPHIC.

When a column is defined using the LONG VARCHAR or LONG VARGRAPHIC syntax, Db2 determines the length attribute of the column. You can use the following information, which is provided for existing applications that require the use of the LONG VARCHAR or LONGVARGRAPHIC syntax, to calculate the byte count and the character count of the column.

To calculate the byte count, use this formula:

$$2 * (\text{INTEGER}((\text{INTEGER}((m - i - k) / j)) / 2))$$

Where:

m

Is the maximum row size (8 less than the maximum record size)

i

Is the sum of the byte counts of all columns in the table that are not LONG VARCHAR or LONG VARGRAPHIC

j

is the number of LONG VARCHAR and LONG VARGRAPHIC columns in the table

k

k is the number of LONG VARCHAR and LONG VARGRAPHIC columns that allow nulls

To find the character count:

1. Find the byte count.
2. Subtract 2.

3. If the data type is LONG VARCHAR, divide the result by 2. If the result is not an integer, drop the fractional part.

Defining a system-period temporal table

A system-period temporal table definition includes the following:

- A system period named SYSTEM_TIME which is defined using a row-begin column and a row-end column.
- A transaction-start-ID column.
- A system-period data versioning definition which includes the name of the associated history table, which is specified in a subsequent ALTER TABLE statement.

To ensure that the history table cannot be implicitly dropped when a system-period temporal table is dropped, use the WITH RESTRICT ON DROP clause in the definition of the history table.

Defining an application-period temporal table

An application-period temporal table definition includes an application period named BUSINESS_TIME. The application period is defined using a begin timestamp column and an end timestamp column.

Data change operations on an application-period temporal table might result in an automatic insert of one or two additional rows when a row is updated or deleted. When an update or delete of a row in an application-period temporal table is specified for a portion of the period that is represented by that row, the row is updated or deleted and one or two rows are automatically inserted to represent the portion of the row that is not changed. New values are generated for each generated column in an application-period temporal table for each row that is automatically inserted as a result of an update or delete operation on the table. If a generated column is defined as part of a unique or primary key, parent key in a referential constraint, or unique index, it is possible that an automatic insert will violate a constraint or index, in which case an error is returned.

Bitemporal tables

A table that is defined for system-period data versioning and contains a BUSINESS_TIME period is referred to as a bitemporal table.

Considerations for transaction-start-ID columns

A transaction-start-ID column contains a null value if the column allows null values. A row-begin column which is unique from other row-begin column values that are generated for other transactions exists with the transaction-start-ID column. Given that the column might contain null values, consider using one of the following methods when retrieving a value from the column:

```
COALESCE ( transaction_start_id_col, row_begin_col)
CASE WHEN transaction_start_id_col IS NOT NULL
      THEN transaction_start_id_col
      ELSE row_begin_col
END
```

Implicitly created indexes

When the PRIMARY KEY or UNIQUE clause is used in the CREATE TABLE statement and the CREATE TABLE statement is processed by the schema processor or the table space that contains the table is implicitly created, Db2 implicitly creates the unique indexes used to enforce the uniqueness of the primary or unique keys.

When a ROWID column is defined as GENERATED BY DEFAULT in the CREATE TABLE statement, and the CREATE TABLE statement is processed by SET CURRENT RULES = 'STD' or the table space that contains the table is implicitly created, Db2 implicitly creates the unique indexes used to enforce the uniqueness of the ROWID column.

The privilege set must include the USE privilege of the buffer pool.

Each index is created as if the following CREATE INDEX statement were issued:

```
CREATE UNIQUE INDEX xxx ON table-name (column1,...)
```

Where:

- *xxx* is the name of the index that Db2 generates.
- *table-name* is the name of the table specified in the CREATE TABLE statement.
- (*column1*,...) is the list of column names that were specified in the UNIQUE or PRIMARY KEY clause of the CREATE TABLE statement, or the column is a ROWID column that is defined as GENERATED BY DEFAULT.

For more information about the schema processor, see [Creating a schema by using the schema processor \(Db2 Administration Guide\)](#).

In addition, if a table space that contains the table is implicitly created, Db2 will check the DEFINE DATA SET subsystem parameter to determine whether to define the underlying data set for the index space of the implicitly created index on the base table.

If DEFINE DATA SET is NO, the index is created as if the following CREATE INDEX statement is issued:

```
CREATE UNIQUE INDEX xxx ON table-name (column1,...) DEFINE NO
```

Maximum record size

The maximum record size of a table depends on the page size of the table space and whether the EDITPROC clause is specified, as shown in [Table 14 on page 182](#).

The initial page size of the table space is the size of its buffer, which is determined by the BUFFERPOOL clause that was explicitly or implicitly specified when the table space was created. When the record size reaches 90 percent of the maximum record size for the page size of the table space, the next largest page size is automatically used.

Table 14. Maximum record size, in bytes

	Page Size = 4KB	Page Size = 8KB	Page Size = 16KB	Page Size = 32KB
Table without EDITPROC=YES	4056	8138	16330	32714
Table with EDITPROC=YES	4046	8128	16320	32704

The maximum record size corresponds to the maximum length of a VARCHAR column if that column is the only column in the table.

If the table space that contains the table is implicitly created, the proper buffer pool size is chosen according to the actual record size. If the record size reaches 90% of the maximum record size for the page size of the table space, the next largest page size will be used. [Table 15 on page 182](#) shows 90% of the maximum record size:

Table 15. 90% of Maximum record size, in bytes

	Page Size = 4KB	Page Size = 8KB	Page Size = 16KB	Page Size = 32KB
Table without EDITPROC=YES	3650	7324	14697	29443
Table with EDITPROC=YES	3641	7315	14688	29434

A row in a table with PAGENUM RELATIVE or in a table space with PAGENUM RELATIVE must have a minimum data size of 3 bytes. Rows with data that compresses to less than 3 bytes, will not be compressed when stored in the table.

Byte counts

The sum of the byte counts of the columns must not exceed the maximum row size of the table. The maximum row size is eight less than the maximum record size.

For columns that do not allow null values, Table 16 on page 183 gives the byte counts of columns by data type. For columns that allow null values, the byte count is one more than shown in the table.

Table 16. Byte counts of columns by data type

Data Type	Byte Count
INTEGER	4
SMALLINT	2
BIGINT	8
FLOAT(<i>n</i>)	If <i>n</i> is between 1 and 21, the byte count is 4. If <i>n</i> is between 22 and 53, the byte count is 8.
DECIMAL	INTEGER($p/2$)+1, where <i>p</i> is the precision
DECFLOAT(16)	9
DECFLOAT(34)	17
CHAR(<i>n</i>)	<i>n</i>
VARCHAR(<i>n</i>)	<i>n</i> +2
CLOB	6
Inline CLOB	6 + inline byte count
GRAPHIC(<i>n</i>)	2 <i>n</i>
VARGRAPHIC(<i>n</i>)	2 <i>n</i> +2
DBCLOB	6
Inline DBCLOB	6 + (inline char count * 2)
BINARY(<i>n</i>)	<i>n</i>
VARBINARY(<i>n</i>)	<i>n</i> +2
BLOB	6
Inline BLOB	6 + inline byte count
DATE	4
TIME	3
TIMESTAMP(<i>p</i>) WITHOUT TIME ZONE	INTEGER($(p+1)/2$) + 7 where <i>p</i> is the precision
TIMESTAMP(<i>p</i>) WITH TIME ZONE	INTEGER($(p+1)/2$) + 9 where <i>p</i> is the precision
ROWID	19
distinct type	The length of the source data type upon which the distinct type was based
XML	6 - If column cannot contain multiple versions of an XML document. 14 - If column can contain multiple versions of an XML document.

For more information, see [XML versions \(Db2 Programming for XML\)](#).

Creating a materialized query table

If the fullselect in the CREATE TABLE statement contains a SELECT *, the select list of the subselect is determined at the time the materialized query table is created. In addition, any references to user-defined functions are resolved at the same time. The isolation level at the time when the CREATE TABLE statement is executed is the isolation level for the materialized query table. After a materialized query table is created, the REFRESH_TIME column of the row for the table in the SYSIBM.SYSVIEWS catalog table contains the default timestamp.

The owner of a materialized query table has all the table privileges with the grant option on the table irrespective of whether the owner has the necessary privileges on the base tables, views, functions, and sequences.

No unique constraints or unique indexes can be created for materialized query tables. Thus, a materialized query table cannot be a parent table in a referential constraint.

When you are creating user-maintained materialized query tables, you should create the materialized query table with query optimization disabled and then enable the table for query optimization after it is populated. Otherwise, Db2 might rewrite queries to use the empty materialized query table, and you will not get accurate results.

Considerations for implicitly hidden columns

A column that is defined as implicitly hidden is not part of the result table of a query that specifies * in a SELECT list. However, an implicitly hidden column can be explicitly referenced in a query. For example, an implicitly hidden column can be referenced in the SELECT list or in a predicate in a query. Additionally, an implicitly hidden column can be explicitly referenced in a COMMENT, CREATE INDEX statement, ALTER TABLE statement, INSERT statement, MERGE statement, UPDATE statement, or RENAME statement. An implicitly hidden column can be referenced in a referential constraint. A REFERENCES clause that does not contain a column list refers implicitly to the primary key of the parent table. It is possible that the primary key of the parent table includes a column defined as implicitly hidden. Such a referential constraint is allowed.

If the SELECT list of the fullselect of a materialized query definition explicitly refers to an implicitly hidden column, that column will be part of the materialized query table.

If the SELECT list of the fullselect of a view definition (CREATE VIEW statement) explicitly refers to an implicitly hidden column, that column will be part of the view, however the view column is not considered 'hidden'.

Restrictions on field procedures, edit procedures, and validation exit procedures

Field procedures, edit procedures, and validation exit procedures cannot be used on tables that have column names that are larger than 18 EBCDIC bytes. If you have tables that have field procedures or validation exit procedures and you add a column where the column name is larger than 18 bytes, the field procedures and validation exit procedures for the table will be invalidated.

Consider using triggers to replace the functionality on field procedures, edit procedures, and validation exit procedures on tables where the column names are larger than 18 EBCDIC bytes.

Restrictions on certain SQL statements in the same unit of work as CREATE TABLE

- A CREATE TABLE statement that contains a PARTITION BY clause should not be followed in the same unit of work by SQL statements that change data.
- A CREATE TABLE statement that contains an IN ACCELERATOR clause should be issued in a separate unit of work from other SQL statements.

Creating a table while a utility runs

You cannot use CREATE TABLE while a Db2 utility has control of the table space implicitly or explicitly specified by the IN clause.

Restrictions involving pending definition changes

A CREATE TABLE statement is not allowed if there are pending changes to the definition of the table space, if the CREATE TABLE statement specifies a FOREIGN KEY clause that reference a column for which there are pending definition changes, or if the CREATE TABLE statement specifies

a materialized query table definition that references a table for which there are pending definition changes.

Key label requirement

To use a key label for encryption, the VSAM data sets for the page sets need to be associated with an SMS Data Class that has extended format capability (EF enabled).

Determining a key label for base table space and associated objects

When a key label is specified at the table level, Db2 provides the key label to DFSMS to encrypt all the table spaces and index spaces associated with the table. This includes base table space, auxiliary table spaces, XML table spaces, index spaces, and clone table space, regardless of whether the base table space or associated objects are explicitly or implicitly created. Db2 does not enforce any key label relationship between the base table and an associated history or archive table. The key label for the archive and the history tables has to be set independent of the base table. If there is no key label specified at the table level, Db2 will provide the key label to DFSMS specified for the storage group.

When Db2 calls DFSMS to allocate the dataset for table space or index space, DFSMS uses its order of precedence to determine the key label and can override the key label specified by Db2.

DFSMS order of precedence:

- RACF data set profile
- JCL, dynamic allocation, TSO ALLOCATE
- SMS data class construct

If the security administrator has specified a key label for the RACF data set profile, that key label takes precedence over the Db2 provided key label. The REPORT utility can be run to determine the key label used for encryption.

Description of key label in effect in DB2

Table 17. Example scenarios for a partition-by-growth table space, that describe the key label in effect in DB2. This is the key label provided to DFSMS during allocation of data set for table spaces and index spaces.

Scenarios	Catalog key label value	Key label provided to DFSMS during data set allocation
Create storage group, SG01 with key label, SGKL01.	SYSSTOGROUP record - KEY LABEL: SGKL01	
Create table space, TBSP01 using storage group, SG01 – Creates Partition 1		SGKL01
Create table, TB01 in table space, TBSP01 with key label, TBKL01	SYSTABLESPACE record for TBSP01 / SYSTABLES record for TBKL01 – KEY LABEL: TBKL01	
REORG TABLESPACE TBSP01 – Reorgs Partition 1		TBKL01
Create index, IX01 on table, TB01 creates index space	SYSINDEXES record for IX01 – KEY LABEL: TBKL01	TBKL01
Insert data into TB01 – Creates Partition 2		TBKL01
Alter table, TB01 to specify NO KEY LABEL	SYSTABLESPACE record for TBSP01 / SYSTABLES record for TBKL01 / SYSINDEXES record for IX01 – KEY LABEL: Empty string	
Insert data into TB01 – Creates partition 3		SGKL01

Table 17. Example scenarios for a partition-by-growth table space, that describe the key label in effect in DB2. This is the key label provided to DFSMS during allocation of data set for table spaces and index spaces. (continued)

Scenarios	Catalog key label value	Key label provided to DFSMS during data set allocation
REORG TABLESPACE TBSP01 with REUSE option – Resets and reuses DB2-managed data sets. No change to key label		

Key label considerations

If the last table is dropped from a segmented table space, the table space and its underlying data set will remain. If key label is in effect, the KEYLABEL column for the table space's SYSTABLESPACE record will be cleared. If a new table is created in this table space, it will be encrypted with the previous key label. If the table has to be created as unencrypted, execute the REORG TABLESPACE utility for the table space.

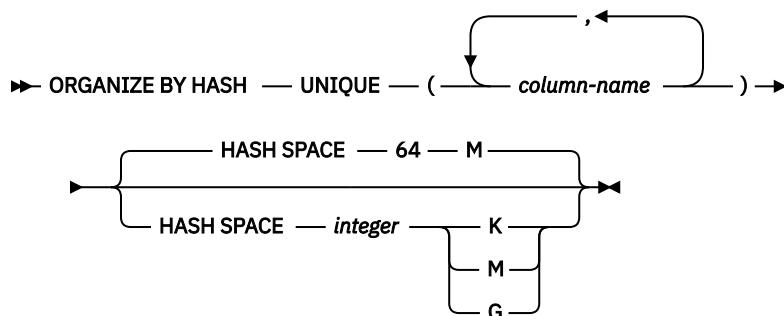
If a table space is explicitly created with the DEFINE YES option and a table with a key label is defined in that table space, then the data sets associated with the table space will not be encrypted. A subsequent REORG is necessary to encrypt the data sets. Users that want immediate encryption of the data sets associated with the table space must to define table spaces with the DEFINE NO option.

Syntax and descriptions for hash organization (deprecated)

Deprecated function: Hash-organized tables are deprecated. Beginning in Db2 12, packages bound with APPLCOMPAT(V12R1M504) or higher cannot create hash-organized tables or alter existing tables to use hash-organization. Existing hash organized tables remain supported, but they are likely to be unsupported in the future.

organization-clause

organization-clause



ORGANIZE BY HASH

Specifies that a hash is to be used for the data organization of the table.

If PARTITION BY RANGE is specified, and the IN clause specifies a table space, the table space must be a partition by range universal table space, and cannot be a table space with PAGENUM RELATIVE.

If PARTITION BY RANGE is not specified, and an IN clause is specified, the IN clause must identify a partition-by-growth table space.

ORGANIZE BY HASH must not be specified if the table is defined with APPEND YES.

ORGANIZE BY HASH must not be specified if the table is using basic row format.

If IN ACCELERATOR is specified, ORGANIZE BY HASH must not be specified.

UNIQUE

Specifies that Db2 enforces uniqueness of the hash key columns, preventing the table from containing two or more rows with the same value of the hash key.

(*column-name*, ...)

The list of column names defines the hash key that is used to determine where a row will be placed. Each *column-name* must be an unqualified name that identifies a column of the table. The same column must not be specified more than once and the specified columns must be defined as NOT NULL. The number of specified columns must not exceed 64, and the sum of their length attributes must not exceed 255. A specified column cannot be any of the following types:

- a LOB column
- a DECFLOAT column
- a XML column
- a distinct type column that is based on one of the preceding data types

All character and graphic string columns in the key must use the same encoding scheme.

If PARTITION BY RANGE is also specified, the list of column names must specify all of the column names that are specified in *partition-expression* for the table, and must specify the column names in the same order as *partition-expression*. If the ORGANIZE BY clause contains more columns than *partition-expression*, *partition-expression* determines the partition number.

HASH SPACE *integer*K|M|G

Specifies the amount of fixed hash space to preallocate for the table. If the table is partitioned by range, this is the space for each partition.

The default is 64M for a table in a partition-by-growth table space or 64M for each partition of a partition-by-range table space.

K

Indicates that the *integer* value is multiplied by 1024 to specify the hash space size in bytes. The *integer* value must be between 256 and 268,435,456.

M

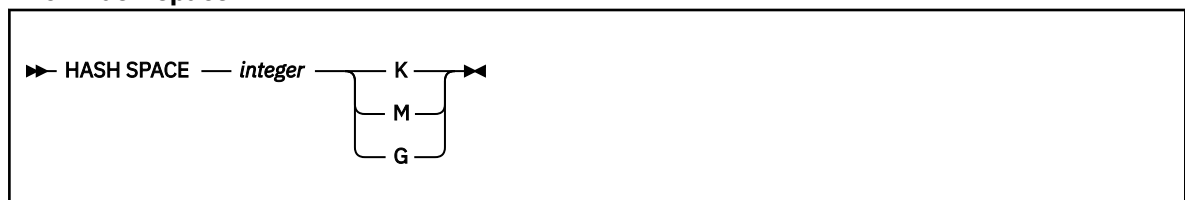
Indicates that the *integer* value is multiplied by 1,048,576 to specify the hash space size in bytes. The *integer* value must be between 1 and 262,144.

G

Indicates that the *integer* value is to be multiplied by 1,073,741,824 to specify the hash space size in bytes. The *integer* value must be between 1 and 256 for a partition by range table and must be between 1 and 131,072 for a non-partitioned table.

If a value greater than 4G is specified, the data sets for the table space are associated with a DFSMS data class that has been specified with extended format and extended addressability.

partition-hash-space



HASH SPACE *integer*K|M|G

Specifies the amount of fixed hash space to preallocate for the partition that is associated with the *partition-element*. If HASH SPACE is omitted from the partition element, the HASH SPACE value from the ORGANIZE BY clause is used. If IN ACCELERATOR is specified, HASH SPACE must not be specified.

If HASH SPACE is not specified, each partition will use the HASH SPACE value specified in *organization-clause*.

The HASH SPACE keyword in *partition-element* must only be specified if *organization-clause* is also specified.

K

Indicates that the *integer* value is multiplied by 1024 to specify the hash space size in bytes. The *integer* value must be between 256 and 268,435,456.

M

Indicates that the *integer* value is multiplied by 1,048,576 to specify the hash space size in bytes. The *integer* value must be between 1 and 262,144.

G

Indicates that the *integer* value is to be multiplied by 1,073,741,824 to specify the hash space size in bytes. The *integer* value must be between 1 and 256 for a partition by range table and must be between 1 and 131,072 for a non-partitioned table.

If a value greater than 4G is specified, the data sets for the table space are associated with a DFSMS data class that has been specified with extended format and extended addressability.

Notes for hash organization (deprecated)

Deprecated function: Hash-organized tables are deprecated. Beginning in Db2 12, packages bound with APPLCOMPAT(V12R1M504) or higher cannot create hash-organized tables or alter existing tables to use hash-organization. Existing hash organized tables remain supported, but they are likely to be unsupported in the future.

If the IN clause is not specified with ORGANIZE BY HASH

If you do not specify IN DATABASE (either explicitly or by default), Db2 will use the default DSSIZE of 4G for each partition for a partition-by-range table space or use the value that is specified in the partitioning clause. The hash space value that is specified on CREATE TABLE will be validated, per part, to ensure that the specified DSSIZE is adequate. If the DSSIZE is not adequate, an error will be returned.

If the maximum number of partitions needed for the specified hash space is more than the maximum number of partitions allowed, Db2 will return an error.

If the selected buffer pool is not available, an error will be returned.

Creating a table with hash organization and LOB columns

If the table space is a partition-by-growth table space, Db2 will preallocate as many partitions as needed depending on the value specified for HASH SPACE. If Db2 needs to implicitly create the LOB object in a new partition, the privilege set for the implicitly created LOB objects must include the USE privilege on the buffer pool for the LOB table space.

Hash space and Db2 page size

If the specified hash space is less than or equal to 64 MB (the Db2 default), Db2 will add extra space for Db2 system pages. If the specified hash space is greater than 64 MB, Db2 will use part of the hash space for Db2 system pages. The amount of space needed for Db2 system pages depends on SEGSIZE and PAGESIZE. The larger the SEGSIZE and/or PAGESIZE becomes, the larger the requirement for Db2 system pages. Db2 can reserve up to 5 MB for system pages for the highest SEGSIZE value (64) and PAGESIZE value (32K).

Hash space and DSSIZE

Depending on certain table space characteristics, Db2 needs to reserve space for the hash overflow area. Therefore, the amount of hash space cannot be equal to the DSSIZE value. The maximum amount of hash space that can be specified is approximately 20% less than the DSSIZE value. Db2 returns an error if the amount of hash space is too large. If the amount of hash space is too large, specify a larger value of DSSIZE, or decrease the amount of hash space.

Specifying APPEND for hash-organized tables

Append processing is not applicable to tables with hash organization since there is no key clustering in hash organization. For insert operations into tables with hash organization, Db2 will use the internal hash algorithm to determine the location of the row.

Maximum record size for hash-organized tables

For hash-organized tables, the maximum record size on whether the EDITPROC clause is specified, as shown in Table 18 on page 189.

The initial page size of the table space is the size of its buffer, which is determined by the BUFFERPOOL clause that was explicitly or implicitly specified when the table space was created. When the record size reaches 90 percent of the maximum record size for the page size of the table space, the next largest page size is automatically used.

Table 18. Maximum record size, in bytes for hash organized tables

	Page Size = 4KB	Page Size = 8KB	Page Size = 16KB	Page Size = 32KB
Hash table (hash home page)	3817	7899	16091	32475
Hash table with EDITPROC=YES (hash home page)	3807	7889	16081	32465

The maximum record size corresponds to the maximum length of a VARCHAR column if that column is the only column in the table.

If the table space that contains the table is implicitly created, the proper buffer pool size is chosen according to the actual record size.

A row in a table with PAGENUM RELATIVE or in a table space with PAGENUM RELATIVE must have a minimum data size of 3 bytes. Rows with data that compresses to less than 3 bytes, will not be compressed when stored in the table.

Restrictions for tables with hash organization

Tables that use hash organization are subject to the following restrictions:

- A table that is defined to use hash organization cannot be created in a LOB table space or XML table space.
- ORGANIZE BY HASH must not be specified if the table space is defined with the MEMBER CLUSTER clause.
- The MAXROWS clause is applicable only to the hash overflow area of the table space for tables with hash organization. The fixed hash area of each page will contain as many rows as it can hold, up to a maximum of 255.
- The ORGANIZE BY HASH UNIQUE (*column-list*) clause is required when specifying HASH SPACE *integer* K|M|G in the *partition-element*. The *organization-clause* applies to the entire table and the *partition-element* clause applies at the partition level.
- Db2 automatically creates a hash overflow index when a table is created with hash organization.

Alternative syntax and synonyms

To provide compatibility with previous releases of Db2 or other products in the Db2 family, Db2 supports the following clauses:

- NOCACHE (single clause) as a synonym for NO CACHE
- NOCYCLE (single clause) as a synonym for NO CYCLE
- NOMINVALUE (single clause) as a synonym for NO MINVALUE
- NOMAXVALUE (single clause) as a synonym for NO MAXVALUE
- NOORDER (single clause) as a synonym for NO ORDER
- PART *integer* VALUES can be specified as an alternative to PARTITION *integer* ENDING AT.
- VALUES as a synonym for ENDING AT
- DEFINITION ONLY as a synonym for WITH NO DATA

- SUMMARY between CREATE and TABLE
- TIMEZONE can be specified as an alternative to TIME ZONE.

Examples

Example 1

Create a table named DSN8D10.DEPT in the table space DSN8S13D of the database DSN8D13A. Name the five columns DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, and LOCATION, allowing only MGRNO and LOCATION to contain nulls, and designating DEPTNO as the only column in the primary key. All five columns hold character string data. Assuming a value of NO for the field MIXED DATA on installation panel DSNTIPF, all five columns have the subtype SBCS.

```
CREATE TABLE DSN8D10.DEPT
(DEPTNO CHAR(3) NOT NULL,
DEPTNAME VARCHAR(36) NOT NULL,
MGRNO CHAR(6) ,
ADMRDEPT CHAR(3) NOT NULL,
LOCATION CHAR(16) ,
PRIMARY KEY(DEPTNO) )
IN DSN8D13A.DSN8S13D;
```

Example 2

Create a table named DSN8D10.PROJ in an implicitly created table space of the database DSN8D13A. Assign the table a validation procedure named DSN8EAPR.

```
CREATE TABLE DSN8D10.PROJ
(PROJNO CHAR(6) NOT NULL,
PROJNAME VARCHAR(24) NOT NULL,
DEPTNO CHAR(3) NOT NULL,
RESPEMP CHAR(6) NOT NULL,
PRSTAFF DECIMAL(5,2) ,
PRSTDATE DATE ,
PRENDATE DATE ,
MAJPROJ CHAR(6) NOT NULL)
IN DATABASE DSN8D13A
VALIDPROC DSN8EAPR;
```

Example 3

Assume that table PROJECT has a non-primary unique key that consists of columns DEPTNO and RESPEMP (the department number and employee responsible for a project). Create a project activity table named ACTIVITY with a foreign key on that unique key.

```
CREATE TABLE ACTIVITY
(PROJNO CHAR(6) NOT NULL,
ACTNO SMALLINT NOT NULL,
ACTDEPT CHAR(3) NOT NULL,
ACTOWNER CHAR(6) NOT NULL,
ACSTAFF DECIMAL(5,2) ,
ACSTDATE DATE NOT NULL,
ACENDATE DATE ,
FOREIGN KEY (ACTDEPT,ACTOWNER)
REFERENCES PROJECT (DEPTNO,RESPEMP) ON DELETE RESTRICT)
IN DSN8D13A.DSN8S13D;
```

Example 4

Create an employee photo and resume table EMP_PHOTO_RESUME that complements the sample employee table. The table contains a photo and resume for each employee. Put the table in table space DSN8D13A.DSN8S13E. Let Db2 always generate the values for the ROWID column.

```
CREATE TABLE DSN8D10.EMP_PHOTO_RESUME
(EMPNO CHAR(6) NOT NULL,
EMP_ROWID ROWID NOT NULL GENERATED ALWAYS,
EMP_PHOTO BLOB(110K),
RESUME CLOB(5K),
PRIMARY KEY (EMPNO))
```

```
IN DSN8D13A.DSN8S13E
CCSID EBCDIC;
```

Example 5

Create an EMPLOYEE table with an identity column named EMPNO. Define the identity column so that Db2 will always generate the values for the column. Use the default value, which is 1, for the first value that should be assigned and for the incremental difference between the subsequently generated consecutive numbers.

```
CREATE TABLE EMPLOYEE
(EMPNO      INTEGER GENERATED ALWAYS AS IDENTITY,
 ID         SMALLINT,
 NAME      CHAR(30),
 SALARY    DECIMAL(5,2),
 DEPTNO    SMALLINT)
IN DSN8D13A.DSN8S13D;
```

Example 6

Assume a very large transaction table named TRANS contains one row for each transaction processed by a company. The table is defined with many columns. Create a materialized query table for the TRANS table that contain daily summary data for the date and amount of a transaction.

```
CREATE TABLE STRANS AS
(SELECT YEAR AS SYEAR, MONTH AS SMONTH, DAY AS SDAY, SUM(AMOUNT) AS SSUM
FROM TRANS
GROUP BY YEAR, MONTH, DAY)
DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

Example 7

The following example creates a table in a partition-by-growth table space and includes the APPEND option:

```
CREATE TABLE TS01TB
(C1 SMALLINT,
 C2 DECIMAL(9,2),
 C3 CHAR(4))
APPEND YES
IN TS01DB.TS01TS;
```

Example 8

The following example creates a table in a partition-by-growth table space where the table space is implicitly created:

```
CREATE TABLE TS02TB
(C1 SMALLINT,
 C2 DECIMAL(9,2),
 C3 CHAR(4))
PARTITION BY SIZE EVERY 4G
IN DATABASE DSNDB04;
```

Example 9

Create a table, EMP_INFO, that contains a phone number and address for each employee. Include a row change timestamp column in the table to track the modification of employee information:

```
CREATE TABLE EMP_INFO
(EMPNO CHAR(6) NOT NULL,
 EMP_INFOCHANGE NOT NULL
  GENERATED ALWAYS FOR EACH ROW ON UPDATE
  AS ROW CHANGE TIMESTAMP,
 EMP_ADDRESS VARCHAR(300),
 EMP_PHONENO CHAR(4),
 PRIMARY KEY (EMPNO));
```

Example 10

Create a table, TB01, that uses a range partitioning scheme with a segment size of 4 and 4 partitions.

```
CREATE TABLE TB01 (
 ACCT_NUM      INTEGER,
 CUST_LAST_NM  CHAR(15),
```

```

LAST_ACTIVITY_DT VARCHAR(25),
COL2             CHAR(10),
COL3             CHAR(25),
COL4             CHAR(25),
COL5             CHAR(25),
COL6             CHAR(55),
STATE            CHAR(55))
IN DBB.TS01

PARTITION BY (ACCT_NUM)
(PARTITION 1 ENDING AT (199),
 PARTITION 2 ENDING AT (299),
 PARTITION 3 ENDING AT (399),
 PARTITION 4 ENDING AT (MAXVALUE));

```

Example 11

Create a table, `policy_info`, that uses a `SYSTEM_TIME` period and create a history table, `hist_policy_info`. Then issue an `ALTER TABLE` statement to associate the `policy_info` table with the `hist_policy_info` table.

```

CREATE TABLE policy_info
(policy_id CHAR(10) NOT NULL,
 coverage INT NOT NULL,
 sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
 sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
 create_id TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
 PERIOD SYSTEM_TIME(sys_start,sys_end));

```

```

CREATE TABLE hist_policy_info
(policy_id CHAR(10) NOT NULL,
 coverage INT NOT NULL,
 sys_start TIMESTAMP(12) NOT NULL,
 sys_end TIMESTAMP(12) NOT NULL,
 create_id TIMESTAMP(12));

```

```

ALTER TABLE policy_info
ADD VERSIONING USE HISTORY TABLE hist_policy_info;

```

Example 12

Create a table, `policy_info`, that uses a `BUSINESS_TIME` period.

```

CREATE TABLE policy_info
(policy_id CHAR(4) NOT NULL,
 coverage INT NOT NULL,
 bus_start DATE NOT NULL,
 bus_end DATE NOT NULL,
 PERIOD BUSINESS_TIME(bus_start, bus_end));

```

Example 13

Create a table, `policy_info`, that uses both a `SYSTEM_TIME` period and a `BUSINESS_TIME` period to keep historical rows and track a user-specified time period. A table that specifies both a `SYSTEM_TIME` period and a `BUSINESS_TIME` period is sometimes referred to as a *bitemporal table*. To enable retention of historical rows, a history table, `hist_policy_info`, also needs to be created and associated (using the `ALTER TABLE` statement) with the `policy_info` table.

```

CREATE TABLE policy_info
(policy_id CHAR(4) NOT NULL,
 coverage INT NOT NULL,
 bus_start DATE NOT NULL,
 bus_end DATE NOT NULL,
 sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
 sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
 create_id TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
 PERIOD BUSINESS_TIME(bus_start, bus_end),
 PERIOD SYSTEM_TIME(sys_start, sys_end));

```

```

CREATE TABLE hist_policy_info
(policy_id CHAR(4) NOT NULL,
 coverage INT NOT NULL,
 bus_start DATE NOT NULL,
 bus_end DATE NOT NULL,

```

```
sys_start TIMESTAMP(12) NOT NULL,  
sys_end TIMESTAMP(12) NOT NULL),  
create_id TIMESTAMP(12);
```

```
ALTER TABLE policy_info  
ADD VERSIONING USE HISTORY TABLE hist_policy_info;
```

Example 14: Create table EMPLOYEE.PERSONAL with key label EMPKEYLABEL.

```
CREATE TABLE EMPLOYEE.PERSONAL  
(DEPTNO CHAR(3) NOT NULL,  
DEPTNAME VARCHAR(36) NOT NULL,  
MGRNO CHAR(6),  
ADMRDEPT CHAR(3) NOT NULL,  
LOCATION CHAR(16),  
PRIMARY KEY(DEPTNO) )  
IN DSN8D12A.DSN8S12D  
KEY LABEL EMPKEYLABEL;
```

Related concepts

[Types of accelerator tables \(Db2 Performance\)](#)

[Unicode columns in EBCDIC tables \(Db2 SQL\)](#)

[Unicode support in Db2 \(Db2 Installation and Migration\)](#)

[Naming conventions \(Db2 SQL\)](#)

Related tasks

[Creating tables from application programs \(Db2 Application programming and SQL\)](#)

Related reference

[EBCDIC and ASCII support \(Db2 Installation and Migration\)](#)

Related information

[Implementing Db2 tables \(Db2 Administration Guide\)](#)

[Conditions that prevent query routing to an accelerator](#)

CREATE VIEW

The CREATE VIEW statement creates a view on tables or views at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES RUN behavior is in effect. For more information, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).

Authorization

For every table or view identified in the *fullselect*, the privilege set that is defined below must include at least one of the following:

- The SELECT privilege on the table or view
- Ownership of the table or view
- DBADM authority for the database (tables only)
- DATAACCESS authority
- SYSADM authority
- SQLADM authority (catalog tables only)
- System DBADM authority (catalog tables only)
- ACCESSCTRL authority (catalog tables only)
- SYSCTRL authority (catalog tables only)
- SECADM authority (catalog tables only)

If the database is implicitly created, the database privileges must be on the implicit database or on DSNDBO4.

Authority requirements depend in part on the choice of the owner of the view. For information on how to choose the owner, see the description of *view-name* in [ALTER VIEW \(Db2 SQL\)](#).

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the plan or package. If the application is bound in a trusted context with the `ROLE AS OBJECT OWNER` clause specified, a role is the owner. Otherwise, an authorization ID is the owner.

- If this privilege set includes SYSADM authority, the owner of the view can be any authorization ID. If that set includes SYSCTRL but not SYSADM authority, the following is true: the owner of the view can be any authorization ID, provided the view does not refer to user tables or views in the first FROM clause of its defining fullselect. (It could refer instead, for example, to catalog tables or views thereof.)

If the view satisfies the rules in the preceding paragraph, and if no errors are present in the CREATE statement, the view is created, even if the owner has no privileges at all on the tables and views identified in the fullselect of the view definition.

- If the privilege set includes system DBADM authority, the owner of the view can be any authorization ID. However, to create a view on a user table, either the owner of the view or the creator must have the SELECT privilege on all the tables or views in the CREATE VIEW statement.
- If the privilege set lacks system DBADM, SYSADM and SYSCTRL but includes DBADM authority on at least one of the databases that contains a table from which the view is created, the owner of the view can be any authorization ID if all of the following conditions are true:
 - The value of subsystem parameter DBACRVW is set to YES.
 - The view is not based only on views.

Note: The owner of the view must have the SELECT privilege on all tables and views in the CREATE VIEW statement, or, if the owner does not have the SELECT privilege on a table, the creator must have DBADM authority on the database that contains that table.

- If the privilege set lacks SYSADM, SYSCTRL, system DBADM, and DBADM authority, or if the authorization ID of the application plan or package fails to meet any of the previous conditions, the owner of the view must be the owner of the application plan or package.

If `ROLE AS OBJECT OWNER` is in effect, the schema qualifier must be the same as the role, unless the role has the `CREATEIN` privilege on the schema, SYSADM authority, system DBADM authority, or SYSCTRL authority.

If `ROLE AS OBJECT OWNER` is not in effect, one of the following rules applies:

- If the privilege set lacks the `CREATIN` privilege on the schema, SYSADM authority, system DBADM authority, or SYSCTRL authority, the schema qualifier (implicit or explicit) must be the same as one of the authorization ids of the process.
- If the privilege set includes system DBADM authority, SYSADM authority or SYSCTRL authority, the schema qualifier can be any valid schema name.

If the statement is dynamically prepared, the following rules apply:

- If the SQL authorization ID of the process has SYSADM authority, the owner of the view can be any authorization ID. If that authorization ID has SYSCTRL but not SYSADM authority, the following is true: the owner of the view can be any authorization ID, provided the view does not refer to user tables or views in the first FROM clause of its defining fullselect. (It could refer instead, for example, to catalog tables or views thereof.)

If the view satisfies the rules in the preceding paragraph, and if no errors are present in the CREATE statement, the view is created, even if the owner has no privileges at all on the tables and views identified in the fullselect of the view definition.

- If the SQL authorization ID of the process has system DBADM authority, the owner of the view can be any authorization ID. However, to create a view on a user table, either the owner of the view or the

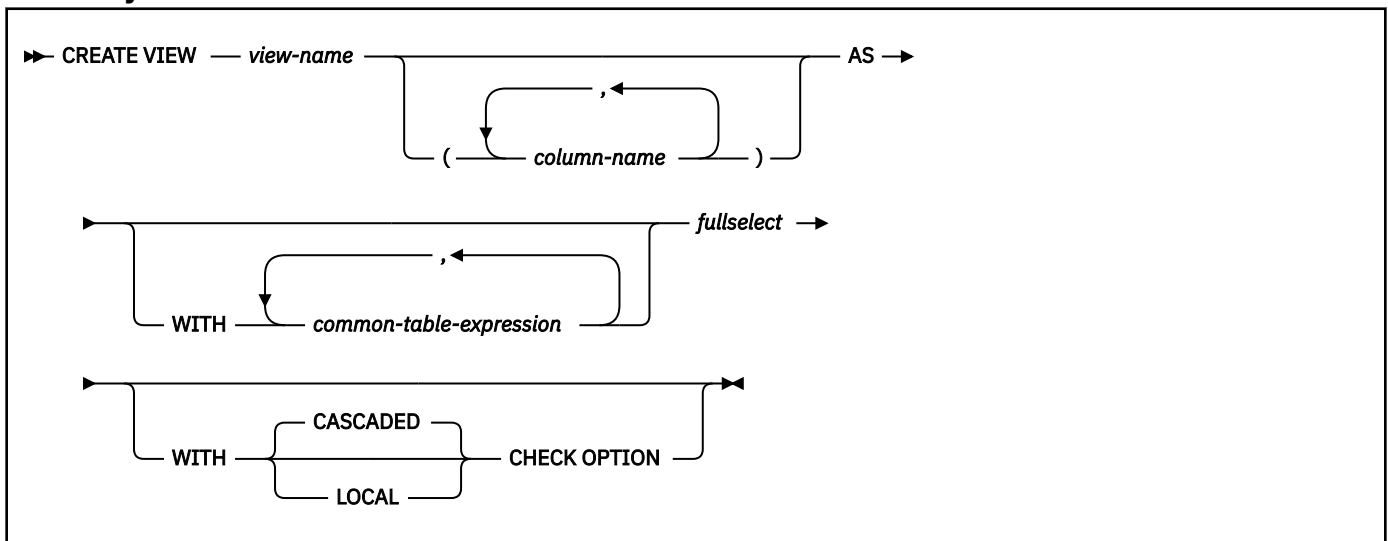
SQL authorization ID must have the SELECT privilege on all the tables or views in the CREATE VIEW statement.

- If SQL authorization ID of the process lacks system DBADM authority, SYSADM and SYSCTRL but includes DBADM authority on at least one of the databases that contains a table from which the view is created, the owner of the view can be different from the SQL authorization ID if all of the following conditions are true:
 - The value of field DBADM CREATE AUTH was set to YES on panel DSNTIPP during Db2 installation.
 - The view is not based only on views.

Note: The owner of the view must have the SELECT privilege on all tables and views in the CREATE VIEW statement, or, if the owner does not have the SELECT privilege on a table, the creator must have DBADM authority on the database that contains that table.

- If the SQL authorization ID of the process lacks SYSADM, SYSCTRL, system DBADM authority, or DBADM authority, or if the SQL authorization ID of the process fails to meet any of the previous conditions, only the authorization IDs of the process can own the view. In this case, the privilege set is the privileges that are held by the authorization ID selected for ownership.

Syntax



Description

view-name

Names the view. The name, including the implicit or explicit qualifier, must not identify a table, view, alias, or synonym that exists at the current server or a table that exists in the SYSIBM.SYSPENDINGOBJECTS catalog table. The unqualified name must not be the same as an existing synonym.

If the name is qualified, the name can be a two-part or three-part name. If a three-part name is used, the first part must match the value of the field Db2 LOCATION NAME of installation panel DSNTIPR at the current server. (If the current server is not the local Db2, this name is not necessarily the name in the CURRENT SERVER special register.)

column-name,...

Names the columns in the view. If you specify a list of column names, it must consist of as many names as there are columns in the result table of the fullselect. Each name must be unique and unqualified. If you do not specify a list of column names, the columns of the view inherit the names of the columns of the result table of the fullselect.

You must specify a list of column names if the result table of the fullselect has duplicate column names or an unnamed column (a column derived from a constant, function, or expression that was not

given a name by the AS clause). For more details about unnamed columns, see the information about names of result columns under [select-clause \(Db2 SQL\)](#).

AS

Identifies the view definition.

WITH common-table-expression

Defines a common table expression for use with the fullselect that follows. The fullselect must not contain a period specification. For an explanation of common table expression, see [common-table-expression \(Db2 SQL\)](#).

fullselect

Defines the view. At any time, the view consists of the rows that would result if the fullselect were executed.

The *fullselect* must conform to the following rules:

- The *fullselect* must not refer to any host variables or parameter markers (question marks), but can refer to global variables.
- The *fullselect* must not refer to any declared temporary tables.
- The *fullselect* must not include an invocation of the UNPACK function.
- The *fullselect* must not include an invocation of the AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY function.
- The *fullselect* must not contain a period specification.
- The FROM clause of the *fullselect* must not include a *data-change-table-reference*.
- The FROM clause of the *fullselect* must not include a view for which an INSTEAD OF trigger is defined.
- The outer SELECT list of the outer *fullselect* must not result in a column that is an array.

For an explanation of *fullselect*, see [fullselect \(Db2 SQL\)](#).

WITH CASCADED CHECK OPTION or WITH LOCAL CHECK OPTIONS

Specifies that every row that is inserted or updated through the view must conform to the definition of the view. A row that does not conform to the definition of the view is a row that cannot be retrieved using that view.

The CHECK OPTION clause must not be specified if the view is read-only, includes a subquery, references a function that is not deterministic or has an external action, or if the fullselect of the view refers to a created temporary table. If the CHECK OPTION clause is specified for an updatable view that does not allow inserts, it applies to updates only.

If the CHECK OPTION clause is omitted, the definition of the view is not used in the checking of any insert or update operations that use the view. Some checking might still occur during insert or update operations if the view is directly or indirectly dependent on another view that includes the CHECK OPTION clause. Because the definition of the view is not used, rows might be inserted or updated through the view that do not conform to the definition of the view.

The difference between the two forms of the check option, CASCADED and LOCAL, is meaningful only when a view is dependent on another view. The default is CASCADED. The view on which another view is directly or indirectly defined is an *underlying view*.

CASCADED

Update and insert operations on view V must satisfy the search conditions of view V and all underlying views, regardless of whether the underlying views were defined with a check option. Furthermore, every updatable view that is directly or indirectly defined on view V inherits those search conditions (the search conditions of view V and all underlying views of V) as a constraint on insert or update operations. WITH CASCADED CHECK OPTION must not be specified if a view on which the specified view definition is dependent has an INSTEAD OF trigger defined.

LOCAL

Update and insert operations on view V must satisfy the search conditions of view V and underlying views that are defined with a check option (either WITH CASCADED CHECK OPTION

or WITH LOCAL CHECK OPTION). Furthermore, every updatable view that is directly or indirectly defined on view V inherits those search conditions (the search conditions of view V and all underlying views of V that are defined with a check option) as a constraint on insert or update operations.

The LOCAL form of the CHECK option lets you update or insert rows that do not conform to the search condition of view V. You can perform these operations if the view is directly or indirectly defined on a view that was defined without a check option.

Table 19 on page 197 illustrates the effect of using the default check option, CASCADED. The information in Table 19 on page 197 is based on the following views:

- CREATE VIEW V1 AS SELECT COL1 FROM T1 WHERE COL1 > 10
- CREATE VIEW V2 AS SELECT COL1 FROM V1 WITH CASCADED CHECK OPTION
- CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100

Table 19. Examples using default check option, CASCADED

SQL statement	Description of result
INSERT INTO V1 VALUES(5)	Succeeds because V1 does not have a check option and it is not dependent on any other view that has a check option.
INSERT INTO V2 VALUES(5)	Results in an error because the inserted row does not conform to the search condition of V1 which is implicitly is part of the definition of V2.
INSERT INTO V3 VALUES(5)	Results in an error because the inserted row does not conform to the search condition of V1.
INSERT INTO V3 VALUES(200)	Succeeds even though it does not conform to the definition of V3 (V3 does not have the view check option specified); it does conform to the definition of V2 (which does have the view check option specified).

The difference between CASCADED and LOCAL is shown best by example. Consider the following updatable views, where x and y represent either LOCAL or CASCADED:

- V1 is defined on Table T0.
- V2 is defined on V1 WITH x CHECK OPTION.
- V3 is defined on V2.
- V4 is defined on V3 WITH y CHECK OPTION.
- V5 is defined on V4.

This example shows V1 as an *underlying view* for V2 and V2 as *dependent* on V1.

Table 20 on page 197 shows the views in which search conditions are checked during an insert or update operation:

Table 20. Views in which search conditions are checked during insert and update operations

View used in INSERT or UPDATE operation	x = LOCAL y = LOCAL	x = CASCADED y = CASCADED	x = LOCAL y = CASCADED	x = CASCADED y = LOCAL
V1	None	None	None	None
V2	V2	V2, V1	V2	V2, V1
V3	V2	V2, V1	V2	V2, V1
V4	V4, V2	V4, V3, V2, V1	V4, V3, V2, V1	V4, V2, V1

Table 20. Views in which search conditions are checked during insert and update operations (continued)

View used in INSERT or UPDATE operation	x = LOCAL y = LOCAL	x = CASCADED y = CASCADED	x = LOCAL y = CASCADED	x = CASCADED y = LOCAL
V5	V4, V2	V4, V3, V2, V1	V4, V3, V2, V1	V4, V2, V1

Notes

Owner privileges

The owner of a view always acquires the SELECT privilege on the view and the authority to drop the view. If all of the privileges that are required to create the view are held with the GRANT option before the view is created, the owner of the view receives the SELECT privilege with the GRANT option. Otherwise, the owner receives the SELECT privilege without the GRANT option. For example, assume that a view definition also refers to a user-defined function. If the owner's EXECUTE privilege on the user-defined function is held without the GRANT option, the owner acquires the SELECT privilege on the view without the GRANT option.

The owner can also acquire INSERT, UPDATE, and DELETE privileges on the view. Acquiring these privileges is possible if the view is not "read-only", which means a single table or view is identified in the first FROM clause of the fullselect. For each privilege that the owner has on the identified table or view (INSERT, UPDATE, and DELETE) before the new view is created, the owner acquires that privilege on the view. The owner receives the privilege with the GRANT option if the privilege is held on the table or view with the GRANT option. Otherwise, the owner receives the privileges without the GRANT option.

With appropriate Db2 authority, a process can create views for those who have no authority to create the views themselves. The owner of such a view has the SELECT privilege on the view, without the GRANT option, and can drop the view.

For more information on the ownership of an object, see [Authorization, privileges, permissions, masks, and object ownership \(Db2 SQL\)](#).

Authorization for views created for other users

When a process with appropriate authority creates a view for another user that does not have authorization for the underlying table or view, the SELECT privilege for the created view is implicitly granted to the user.

Considerations for column names longer than 30 bytes

If a length of a new column name is greater than 30 Unicode bytes, truncation occurs in the SQLNAME field of the SQLDA when the column is described in an application. A column name in UTF8, and its equivalent in the system EBCDIC CCSID, must be 128 bytes or less. For more information about long column names, see [Column names longer than 30 bytes \(Db2 SQL\)](#).

Considerations for row access control and column access control

The view definition might reference a table for which row access control or column access control is activated. If the view definition references a table for which row access control or column access control is activated, the WITH CHECK OPTION clause must not be specified if the search conditions from the view or from the underlying views will be checked during an insert or update operation. Note that the WITH CHECK OPTION clause is ignored if such search conditions do not exist.

Read-only views

A view is *read-only* if one or more of the following statements is true of its definition:

- The first FROM clause identifies more than one table or view, or identifies a table function, a nested table expression, a common table expression, or a collection-derived table.
- The first SELECT clause specifies the keyword DISTINCT.
- The outer fullselect contains a GROUP BY clause.
- The outer fullselect contains a HAVING clause.

- The first SELECT clause contains an aggregate function.
- It contains a subquery such that the base object of the outer fullselect, and of the subquery, is the same table.
- The first FROM clause identifies a read-only view.
- The first FROM clause identifies a system-maintained materialized query table.
- The outer fullselect is not a subselect (contains a set operator).

A read-only view cannot be the object of an SQL data change statement or a TRUNCATE statement. A view that includes GROUP BY or HAVING cannot be referred to in a subquery of a basic predicate.

Insertable views

A view is insertable if an INSTEAD OF trigger for the insert operation has been defined for the view, or if at least one column of the view is updatable (independent of an INSTEAD OF trigger for update).

Considerations for implicitly hidden columns

It is possible that the result table of the fullselect will include a column of a base table that is defined as implicitly hidden. This can occur when the implicitly hidden column is explicitly referenced in the fullselect of the view definition. However, the corresponding column of the view does not inherit the implicitly hidden attribute. Columns of a view cannot be defined as hidden.

Testing a view definition

You can test the semantics of your view definition by executing `SELECT * FROM view-name`.

The two forms of a view definition

Both the source and the operational form of a view definition are stored in the Db2 catalog. Those two forms are not necessarily equivalent because the operational form reflects the state that exists when the view is created. For example, consider the following statement:

```
CREATE VIEW V AS SELECT * FROM S;
```

In this example, S is a synonym or alias for A.T, which is a table with columns C1, C2, and C3[®]. The operational form of the view definition is equivalent to:

```
SELECT C1, C2, C3 FROM A.T;
```

Adding columns to A.T using ALTER TABLE and dropping S does not affect the operational form of the view definition. Thus, if columns are added to A.T or if S is redefined, the source form of the view definition can be misleading.

View restrictions

A view definition cannot contain references to remote objects. A view definition cannot map to more than 15 base table instances. A view definition cannot reference a declared global temporary table.

Restrictions involving pending definition changes

CREATE VIEW is not allowed if the view references a column on which there are pending definition changes.

Considerations for inline LOB columns

If the view references a table that contains an inline LOB column and Db2 determines that the inline attribute can be passed on to the view, the view will then inherit the inline attribute, otherwise the inline attribute is not inherited by the view.

Considerations for XML columns

If the view has an XML column and the column of the underlying base table for the view has an XML type modifier, the view column has the same type modifier. However, if there is an instead of trigger defined on the view, validation of the column, according to XML schemas in the type modifier, is not enforced during insert or update to this view.

Examples

Example 1

Create the view DSN8D10.VPROJRE1. PROJNO, PROJNAME, PROJDEP, RESPEMP, FIRSTNME, MIDINIT, and LASTNAME are column names. The view is a join of tables and is therefore read-only.

```
CREATE VIEW DSN8D10.VPROJRE1
  (PROJNO, PROJNAME, PROJDEP, RESPEMP,
   FIRSTNME, MIDINIT, LASTNAME)
AS SELECT ALL
  PROJNO, PROJNAME, DEPTNO, EMPNO,
  FIRSTNME, MIDINIT, LASTNAME
FROM DSN8D10.PROJ, DSN8D10.EMP
WHERE RESPEMP = EMPNO;
```

In the example, the WHERE clause refers to the column EMPNO, which is contained in one of the base tables but is not part of the view. In general, a column named in the WHERE, GROUP BY, or HAVING clause need not be part of the view.

Example 2

Create the view DSN8D10.FIRSTQTR that is the UNION ALL of three fullselects, one for each month of the first quarter of 2000. The common names are SNO, CHARGES, and DATE.

```
CREATE VIEW DSN8D10.FIRSTQTR (SNO, CHARGES, DATE) AS
SELECT SNO, CHARGES, DATE
FROM MONTH1
WHERE DATE BETWEEN '01/01/2000' and '01/31/2000'
UNION ALL
SELECT SNO, CHARGES, DATE
FROM MONTH2
WHERE DATE BETWEEN '02/01/2000' and '02/29/2000'
UNION ALL
SELECT SNO, CHARGES, DATE
FROM MONTH3
WHERE DATE BETWEEN '03/01/2000' and '03/31/2000';
```

Related concepts

[Db2 views \(Introduction to Db2 for z/OS\)](#)

[Naming conventions \(Db2 SQL\)](#)

Related tasks

[Creating Db2 views \(Db2 Administration Guide\)](#)

DELETE

The DELETE statement deletes rows from a table or view. Deleting a row from a view deletes the row from the table on which the view is based if no INSTEAD OF DELETE trigger is defined for this view. If such a trigger is defined, the trigger is activated instead.

The table or view can be at the current server or any Db2 subsystem with which the current server can establish a connection.

There are two forms of this statement:

- The *searched* DELETE form is used to delete one or more rows, optionally determined by a search condition.
- The *positioned* DELETE form specifies that one or more rows corresponding to the current cursor position are to be deleted.

Invocation

This statement can be embedded in an application program or issued interactively. A positioned DELETE is embedded in an application program. Both the embedded and interactive forms are executable statements that can be dynamically prepared.

Authorization

Authority requirements depend on whether the object identified in the statement is a user-defined table, a catalog table, or a view, and whether the statement is a searched DELETE and SQL standard rules are in effect:

When a table other than a catalog table is identified: The privilege set must include at least one of the following:

- The DELETE privilege on the table
- Ownership of the table
- DBADM authority on the database that contains the table
- SYSADM authority

If the database is implicitly created, the database privileges must be on the implicit database or on DSNDB04.

When a catalog table is identified: The privilege set must include at least one of the following:

- DBADM authority on the catalog database
- SYSCTRL authority
- SYSADM authority

When a view is identified: The privilege set must include at least one of the following:

- The DELETE privilege on the view
- SYSADM authority

If the *search-condition* in a searched DELETE contains a reference to a column of the table or view, or the *expression* in the *assignment-clause* contains a reference to a column of the table or view, the privilege set must include at least one of the following:

- The SELECT privilege on the table or view
- Ownership of the table or view
- DBADM authority on the database that contains the table, if the target is a table and that table that is not a catalog table
- DATAACCESS
- SYSADM authority

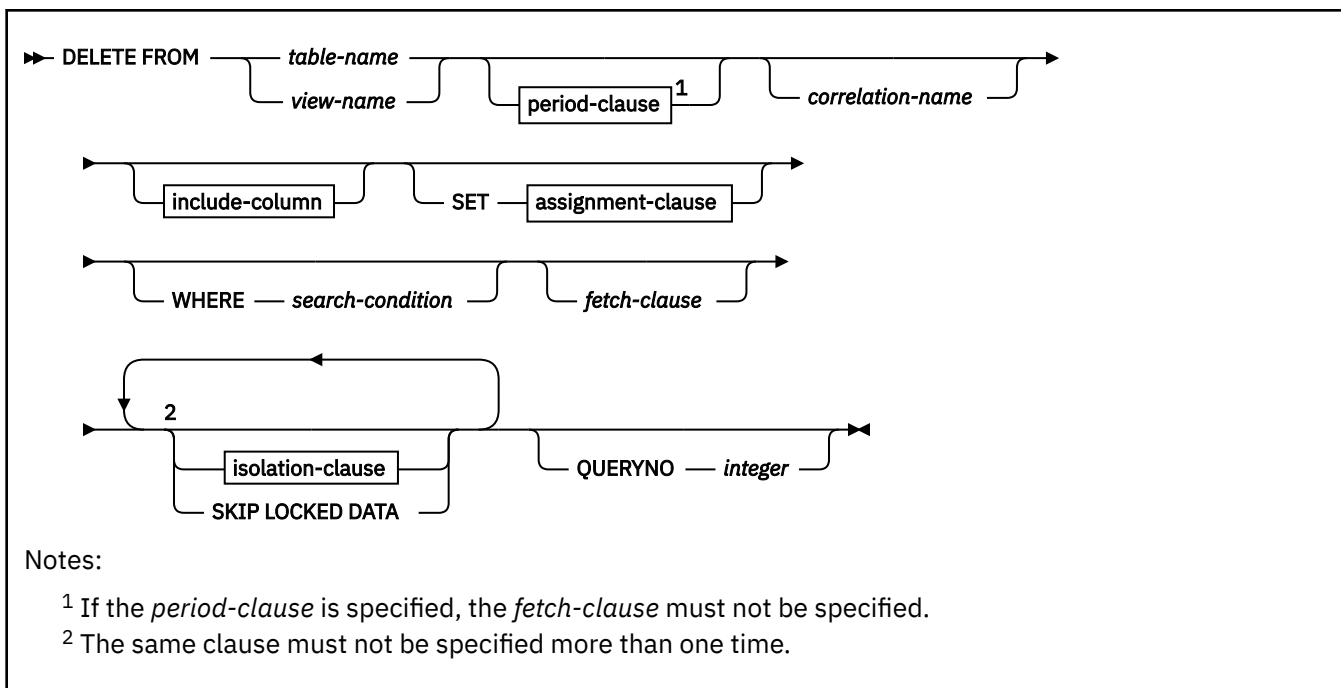
If the *search-condition* in a searched DELETE includes a subquery, or if the *assignment-clause* includes a *scalar-fullselect* or a *row-fullselect*, see [Authorization for queries \(Db2 SQL\)](#) for an explanation of the authorization required.

The owner of a view, unlike the owner of a table, might not have DELETE authority on the view (or might have DELETE authority without being able to grant it to others). The nature of the view itself can preclude its use for DELETE. For more information, see the description of authority in [CREATE VIEW \(Db2 SQL\)](#).

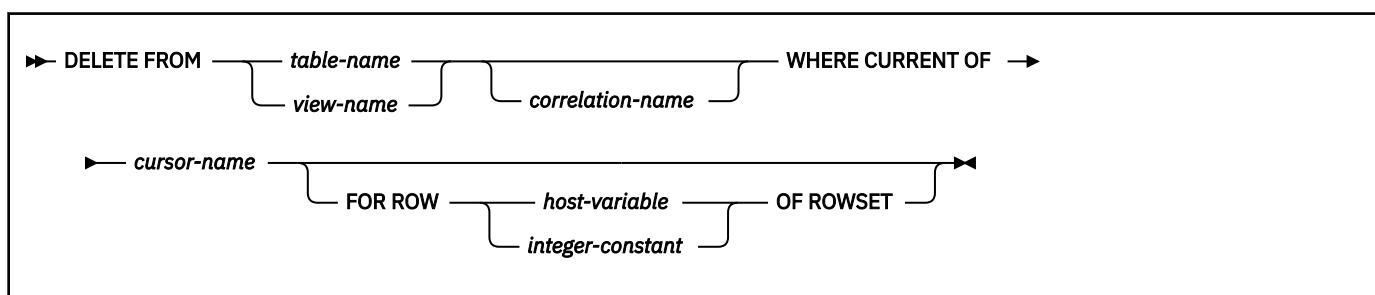
If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in "DYNAMICRULES behaviors and authorization checking" in [Dynamic preparation and execution \(Db2 SQL\)](#). (For more information on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see [Authorization IDs and dynamic SQL \(Db2 SQL\)](#).)

If the statement attempts to delete a row in the SYSIBM.SYSAUDITPOLICIES catalog table that is subject to a tamper-proof audit policy, additional RACF authorization is required. During statement execution, the primary authorization ID or one of the groups associated with the primary authorization ID must be authorized to access the tamper-proof audit policy profile in RACF. For more information on the authorization rules, see [Db2 audit policy \(Managing Security\)](#).

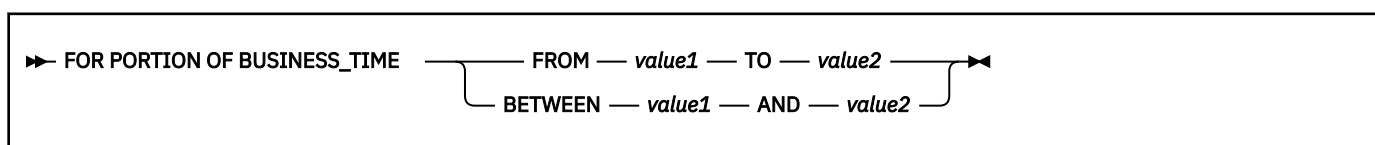
searched delete:



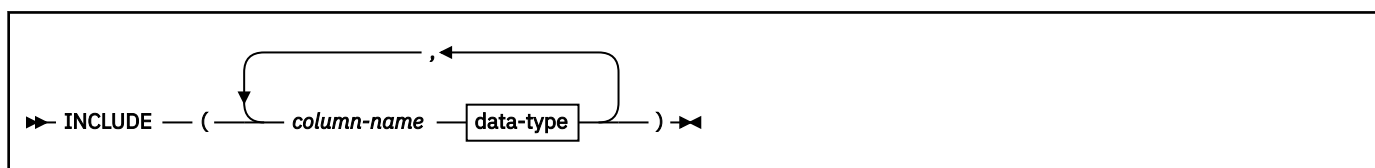
positioned delete:



period-clause:



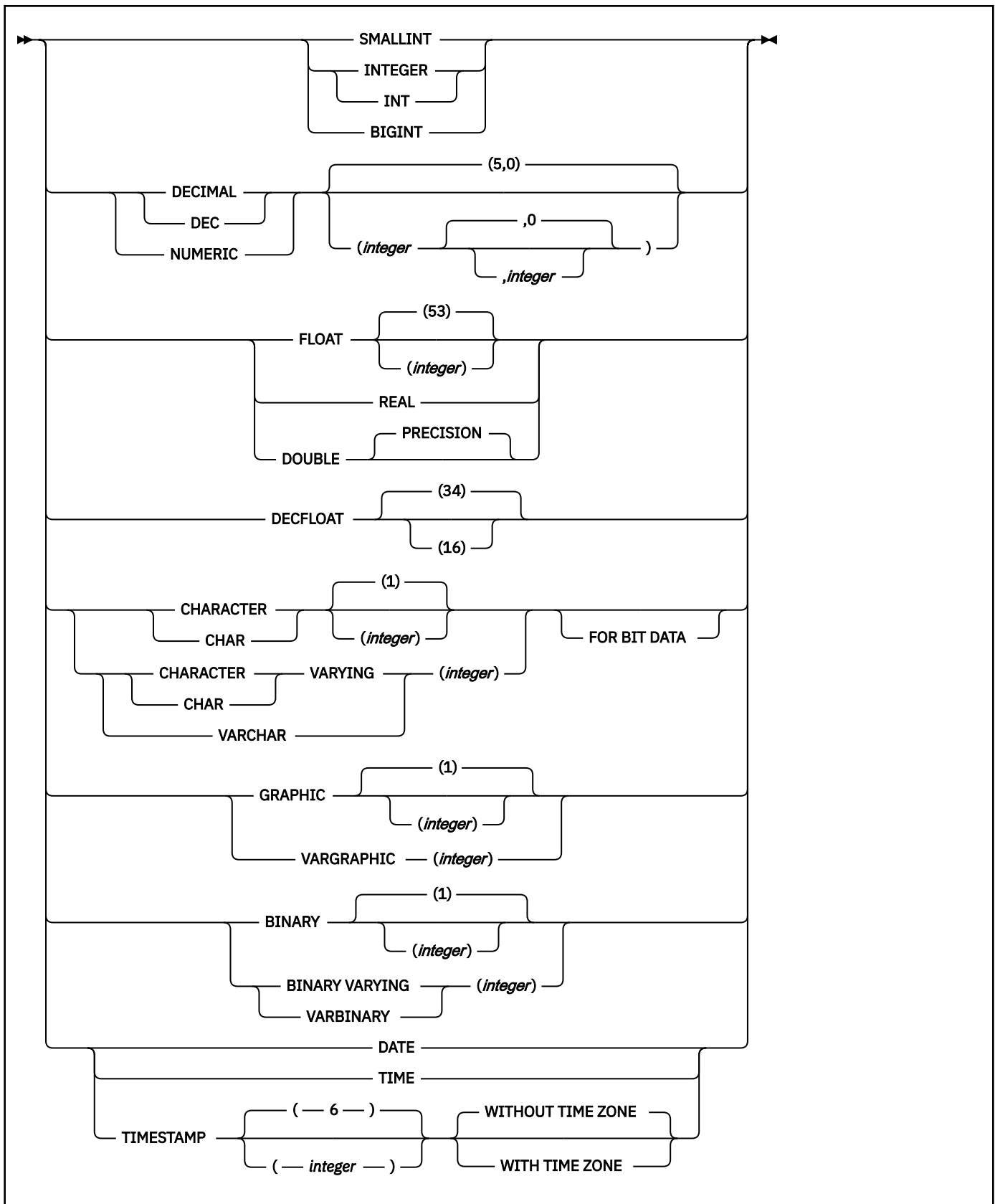
include-column:



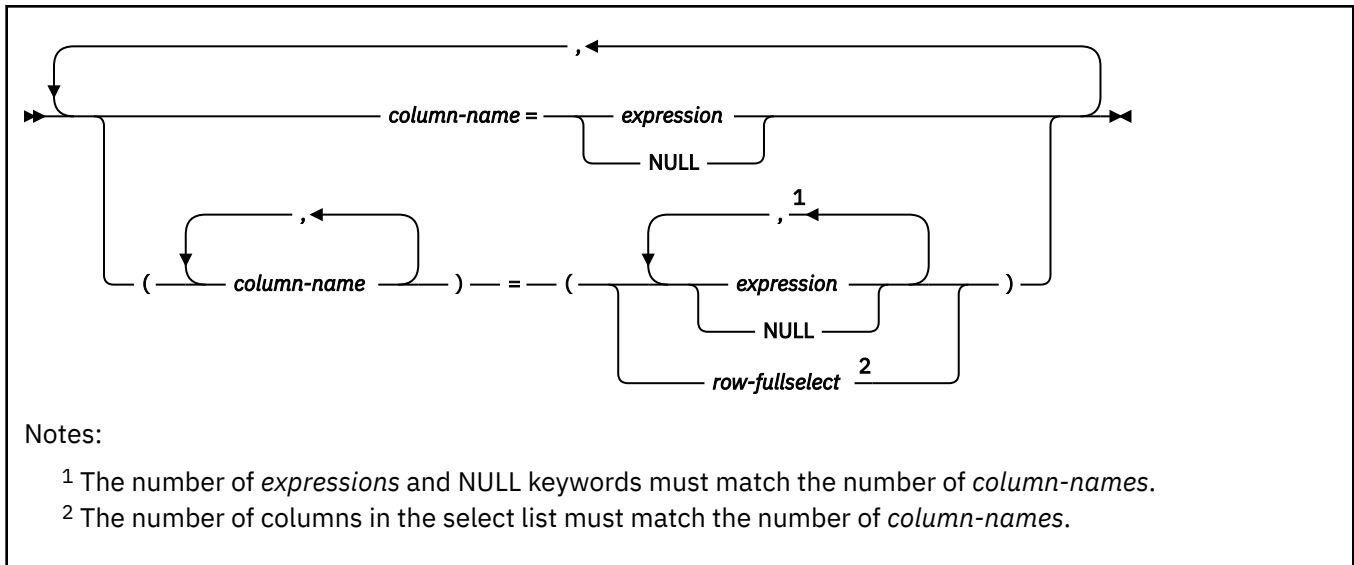
data-type:



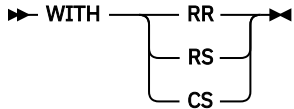
built-in-type:



assignment clause:



isolation-clause:



Description

FROM *table-name* or *view-name*

Identifies the table or view from which rows are to be deleted. The name must identify a table or view that exists at the Db2 subsystem that is identified by the implicitly or explicitly specified location name. The name must not identify:

- An auxiliary table
- A catalog table for which deletes are not allowed
- A view of such a catalog table
- A directory table
- A read-only view (see [CREATE VIEW \(Db2 SQL\)](#))
- A view that is defined with an instead of trigger if the fetch-clause is specified.
- A created global temporary table if the fetch-clause is specified.
- An accelerator-only table if the fetch-clause is specified.
- A system-maintained materialized query table
- A table that is implicitly created for an XML column
- An archive-enabled table if the SYSIBMADM.GET_ARCHIVE global variable is set to Y, the ARCHIVESENSITIVE bind option is set to YES, and the operation is a positioned delete

In an IMS or CICS application, the Db2 subsystem that contains the identified table or view must be a remote server that supports two-phase commit.

period-clause

Specifies that a period clause applies to the target of the delete operation. The same period name must not be specified more than one time. If the target of the delete operation is a view:

- The FROM clause of the outer fullselect of the view definition must include a reference, directly or indirectly, to an application-period temporal table.
- An INSTEAD OF trigger must not be defined for that view.

FOR PORTION OF BUSINESS_TIME

Specifies that the delete only applies to row values for the portion of the BUSINESS_TIME period in the row that is specified by the period clause. BUSINESS_TIME must be a period that is defined on the table.

FOR PORTION OF BUSINESS_TIME must not be specified if the value of the CURRENT TEMPORAL BUSINESS_TIME special register is not NULL when the BUSTIMESENSITIVE bind option is set to YES.

FROM *value1* TO *value2*

Specifies that the delete operation applies to rows for the period that is specified from *value1* to *value2*. No rows are deleted if *value1* is greater than or equal to *value2*, or if *value1* or *value2* is the null value.

This clause must not be specified for an inclusive-inclusive period.

For the period condition that is specified with FROM *value1* TO *value2*, the period that is specified by *period-name* in a row of the target table of the delete:

- Overlaps the beginning of the specified period if the value of the begin column is less than *value1* and the value of the end column is greater than *value1*.
- Overlaps the end of the specified period if the value of the end column is greater than or equal to *value2* and the value of the begin column is less than *value2*.
- Is fully contained within the specified period if the value for the begin column for *period-name* in the row is greater than or equal to *value1* and the value for the corresponding end column in the row is less than or equal to *value2*.
- Is partially contained in the specified period if the row overlaps the beginning of the specified period or the end of the specified period.
- Fully overlaps the specified period if the period in the row overlaps the beginning of the specified period and overlaps the end of the specified period, .
- Is not contained in the period if both columns of *period-name* are less than or equal to *value1* or are greater than *value2*.

If the period *period-name* in a row is not contained in the specified period, the row is not deleted. Otherwise, the delete operation is applied based on the specification of the PORTION OF clause and how the values in the columns of *period-name* overlap the specified period as follows:

- If the period *period-name* in a row is fully contained within the specified period, the row is deleted.
- If the period *period-name* in a row is partially contained in the specified period and overlaps the beginning of the specified period:
 - The row is deleted.
 - A row is inserted using the original values from the row, except that the end column is set to *value1*, and new values are used for other generated columns.
- If the period *period-name* in a row is partially contained in the specified period and overlaps the end of the specified period:
 - The row is deleted.
 - A row is inserted using the original values from the row, except that the begin column is set to *value2*, and new values are used for other generated columns.
- If the period *period-name* in a row fully overlaps the specified period:
 - The row is deleted.
 - A row is inserted using the original values from the row, except that the end column is set to *value1*, a column defined as DATA CHANGE OPERATION is set to 'I', and new values are used for other generated columns.

- An additional row is inserted using the original values from the row, except that the begin column is set to *value2*, a column defined as DATA CHANGE OPERATION is set to 'I', and new values are used for other generated columns.

Any existing delete triggers are activated for the rows that are deleted, and any existing insert triggers are activated for the rows that are implicitly inserted.

BETWEEN *value1* AND *value2*

Specifies that the delete operation applies to rows for the period that is specified from *value1* up to and including *value2*. No rows are deleted if *value1* is greater than *value2*, or if *value1* or *value2* is the null value. This clause must not be specified for an inclusive-exclusive period.

For the period clause that is specified with BETWEEN *value1* AND *value2*, period *period-name* in a row in the target of the delete covers one of the following ranges:

- Overlaps the beginning of the specified period if the value of the begin column is less than *value1* and the value of the end column is greater than or equal to *value1*.
- Overlaps the end of the specified period if the value of the end column is greater than *value2* and the value of the begin column is less than or equal to *value2*.
- Is fully contained within the specified period if the value for the begin column for *period-name* in the row is greater than or equal to *value1* and the value for the corresponding end column in the row is less than or equal to *value2*.
- Is partially contained in the specified period if the row overlaps the beginning of the specified period or the end of the specified period, but not both .
- fully overlaps the specified period if the period in the row overlaps the beginning of the specified period and overlaps the end of the specified period.
- Is not contained in the period if both columns of *period-name* are less than *value1* or greater than *value2*.

If the period *period-name* in a row is not contained in the specified period, the row is not deleted. Otherwise, the delete operation is based on the following items:

- The specification of the PORTION OF clause.
- How the values in the columns of *period-name* overlap the specified period.
- *spu* (smallest period unit), which depends on the data type of the columns of the period as follows:
 - For a period containing DATE columns, spu is 1 day.
 - For a period containing TIMESTAMP(6) columns, spu is 1 microsecond.

Based on those items, the delete operation is applied as follows:

- If the period *period-name* in a row is fully contained within the specified period, the row is deleted.
- If the period *period-name* in a row is partially contained in the specified period and overlaps the beginning of the specified period:
 - The row is deleted.
 - A row is inserted using the original values from the row, except that the end column is set to *value1* - spu, and new values are used for other generated columns.
- If the period *period-name* in a row is partially contained in the specified period and overlaps the end of the specified period:
 - The row is deleted.
 - A row is inserted using the original values from the row, except that the begin column is set to *value2* + spu, and new values are used for other generated columns.
- If the period *period-name* in a row fully overlaps the specified period:
 - The row is deleted.

- A row is inserted using the original values from the row, except that the end column is set to *value1* - spu, and new values are used for other generated columns.
- An additional row is inserted using the original values from the row, except that the begin column is set to *value2* + spu, and new values are used for other generated columns.

value1, value2

Specifies expressions that return a value of a built-in data type. The result of each expression must be comparable to the data type of the columns of the specified period. See the comparison rules described in [Assignment and comparison \(Db2 SQL\)](#). Each expression can contain any of the following supported operands:

- A constant
- A special register
- A variable
- An array element specification
- A built-in scalar function whose arguments are supported operands. The scalar function must not be AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.
- A CAST specification where the cast operand is a supported operand
- An expression that uses arithmetic operators and operands

Each expression must not have a timestamp precision that is greater than the precision of the columns for the period.

If the begin and end columns of the period are defined as `TIMESTAMP WITHOUT TIME ZONE`, each expression must not return a value of a timestamp with a time zone.

A period clause for a view must not contain an untyped parameter marker.

correlation-name

Specifies an alternate name that can be used within the *search-condition* to designate the table or view. (For an explanation of correlation names, see [Correlation names \(Db2 SQL\)](#).)

include-column

Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the result table of the DELETE statement when it is nested in the FROM clause of the outer fullselect that is used in a subselect, SELECT statement, or in a SELECT INTO statement. The included columns are appended to the end of the list of columns that is identified by *table-name* or *view-name*. If no value is assigned to a column that is specified by an *include-column*, a NULL value is returned for that column.

INCLUDE

Introduces a list of columns that are to be included in the result table of the DELETE statement.

The included columns are only available if the DELETE statement is nested in the FROM clause of a SELECT statement or a SELECT INTO statement.

column-name

Specifies the name for a column of the result table of the DELETE statement that is not the same name as another included column nor a column in the table or view that is specified in *table-name* or *view-name*.

data-type

Specifies the data type of the included column. The included columns are nullable.

built-in-type

Specifies a built-in data type. See [CREATE TABLE \(Db2 SQL\)](#) for a description of each built-in type.

The CCSID 1208 and CCSID 1200 clauses must not be specified for an INCLUDE column.

distinct-type

Specifies a distinct type. Any length, precision, or scale attributes for the column are those of the source type of the distinct type as specified by using the CREATE TYPE statement.

SET

Introduces the assignment of values to columns.

assignment-clause

The assignment-clause introduces a list of one or more *column-names* and the values that are to be assigned to the columns. The *column-names* are the only columns that can be set using the *assignment-clause*.

column-name

Identifies an INCLUDE column.

Assignments to included columns are only processed when the DELETE statement is nested in the FROM clause of a SELECT statement or a SELECT INTO statement. The columns that are named in the INCLUDE clause are the only columns that can be set using the SET clause. The null value is returned for an included column that is not set by using an explicit SET clause.

expression

Indicates the new value of the column. The *expression* is any expression of the type described in [Expressions \(Db2 SQL\)](#). It must not include an aggregate function.

A *column-name* in an expression must identify a column of the table or view. For each row that is deleted, the value of the column in the expression is the value of the column in the row before the row is deleted.

NULL

Specifies the null value as the new value of the column. Specify NULL only for nullable columns.

row-fullselect

Specifies a fullselect that returns a single row. The column values are assigned to each of the corresponding *column-names*. If the fullselect returns no rows, the null value is assigned to each column; an error occurs if any column that is to be deleted is not nullable. An error also occurs if there is more than one row in the result.

If the fullselect refers to columns that are to be deleted, the value of such a column in the fullselect is the value of the column in the row before the row is deleted.

WHERE

Specifies the rows to be deleted. You can omit the clause, give a search condition, or specify a cursor. For a created temporary table or a view of a created temporary table, you must omit the clause. When the clause is omitted, all the rows of the table or view are deleted.

search-condition

Is any search condition as described in [Language elements \(Db2 SQL\)](#). Each *column-name* in the search condition, other than in a subquery, must identify a column of the table or view.

The search condition is applied to each row of the table or view and the deleted rows are those for which the result of the search condition is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed just once, whereas it is possible that a subquery with a correlated reference must be executed once for each row.

Let T2 denote the object table of a DELETE statement and let T1 denote a table that is referred to in the FROM clause of a subquery of that statement. T1 must not be a table that can be affected by the DELETE on T2. Thus, the following rules apply:

- T1 must not be a dependent of T2 in a relationship with a delete rule of CASCADE or SET NULL, unless the result of the subquery is materialized before the DELETE action is executed.
- T1 must not be a dependent of T3 in a relationship with a delete rule of CASCADE or SET NULL if deletes of T2 cascade to T3.

fetch-clause

Limits the effects of the delete to a subset of the qualifying rows. See [fetch-clause \(Db2 SQL\)](#) for details.

WHERE CURRENT OF *cursor-name*

Identifies the cursor to be used in the delete operation. *cursor-name* must identify a declared cursor as explained in the description of the DECLARE CURSOR statement in [DECLARE CURSOR \(Db2 SQL\)](#). If the DELETE statement is embedded in a program, the DECLARE CURSOR statement must include *select-statement* rather than *statement-name*.

The table or view named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must be capable of being deleted. For an explanation of read-only result tables, see Read-only cursors in [DECLARE CURSOR \(Db2 SQL\)](#). Note that the object of the DELETE statement must not be identified as the object of the subquery in the WHERE clause of the SELECT statement of the cursor.

If the cursor is ambiguous and the plan or package was bound with CURRENTDATA(NO), Db2 might return an error to the application if DELETE WHERE CURRENT OF is attempted for any of the following:

- A cursor that is using block fetching
- A cursor that is using query parallelism
- A cursor that is positioned on a row that has been modified by this or another application process

When the DELETE statement is executed, the cursor must be open and positioned on a row or rowset of the result table.

- If the cursor is positioned on a single row, that row is the one deleted, and after the deletion the cursor is positioned before the next row of its result table. If there is no next row, the cursor positioned after the last row.
- If the cursor is positioned on a rowset, all rows corresponding to the rows of the current rowset are deleted, and after the deletion the cursor is positioned before the next rowset of its result table. If there is no next rowset, the cursor positioned after the last rowset.

A positioned DELETE must not be specified for a cursor that references a view on which an instead of delete trigger is defined, even if the view is an updatable view.

FOR ROW *n* OF ROWSET

Specifies which row of the current rowset is to be deleted. The corresponding row of the rowset is deleted, and the cursor remains positioned on the current rowset. If the rowset consists of a single row, or all other rows in the rowset have already been deleted, then the cursor is positioned before the next rowset of the result table. If there is no next rowset, the cursor is positioned after the last rowset.

host-variable or *integer-constant* is assigned to an integral value *k*. If *host-variable* is specified, it must be an exact numeric type with scale zero, must not include an indicator variable, and *k* must be in the range 1 - 32767. The cursor must be positioned on a rowset, and the specified value must be a valid value for the set of rows most recently retrieved for the cursor.

If the specified row cannot be deleted, an error is returned. It is possible that the specified row is within the bounds of the rowset most recently requested, but the current rowset contains less than the number of rows that were implicitly or explicitly requested when that rowset was established.

If, via a positioned delete against a sensitive static cursor that specifies a particular row of the current rowset, and that row has been identified as a delete hole (that is, a row in the result table whose corresponding row has deleted from the base table), an error is returned.

If, via a positioned delete against a sensitive static cursor that specifies a particular row of the current rowset, and that row has been identified as an update hole (that is, a row in the result table whose corresponding row has been updated so that it no longer satisfies a predicate of the SELECT statement), an error is returned.

It is possible for another application process to delete a row in the base table of the SELECT statement so that the specified row of the cursor no longer has a corresponding row in the base table. An attempt to delete such a row results in an error.

If the FOR ROW *n* OF ROWSET clause is not specified, the current position of cursor determines the rows that are affected by the statement:

- If the cursor is positioned on a single row, that row is the one deleted. After the row is deleted, the cursor is positioned before the next row of its result table. If there is no next row, the cursor positioned after the last row.
- If the cursor is positioned on a rowset, all rows corresponding to the rows of the current rowset are deleted. After the rows are deleted, the cursor is positioned before the next rowset of its result table. If there is no next rowset, the cursor positioned after the last rowset.

isolation-clause

Specifies the isolation level used when locating the rows to be deleted by the statement.

WITH

Introduces the isolation level, which may be one of the following:

RR

Repeatable read

RS

Read stability

CS

Cursor stability

The default isolation level of the statement is the isolation level of the package or plan in which the statement is bound, with the package isolation taking precedence over the plan isolation. When a package isolation is not specified, the plan isolation is the default.

SKIP LOCKED DATA

The SKIP LOCKED DATA clause specifies that rows are skipped when incompatible locks are held on the row by other transactions. These rows can belong to any accessed table that is specified in the statement. SKIP LOCKED DATA can be used only when isolation CS or RS is in effect and applies only to row level or page level locks.

For DELETE statements, SKIP LOCKED DATA can be specified only in the searched form of the DELETE statement. SKIP LOCKED DATA is ignored if it is specified when the isolation level that is in effect is repeatable read (WITH RR) or uncommitted read (WITH UR). The default isolation level of the statement depends on the isolation level of the package or plan with which the statement is bound, with the package isolation taking precedence over the plan isolation. When a package isolation is not specified, the plan isolation is the default.

QUERYNO *integer*

Specifies the number to be used for this SQL statement in EXPLAIN output and trace records. The number is used for the QUERYNO column of the plan table for the rows that contain information about this SQL statement. This number is also used in the QUERYNO column of the SYSIBM.SYSSTMT and SYSIBM.SYSPACKSTMT catalog tables.

If the clause is omitted, the number associated with the SQL statement is the statement number assigned during precompilation. Thus, if the application program is changed and then precompiled, that statement number might change.

Using the QUERYNO clause to assign unique numbers to the SQL statements in a program is helpful:

- For simplifying the use of optimization hints for access path selection
- For correlating SQL statement text with EXPLAIN output in the plan table

For more information about enabling and using optimization hints, see [Influencing access path selection \(Db2 Performance\)](#)

For information on accessing the plan table, see [Investigating SQL performance by using EXPLAIN \(Db2 Performance\)](#).

Notes

Delete operation errors:

If an error occurs during the execution of any delete operation, no changes are made. If an error occurs during the execution of a positioned delete, the position of the cursor is unchanged. However, it is possible for an error to make the position of the cursor invalid, in which case the cursor is closed. It is also possible for a delete operation to cause a rollback, in which case the cursor is closed.

Position of cursor:

If an application process deletes a row on which any of its cursors are positioned, those cursors are positioned before the next row of the result table. Let C be a cursor that is positioned before row R (as a result of an OPEN, a DELETE through C, a DELETE through some other cursor, or a searched DELETE). In the presence of an SQL data change statements that affect the base table from which R is derived, the next FETCH operation referencing C does not necessarily position C on R. For example, the operation can position C on R', where R' is a new row that is now the next row of the result table.

Locking:

Unless appropriate locks already exist, one or more exclusive locks are acquired during the execution of a successful delete operation. Until the locks are released by a commit or rollback operation, the effect of the delete operation can only be perceived by the application process that performed the deletion and the locks can prevent other application processes from performing operations on the table. Locks are not acquired when rows are deleted from a declared temporary table unless all the rows are deleted (DELETE FROM T). When all the rows are deleted from a declared temporary table, a segmented table lock is acquired on the pages for the table and no other table in the table space is affected.

Triggers:

Delete operations can cause triggers to be activated. A trigger might cause other statements to be executed or might raise error conditions that are based on the deleted rows. If a DELETE statement on a view causes an INSTEAD OF trigger to be activated, referential integrity is checked against the updates that are performed in the trigger and not against the underlying tables of the view that cause the trigger to be activated.

Triggers defined on a table for which row or column access control is also enforced:

Row permissions and column masks are not applied to the initial values of transition variables and transition tables. Row and column access control that is enforced for the triggering table is also ignored for any transition variables or transition tables that are referenced in the trigger body or that are passed as arguments to user-defined functions that are invoked in the trigger body. To ensure that no security concern exists for SQL statements in the trigger action (access to sensitive data in transition variables and transition tables, for example), the trigger must be secure. For information about securing a trigger, see [CREATE TRIGGER \(basic\) \(Db2 SQL\)](#) and [ALTER TRIGGER \(basic\) \(Db2 SQL\)](#).

Referential integrity:

If the identified table or the base table of the identified view is a parent, the rows selected must not have any dependents in a relationship with a delete rule of RESTRICT or NO ACTION. In addition, the delete operation must not cascade to descendent rows that have dependents in a relationship with a delete rule of RESTRICT or NO ACTION.

If the delete operation is not prevented by a RESTRICT or NO ACTION delete rule, the selected rows are deleted and any rows that are dependents of the selected rows are also deleted.

- The nullable columns of foreign keys in any rows that are their dependents in a relationship governed by a delete rule of SET NULL are set to the null value.
- Any rows that are their dependents in a relationship governed by a delete rule of CASCADE are also deleted, and these rules apply, in turn, to those rows.

The only difference between NO ACTION and RESTRICT is when the referential constraint is enforced. RESTRICT (IBM SQL rules) enforces the rule immediately, and NO ACTION (SQL standard rules) enforces the rule at the end of the statement. This difference matters only in the case of a searched DELETE involving a self-referencing constraint that deletes more than one row. NO ACTION might allow the DELETE to be successful where RESTRICT (if it were allowed) would prevent it.

Check constraint:

A check constraint can prevent the deletion of a row in a parent table when there are dependents in a relationship with a delete rule of SET NULL. If deleting a row in the parent table would cause a column in a dependent table to be set to null and there is a check constraint that specifies that the column must not be null, the row is not deleted.

Referential constraints defined on a table for which row or column access control is enforced:

Row and column access controls do not effect referential constraints.

Nesting user-defined functions or stored procedures:

A DELETE statement can implicitly or explicitly refer to user-defined functions or stored procedures. This is known as *nesting* of SQL statements. A user-defined function or stored procedure that is nested within the DELETE must not access the table from which you are deleting rows.

Indexes with VARBINARY columns:

If the identified table has an index on a VARBINARY column or a column that is a distinct type that is based on VARBINARY data type, that index column cannot specify the DESC attribute. To use the SQL data change operation on the identified table, either drop the index or alter the data type of the column to BINARY and then rebuild the index.

Number of rows deleted:

Except as noted below, a delete operation sets SQLERRD(3) in the SQLCA to the number of deleted rows. This number does not include any rows that were deleted as a result of a CASCADE delete rule or a trigger.

DELETE FROM T without a WHERE clause deletes all rows of T. If a table T is contained in a segmented table space and is not a parent table, this deletion will be performed without accessing T. The SQLERRD(3) field is set to -1. (For a complete description of the SQLCA, including exceptions to the above, see [SQL communication area \(SQLCA\) \(Db2 SQL\)](#)).

Rules for positioned DELETE with SENSITIVE STATIC scrollable cursor:

When a SENSITIVE STATIC scrollable cursor has been declared, the following rules apply:

- *Delete attempt of delete holes or update holes.* If, with a positioned delete against a SENSITIVE STATIC scrollable cursor, an attempt is made to delete a row that has been identified as a delete hole (that is, a row in the result table whose corresponding row has been deleted from the base table), an error occurs.

If an attempt is made to delete a row that has been identified as an update hole (that is, a row in the result table whose corresponding row has been updated so that it no longer satisfies the predicate of the SELECT statement), an error occurs.

- *Delete operations.* Positioned delete operations with SENSITIVE STATIC scrollable cursors perform as follows:
 1. The SELECT list items in the target row of the base table of the cursor are compared with the values in the corresponding row of the result table (that is, the result table must still agree with the base table). If the values are not identical, the delete operation is rejected and an error occurs. The operation can be attempted again after a successful FETCH SENSITIVE has occurred for the target row.
 2. The WHERE clause of the SELECT statement is re-evaluated to determine whether the current values in the base table still satisfy the search criteria. The values in the SELECT list are compared to determine that these values have not changed. If the WHERE clause evaluates as true, and the values in the SELECT list have not changed, the delete operation is allowed to proceed. Otherwise, an error occurs, the delete operation is rejected, and an update hole appears in the cursor.
 3. After the base table row is successfully deleted, the temporary result table is updated and the row is marked as a delete hole.
- *Rollback of delete holes.* Delete holes are usually permanent. Once a delete hole is identified, it remains a delete hole until the cursor is closed. However, if a positioned delete using *this* cursor actually caused the creation of the hole (that is, this cursor was used to make the changes that

resulted in the hole) and the delete was subsequently rolled back, then the row is no longer considered a delete hole.

- *Result table.* Any deletes, either positioned or searched, to rows of the base table on which a SENSITIVE STATIC scrollable cursor is defined are reflected in the result table if a positioned update or positioned delete is attempted with the scrollable cursor. A SENSITIVE STATIC scrollable cursor sees these deletes when a FETCH SENSITIVE is attempted.

If the FOR ROW *n* OF ROWSET clause is not specified, the entire rowset fetched by the most recent FETCH statement that returned data for the specified cursor is deleted.

Referencing columns that will be updated:

If a cursor uses FETCH statements to retrieve columns that will be updated later, specify FOR UPDATE OF when you select the columns. Then specify WHERE CURRENT OF in the subsequent UPDATE or DELETE statements. These clauses prevent Db2 from selecting access through an index on the columns that are being updated, which might otherwise cause Db2 to read the same row more than once.

For more information, see [Updating previously retrieved data \(Db2 Application programming and SQL\)](#).

Deleting rows from a table with multilevel security:

When you delete rows from a table with multilevel security, Db2 compares the security label of the user (the primary authorization ID) to the security label of the row. The delete proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is deleted.
- If the security label of the user dominates the security label of the row, the user's write-down privilege determines the security the result of the DELETE statement:
 - If the user has write-down privilege or write-down control is not enabled, the row is deleted.
 - If the user does not have write-down privilege and write-down control is enabled, the row is not deleted.
- If the security label of the row dominates the security label of the user, the row is not deleted.

Deleting rows from a table for which row and column access control is enforced:

When a DELETE statement is issued for a table for which row access control is enforced, the rules specified in the row permissions affect whether a row can be deleted. Typically those rules are based on the authorization ID or role of the process.

A table for which row access control is enforced has at least one row permission, the default row permission that prevents all access to the table. When multiple row permissions are defined and enabled for a table, a row access control search condition is derived by using the logical OR operator to the search condition in each enabled permission. This row access control search condition is applied to the table to determine which rows are accessible to the authorization ID or role of the DELETE statement. If the WHERE clause is specified in the DELETE statement, the user-specified predicates are applied on the accessible rows to determine the rows to be deleted. If there is no WHERE clause, the accessible rows are the rows to be deleted.

If there are rows to be deleted, and there is a DELETE trigger for the table, the trigger is activated.

When a DELETE statement is issued for a table for which column access control is enforced, column masks do not affect the DELETE statement.

The preceding rules are not applicable to *include-columns*. *include-columns* are subject to the rules for the select list because they are not the columns of the object table of the DELETE statement.

Other SQL statements in the same unit of work:

The following statements cannot follow a DELETE statement in the same unit of work:

- An ALTER TABLE statement that changes the data type of a column (ALTER COLUMN SET DATA TYPE)

- An ALTER INDEX statement that changes the padding attribute of an index with varying-length columns (PADDED to NOT PADDED or vice versa)
- A CREATE TABLE statement that creates an accelerator-only table.
- An INSERT, UPDATE or DELETE statement that updates an accelerator-only table from a different accelerator

Considerations for a system-period temporal table:

If the DELETE statement has a search condition that contains a correlated subquery that references the history table (explicitly referencing the name of the history table or implicitly referenced through the use of a period specification in the FROM clause), the deleted rows that are stored as historical rows are potentially visible for delete operations for the rows that are subsequently processed for the statement.

The mass delete operation is not used for a DELETE statement that does not contain a search condition if the table is defined as a system-period temporal table.

If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, the underlying target of the DELETE statement cannot be a system-period temporal table. This restriction applies regardless of whether the system-period temporal table is directly or indirectly referenced.

Considerations for a history table:

When a row of a system-period temporal table is deleted, a historical copy of the row is inserted into the corresponding history table and the end timestamp of the historical row is captured in the form of a system determined value that corresponds to the time of the data change operation. If the value of the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME built-in global variable at the time of the data change operation is null, the value is generated using a reading of the time-of-day clock during execution of the first data change statement in the unit of work that requires a value to be assigned to a row-begin column or transaction-start-ID column in a table, or a row in a systemperiod temporal table is deleted. Otherwise, the value is assigned from the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME built-in global variable at the time of the data change operation. If a conflicting transaction is updating the same row in the system-period temporal table and the row that is to be inserted into the associated history table will have a value for the end column that is greater than the value of the corresponding begin column, an error is returned.

Considerations for an application-period temporal table:

A DELETE statement that contains a FOR PORTION OF BUSINESS_TIME clause for an application-period temporal table indicates the two points in time between which the specified delete operations are effective.

Suppose that FOR PORTION OF BUSINESS_TIME is specified and the period value for a row is only partially contained in the period that is specified from *value1* up to *value2* or between *value1* and *value2*. (The period value for a row is specified by the values of the begin column and end column.) In this case, the row is deleted and one or two rows are automatically inserted to represent the portion of the row that is not deleted. For each row that is automatically inserted as a result of a delete operation on the table, new values are generated for each generated column in the application-period temporal table. If a generated column is defined as part of a unique or primary key, parent key in a referential constraint, or unique index, an automatic insert might violate a constraint or index. In this case, an error is returned.

When an application-period table is the target of a DELETE statement and the value in effect for the CURRENT TEMPORAL BUSINESS_TIME special register is not the null value, Db2 adds the following additional predicates to the statement:

- inclusive-exclusive period:

```
bt_begin <= CURRENT TEMPORAL BUSINESS_TIME AND
bt_end > CURRENT TEMPORAL BUSINESS_TIME
```

- inclusive-inclusive period:

```
bt_begin <= CURRENT TEMPORAL BUSINESS_TIME AND
bt_end >= CURRENT TEMPORAL BUSINESS_TIME
```

In the preceding code, `bt_begin` and `bt_end` are the begin and end columns of the `BUSINESS_TIME` period of the target table of the `DELETE` statement.

Deleting rows from archive-enabled tables:

If the target of the `DELETE` statement is an archive-enabled table, existing rows in the associated archive table are not affected.

When a row of an archive-enabled table is deleted, the effect on the associated archive table is determined by the setting of the `SYSIBMADM.MOVE_TO_ARCHIVE` global variable. If the global variable is set to `Y`, a copy of a deleted row is inserted into the associated archive table. Otherwise, a copy of a deleted row is not inserted into the associated archive table.

A data change statement cannot reference an archive-enabled table when a system-period temporal table or application-period temporal table is also referenced.

Syntax alternatives:

For compatibility with other SQL implementations, the `FROM` keyword that immediately follows the `DELETE` keyword can be omitted.

Examples

Assume that the statements in the examples are embedded in PL/I programs.

Example 1

From the table `DSN8D10.EMP` delete the row on which the cursor `C1` is currently positioned.

```
EXEC SQL DELETE FROM DSN8D10.EMP WHERE CURRENT OF C1;
```

Example 2

From the table `DSN8D10.EMP`, delete all rows for departments `E11` and `D21`.

```
EXEC SQL DELETE FROM DSN8D10.EMP  
WHERE WORKDEPT = 'E11' OR WORKDEPT = 'D21';
```

Example 3

From employee table `X`, delete the employee who has the most absences.

```
EXEC SQL DELETE FROM EMP X  
WHERE ABSENT = (SELECT MAX(ABSENT) FROM EMP Y  
WHERE X.WORKDEPT = Y.WORKDEPT);
```

Example 4

Assuming that cursor `CS1` is positioned on a rowset consisting of 10 rows of table `T1`, delete all 10 rows in the rowset.

```
EXEC SQL DELETE FROM T1 WHERE CURRENT OF CS1;
```

Example 5

Assuming cursor `CS1` is positioned on a rowset consisting of 10 rows of table `T1`, delete the fourth row of the rowset.

```
EXEC SQL DELETE FROM T1 WHERE CURRENT OF CS1 FOR ROW 4 OF ROWSET;
```

Example 6

Delete rows in table `T1` if the value for column `COL2` matches the cardinality of array `INTA`. The array `INTA` is specified as an argument for the `CARDINALITY` function in the `DELETE` statement.

```
CREATE TYPE INTARRAY AS INTEGER ARRAY[6];  
CREATE VARIABLE INTA AS INTARRAY;  
SET INTA = ARRAY[1, 2, 3, 4, 5];  
CREATE TABLE T1 (COL1 CHAR(7), COL2 INT);  
INSERT INTO T1 VALUES('abc', 10);  
DELETE FROM T1 WHERE COL2 = CARDINALITY(INTA);
```

Example 7

Delete only 3 rows from table T1 where the value of column C2 is greater than 10.

```
DELETE FROM T1
WHERE C2 > 10
FETCH FIRST 3 ROWS ONLY;
```

SET CURRENT TEMPORAL BUSINESS_TIME

The SET CURRENT TEMPORAL BUSINESS_TIME statement changes the value of the CURRENT TEMPORAL BUSINESS_TIME special register.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax

```

-- SET -- CURRENT TEMPORAL BUSINESS_TIME -- (= --
--                                     -- NULL --
--                                     -- expression --
--

```

Description

NULL

Specifies the null value.

expression

Specifies an expression that returns the null value or the value of one of the following built-in data types:

- Timestamp
- Character string
- Graphic string

If the expression is a character or graphic string, it must meet the following requirements:

- It must not be a CLOB or DBCLOB.
- The value of the expression must be a valid character-string or graphic-string representation of a timestamp.
- The result of the expression must be castable to `TIMESTAMP(12)`.

expression can contain any of the following supported operands:

- Constant
- Special register
- Variable (host variable, SQL parameter, SQL variable, or global variable)
- Scalar function whose arguments are supported operands. The scalar function must not be `AI_ANALOGY`, `AI_COMMONALITY`, `AI_SEMANTIC_CLUSTER` or `AI_SIMILARITY`.
- CAST specification where the cast operand is a supported operand
- Expression that uses arithmetic operators and operands

For more information, see:

[String representations of datetime values \(Db2 SQL\)](#)

[Casting between data types \(Db2 SQL\)](#)

Notes

Transactions

The SET CURRENT TEMPORAL BUSINESS_TIME statement is not a committable operation. The ROLLBACK statement has no effect on CURRENT TEMPORAL BUSINESS_TIME.

Effects on other special registers

The setting of the CURRENT TEMPORAL BUSINESS_TIME special register does not affect other special registers, such as the CURRENT DATE and CURRENT TIMESTAMP special registers.

Examples

Example of setting the special register to a valid value

Both of the following statements set the CURRENT TEMPORAL BUSINESS_TIME special register to '2008-01-06-00.00.00.000000000000'.

```
SET CURRENT TEMPORAL BUSINESS_TIME = TIMESTAMP('2008-01-01') + 5 DAYS ;  
SET CURRENT TEMPORAL BUSINESS_TIME = '2008-01-06-00.00.00.000000000000' ;
```

Example of how setting the special register affects subsequent SQL statements

In the following example, the first statement sets the CURRENT TEMPORAL BUSINESS_TIME special register to last month. Assume that table att1 is an application-period temporal table. The setting of the CURRENT TEMPORAL BUSINESS_TIME special register affects the update of att1.

```
SET CURRENT TEMPORAL BUSINESS_TIME = CURRENT TIMESTAMP - 1 MONTH  
UPDATE att1 SET c1 = 5 WHERE pk = 100
```

Assume that the att1 table has columns bt_begin and bt_end to indicate the beginning and end of the BUSINESS_TIME period. In this example, Db2 interprets the UPDATE statement as follows:

```
UPDATE att1 SET c1 = 5 WHERE pk = 100  
AND bt_begin <= CURRENT TEMPORAL BUSINESS_TIME  
AND bt_end > CURRENT TEMPORAL BUSINESS_TIME
```

Example of setting the special register so that it does not affect subsequent SQL statements

The following statement sets the CURRENT TEMPORAL BUSINESS_TIME special register to the null value. Subsequent SQL statements that reference application-period temporal tables are not affected by the CURRENT TEMPORAL BUSINESS_TIME special register.

```
SET CURRENT TEMPORAL BUSINESS_TIME = NULL
```

Related concepts

[Data types \(Db2 SQL\)](#)

Related reference

[CURRENT TEMPORAL BUSINESS_TIME \(Db2 SQL\)](#)

SET CURRENT TEMPORAL SYSTEM_TIME

The SET CURRENT TEMPORAL SYSTEM_TIME statement changes the value of the CURRENT TEMPORAL SYSTEM_TIME special register.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax

```
➡ SET — CURRENT TEMPORAL SYSTEM_TIME —=— NULL —➡  
                                     |  
                                     expression
```

Description

NULL

Specifies the null value.

expression

Specifies an expression that returns the null value or the value of one of the following built-in data types:

- Timestamp
- Character string
- Graphic string

If the expression is a character or graphic string, it must meet the following requirements:

- It must not be a CLOB or DBCLOB.
- The value of the expression must be a valid character-string or graphic-string representation of a timestamp.
- The result of the expression must be castable to `TIMESTAMP(12)`.

expression can contain any of the following supported operands:

- Constant
- Special register
- Variable (host variable, SQL parameter, SQL variable, or global variable)
- Scalar function whose arguments are supported operands
- Scalar function whose arguments are supported operands. The scalar function must not be `AI_ANALOGY`, `AI_COMMONALITY`, `AI_SEMANTIC_CLUSTER` or `AI_SIMILARITY`.
- `CAST` specification where the cast operand is a supported operand
- Expression that uses arithmetic operators and operands

For more information, see:

[String representations of datetime values \(Db2 SQL\)](#)

[Casting between data types \(Db2 SQL\)](#)

Notes

Transactions

The `SET CURRENT TEMPORAL SYSTEM_TIME` statement is not a committable operation. The `ROLLBACK` statement has no effect on `CURRENT TEMPORAL SYSTEM_TIME`.

Effects on other special registers

The setting of the `CURRENT TEMPORAL SYSTEM_TIME` special register does not affect other special registers, such as the `CURRENT DATE` and `CURRENT TIMESTAMP` special registers.

Examples

Example of setting the special register to a valid value

Both of the following statements set the CURRENT TEMPORAL SYSTEM_TIME special register to '2008-01-06-00.00.00.000000000000'.

```
SET CURRENT TEMPORAL SYSTEM_TIME = TIMESTAMP('2008-01-01') + 5 DAYS;  
SET CURRENT TEMPORAL SYSTEM_TIME = '2008-01-06-00.00.00.000000000000';
```

Example of setting the special register so that it does not affect subsequent SQL statements

The following statement sets the CURRENT TEMPORAL SYSTEM_TIME special register to the null value. Subsequent SQL statements that reference system-period temporal tables are not affected by the CURRENT TEMPORAL SYSTEM_TIME special register.

```
SET CURRENT TEMPORAL SYSTEM_TIME = NULL
```

Related concepts

[Data types \(Db2 SQL\)](#)

Related reference

[CURRENT TEMPORAL SYSTEM_TIME \(Db2 SQL\)](#)

UPDATE

The UPDATE statement updates the values of specified columns in rows of a table or view. Updating a row of a view updates a row of its base table if no INSTEAD OF UPDATE trigger is defined for this view. If such a trigger is defined, the trigger is activated instead.

The table or view can exist at the current server or at any Db2 subsystem with which the current server can establish a connection.

There are two forms of this statement:

- The *searched* UPDATE form is used to update one or more rows optionally determined by a search condition.
- The *positioned* UPDATE form specifies that one or more rows corresponding to the current cursor position are to be updated.

Invocation

This statement can be embedded in an application program or issued interactively. A positioned UPDATE can be embedded in an application program. Both forms are executable statements that can be dynamically prepared.

Authorization

Authority requirements depend on whether the object identified in the statement is a user-defined table, a catalog table for which updates are allowed, or a view, and whether SQL standard rules are in effect:

When a user-defined table is identified: The privilege set must include at least one of the following:

- DATAACCESS authority
- The UPDATE privilege on the table
- The UPDATE privilege on each column to be updated
- Ownership of the table
- DBADM authority on the database that contains the table
- SYSADM authority

If the database is implicitly created, the database privileges must be on the implicit database or on DSNUB04.

When a catalog table is identified: The privilege set must include at least one of the following:

- ACCESSCTRL authority
- DATAACCESS authority
- The UPDATE privilege on each column to be updated
- DBADM authority on the catalog database
- Installation SYSOPR authority
- SYSCTRL authority
- SYSADM authority
- SYSADM authority
- System DBADM authority

When a view is identified: The privilege set must include at least one of the following:

- DATAACCESS authority
- SYSADM authority
- UPDATE privilege on the view
- UPDATE privilege on each column to be updated

If the *expression* in the *assignment-clause* contains a reference to a column of the table or view, or if the *search-condition* in a searched UPDATE contains a reference to a column of the table or view, the privilege set must include at least one of the following:

- The SELECT privilege on the table or view
- Ownership of the table or view
- DBADM authority on the database that contains the table, if the target is a table and that table that is not a catalog table
- DATAACCESS
- SYSADM authority

When FOR PORTION OF BUSINESS_TIME is specified: The privilege set must include at least one of the following:

- The UPDATE privilege on the columns of the BUSINESS_TIME period
- The UPDATE privilege on the table
- Ownership of the table or view
- DBADM authority on the database that contains the table, if the target is a table and that table that is not a catalog table
- DATAACCESS
- SYSADM authority

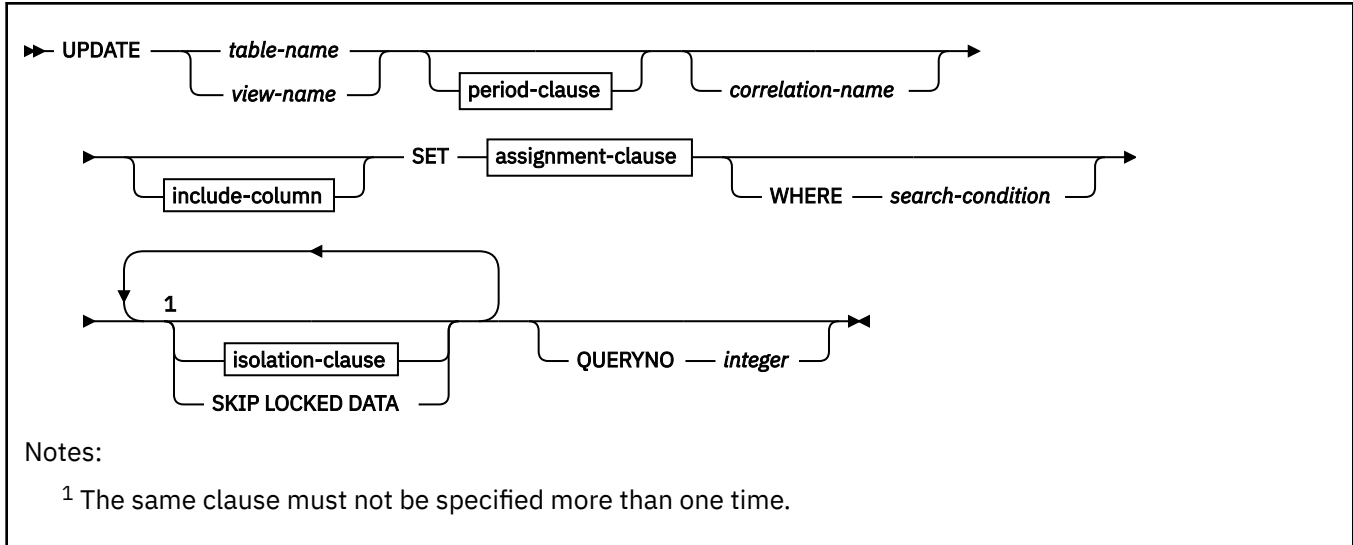
If the *search-condition* in a searched UPDATE includes a subquery, or if the *assignment-clause* includes a *scalar-fullselect* or a *row-fullselect*, see [Authorization for queries \(Db2 SQL\)](#) for an explanation of the authorization required.

The owner of a view, unlike the owner of a table, might not have UPDATE authority on the view (or might have UPDATE authority without being able to grant it to others). The nature of the view itself can preclude its use for UPDATE. For more information, see the discussion of authority in [CREATE VIEW \(Db2 SQL\)](#).

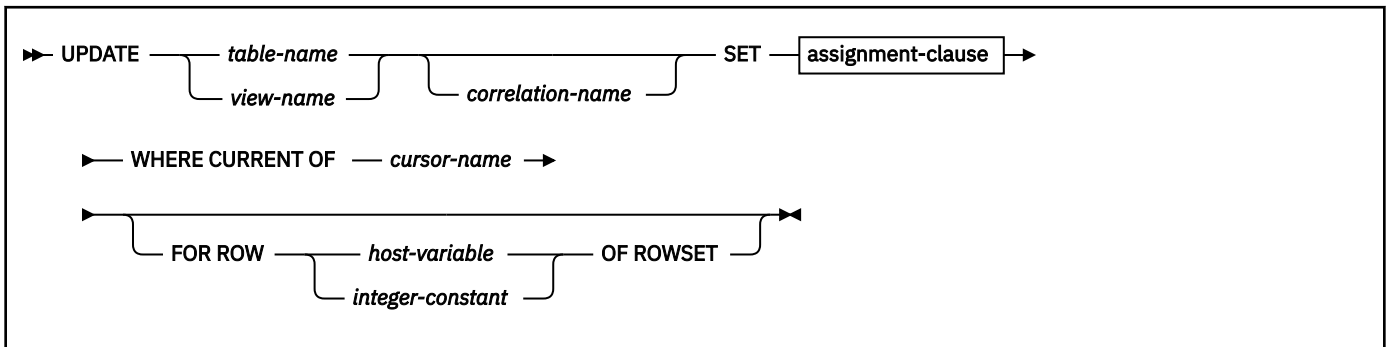
If the statement attempts to update a row in the SYSIBM.SYSAUDITPOLICIES catalog table that is subject to a tamper-proof audit policy, additional RACF authorization is required. During statement execution, the primary authorization ID or one of the groups associated with the primary authorization ID must be authorized to access the tamper-proof audit policy profile in RACF. For more information on the authorization rules, see [Db2 audit policy \(Managing Security\)](#).

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in *Dynamic preparation and execution (Db2 SQL)*. (For more information on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see *Authorization IDs and dynamic SQL (Db2 SQL)*).

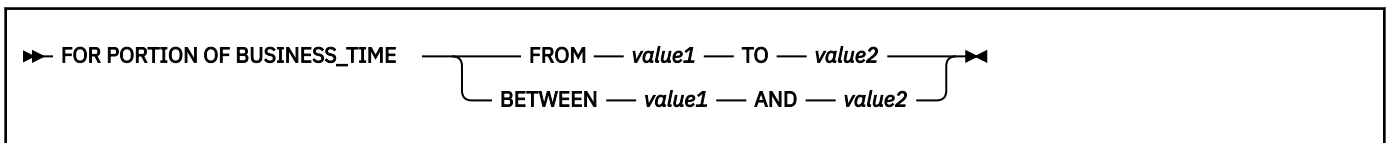
searched update:



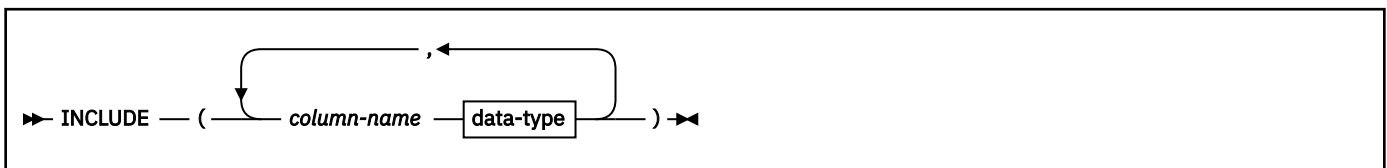
positioned update:



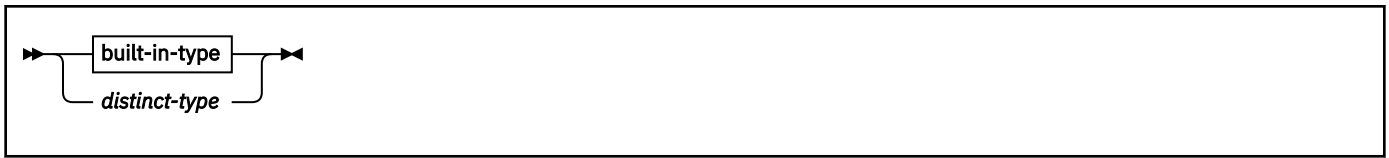
period-clause:



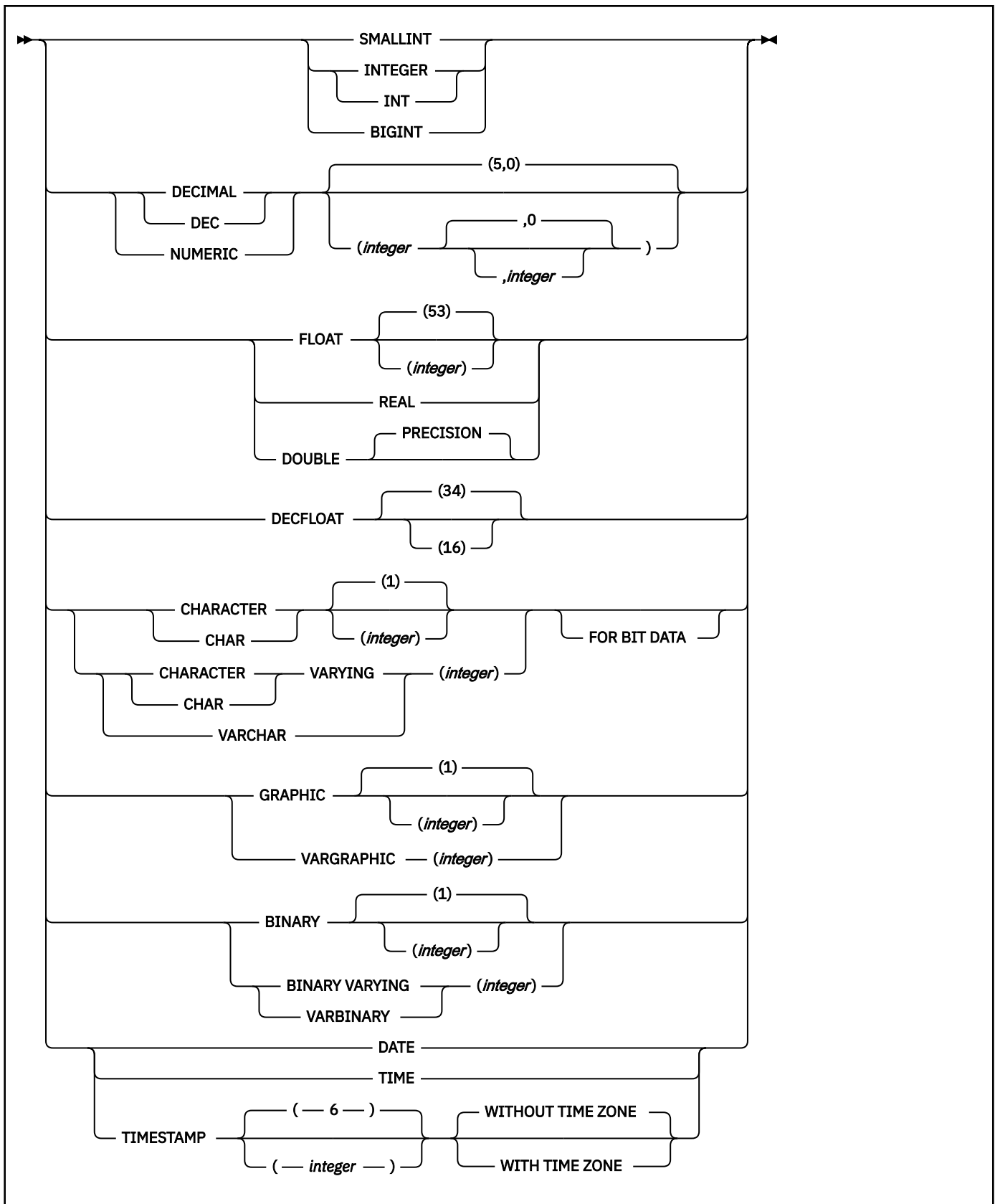
include-column:



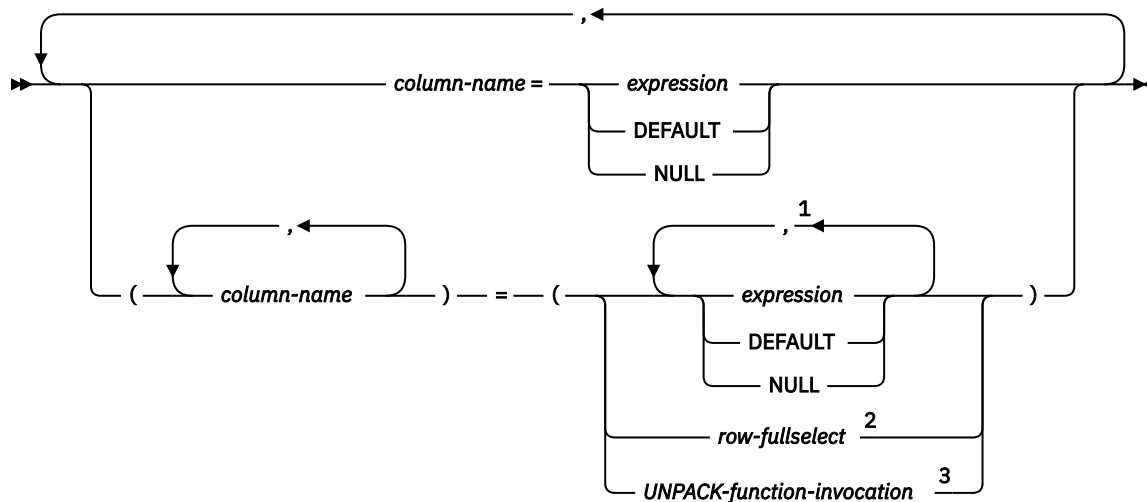
data-type:



built-in-type:



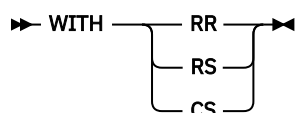
assignment clause:



Notes:

- ¹ The number of *expressions*, *DEFAULT*, and *NULL* keywords must match the number of *column-names*. *Expressions* must not refer to *UNPACK-function-invocation*.
- ² The number of columns in the select list must match the number of *column-names*.
- ³ The number of items returned from *UNPACK-function-invocation* must match the number of column names.

isolation-clause:



Description

table-name or view-name

Identifies the object of the UPDATE statement. The name must identify a table or view that exists at the Db2 subsystem that is identified by the implicitly or explicitly specified location name. The name must not identify one of the following tables:

- An auxiliary table
- A created temporary table or a view of a created temporary table
- A catalog table with no updatable columns or a view of a catalog table with no updatable columns
- A directory table
- A read-only view that has no INSTEAD OF trigger defined for its update operations. (For a description of a read-only view, see [CREATE VIEW \(Db2 SQL\)](#).)
- A system-maintained materialized query table
- A table that is implicitly created for an XML column
- An archive-enabled table if any of the following conditions are true:
 - The SYSIBMADM.MOVE_TO_ARCHIVE global variable is set to Y.
 - The SYSIBMADM.GET_ARCHIVE global variable is set to Y, the ARCHIVESENSITIVE bind option is set to YES, and the operation is a positioned update.

In an IMS or CICS application, the Db2 subsystem that contains the identified table or view must be a remote server that supports two-phase commit.

A catalog table or a view of a catalog table can be identified if every column identified in the SET clause is an updatable column. If a column of a catalog table is updatable, its description in [Db2 catalog tables \(Db2 SQL\)](#) indicates that the column can be updated. If the object table is SYSIBM.SYSSTRINGS, any column other than IBMREQD can be updated, but the rows that are selected for update must be rows that are provided by the user (the value of the IBMREQD column is N) and only certain values can be specified as explained in [How an entry in SYSIBM.SYSSTRINGS works with character conversion \(Db2 Installation and Migration\)](#).

period-clause

Specifies that a period clause applies to the target of the update operation. The same period name must not be specified more than one time. If the target of the update operation is a view:

- The FROM clause of the outer fullselect of the view definition must include a reference, directly or indirectly, to an application-period temporal table.
- The result table of the outer fullselect of the view definition must include, explicitly or implicitly, the start and end columns of the BUSINESS_TIME period.
- An INSTEAD OF trigger must not be defined for the view.

FOR PORTION OF BUSINESS_TIME

Specifies that the update only applies to row values for the portion of the BUSINESS_TIME period in the row that is specified by the period clause. BUSINESS_TIME must be a period that is defined on the table.

FOR PORTION OF BUSINESS_TIME must not be specified if the value of the CURRENT TEMPORAL BUSINESS_TIME special register is not NULL when the BUSTIMESENSITIVE bind option is set to YES.

FROM *value1* TO *value2*

Specifies that the update applies to rows for the period that is specified from *value1* to *value2*. No rows are updated if *value1* is greater than or equal to *value2* or if *value1* or *value2* is the null value.

This clause must not be specified for an inclusive-inclusive period.

For the period condition that is specified with FROM *value1* TO *value2*, the period that is specified with *period-name* in a row of the target update:

- Overlaps the beginning of the specified period if the value of the begin column is less than *value1* and the value of the end column is greater than *value1*.
- Overlaps the end of the specified period if the value of the end column is greater than or equal to *value2* and the value of the begin column is less than *value2*.
- Is fully contained within the specified period if the value for the begin column is greater than or equal to *value1* and the value for the end column is less than or equal to *value2*.
- Is not contained in the period if both columns of *period-name* are less than *value1* or greater than or equal to *value2*.
- Is partially contained in the specified period if the period in the row overlaps the beginning of the specified period or the end of the specified period, but not both.
- Fully overlaps the specified period if the period in the row overlaps both the beginning and the end of the specified period.

If the period, *period-name* in a row is not contained in the specified period, the row is not updated. Otherwise, the update is applied based on the specification of PORTION OF and how the values in the columns of *period-name* overlap the specified period as follows:

- If the period, *period-name* in a row is fully contained within the specified period, the row is updated and the values of the begin column and end column of *period-name* are unchanged.
- If the period, *period-name* in a row is partially contained in the specified period and overlaps the beginning of the specified period:

- The row is updated. In the updated row, the value of the begin column is set to *value1* and the value of the end column is the original value of the end column.
- An additional row is inserted using the original values from the row, except that the end column is set to *value1*, and new values are used for other generated columns.
- If the period, *period-name* in a row is partially contained in the specified period and overlaps the end of the specified period:
 - The row is updated. In the updated row, the value of the begin column is the original value of the begin column and the end column is set to *value2*.
 - An additional row is inserted using the original values from the row, except that the begin column is set to *value2*, and new values are used for other generated columns.
- If the period, *period-name* in a row fully overlaps the specified period:
 - The row is updated. In the updated row, the value of the begin column is set to *value1* and the value of the end column is set to *value2*.
 - An additional row is inserted using the original values from the row, except that the end column is set to *value1*, a column defined as DATA CHANGE OPERATION is set to 'I', and new values are used for other generated columns.
 - An additional row is inserted using the original values from the row, except that the begin column is set to *value2*, a column defined as DATA CHANGE OPERATION is set to 'I', and new values are used for other generated columns.

Any existing update triggers are activated for the updated rows and any existing insert triggers are activated for rows that are implicitly inserted.

BETWEEN *value1* AND *value2*

Specifies that the update operation applies to rows for the period that is specified from *value1* up to and including *value2*. No rows are updated if *value1* is greater than *value2*, or if *value1* or *value2* is the null value. This clause must not be specified for an inclusive-exclusive period.

For the period clause that is specified with BETWEEN *value1* AND *value2*, period *period-name* in a row in the target of the update operation:

- Overlaps the beginning of the specified period if the value of the begin column is less than *value1* and the value of the end column is greater than *value1*.
- Overlaps the end of the specified period if the value of the end column is greater than or equal to *value2* and the value of the begin column is less than *value2*.
- Is fully contained within the specified period if the value for the begin column for *period-name* in the row is greater than or equal to *value1* and the value for the corresponding end column in the row is less than or equal to *value2*.
- Is partially contained in the specified period if the row overlaps the beginning of the specified period or the end of the specified period, but not both.
- Fully overlaps the specified period if the period in the row overlaps the beginning of the specified period and overlaps the end of the specified period.
- Is not contained in the period if both columns of *period-name* are less than *value1* or greater than *value2*.

If the period *period-name* in a row is not contained in the specified period, the row is not updated. Otherwise, the update operation is based on the following items:

- The specification of the PORTION OF clause.
- How the values in the columns of *period-name* overlap the specified period.
- *spu* (smallest period unit), which depends on the data type of the columns of the period as follows:
 - For a period containing DATE columns, spu is 1 day.
 - For a period containing TIMESTAMP(6) columns, spu is 1 microsecond.

Based on those items, the update operation is applied as follows:

- If the period *period-name* in a row is fully contained within the specified period, the row is updated and the values of the begin column and end column of *period-name* are unchanged.
- If the period *period-name* in a row is partially contained in the specified period and overlaps the beginning of the specified period:
 - The row is updated. In the updated row the value of the begin column is set to *value1* and the value of the end column is the original value of the end column.
 - A row is inserted using the original values from the row, except that the end column is set to *value1* - spu, and new values are used for other generated columns.
- If the period *period-name* in a row is partially contained in the specified period and overlaps the end of the specified period:
 - The row is updated. In the updated row the value of the begin column is the original value of the begin column and the end column is set to *value2*
 - A row is inserted using the original values from the row, except that the begin column is set to *value2* + spu, and new values are used for other generated columns.
- If the period *period-name* in a row fully overlaps the specified period:
 - The row is updated. In the updated row the value of the begin column is set to *value1* and the value of the end column is set to *value2*.
 - A row is inserted using the original values from the row, except that the end column is set to *value1* - spu, a column defined as DATA CHANGE OPERATION is set to 'I', and new values are used for other generated columns.
 - An additional row is inserted using the original values from the row, except that the begin column is set to *value2* + spu, a column defined as DATA CHANGE OPERATION is set to 'I', and new values are used for other generated columns.

value1, value2

Specifies expressions that return a value of a built-in data type. The result of each expression must be comparable to the data type of the columns of the specified period. See the comparison rules described in [Assignment and comparison \(Db2 SQL\)](#). Each expression can contain any of the following supported operands:

- A constant
- A special register
- A variable
- An array element specification
- A built-in scalar function whose arguments are supported operands. The scalar function must not be AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.
- A CAST specification where the cast operand is a supported operand
- An expression that uses arithmetic operators and operands

Each expression must not have a timestamp precision that is greater than the precision of the columns for the period.

If the begin and end columns of the period are defined as `TIMESTAMP WITHOUT TIME ZONE`, each expression must not return a value of a timestamp with a time zone.

A period clause for a view must not contain an untyped parameter marker.

correlation-name

Can be used within *search-condition* or *assignment-clause* to designate the table or view. (For an explanation of *correlation-name*, see [Correlation names \(Db2 SQL\)](#).)

include-column

Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the result table of the UPDATE statement when it is nested in the FROM clause of the outer fullselect

that is used in a subselect, SELECT statement, or in a SELECT INTO statement. The included columns are appended to the end of the list of columns that is identified by *table-name* or *view-name*. If no value is assigned to a column that is specified by an *include-column*, a NULL value is returned for that column.

INCLUDE

Introduces a list of columns that are to be included in the result table of the UPDATE statement. The included columns are only available if the UPDATE statement is nested in the FROM clause of a SELECT statement or a SELECT INTO statement.

column-name

Specifies the name for a column of the result table of the UPDATE statement that is not the same name as another included column nor a column in the table or view that is specified in *table-name* or *view-name*.

data-type

Specifies the data type of the included column. The included columns are nullable.

built-in-type

Specifies a built-in data type. See [CREATE TABLE \(Db2 SQL\)](#) for a description of each built-in type.

The CCSID 1208 and CCSID 1200 clauses must not be specified for an INCLUDE column.

distinct-type

Specifies a distinct type. Any length, precision, or scale attributes for the column are those of the source type of the distinct type as specified by using the CREATE TYPE statement.

SET

Introduces the assignment of values to column names.

assignment-clause

If *row-fullselect* is specified, the number of columns in the result of *row-fullselect* must match the number of *column-names* that are specified. If *row-fullselect* is not specified, the number of expressions, and NULL and DEFAULT keywords must match the number of *column-names* that are specified.

column-name

Identifies a column that is to be updated. *column-name* must identify a column of the specified table or view. If extended indicators are not enabled, that column must be an updatable column. The column must not identify a generated column or a view column where the column is derived from a scalar function, constant, or expression. *column-name* can also identify an INCLUDE column that must not be qualified. The same column name must not be specified more than once.

A column that is defined as part of a BUSINESS_TIME period must not be specified if the UPDATE statement contains a *period-clause*.

Assignments to included columns are only processed when the UPDATE statement is nested in the FROM clause of a SELECT statement or a SELECT INTO statement. There must be at least one assignment clause that specifies a *column-name* that is not an INCLUDE column. The null value is returned for an included column that is not set by using an explicit SET clause.

For a positioned update, allowable column names can be further restricted to those in a certain list. This list appears in the FOR UPDATE clause of the SELECT statement for the associated cursor. The clause can be omitted by using the conditions that are described in [Positioned updates of columns \(Db2 SQL\)](#).

A view column that is derived from the same column as another column of the view can be updated, but both columns cannot be updated in the same UPDATE statement.

expression

Indicates the new value of the column. The *expression* is any expression of the type described in [Expressions \(Db2 SQL\)](#). It must not include an aggregate function.

A *column-name* in an expression must identify a column of the table or view. For each row that is updated, the value of the column in the expression is the value of the column in the row before the row is updated.

If *expression* is a single host variable, the host variable can include an indicator with an extended indicator value. If extended indicators are enabled, and an expression in the assignment clause is not a single host variable, the extended indicator values of DEFAULT and UNASSIGNED must not be used.

A CAST specification can be used if either of the following is true:

- The target column is defined as nullable.
- The target column is defined as NOT NULL with a non-null default, the source of the CAST specification is a single host variable, and the data attributes (data type, length, precision, and scale) of the host variable are the same as the result of the cast specification.

DEFAULT

Specifies that the default value is used based on how the corresponding column is defined in the table. DEFAULT must not be specified for a ROWID column. The value that is assigned depends on how the column is defined.

- If the column is a generated expression, the column value will be generated by the Db2 subsystem based on the result of the expression.
- If the column is an identity column, row change timestamp column, row-begin column, row-end column, or transaction-start-ID column, the Db2 subsystem will generate a new value.
- If the column is defined using the WITH DEFAULT clause, the value is set to the default that is defined for the column.
- If the column is defined without specifying the WITH DEFAULT clause, the GENERATED clause, or the NOT NULL clause, the value is NULL.
- If the column is specified in the INCLUDE column list, the column value is set to null.

DEFAULT must be specified for a column that was defined as GENERATED ALWAYS. A valid value can be specified for a column that was defined as GENERATED BY DEFAULT.

If the column is defined using the NOT NULL clause and the GENERATED clause is not used, or the WITH DEFAULT clause is not used, the DEFAULT keyword cannot be specified for that column.

NULL

Specifies the null value as the new value of the column. Specify NULL only for nullable columns.

row-fullselect

Specifies a fullselect that returns a single row. The column values are assigned to each of the corresponding *column-names*. If the fullselect returns no rows, the null value is assigned to each column; an error occurs if any column to be updated is not nullable. An error also occurs if there is more than one row in the result.

For a positioned update, if the table or view that is the object of the UPDATE statement is used in the fullselect, a column from the instance of the table or view in the fullselect cannot be the same as *column-name*, a column being updated.

If the fullselect refers to columns to be updated, the value of such a column in the fullselect is the value of the column in the row before the row is updated.

UNPACK-function-invocation

Specifies an invocation of the UNPACK built-in function. The number of fields that are returned by the UNPACK function invocation must be the same as the number of *column-names*.

WHERE

Specifies the rows to be updated. You can omit the clause, give a search condition, or specify a cursor. If you omit the clause, all rows of the table or view are updated.

search-condition

Specifies any search condition described in [Language elements \(Db2 SQL\)](#). Each *column-name* in the search condition, other than in a subquery, must identify a column of the table or view.

The *search-condition* is applied to each row of the table or view and the updated rows are those for which the result of the *search-condition* is true. If the unique key or primary key is a parent key, the constraints are effectively checked at the end of the operation.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed just once, whereas it is possible that a subquery with a correlated reference must be executed once for each row.

WHERE CURRENT OF *cursor-name*

Identifies the cursor to be used in the update operation. *cursor-name* must identify a declared cursor as explained in the description of the DECLARE CURSOR statement in [DECLARE CURSOR \(Db2 SQL\)](#). If the UPDATE statement is embedded in a program, the DECLARE CURSOR statement must include *select-statement* rather than *statement-name*.

The object of the UPDATE statement must also be identified in the FROM clause of the SELECT statement of the cursor. The columns to be updated can be identified in the FOR UPDATE clause of that SELECT statement though they do not have to be identified. If the columns are not specified, the columns that can be updated include all the updatable columns of the table or view that is identified in the first FROM clause of the fullselect.

The result table of the cursor must not be read-only. For an explanation of read-only result tables, see [Read-only cursors](#). Note that the object of the UPDATE statement must not be identified as the object of the subquery in the WHERE clause of the SELECT statement of the cursor.

When the UPDATE statement is executed, the cursor must be open and positioned on a row or rowset of the result table.

- If the cursor is positioned on a single row, that row is the one updated.
- If the cursor is positioned on a rowset, all rows corresponding to the rows of the current rowset are updated.

A positioned UPDATE must not be specified for a cursor that references a view on which an instead of update trigger is defined, even if the view is an updatable view.

FOR ROW *n* OF ROWSET

Specifies which row of the current rowset is to be updated. The corresponding row of the rowset is updated, and the cursor remains positioned on the current rowset.

host-variable or *integer-constant* is assigned to an integral value *k*. If *host-variable* is specified, it must be an exact numeric type with scale zero, must not include an indicator variable, and *k* must be in the range 1 - 32767.

The cursor must be positioned on a rowset, and the specified value must be a valid value for the set of rows most recently retrieved for the cursor. If the specified row cannot be updated, an error is returned. It is possible that the specified row is within the bounds of the rowset most recently requested, but the current rowset contains less than the number of rows that were implicitly or explicitly requested when that rowset was established.

If this clause is not specified, the cursor position determines the rows that will be affected. If the cursor is positioned on a single row, that row is the one updated. In the case where the most recent FETCH statement returned multiple rows of data (but not as a rowset), this position would be on the last row of data that was returned. If the cursor is positioned on a rowset, all rows corresponding to the current rowset are updated. The cursor position remains unchanged.

It is possible for another application process to update a row in the base table of the SELECT statement so that the specified row of the cursor no longer has a corresponding row in the base table. An attempt to update such a row results in an error.

isolation-clause

Specifies the isolation level used when locating the rows to be updated by the statement.

WITH

Introduces the isolation level, which may be one of the following:

RR

Repeatable read

RS

Read stability

CS

Cursor stability

The default isolation level of the statement is the isolation level of the package or plan in which the statement is bound, with the package isolation taking precedence over the plan isolation. When a package isolation is not specified, the plan isolation is the default.

SKIP LOCKED DATA

Specifies that rows are skipped when incompatible locks are held on the row by other transactions. These rows can belong to any accessed table that is specified in the statement. **SKIP LOCKED DATA** can be used only when isolation **CS** or **RS** is in effect and applies only to row level or page level locks.

SKIP LOCKED DATA can be specified only in the searched **UPDATE** statement (or the searched update operation of a **MERGE** statement). **SKIP LOCKED DATA** is ignored if it is specified when the isolation level that is in effect is repeatable read (**WITH RR**) or uncommitted read (**WITH UR**). The default isolation level of the statement depends on the isolation level of the package or plan with which the statement is bound, with the package isolation taking precedence over the plan isolation. When a package isolation is not specified, the plan isolation is the default.

QUERYNO integer

Specifies the number to be used for this SQL statement in **EXPLAIN** output and trace records. The number is used for the **QUERYNO** column of the plan table for the rows that contain information about this SQL statement. This number is also used in the **QUERYNO** column of the **SYSIBM.SYSSTMT** and **SYSIBM.SYSPACKSTMT** catalog tables.

If the clause is omitted, the number associated with the SQL statement is the statement number assigned during precompilation. Thus, if the application program is changed and then precompiled, that statement number might change.

Using the **QUERYNO** clause to assign unique numbers to the SQL statements in a program is helpful:

- For simplifying the use of optimization hints for access path selection
- For correlating SQL statement text with **EXPLAIN** output in the plan table

For more information about enabling and using optimization hints, see [Influencing access path selection \(Db2 Performance\)](#)

For information on accessing the plan table, see [Investigating SQL performance by using EXPLAIN \(Db2 Performance\)](#).

Notes**Update rules:**

Update values must satisfy the following rules. If they do not, or if other errors occur during the execution of the **UPDATE** statement, no rows are updated and the position of the cursors are not changed.

- *Assignment.* Update values are assigned to columns using the assignment rules described in [Language elements \(Db2 SQL\)](#).
- *Validity.* Updates must obey the following rules. If they do not, or if any other errors occur during the execution of the **UPDATE** statement, no rows are updated.

- *Fullselects*: The row-fullselect and expressions that contain a *scalar-fullselect* must return no more than one row.
- *Unique constraints and unique indexes*: If the identified table (or base table of the identified view) has any unique indexes or unique constraints, each row that is updated in the table must conform to the limitations that are imposed by those indexes and constraints.

All uniqueness checks are effectively made at the end of the statement. In the case of a multi-row update, this validation occurs after all the rows are updated.

- *Check constraints*: If the identified table (or base table of the identified view) has any check constraints, each check constraint must be true or unknown for each row that is updated in the table.

All checks constraints are effectively validated at the end of the statement. In the case of a multi-row update, this validation occurs after all the rows are updated.

- *Views and the WITH CHECK OPTION*. For views defined with WITH CHECK OPTION, an updated row must conform to the definition of the view. If the view you name is dependent on other views whose definitions include WITH CHECK OPTION, the updated rows must also conform to the definitions of those views. For an explanation of the rules governing this situation, see [CREATE VIEW \(Db2 SQL\)](#).

For views that are not defined with WITH CHECK OPTION, you can change the rows so that they no longer conform to the definition of the view. Such rows are updated in the base table of the view and no longer appear in the view.

- *Field and validation procedures*. The updated rows must conform to any constraints imposed by any field or validation procedures on the identified table (or on the base table of the identified view).
- *Referential constraints*. The value of the parent key in a parent row must not be changed. If the update value produces a foreign key that is nonnull, the foreign key must be equal to some value of the parent key of the parent table of the relationship.

All referential constraints are effectively checked at the end of the statement. In the case of a multi-row update, this validation occurs after all the rows are updated.

- *Indexes with VARBINARY columns*. If the identified table has an index on a VARBINARY column or a column that is a distinct type that is based on VARBINARY data type, that index column cannot specify the DESC attribute. To use the SQL data change operation on the identified table, either drop the index or alter the data type of the column to BINARY and then rebuild the index.
- *Triggers*. An UPDATE statement might cause triggers to activate. A trigger might cause other statements to be executed or raise error conditions based on the update values. If an UPDATE statement for a view causes an instead of trigger to activate, validity, referential integrity, and check constraints are checked against the data changes that are performed in the trigger and not against the view that causes the trigger to activate or its underlying base tables.

Number of rows updated:

Normally, after an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows updated. (For a complete description of the SQLCA, including exceptions to the preceding sentence, see [SQL communication area \(SQLCA\) \(Db2 SQL\)](#).)

Nesting user-defined functions or stored procedures:

An UPDATE statement can implicitly or explicitly refer to user-defined functions or stored procedures. This is known as *nesting* of SQL statements. A user-defined function or stored procedure that is nested within the UPDATE must not access the table being updated.

Locking:

Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful update operation. Until a commit or rollback operation releases the locks, only the application process that performed the insert can access the updated row. If LOBs are not updated, application processes that are running with uncommitted read can also access the updated row. The locks can also prevent other application processes from performing operations on the table. However, application processes that are running with uncommitted read can access locked pages and rows.

Locks are not acquired on declared temporary tables.

Datetime representation when using datetime registers:

As explained under [Datetime special registers](#), when two or more datetime registers are implicitly or explicitly specified in a single SQL statement, they represent the same point in time. This is also true when multiple rows are updated.

Rules for positioned UPDATE with a SENSITIVE STATIC scrollable cursor:

When a SENSITIVE STATIC scrollable cursor has been declared, the following rules apply:

- *Update attempt of delete holes.* If, with a positioned update against a SENSITIVE STATIC scrollable cursor, an attempt is made to update a row that has been identified as a delete hole, an error occurs.
- *Update operations.* Positioned update operations with SENSITIVE STATIC scrollable cursors perform as follows:
 1. The SELECT list items in the target row of the base table of the cursor are compared with the values in the corresponding row of the result table (that is, the result table must still agree with the base table). If the values are not identical, then the update operation is rejected, and an error occurs. The operation may be attempted again after a successful FETCH SENSITIVE has occurred for the target row.
 2. The WHERE clause of the SELECT statement is re-evaluated to determine whether the current values in the base table still satisfy the search criteria. The values in the SELECT list are compared to determine that these values have not changed. If the WHERE clause evaluates as true, and the values in the SELECT have not changed, the update operation is allowed to proceed. Otherwise, the update operation is rejected, an error occurs, and an *update hole* appears in the cursor.
- *Update of update holes.* Update holes are not permanent. It is possible for another process, or a searched update in the same process, to update an update hole row so that it is no longer an update hole. Update holes become visible with a FETCH SENSITIVE for positioned updates and positioned deletes.
- *Result table.* After the base table is updated, the row is re-evaluated and updated in the temporary result table. At this time, it is possible that the positioned update changed the data such that the row does not qualify the search condition, in which case the row is marked as an update hole for subsequent FETCH operations.

Referencing columns that will be updated:

If a cursor uses FETCH statements to retrieve columns that will be updated later, specify FOR UPDATE OF when you select the columns. Then specify WHERE CURRENT OF in the subsequent UPDATE or DELETE statements. These clauses prevent Db2 from selecting access through an index on the columns that are being updated, which might otherwise cause Db2 to read the same row more than once.

For more information, see [Updating previously retrieved data \(Db2 Application programming and SQL\)](#).

Updating rows in a table with multilevel security:

When you update rows in a table with multilevel security, Db2 compares the security label of the user (the primary authorization ID) to the security label of the row. The update proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is updated and the value of the security label is determined by whether the user has write-down privilege:
 - If the user has write-down privilege or write-down control is not enabled, the user can set the security label of the row to any valid security label. The value that is specified for the security label column must be assignable to a column that is defined as CHAR(8) FOR SBCS DATA NOT NULL.
 - If the user does not have write-down privilege and write-down control is enabled, the security label of the row is set to the value of the security label of the user.

- If the security label of the user dominates the security label of the row, the result of the UPDATE statement is determined by whether the user has write-down privilege:
 - If the user has write-down privilege or write-down control is not enabled, the row is updated and the user can set the security label of the row to any valid security label.
 - If the user does not have write-down privilege and write-down control is enabled, the row is not updated.
- If the security label of the row dominates the security label of the user, the row is not updated.

Updating rows in a table for which row or column access control is enforced:

When an UPDATE statement is issued for a table for which row or column access control is enforced, the rules specified in the enabled row permissions or column masks determine whether the row can be updated. Typically those rules are based on the authorization ID or role of the process. The following describes how enabled row permissions and column masks are used during UPDATE:

- Row permissions are used to identify the set of rows to be updated.

When multiple enabled row permissions are defined for a table, a row access control search condition is derived by application of the logical OR operator to the search condition in each enabled permission. This row access control search condition is applied to the table to determine which rows are accessible to the authorization ID or role of the UPDATE statement. If the WHERE clause is specified in the UPDATE statement, the user-specified predicates are applied on the accessible rows to determine the rows to be updated. If there is no WHERE clause, the accessible rows are the rows to be updated.

Column masks are not applicable in this step.

If the table is not enforced by row access control, the WHERE clause determines the rows to be updated, otherwise all rows in the table are to be updated.

- If there are rows to be updated, the following rules determine whether those rows can be updated:
 - For every column to be updated, the new value of the column must not be affected by enabled column masks whose columns are referenced when deriving the new value.

When a column is referenced while deriving the values of a new row, if the column has an enabled column mask, the masked value is used to derive the new values. If the object table is also column access control activated, the column mask applied to derive the new values must ensure the evaluation of the access control rules defined in the column mask resolves the column to itself, not to a constant or an expression. If the column mask does not mask the column to itself, the new value cannot be used for update and an error is returned at run time.

- If the rows are updatable, and there is a BEFORE UPDATE trigger for the table, the trigger is activated.

Within the trigger actions, the new values for update might be modified in transition variables. When the final values are returned from the trigger, the new values are used for the update.

- The rows that are to be updated must conform to the enabled row permissions:

For each row that is to be updated, the old values are replaced with the new values that were specified in the UPDATE statement. A row that conforms to the enabled row permissions is a row that, if updated, can be retrieved using the derived row access control search condition.

- If the rows are updatable, and there is an AFTER UPDATE trigger for the table, the trigger is activated.

The above rules are not applicable to the included columns. The included columns are subject to the rules for the select list because they are not the columns of the object table of the UPDATE statement.

Extended indicators usage:

When extended indicators are enabled, indicator values other than positive values and 0 (zero) through -7 must not be specified. The DEFAULT and UNASSIGNED extended indicator values must not appear in contexts where they are not supported.

Extended indicators:

Specifying an indicator value with the extended indicator value of UNASSIGNED has the same effect as if the column had not been specified in the statement. Assigning an extended indicator value of DEFAULT assigns the default value to the column, and must only be specified for a column that is defined with a default value.

If a target column is not updatable, such as an identity column that is defined as GENERATED ALWAYS, it must be assigned the extended indicator value of UNASSIGNED.

An UPDATE statement must not specify the extended indicator value of UNASSIGNED for all target columns.

Extended indicators and update triggers:

If the indicator value for a target column is UNASSIGNED, that column is not considered to have been updated. That column is treated as if it had not been specified in the OF *column-name* list of any update trigger that is defined on the target table or view.

Extended indicators and deferred error checks:

When extended indicators are enabled, validation that would normally be done during statement preparation to recognize an insert into a non-updatable column is deferred until the statement is executed.

Considerations for a generated column:

A generated column that is defined as GENERATED ALWAYS should not be specified as the target of an assignment clause unless the value that is to be assigned is specified with the DEFAULT keyword or an extended indicator that specifies that a default value is to be assigned.

Considerations for a system-period temporal table:

When a row of a system-period temporal table is updated, Db2 updates the values of the row-begin and transaction-start-ID columns as follows:

- A row-begin column is assigned a value for the data type of the column. If the value of the SYSIBM. TEMPORAL_LOGICAL_TRANSACTION_TIME built-in global variable at the time of the update is null, the value is generated using a reading of the time-of-day clock during execution of the first data change statement in the unit of work that requires a value to be assigned to a row-begin column or transaction-start-ID column in a table, or a row in a system-period temporal table is deleted. Otherwise, the row-begin column is assigned the value of the SYSIBM. TEMPORAL_LOGICAL_TRANSACTION_TIME built-in global variable at the time of the insert.
- A transaction-start-ID column is assigned a unique timestamp value per unit of work or the null value. The null value is assigned to the transaction-start-ID column if the column is nullable. Otherwise, the value is generated by using the time-of-day clock during execution of the first data change statement in the unit of work that requires a value to be assigned to a row-begin column or transaction-start-ID column in a table. This also occurs when a row in a system-period temporal table is deleted. If multiple rows are updated within a single SQL unit of work, the values for the transaction-start-ID column are the same for all the rows and are unique from the values that are generated for the column for another unit of work.

If the UPDATE statement has a search condition that contains a correlated subquery that references historical rows (explicitly referencing the name of the history table or implicitly referenced through the use of a period specification in the FROM clause), the old version of the updated rows that are inserted as historical rows (into the history table) are potentially visible to update operations for the rows that are subsequently processed for the statement.

If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, the underlying target of the UPDATE statement cannot be a system-period temporal table. This restriction applies regardless of whether the system-period temporal table is directly or indirectly referenced.

Considerations for a history table:

When a row of a system-period temporal table is updated, a historical copy of the row is inserted into the corresponding history table and the end timestamp of the historical row is captured in the form of a system determined value that corresponds to the time of the data change operation. Db2 generates the value by using the time-of-day clock during the execution of the first data change statement in the transaction that requires a value to be assigned to a row-begin or transaction-start-

ID column in a table. This also occurs when a row in a system-period temporal table is deleted. If the value of the SYSIBM. TEMPORAL_LOGICAL_TRANSACTION_TIME built-in global variable at the time of the data change operation is null, the value is generated using a reading of the time-of-day clock during execution of the first data change statement in the unit of work that requires a value to be assigned to a row-begin column or transaction-start-ID column in a table, or a row in a system-period temporal table is deleted. Otherwise, the value is assigned from the SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME built-in global variable at the time of the data change operation.

Considerations for an application-period temporal table:

An UPDATE statement that contains a FOR PORTION OF BUSINESS_TIME clause for an application-period temporal table indicates the two points in time between which the specified updates are effective.

Suppose that FOR PORTION OF BUSINESS_TIME is specified, and the period value for a row is only partially contained in the period that is specified from *value1* up to *value2* or between *value1* and *value2*. (The period value for a row is specified by the values of the begin column and end column for the BUSINESS_TIME period.) In this case, the row is updated and one or two rows are automatically inserted to represent the portion of the row that is not changed. For each row that is automatically inserted as a result of an update operation on the table, new values are generated for each generated column in the application-period temporal table. If a generated column is defined as part of a unique or primary key, parent key in a referential constraint, or unique index, an automatic insert might violate a constraint or index. In this case, an error is returned.

When an application-period table is the target of an UPDATE statement and the value in effect for the CURRENT TEMPORAL BUSINESS_TIME special register is not the null value, Db2 adds the following additional predicates to the statement:

- inclusive-exclusive period:

```
bt_begin <= CURRENT TEMPORAL BUSINESS_TIME AND  
bt_end > CURRENT TEMPORAL BUSINESS_TIME
```

- inclusive-inclusive period:

```
bt_begin <= CURRENT TEMPORAL BUSINESS_TIME AND  
bt_end >= CURRENT TEMPORAL BUSINESS_TIME
```

In the preceding code, bt_begin and bt_end are the begin and end columns of the BUSINESS_TIME period of the target table of the UPDATE statement.

Archive-enabled tables:

A reference to an archive-enabled table as the target of the UPDATE statement does not affect rows in the associated archive table.

A data change statement must not reference an archive-enabled table when a system-period temporal table or application-period temporal table is also referenced.

Other SQL statements in the same unit of work:

The following statements cannot follow an UPDATE statement in the same unit of work:

- An ALTER TABLE statement that changes the data type of a column (ALTER COLUMN SET DATA TYPE)
- An ALTER INDEX statement that changes the padding attribute of an index with varying-length columns (PADDED to NOT PADDED or vice versa)
- A CREATE TABLE statement that creates an accelerator-only table.
- An INSERT, UPDATE, or DELETE statement that updates accelerator-only tables from a different accelerator.

Using UPDATE to reset AREO* status on a table:

An UPDATE statement will reset the AREO* state of a table if all conditions are true:

- The statement is a searched UPDATE statement. An UPDATE statement within a SELECT statement will not reset the AREO* state.
- The expression in the SET clause is not a *scalar-fullselect* or *row-fullselect*
- The update operation is against a table in a universal table space
- The table does not have row access control activated
- The SKIP LOCKED DATA clause is not specified
- The WHERE clause is not specified
- A resource unavailable condition is not encountered.

No error or warning SQLCODE is returned if a resource unavailable condition is encountered. Only a resource unavailable console message will be displayed.

A DISPLAY DATABASE command can be used to determine if AREO* is reset.

Examples

Example 1

Change employee 000190's telephone number to 3565 in DSN8D10.EMP.

```
UPDATE DSN8D10.EMP
SET PHONENO='3565'
WHERE EMPNO='000190';
```

Example 2

Give each member of department D11 a 100-dollar raise.

```
UPDATE DSN8D10.EMP
SET SALARY = SALARY + 100
WHERE WORKDEPT = 'D11';
```

Example 3

Employee 000250 is going on a leave of absence. Set the employee's pay values (SALARY, BONUS, and COMMISSION) to null.

```
UPDATE DSN8D10.EMP
SET SALARY = NULL, BONUS = NULL, COMM = NULL
WHERE EMPNO='000250';
```

Alternatively, the statement could also be written as follows:

```
UPDATE DSN8D10.EMP
SET (SALARY, BONUS, COMM) = (NULL, NULL, NULL)
WHERE EMPNO='000250';
```

Example 4

Assume that a column named PROJSIZE has been added to DSN8D10.EMP. The column records the number of projects for which the employee's department has responsibility. For each employee in department E21, update PROJSIZE with the number of projects for which the department is responsible.

```
UPDATE DSN8D10.EMP
SET PROJSIZE = (SELECT COUNT(*)
                FROM DSN8D10.PROJ
                WHERE DEPTNO = 'E21')
WHERE WORKDEPT = 'E21';
```

Example 5

Double the salary of the employee represented by the row on which the cursor C1 is positioned.

```
EXEC SQL UPDATE DSN8D10.EMP
SET SALARY = 2 * SALARY
WHERE CURRENT OF C1;
```

Example 6

Assume that employee table EMP1 was created with the following statement:

```
CREATE TABLE EMP1
(EMP_ROWID    ROWID GENERATED ALWAYS,
 EMPNO        CHAR(6),
 NAME         CHAR(30),
 SALARY       DECIMAL(9,2),
 PICTURE      BLOB(250K),
 RESUME       CLOB(32K));
```

Assume that host variable *HV_EMP_ROWID* contains the value of the ROWID column for employee with employee number '350000'. Using that ROWID value to identify the employee and user-defined function UPDATE_RESUME, increase the employee's salary by \$1000 and update that employee's resume.

```
EXEC SQL UPDATE EMP1
SET SALARY = SALARY + 1000,
    RESUME = UPDATE_RESUME(:HV_RESUME)
WHERE EMP_ROWID = :HV_EMP_ROWID;
```

Example 7

In employee table X, give each employee whose salary is below average a salary increase of 10%.

```
EXEC SQL UPDATE EMP X
SET SALARY = 1.10 * SALARY
WHERE SALARY < (SELECT AVG(SALARY) FROM EMP Y
WHERE X.JOBCODE = Y.JOBCODE);
```

Example 8

Raise the salary of the employees in department 'E11' whose salary is below average to the average salary.

```
EXEC SQL UPDATE EMP T1
SET SALARY = (SELECT AVG(T2.SALARY) FROM EMP T2)
WHERE WORKDEPT = 'E11' AND
    SALARY < (SELECT AVG(T3.SALARY) FROM EMP T3);
```

Example 9

Give the employees in department 'E11' a bonus equal to 10% of their salary.

```
EXEC SQL
DECLARE C1 CURSOR FOR
SELECT BONUS
FROM DSN8710.EMP
WHERE WORKDEPT = 'E12'
FOR UPDATE OF BONUS;
EXEC SQL
UPDATE DSN8710.EMP
SET BONUS = ( SELECT .10 * SALARY FROM DSN8710.EMP Y
WHERE EMPNO = Y.EMPNO )
WHERE CURRENT OF C1;
```

Example 10

Assuming that cursor CS1 is positioned on a rowset consisting of 10 rows in table T1, update all 10 rows in the rowset.

```
EXEC SQL UPDATE T1 SET C1 = 5 WHERE CURRENT OF CS1;
```

Example 11

Assuming that cursor CS1 is positioned on a rowset consisting of 10 rows in table T1, update the fourth row of the rowset.

```
short ind1, ind2;
int n, updt_value;
stmt = 'UPDATE T1 SET C1 = ? WHERE CURRENT OF CS1 FOR ROW ? OF ROWSET'
```

```

ind1 = 0;
ind2 = 0;
n = 4;
updt_value = 5;
...
strcpy(my_sqlda.sqldaid,"SQLDA");
my_sqlda.sqln = 2;
my_sqlda.sqld = 2;
my_sqlda.sqlvar[0].sqltype = 497;
my_sqlda.sqlvar[0].sqllen = 4;
my_sqlda.sqlvar[0].sqldata = (int *) &updt_value;
my_sqlda.sqlvar[0].sqlind = (short *) &ind1;

my_sqlda.sqlvar[1].sqltype = 497;
my_sqlda.sqlvar[1].sqllen = 4;
my_sqlda.sqlvar[1].sqldata = (int *) &n;
my_sqlda.sqlvar[1].sqlind = (short *) &ind2;

EXEC SQL PREPARE S1 FROM :stmt;
EXEC SQL EXECUTE S1 USING DESCRIPTOR :my_sqlda;

```

Example 12

Assume that table POLICY exists and that it is defined with a single inclusive-exclusive period, BUSINESS_TIME. The table contains a row where column BK has a value of 'P138', column CLIENT has a value of 'C882', column TYPE has a value of 'PPO', and the period has value ('2013-01-01', '2020-12-31'). Update the portion of the row beginning from '2014-01-01' to set the TYPE column to 'HMO':

```

UPDATE POLICY
  FOR PORTION OF BUSINESS_TIME
    FROM '2014-01-01' TO '9999-12-31'
    SET TYPE='HMO'
  WHERE BK='P138', CLIENT='C882';

```

After the UPDATE statement is processed, the table contains 2 rows in place of the original row. One row with period value ('2013-01-01', '2014-01-01') represents a value of 'PPO' for the TYPE column (the value before the update) and the other row with period value ('2014-01-01', '2020-12-31') represents a value of 'HMO' for the TYPE column (that began with the UPDATE statement).

Example 13

Suppose that the INTARRAY and CHARARRAY array types, the INTA, CHARA, and SI variables, and the T1 table are defined as follows:

```

CREATE TYPE INTARRAY AS INTEGER ARRAY [6];
CREATE TYPE CHARARRAY AS CHAR(20) ARRAY [7];
CREATE VARIABLE INTA AS INTARRAY;
CREATE VARIABLE CHARA AS CHARARRAY;
CREATE VARIABLE SI INT;
CREATE TABLE T1 (COL1 CHAR(7), COL2 INT);

```

Assign values to CHARA, INTA, and SI.

```

SET CHARA = ARRAY [ 'a', 'b', 'c' ];
SET INTA = ARRAY [ 1, 2, 3, 4, 5 ];
SET SI = 1;

```

Insert a row into table T1, and then update the row values using values from the CHARA and INTA arrays, which are indexed by the value of variable SI.

```

INSERT INTO T1 VALUES ('abc', 10);
UPDATE T1
  SET COL1 = CHARA[SI],
      COL2 = INTA[SI];

```

In the table row, COL1 now contains 'a', and COL2 contains 1.

Set the value of column COL2 for all rows to the cardinality of array INTA.

```
UPDATE T1  
SET COL2 = CARDINALITY(INTA);
```

In the table row, COL2 now contains 5.

Example 14

Assume that table POLICY exists and that it is defined with a single inclusive-inclusive period, BUSINESS_TIME. The table contains a row where column BK has a value of 'P138', column CLIENT has a value of 'C882', column TYPE has a value of 'PPO', and period has value ('2013-01-01', '2020-12-31'). Suppose that you issue the following UPDATE statement:

```
UPDATE POLICY  
FOR PORTION OF BUSINESS_TIME  
BETWEEN '2014-01-01' AND '9999-12-31'  
SET TYPE='HMO'  
WHERE BK='P138', CLIENT='C882';
```

After the UPDATE statement is processed, the table contains 2 rows in place of the original row. One row with period value ('2013-01-01', '2013-12-31') has a value of 'PPO' for the TYPE column (the value before the update) and the other row with period value ('2014-01-01', '2020-12-31') has a value of 'HMO' for the TYPE column.

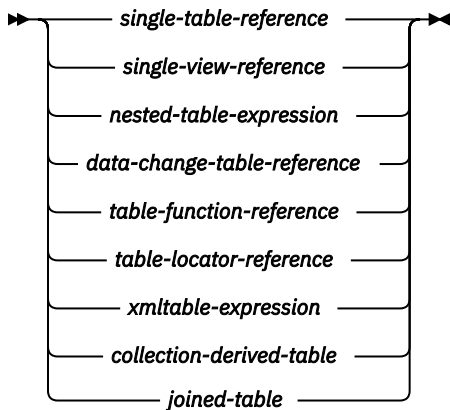
Chapter 10. Db2 queries for SQL DI

Db2 updates the following subselects of the SELECT statement to support SQL DI.

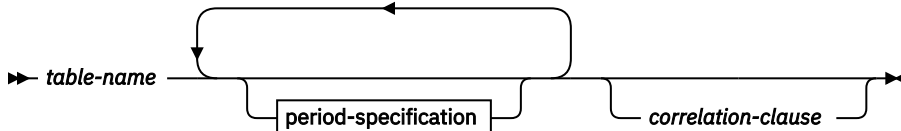
table-reference

A *table-reference* specifies a result table as either a table or view, or an intermediate table.

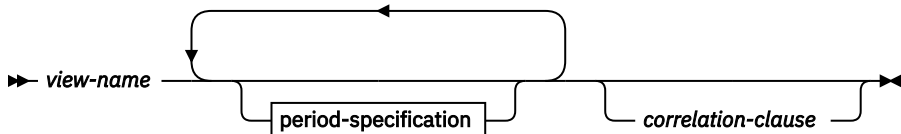
table-reference:



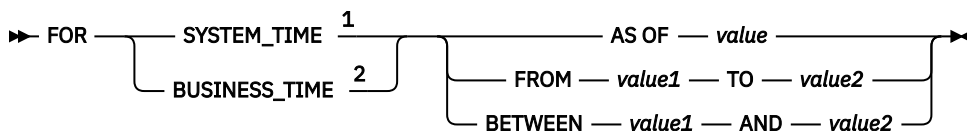
single-table-reference:



single-view-reference:



period-specification:

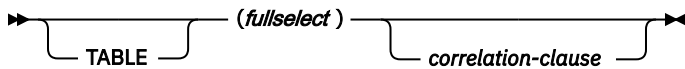


Notes:

¹ AS OF TIMESTAMP can be specified as an alternative and is treated as if FOR SYSTEM_TIME AS OF had been specified.

² SYSTEM_TIME and BUSINESS_TIME cannot be specified more than one time per table.

nested-table-expression:



data-change-table-reference:

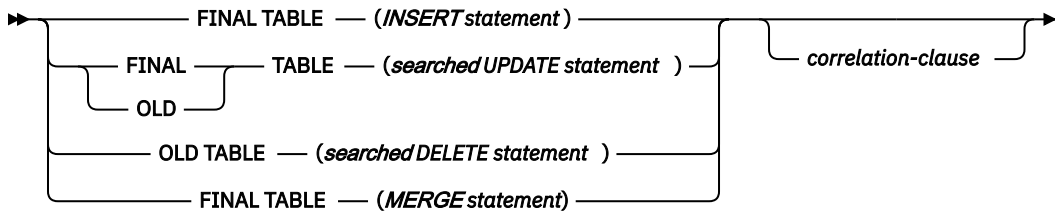
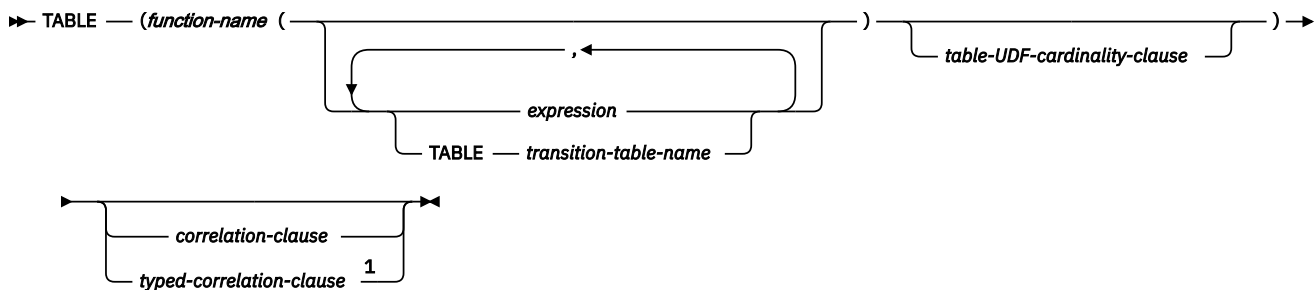


table-function-reference:



Notes:

¹ The *typed-correlation-clause* is required for generic table functions. This clause cannot be specified for any other table functions.

table-UDF-cardinality-clause:

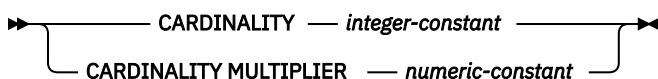
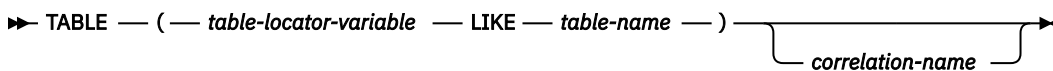
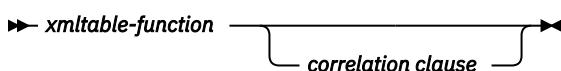


table-locator-reference:



xmltable-expression:



A *table-reference* specifies an intermediate result table.

- If a *single-table-reference* is specified and it is not an archive-enabled table or a temporal table, the intermediate result table is the specified table. If a *period-specification* is also specified, the

intermediate result table consists of the rows of the temporal table where the period matches the specification.

- If a *single-table-reference* is specified and it is an archive-enabled table, the setting of the SYSIBMADM.GET_ARCHIVE global variable and the ARCHIVESENSITIVE bind option determine the contents of the intermediate result table. If the global variable is set to Y and the bind option is set to YES, the intermediate result table includes the rows in the associated archive table. Otherwise, the intermediate result table does not include rows in the associated archive table.
- If a *single-view-reference* is specified without a *period-specification*, the intermediate result table is that view. If a *period-specification* is specified, temporal table references in the view consider only the rows where the period matches the specification.
- If a *nested-table-expression* is specified, the result table is the result of the specified fullselect. The columns of the result do not need unique names, but a column with a non-unique name cannot be explicitly referenced.
- If a *data-change-table-reference* is specified, the intermediate result table is the set of rows that are directly affected by the data change statement.
- If a *table-function-reference* is specified, the intermediate result table is the set of rows that are returned by the table function.
- If a *table-locator-reference* is specified, the host variable represents the intermediate result table. The intermediate result table has the same structure as the table identified in *table-name*.
- If a *collection-derived-table* is specified, the intermediate result table is a set of rows from one or more array values. For more information, see [collection-derived-table \(Db2 SQL\)](#).
- If an *xmltable-expression* is specified, the intermediate result table is the set of rows that are returned by the XMLTABLE (Db2 SQL) function.
- If a *joined-table* is specified, the intermediate result table is the result of one or more join operations. For more information, see [joined-table \(Db2 SQL\)](#).

Each *table-name* or *view-name* specified in every FROM clause of the same SQL statement must identify a table or view that exists at the same Db2 subsystem. If a FROM clause is specified in a subquery of a basic predicate, a view that includes GROUP BY or HAVING must not be identified.

A *table-reference* must not identify a table that was implicitly created for an XML column.

table-locator-variable

table-locator-variable must specify a variable with a table locator type. The only way to assign a value to a table locator is to pass the old or new transition table of a trigger to a user-defined function or stored procedure. A table locator host variable must not have a null indicator. A table locator variable must not be a parameter marker. In addition, a table locator can be used only in a manipulative SQL statement. *table-locator-reference* must not be specified in the body of a trigger.

table-name must refer to an EBCDIC table with a Unicode column if the transition table that is identified by *table-locator-variable* is for a trigger that is defined on an EBCDIC table with a Unicode column.

nested-table-expression

A *fullselect* in parentheses is called a *nested table expression*. If a nested table expression is specified, the result table is the result of that *nested-table-expression*. The columns of the result do not need unique names, but a column with a non-unique name cannot be referenced. At any time, the table consists of the rows that would result if the fullselect were executed.

table-function-reference

If a *function-name* is specified, the result table is the set of rows returned by the table function.

expression must not contain a scalar fullselect, a function, or a reference to a column.

Each *function-name*, together with the types of its arguments, must resolve to a table function that exists at the same Db2 subsystem. An algorithm called function resolution, which is described in [Function resolution \(Db2 SQL\)](#), uses the function name and the arguments to determine the exact function to use. Unless given column names in the *correlation-clause*, the column names for a table

function are those specified on the RETURNS clause of the CREATE FUNCTION statement. This is analogous to the column names of a table, which are defined in the CREATE TABLE statement.

If a column mask is used to mask the column values in the final result table, and if the result of the table function is used to derive the final result table, the column mask cannot be applied to a column that is specified in the argument of the table function.

table-UDF-cardinality-clause

The *table-UDF-cardinality-clause* can be specified to each user-defined table function reference within the table spec of the FROM clause in a subselect. This option indicates the expected number of rows to be returned only for the SELECT statement that contains it.

CARDINALITY *integer-constant* specifies an estimate of the expected number of rows returned by the reference to the user-defined function. The value of *integer-constant* must range 0 - 2147483647.

The value set in the CARDINALITY field of SYSIBM.SYSROUTINES for the table function name is used as the reference cardinality value. The product of the specified CARDINALITY MULTIPLIER *numeric-constant* and the reference cardinality value are used by Db2 as the expected number of rows returned by the table function reference.

In this case, the *numeric-constant* can be in the integer, decimal, or floating-point format. The value must be greater than or equal to zero. If the decimal number notation is used, the number of digits can be up to 31. An integer value is treated as a decimal number with no fraction. The maximum value allowed for a floating-point number is about 7.237E + 75. If no value has been set in the CARDINALITY field of SYSIBM.SYSROUTINES, its default value is used as the reference cardinality value. If zero is specified or the computed cardinality is less than 1, Db2 assumes that the cardinality of the reference to the user-defined table function is 1.

Only a numeric constant can follow the keyword CARDINALITY or CARDINALITY MULTIPLIER. No host variable or parameter marker is allowed in a cardinality option. Specifying a cardinality option in a table function reference does not change the corresponding CARDINALITY field in SYSIBM.SYSROUTINES. The CARDINALITY field value in SYSIBM.SYSROUTINES can be initialized by the CARDINALITY option in the CREATE FUNCTION (external table) statement when a user-defined table function is created. It can be changed by the CARDINALITY option in the ALTER FUNCTION statement or by a direct update operation to SYSIBM.SYSROUTINES.

data-change-table-reference

A *data-change-table-reference* clause specifies an intermediate result table. This table is based on the rows that are directly changed by the SQL data change statement that is included in the clause. A *data-change-table-reference* can only be specified as the only *table-reference* in the FROM clause of the outer fullselect that is used in a select-statement and that fullselect must be in a subselect, or a SELECT INTO statement. A *data-change-table-reference* in a SELECT statement of a cursor makes the cursor read only. The target table or view of the SQL data change statement is a table or view that is reference in the query. The privileges that are held by the authorization ID of the statement must include the SELECT privilege on that target table or view. The encoding scheme of the result table of the SELECT must be the same as the encoding scheme of the target table or view of the *data-change-table-reference*.

If row access control is enforced for the target of the data change statement, the rows in the intermediate result table already satisfy the rules that are specified in the enabled row permissions. If column access control is enforced for the target of the data change statement, the enabled column masks are applied to the outermost select list. For more information, see [select-clause \(Db2 SQL\)](#). If an INCLUDE clause is specified as part of the SQL data change statement, and these additional columns appear in the outermost select list, the column values must not be derived from columns for which column masks are defined.

Expressions in the select list of a view in a table reference can only be selected if OLD TABLE is specified or if the expression does not include any of the following objects:

- a function that is defined to read or modify SQL data
- a function that is defined as not deterministic or has an external action

- a NEXT VALUE expression for a sequence

FINAL TABLE

Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement as they appear at the completion of the SQL data change statement. If there are AFTER triggers that result in further operations on the table that is the target of the SQL data change statement, an error is returned. If the target of the SQL data change statement is a view that is defined with an INSTEAD OF trigger for the type of data change, an error is returned.

OLD TABLE

The rows of the intermediate result table represent the set of affected rows as they exist prior to the application of the SQL data change statement.

INSERT statement

Specifies an INSERT statement as described in [INSERT \(Db2 SQL\)](#). A fullselect in the INSERT statement cannot contain correlated references to columns that are outside of the fullselect of the INSERT statement. The target of the INSERT statement must be a base table, a view that is defined with the WITH CASCADED CHECK clause, or a view where the view definition has no WHERE clause. If there are input variables elsewhere in the fullselect, the INSERT statement cannot be a multiple row not atomic insert, or a multiple row atomic insert that specifies the USING DESCRIPTOR clause.

MERGE statement

Specifies a MERGE statement as described in [MERGE \(Db2 SQL\)](#). The MERGE statement must conform to the following rules:

- The target of the MERGE statement must be a base table, a view that is defined with the WITH CASCADED CHECK clause, or a view where the view definition has no WHERE clause.
- The target table or view of the MERGE statement must not have a column with a ROWID data type. Additionally, when NOT ATOMIC CONTINUE ON SQLEXCEPTION is specified, the target table or view of the MERGE statement must not have a column with a LOB or XML data type.
- If *table-reference* is specified in the MERGE statement, it must not contain correlated references to columns that are outside of the table reference in the MERGE statement.
- If *table-reference* is specified in the MERGE statement, AFTER triggers that result in further operations on the target table must not exist.
- When NOT ATOMIC CONTINUE ON SQLEXCEPTION is specified in the MERGE statement, or the NOT ATOMIC CONTINUE ON SQLEXCEPTION clause is not specified, and *source-values* (VALUES) is specified, the MERGE statement must not include a delete operation.

searched UPDATE statement

Specifies a searched UPDATE statement as described in [UPDATE \(Db2 SQL\)](#). A WHERE clause or a SET clause in the UPDATE statement cannot contain correlated references to columns that are outside of the UPDATE statement. The target of the UPDATE statement must be a base table, a symmetric view, or a view where the view definition has no WHERE clause.

If the searched UPDATE statement is used in the SELECT statement and the UPDATE statement references a view, the view must be defined using the WITH CASCADED CHECK OPTION clause.

A searched UPDATE statement in a SELECT statement will not clear the AREO* status of a table.

AFTER triggers that result in further operations on the target table cannot exist on the target table.

searched DELETE statement

Specifies a searched DELETE statement as described in [DELETE \(Db2 SQL\)](#). A WHERE clause in the DELETE statement cannot contain correlated references to columns that are outside of the DELETE statement. The target of the DELETE statement must be a base table, a symmetric view, or a view where the view definition has no WHERE clause.

If the searched DELETE statement is used in the SELECT statement and the DELETE statement references a view, the view must be defined using the WITH CASCADED CHECK OPTION clause.

AFTER triggers that result in further operations on the target table cannot exist on the target table.

The content of the intermediate result table for a table reference that contains an SQL data change statement is determined when the cursor is opened. The intermediate result table includes a column for each of the columns of the target table (including implicitly hidden columns) or view. All of the columns of the target table or view of an SQL data change statement are accessible by using the names of the columns from the target table or view unless the columns are renamed by using the correlation clause. If an INCLUDE clause is specified as part of the SQL data change statement, the intermediate result table will contain these additional columns.

correlation-clause

Each *correlation-name* in a *correlation-clause* defines a designator for the immediately preceding result table, which can be used to qualify references to the columns of the table. For more information, see [correlation-clause \(Db2 SQL\)](#).

The exposed names of all table references in the FROM clause should be unique. An *exposed name* is considered to be any of the following names:

- A *correlation-name*
- A *table-name* that is not followed by a *correlation-name*
- A *view-name* that is not followed by a *correlation-name*
- A *function-name* that is not followed by a *correlation-name*
- The table name that is specified after LIKE when a *table-locator* is not followed by a *correlation-name*
- The target table or view name for a *data-change-table-reference* that is not followed by a *correlation-name*
- An *alias-name* that is not followed by a *correlation-name*
- A *synonym-name* that is not followed by a *correlation-name*

If a *correlation-clause* clause does not follow an *xmltable-expression* reference, a *nested-table-expression* reference, or a *collection-derived-table-reference*, there is no exposed name for that table reference.

Any qualified reference to a column must use the exposed name. If the same name is specified twice, at least one specification should be followed by a *correlation-name*. The *correlation-name* is used to qualify references to the columns of the table or view. When a *correlation-name* is specified, column names can also be specified to give names to the columns of the table reference. If the *correlation-clause* does not include column names, the exposed column names are determined as follows:

- Column names of the referenced table or view when the *table-reference* is *table-name*, *view-name*, *alias-name*, or *synonym-name*
- Column names specified in the RETURNS clause of the CREATE FUNCTION statement when the *table-reference* is a *function-name* reference
- Column names of the table referenced after LIKE when the *table-reference* is a *table-locator*
- Column names specified in the COLUMNS clause of the *xmltable-expression* when the *table-reference* is an *xmltable-expression*
- Column names returned by the fullselect when the *table-reference* is a *nested-table-expression*
- Column names from the target table of the data change statement, along with any defined INCLUDE columns, when the *table-reference* is a *data-change-table-reference*

Otherwise, there are no exposed names for the columns of that table reference.

typed-correlation-clause

A *typed-correlation-clause* defines the appearance and contents of the table generated by a generic table function. This clause must be specified when the *table-function-reference* is a generic table function and cannot be specified for any other table reference. For more information, see [typed-correlation-clause \(Db2 SQL\)](#).

xmltable-expression

Specifies an invocation of the built-in XMLTABLE function. For more information, see [XMLTABLE \(Db2 SQL\)](#).

If a column mask is used to mask the column values in the final result table, and if the result of the XMLTABLE function is used to derive the final result table, the column mask cannot be applied to a column that is specified in the PASSING clause of the XMLTABLE function.

collection-derived-table

A *collection-derived-table* is used to convert the elements of one or more arrays into column values in separate rows of an intermediate result table, as explained in [collection-derived-table \(Db2 SQL\)](#).

joined-table

If a *joined-table* is specified, the result table is the result of one or more join operations as explained in [joined-table \(Db2 SQL\)](#).

period-specification

Specifies that a period specification applies to the *table-reference*. The same period name (SYSTEM_TIME or BUSINESS_TIME) must not be specified more than one time for the same table. If the table reference specifies a view, the definition of that view must not reference a user-defined function.

The rows of the table reference are derived by application of the specified period specification. The intermediate result table does not include rows in the associated history table that were added for the ON DELETE ADD EXTRA ROW attribute in the system-period temporal table definition.

Note: History tables are intended to include only rows that Db2 stores to record the history of the associated system-period temporal table. However, if the history table contains other rows with the same value in the two columns that correspond to the row-begin and row-end columns in the system-period temporal table, the intermediate result table might include these rows. These rows might be included in the following cases:

- The system-period temporal table is defined with the ON DELETE ADD EXTRA ROW attribute, the table contains a DATA CHANGE OPERATION column, and the value of the corresponding column in the history table is not 'D'.
- The system-period temporal table is not defined with the ON DELETE ADD EXTRA ROW attribute.

The rows of a view reference are derived by application of the specified period specifications to all of the temporal tables that are accessed when computing the result table of the view. If the view does not access any temporal tables, the period specification has no effect on the result table of the view.

If the table is a bitemporal table and a *period-specification* is not specified for both SYSTEM_TIME or BUSINESS_TIME, the table reference includes all current rows of the table and does not include any historical rows of the table.

If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a value other than the null value, a *period-specification* for a table or view cannot reference SYSTEM_TIME. This restriction applies even if the view body does not reference a system-period temporal table. The exception is if the value in effect for the SYSTIMESENSITIVE bind option is NO. In this case, the *period-specification* can reference SYSTEM_TIME.

If the CURRENT TEMPORAL BUSINESS_TIME special register is set to a value other than the null value, a *period-specification* for a table or view cannot reference BUSINESS_TIME. This restriction applies even if the view body does not reference an application-period temporal table. The exception is if the value in effect for the BUSTIMESENSITIVE bind option is NO. In this case, the *period-specification* can reference BUSINESS_TIME.

For more information, see:

[CURRENT TEMPORAL BUSINESS_TIME \(Db2 SQL\)](#)

[CURRENT TEMPORAL SYSTEM_TIME \(Db2 SQL\)](#)

FOR SYSTEM_TIME

Specifies that the SYSTEM_TIME period is used for the *period-specification*. The table reference must be a system-period temporal table or a view.

Do not specify FOR SYSTEM_TIME if the value of the CURRENT TEMPORAL SYSTEM_TIME special register is not NULL and the SYSTIMESENSITIVE bind option is set to YES .

FOR BUSINESS_TIME

Specifies that the BUSINESS_TIME period is used for the *period-specification*. The table reference must be an application-period temporal table or a view.

Do not specify FOR BUSINESS_TIME if the value of the CURRENT TEMPORAL BUSINESS_TIME special register is not NULL and the BUSTIMESENSITIVE bind option is set to YES .

AS OF value

Specifies that the *table-reference* includes rows that exist at the time that is specified by *value* as follows:

- For an inclusive-exclusive period, a row is included if the begin value for the specified period is less than or equal to *value*, and the end value for the period is greater than *value*. If *value* is the null value, the table reference is an empty table.
- For an inclusive-inclusive period, a row is included if the begin value for the specified period is less than or equal to *value*, and the end value for the period is greater than or equal to *value*. If *value* is the null value, the table reference is an empty table.

value

Specifies an expression that returns a value of a built-in data type. The result of the expression must be comparable to the data type of the columns of the specified period according to the comparison rules specified in [Assignment and comparison \(Db2 SQL\)](#).

The expression must not have a timestamp precision that is greater than the precision of the columns for the period.

If the begin and end columns of the period are defined as TIMESTAMP WITHOUT TIME ZONE, the expression must not return a value of a timestamp with a time zone.

The expression can contain any of the following supported operands:

- A constant
- A special register
- A variable
- An array element specification
- A built-in scalar function whose arguments are supported operands

Note: The scalar function must not be AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.

- A CAST specification where the cast operand is a supported operand
- An expression that uses arithmetic operators and operands

A period specification for a view must not contain an untyped parameter marker.

FROM value1 TO value2

Specifies that the *table-reference* includes rows that exist for the period that is specified from *value1* up to *value2*.

- For an inclusive-exclusive period, a row is included in the *table-reference* if the start value for the period in the row is less than *value2*, and the end value for the period in the row is greater than *value1*. The *table-reference* contains zero rows if *value1* is greater than or equal to *value2*. If *value1* or *value2* is the null value, the table reference is an empty table.
- For an inclusive-inclusive period, a row is included in the *table-reference* if the start value for the period in the row is less than *value2*, and the end value for the period in the row is greater

than or equal to *value1*. The *table-reference* contains zero rows if *value1* is greater than or equal to *value2*. If *value1* or *value2* is the null value, the table reference is an empty table.

value1* or *value2

Specifies an expression that returns a value of a built-in data type. The result of the expression must be comparable to the data type of the columns of the specified period according to the comparison rules specified in [Assignment and comparison \(Db2 SQL\)](#).

The expression must not have a timestamp precision that is greater than the precision of the columns for the period.

If the begin and end columns of the period are defined as `TIMESTAMP WITHOUT TIME ZONE`, the expression must not return a value of a timestamp with a time zone.

The expression can contain any of the following supported operands:

- A constant
- A special register
- A variable
- An array element specification
- A built-in scalar function whose arguments are supported operands

Note: The scalar function must not be `AI_ANALOGY`, `AI_COMMONALITY`, `AI_SEMANTIC_CLUSTER`, or `AI_SIMILARITY`.

- A `CAST` specification where the cast operand is a supported operand
- An expression that uses arithmetic operators and operands

A period specification for a view must not contain an untyped parameter marker.

BETWEEN *value1* AND *value2*

Specifies that the *table-reference* includes rows in which the specified period overlaps at any point in time between *value1* and *value2*.

- For an inclusive-exclusive period, a row is included in the *table-reference* if the start value for the period in the row is less than or equal to *value2* and the end value for the period in the row is greater than *value1*. The *table-reference* contains zero rows if *value1* is greater than *value2*. If *value1* = *value2*, the expression is equivalent to `AS OF value1`. If *value1* or *value2* is the null value, the *table-reference* is an empty table.
- For an inclusive-inclusive period, a row is included in the *table-reference* if the start value for the period in the row is less than or equal to *value2* and the end value for the period in the row is greater than or equal to *value1*. The *table-reference* contains zero rows if *value1* is greater than *value2*. If *value1* = *value2*, the expression is equivalent to `AS OF value1`. If *value1* or *value2* is the null value, the *table-reference* is an empty table.

value1* or *value2

Specifies an expression that returns a value of a built-in data type. The result of the expression must be comparable to the data type of the columns of the specified period according to the comparison rules specified in [Assignment and comparison \(Db2 SQL\)](#).

The expression must not have a timestamp precision that is greater than the precision of the columns for the period.

If the begin and end columns of the period are defined as `TIMESTAMP WITHOUT TIME ZONE`, the expression must not return a value of a timestamp with a time zone.

The expression can contain any of the following supported operands:

- A constant
- A special register
- A variable
- An array element specification

- A built-in scalar function whose arguments are supported operands
Note: The scalar function must not be AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.
- A CAST specification where the cast operand is a supported operand
- An expression that uses arithmetic operators and operands

A period specification for a view must not contain an untyped parameter marker.

Notes

Correlated references in *table-reference*:

In general, nested table expressions and table functions can be specified in any FROM clause. Columns from the nested table expressions and table functions can be referenced in the select list and in the rest of the fullselect using the correlation name. The scope of this correlation name is the same as correlation names for other table or view names in the FROM clause. The basic rule that applies for both these cases is that the correlated reference must be from a *table-reference* at a higher level in the hierarchy of subqueries.

Nested table expressions can be used in place of a view to avoid creating a view when general use of the view is not required. They can also be used when the result table is based on host variables.

For table functions, an additional capability exists. A table function can contain one or more correlated references to other tables in the same FROM clause if the referenced tables precede the reference in the left-to-right order of the tables in the FROM clause. The same capability exists for nested table expressions if the optional keyword TABLE is specified; otherwise, only references to higher levels in the hierarchy of subqueries is allowed.

A nested table expression or table function that contains correlated references to other tables in the same FROM clause:

- Cannot participate in a FULL OUTER JOIN or a RIGHT OUTER JOIN
- Can participate in LEFT OUTER JOIN or an INNER JOIN if the referenced tables precede the reference in the left-to-right order of the tables in the FROM clause

The following table shows some examples of valid and invalid correlated references. TABF1 and TABF2 represent table functions.

Table 21. Examples of correlated references		
Subselect	Valid	Reason
SELECT T.C1, Z.C5 FROM TABLE(TABF1(T.C2)) AS Z, T WHERE T.C3 = Z.C4;	No	T . C2 cannot be resolved because T does not precede TABF1 in FROM
SELECT T.C1, Z.C5 FROM T, TABLE(TABF1(T.C2)) AS Z WHERE T.C3 = Z.C4;	Yes	T precedes TABF1 in FROM, making T . C2 known
SELECT A.C1, B.C5 FROM TABLE(TABF2(B.C2)) AS A, TABLE(TABF1(A.C6)) AS B WHERE A.C3 = B.C4;	No	B in B . C2 cannot be resolved because the table function that would resolve it, TABF1, follows its reference in TABF2 in FROM

Table 21. Examples of correlated references (continued)

Subselect	Valid	Reason
<pre>SELECT D.DEPTNO, D.DEPTNAME, EMPINFO.AVGSAL, EMPINFO.EMPCOUNT FROM DEPT D, (SELECT AVG(E.SALARY) AS AVGSAL, COUNT(*) AS EMPCOUNT FROM EMP E WHERE E.WORKDEPT = D.DEPTNO) AS EMPINFO;</pre>	No	DEPT precedes nested table expression, but keyword TABLE is not specified, making D.DEPTNO unknown
<pre>SELECT D.DEPTNO, D.DEPTNAME, EMPINFO.AVGSAL, EMPINFO.EMPCOUNT FROM DEPT D, TABLE (SELECT AVG(E.SALARY) AS AVGSAL, COUNT(*) AS EMPCOUNT FROM EMP E WHERE E.WORKDEPT = D.DEPTNO) AS EMPINFO;</pre>	Yes	DEPT precedes nested table expression and keyword TABLE is specified, making D.DEPTNO known

Affects of special registers:

The setting of the CURRENT TEMPORAL BUSINESS_TIME and CURRENT TEMPORAL SYSTEM_TIME special registers might affect the result of a query, as described in the following situations:

- Assume the following conditions:
 - A table reference is an application-period temporal table.
 - The columns of the BUSINESS_TIME period are defined as TIMESTAMP(6).
 - The CURRENT TEMPORAL BUSINESS_TIME special register is set to a non-null value.

In this case, a query is executed as if it contained the following specification:

```
FOR BUSINESS_TIME AS OF CURRENT TEMPORAL BUSINESS_TIME
```

- Assume the following conditions:
 - A table reference is an application-period temporal table.
 - The columns of the BUSINESS_TIME period are defined as DATE.
 - The CURRENT TEMPORAL BUSINESS_TIME special register is set to a non-null value.

In this case, a query is executed as if it contained the following specification:

```
FOR BUSINESS_TIME AS OF CAST(CURRENT TEMPORAL BUSINESS_TIME AS DATE)
```

- If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, a query is executed as if it contained the following specification:

```
FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME
```

Related reference

[Examples of subselects \(Db2 SQL\)](#)

Chapter 11. Db2 SQL codes for SQL DI

Db2 updates the following SQL codes to support SQL DI.

-154	THE STATEMENT FAILED BECAUSE VIEW OR TABLE DEFINITION IS NOT VALID	<i>expression-number</i> The number of the invalid column or key expression. <i>reason-code</i> A numeric value that indicates the reason for the failure.
Explanation The CREATE VIEW or DECLARE GLOBAL TEMPORARY TABLE statement is not valid for one of the following reasons: <ul style="list-style-type: none">• The CREATE VIEW or DECLARE GLOBAL TEMPORARY TABLE statement references a remote object.• The CREATE VIEW statement references one of the following scalar functions:<ul style="list-style-type: none">– AI_ANALOGY– AI_COMMONALITY– AI_SEMATIC_CLUSTER– AI_SIMILARITY– UNPACK		1 Contains a subquery. 2 Does not contain at least one reference to a column. 3 References a special register. 4 Includes a CASE expression. 5 Includes a user-defined function. 6 Appears more than once in the index. 7 References a qualified column name. 8 References a column that is defined with a FIELDPROC. 9 References the LOWER or UPPER function without a locale name or the input string-expression is FOR BIT DATA. 10 References the TRANSLATE function without an output translation table. 11 The encoding scheme of the result of a column or key expression is different than the CCSID encoding scheme of the table. 12 The SUBSTR built-in function is allowed to reference the inline portion of a LOB column in the specified context. In addition, the START and LENGTH arguments of the SUBSTR function must be constants. 13 References one of the following built-in functions: <ul style="list-style-type: none">• VERIFY_GROUP_FOR_USER• VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER
System action The statement cannot be processed. The specified object is not defined.		
Programmer response The implied function is not supported.		
SQLSTATE 42909		
Related reference CREATE TABLE (Db2 SQL) CREATE VIEW (Db2 SQL) DECLARE GLOBAL TEMPORARY TABLE (Db2 SQL)		
-356	COLUMN OR KEY EXPRESSION <i>expression-number</i> IS NOT VALID, REASON CODE = <i>reason-code</i>	
Explanation The CREATE INDEX statement cannot be processed because a column or key expression is not valid.		

- VERIFY_ROLE_FOR_USER.

14

Contains an expression that requires the use of an implicit time zone value. For example, the key expression might include an explicit cast of a TIMESTAMP WITHOUT TIME ZONE value to a TIMESTAMP WITH TIME ZONE value.

15

References a global variable.

25

A specification for an index on an EBCDIC table includes one or more Db2 11 Unicode columns and one or more Db2 12 or later Unicode columns. For more information, see [Unicode columns in EBCDIC tables \(Db2 SQL\)](#).

116

In an invocation of the JSON_VAL built-in function in a *key-expression* for an index, the third argument of the function must end in ':na', to indicate that the first argument does not contain a JSON array.

117

In an invocation of the JSON_VAL built-in function in a *key-expression* for an index, if the first argument of the function is a column, that column must be contained in a table in a partition-by-growth table space.

118

If there is an invocation of the JSON_VAL built-in function in a *key-expression* for an index, the CREATE INDEX statement must not reference a LOB column other than the LOB column that is the argument to the JSON_VAL function. Such a CREATE INDEX statement can refer only to a single LOB column.

119

If a *key-expression* for an index contains an invocation of the JSON_VAL function, the invocation must be the outermost expression for *key-expression*.

120

The *key-expression* must not reference one of the following built-in functions:

- AI_ANALOGY
- AI_COMMONALITY
- AI_SEMANTIC_CLUSTER
- AI_SIMILARITY

System action

The statement cannot be processed.

Programmer response

Correct the error in the key expression, and reissue the statement.

SQLSTATE

429BX

-390

OBJECT *object-name*, SPECIFIC NAME *specific-name*, IS NOT VALID IN THE CONTEXT WHERE IT IS USED

Explanation

One of the following situations occurred:

- A function resolved to a specific function that is not valid in the context where it is used.
- UNNEST was used in a context in which it is not allowed.

object-name

The name of the object.

specific-name

The specific name. If *specific-name* is an empty string, then the function resolved to the built-in function identified by *function-name*.

If the error is for an invalid use of UNNEST, *specific-name* is *N.

Possible causes for this error include:

- A scalar or aggregate function is referenced where only a table function is allowed (such as in the FROM clause of a query).
- A table function is referenced where only a scalar or aggregate function is allowed (such as in an expression).
- A function is referenced in a SOURCE clause of a CREATE FUNCTION statement, but a source function cannot be defined on that function (or on that specific function signature).
- Function XMLMODIFY is referenced where it is not the topmost expression on the right side of the SET assignment clause in an update.
- A generic table function is referenced, but a *typed-correlation-clause* is not specified.
- A *typed-correlation-clause* is specified, but the referenced function is not a generic table function.
- UNNEST was specified in an unsupported context.
- A CORRELATION, COVARIANCE, COVARIANCE_SAMP or ARRAY_AGG set function is referenced where a CUBE, ROLLUP or GROUPING SETS clause exists in the same SQL statement.

- ARRAY_AGG is referenced in a fullselect that includes an ORDER BY clause or a DISTINCT clause.
- LISTAGG is referenced in a fullselect that includes a DISTINCT clause.
- AI_ANALOGY, AI_SEMATIC_CLUSTER, or AI_SIMILARITY is referenced in one of the following statements where they are not allowed:
 - CREATE FUNCTION (sourced)
 - CREATE FUNCTION (inlined SQL scalar)
 - CREATE FUNCTION (SQL table)
- The fullselect must not contain a period specification.
- The object that is specified in the FROM clause of the fullselect cannot be a view with columns of length 0.
- The fullselect cannot contain a reference to a created global temporary table, a declared global temporary table, an accelerator-only table, or another materialized query table.
- The fullselect cannot directly or indirectly reference a base table that has been activated for the row or column access control or a base table for which a row permission or a column mask has been defined.
- The fullselect must not refer to host variables or include parameter markers.
- The fullselect must not refer to global variables.
- The fullselect must not include the following built-in functions: AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.

System action

The statement cannot be processed.

Programmer response

For a function, ensure that the correct function name and arguments are specified and that the SQL path includes the schema where the correct function is defined. You might need to change the function name, arguments, or SQL path (using SET CURRENT PATH or the PATH bind option), or change the context in which the function is used.

SQLSTATE

42887

Related concepts
[Functions \(Db2 SQL\)](#)
Related reference
[table-reference \(Db2 SQL\)](#)

-20058 THE FULLSELECT SPECIFIED FOR MATERIALIZED QUERY TABLE table-name IS NOT VALID.

Explanation

The materialized query table definition has specific rules regarding the contents of the fullselect, and the fullselect that was specified did not conform to these rules.

table-name
The name of the materialized query table.

General restrictions: The following restrictions apply:

- The length of each result column of the fullselect must not be 0.
- The fullselect cannot contain a column of a LOB or XML data type.
- No more than one table in the fullselect can contain a security label column.

Additional restrictions when ENABLE QUERY OPTIMIZATION is in effect:

- The fullselect must be a subselect.
- The outermost SELECT list of the subselect must not reference data that is encoded with different CCSID sets.
- The subselect cannot include the following:
 - A special register
 - A scalar fullselect
 - A row change timestamp column
 - A ROW CHANGE expression
 - An expression for which implicit time zone values apply (for example, cast a timestamp to a timestamp with time zone)
 - The RAND built-in function
 - The RID built-in function
 - A user-defined scalar or table function that is not deterministic or that has external actions
 - Any predicates that include a subquery
 - A row-value-expression in a predicate
 - A join using the INNER JOIN syntax, or an outer join
 - A lateral correlation
 - a nested table expression or view that requires temporary materialization
 - A direct or indirect reference to a table that uses activated row or column access controls, or a table for which row or column access controls have been defined.
 - A FETCH FIRST clause

- A reference to a global variable
- A collection-derived table (UNNEST)
- A GROUPING SETS or *super-groups* clause
- A reference to the LISTAGG, MEDIAN, PERCENTILE_CONT, or PERCENTILE_DISC function
- If a table with a security label is referenced, the security label column must be referenced in the outer select list of the subselect.
- If the subselect references a view, the fullselect in the view definition must satisfy all other restrictions.

System action

The statement cannot be processed.

Programmer response

Change the fullselect in the CREATE TABLE or ALTER TABLE statement so that it conforms to the rules listed above.

SQLSTATE

428EC

Related reference

[CREATE TABLE \(Db2 SQL\)](#)

[ALTER TABLE \(Db2 SQL\)](#)

-20474 **PERMISSION OR MASK CANNOT BE CREATED FOR THE *object-name* OBJECT OF THE *object-type* TYPE. REASON CODE *reason-code*.**

Explanation

The CREATE PERMISSION or CREATE MASK statement cannot be processed.

object-name

The name of the object.

object-type

The type of object.

reason-code

The reason for the message or SQL code, indicated by one of the following values:

1

The definition references the table for which the row permission or the column mask is being defined.

2

The definition references a table function or a collection-derived table (UNNEST).

3

The definition references a user-defined function that is not secure.

4

The definition references one of the following functions:

- A function that is defined as not deterministic
- A function that is defined to have an external action
- A function that is defined with the MODIFIES SQL DATA option

5

The definition references an OLAP specification.

6

The definition references an XMLEXISTS predicate.

7

The definition references a ROW CHANGE expression.

8

The definition references a sequence reference.

9

The definition references a created or declared temporary table.

10

The definition references a table that was implicitly created for an XML column.

11

The definition references * or name . * in a SELECT clause.

12

The definition references a column that is defined with a FIELDPROC.

13

The definition references a language element that requires multiple encoding scheme processing.

14

The definition references an ordinary SQL identifier that contains a dash (-).

16

The body of a row permission or column mask includes a period specification.

17

One of the following situations occurred:

- An attempt was made to create a row permission or column mask on one of the following table types:
 - A table that is defined with a period

- A history table
- An archive-enabled table
- An archive table
- An accelerator-only table

You cannot create row permissions or column masks on these types of tables.

- An attempt was made to implicitly create a default row permission for an ACTIVATE ROW ACCESS CONTROL specification.

18

The definition references a Db2 11 Unicode column in an EBCDIC table.

19

A column mask cannot be defined for a Db2 11 Unicode column in an EBCDIC table.

32

The column for which the mask is defined was done so with a field procedure (FIELDPROC).

33

The data type of the return expression is not the same as the data type of the column on which the column mask is defined.

34

The length attribute of the return expression is not the same as the length attribute of the column on which the column mask is defined.

35

The null attribute of the return expression is not the same as the null attribute of the column on which the column mask is defined.

36

The subtype, encoding scheme, or CCSID of the return expression is not the same as the corresponding attribute of the column on which the column mask is defined.

37

An attribute of the return expression is not the same as the corresponding attribute of the column on which the column mask is defined. The attribute is not one of the attributes that are described in reason codes 33-36.

39

The definition references an accelerator-only table.

51

A row permission cannot be created for a table that has a security label column.

52

A row permission cannot reference a table that has a security label column.

53

GROUPING SETS or *super-groups* cannot be specified in the definition of a column mask or a row permission control.

54

The definition references the AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY function.

System action

The statement cannot be processed.

Programmer response

Correct the syntax, and reissue the statement.

SQLSTATE

428HB

Related concepts

[Column mask \(Managing Security\)](#)

[Row permission \(Managing Security\)](#)

[Temporal tables and data versioning \(Db2 Administration Guide\)](#)

[Archive-enabled tables and archive tables \(Introduction to Db2 for z/OS\)](#)

Related reference

[CREATE MASK \(Db2 SQL\)](#)

[CREATE PERMISSION \(Db2 SQL\)](#)

-20524

**INVALID PERIOD SPECIFICATION
OR PERIOD CLAUSE FOR PERIOD
period-name. REASON CODE =
reason-code.**

Explanation

A period specification or period clause is invalid.

period-name

The period that is invalid.

reason-code

A numeric value that indicates why the period is invalid:

1

The period name was specified more than one time for the table reference.

2

The SYSTEM_TIME period was specified, but the table is not a system-period temporal table.

3

period-name violated the following requirement: each expression must return a

value of a built-in data type and can contain any of the following supported operands:

- A constant
- A special register
- A variable, which can be either a host variable, an SQL variable, an SQL parameter, a transition variable, or a global variable
- A parameter marker
- A CAST specification, where the cast operand is a supported operand
- An expression that uses arithmetic operators and operands
- A scalar function whose arguments are supported operands (Nested function invocations are not permitted.)

These rules have the following exceptions:

- A period specification or period clause for a view must not contain an untyped parameter marker.
- The source expression of SET CURRENT TEMPORAL BUSINESS_TIME and SET CURRENT TEMPORAL SYSTEM_TIME statements must not contain a parameter marker or a transition variable.
- The scalar functions must not be AI_ANALOGY, AI_COMMONALITY, AI_SEMANTIC_CLUSTER, or AI_SIMILARITY.

4

The period specification or period clause was specified for a view where the view definition includes a user-defined function.

5

The precision of an expression must be greater than the precision of the columns of the period. If the expression is a string, it is first converted to a timestamp, and the value must not contain more significant fractional seconds digits than the precision of the column.

6

FOR SYSTEM_TIME was specified. However, the value of the CURRENT TEMPORAL SYSTEM_TIME special register is not null, and the SYSTIMESENSITIVE bind option is set to YES. Therefore, you cannot also explicitly specify FOR SYSTEM_TIME.

7

FOR BUSINESS_TIME was specified. However, the value of the CURRENT TEMPORAL BUSINESS_TIME special register is not null, and the BUSTIMESENSITIVE bind option is set to YES. Therefore, you cannot also explicitly specify FOR BUSINESS_TIME.

8

The period specification or period clause was specified for one of the following items:

- A table that is not an application-period temporal table
- A view for which an application-period temporal table is not referenced in the outermost FROM clause of the view definition, or an INSTEAD OF trigger is defined on the view.

9

An expression must not return a value with a time zone if the begin and end columns of the specified period are defined as timestamp without time zone.

System action

The statement cannot be processed.

User response

Correct the syntax and resubmit the statement.

SQLSTATE

428HY

Related tasks

[Querying temporal tables \(Db2 Administration Guide\)](#)

Related reference

[CURRENT TEMPORAL BUSINESS_TIME \(Db2 SQL\)](#)

[CURRENT TEMPORAL SYSTEM_TIME \(Db2 SQL\)](#)

-20577

SQL DATA INSIGHTS HAS NOT BEEN CONFIGURED IN Db2, REASON *reason-code*.

Explanation

An attempt was made to use an SQL Data Insights function, but failed due to one or more of the following reasons:

reason-code

3

SQL DI is not configured for this Db2.

4

SQL DI detects an incompatible SQL DI table.

5

SQL DI detects invalid values in the SQL DI tables.

SQL DI detects an incompatible code in z/OS.

System action

- For RC 3, run the DSNTIJAI job to configure this Db2 system to use SQL DI. Make sure that you have the database administrator authority.
- For RC 4, check that your Db2 code level is compatible with your SQL DI catalog tables. Update your Db2 by following the ++HOLD instructions in the latest PTF and retrain the model.
- For RC 5, verify that you have applied the latest PTFs for your SQL DI UI and z/OS. Retrain the model after both PTFs are successfully applied.
- For RC 6, check that the code levels of your Db2 and z/OS systems are compatible. Apply the latest z/OS PTF and retrain the model.

SQLSTATE

0A502

-20578 **MODEL COLUMNS CANNOT BE DETERMINED FOR FUNCTION *function-name*.**

Explanation

No model columns can be determined for the arguments of the SQL Data Insights function.

function-name

The name of the SQL Data Insights function.

Arguments of the SQL Data Insights functions must specify model columns. If the argument is a column name, the model column is that column name. The model column may also be specified on the function invocation using the USING MODEL COLUMN clause. Some SQL Data Insights functions infer the model column of arguments from other arguments of the function. Refer to the documentation of the specific function being invoked to understand how the model column of the arguments are determined.

System action

The statement that contains the function cannot be processed.

Programmer response

Correct the invocation of the function to ensure that the arguments have specified a model column or that they specify the arguments such that Db2 is able to infer the model columns. Once the corrections to the SQL are made, reissue the statement. Refer

to the documentation of the specific function being invoked to understand how the model column of the arguments are determined.

SQLSTATE

428ID

-20579 **IN FUNCTION *function-name* MODEL COLUMN *column-name* FROM MODEL *model-name* CANNOT BE USED, REASON *reason-code*.**

Explanation

In an invocation of an SQL Data Insights function, a model column or model was not found or found to be not usable.

function-name

The name of the SQL Data Insights function in which *model-name* and *column-name* is referenced.

column-name

The model column that is not included in the model. *column-name* may be indicated as *N when it cannot be determined for some of reasons below.

model-name

The model table that does not include the model column *column-name*.

reason-code

A numeric value which indicates why the model or column cannot be used:

1

There is no model indicated by *model-name*.

2

The model *model-name* exists, but *column-name* is not a configured column in *model-name*.

3

The model *model-name* is not enabled.

4

The model *model-name* is incomplete.

5

The model columns specified in *function-name* are from different models. *model-name* and *column-name* indicate one model column that is different from other model columns specified in the function.

6

Illegal use of model column qualifier in *column-name*. The qualifier must refer to a table name or view name, or an alias to a table name or view name. It may not refer to a synonym name, or the correlation name of a table expression.

7

One argument to AI_SIMILARITY has a model column that was indicated as a primary key column during model training, and the other argument specifies a different model column.

8

For AI_ANALOGY, arguments *source-1* and *source-2* must use the same model column, and arguments *target-1* and *target-2* must use the same model column.

System action

The statement that contains the function cannot be processed.

Programmer response

Review the reason indicated by *reason-code* and ensure that the model and model column referenced in the SQL Data Insights function refers to a model that has completed training. Correct the usage of the model columns in the function, if necessary.

SQLSTATE

428ID

-20580

**IN AI FUNCTION *function-name*,
ARGUMENT *n* IS NOT USABLE,
REASON *reason-code*.**

Explanation

An SQL Data Insights function specified an argument that cannot be used.

function-name

The name of the SQL DI function.

n

The numeric position of the argument that cannot be used.

reason-code

10

The *source-1* and *source-2* arguments to AI_ANALOGY must not be the same value. If the SQL DI data types of the model columns for *source-1* and *source-2* are numeric, their values, though different, may be treated as the same because they may belong to the same cluster during model training. See the Model details page of the SQL DI UI for more information how numeric values in a cluster are processed in function arguments.

SQLSTATE

428ID

Information resources for Db2 for z/OS and related products

You can find the online product documentation for Db2 13 for z/OS and related products in IBM Documentation.

For all online product documentation for Db2 13 for z/OS, see [IBM Documentation](https://www.ibm.com/docs/en/db2-for-zos/13) (<https://www.ibm.com/docs/en/db2-for-zos/13>).

For other PDF manuals, see PDF format manuals for Db2 13 for z/OS (<https://www.ibm.com/docs/en/db2-for-zos/13?topic=zos-pdf-format-manuals-db2-13>).

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as shown below:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. (enter the year or years).

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)® are trademarks or registered marks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at: <http://www.ibm.com/legal/copytrade.shtml>.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions:

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

Glossary

The glossary is available in IBM Documentation

For definitions of Db2 for z/OS terms, see [Db2 glossary \(Db2 Glossary\)](#).

Index

A

- accelerator tables [169](#)
- accessibility
 - keyboard [vi](#)
 - shortcut keys [vi](#)
- adding
 - AI object [22](#)
 - object [22](#)
- administering SQL Data Insights [29](#)
- AI object
 - adding [22](#)
 - AI query [22](#), [26](#)
- AI query
 - enabling [22](#), [26](#)
 - running [25](#)
- AI_ANALOGY scalar function [43](#)
- AI_COMMONALITY scalar function [45](#)
- AI_SEMANTIC_CLUSTER scalar function [49](#)
- AI_SIMILARITY scalar function [47](#)
- APPEND
 - clause of CREATE TABLE statement [175](#)
- AS clause
 - CREATE VIEW statement [196](#)
- AS IDENTITY clause
 - CREATE TABLE statement [150](#)
- AS LOCATOR clause
 - CREATE FUNCTION statement [55](#)
- AS SECURITY LABEL clause
 - CREATE TABLE statement [154](#)
- ASC clause
 - CREATE INDEX statement [87](#)
 - CREATE TABLE statement [170](#)
- AUDIT
 - clause of CREATE TABLE statement [172](#)
- auditing
 - CREATE TABLE statement [172](#)

B

- BIGINT
 - data type
 - CREATE TABLE statement [138](#)
- BINARY
 - data type [138](#)
- BLOB (binary large object)
 - data type [138](#)
- BLOB LARGE OBJECT data type [138](#)
- BUFFERPOOL
 - clause of CREATE TABLE statement [176](#)
- BUFFERPOOL clause
 - CREATE INDEX statement [99](#)

C

- CALLED ON NULL INPUT clause
 - CREATE FUNCTION (inlined SQL scalar) statement [68](#)

- capturing changed data
 - CREATE TABLE statement [173](#)
- CARDINALITY clause [244](#)
- CARDINALITY MULTIPLIER clause [244](#)
- CASCADE delete rule
 - CREATE TABLE statement [159](#)
- catalog name
 - VCAT clause
 - CREATE INDEX statement [93](#)
- CCSID
 - clause of CREATE FUNCTION (inlined SQL scalar) statement [66](#)
 - clause of CREATE FUNCTION statement [54](#)
 - clause of CREATE TABLE statement [173](#)
- CHAR LARGE OBJECT data type [138](#)
- CHAR VARYING data type [138](#)
- CHARACTER data type
 - CREATE TABLE statement [138](#)
- CHARACTER LARGE OBJECT data type [138](#)
- CHARACTER VARYING data type [138](#)
- CHECK
 - clause of CREATE TABLE statement [160](#)
- CLOB (character large object)
 - description [138](#)
- CLOSE
 - clause of CREATE INDEX statement
 - description [99](#)
- CLUSTER clause
 - CREATE INDEX statement [92](#)
- column
 - derived
 - CREATE VIEW statement [195](#)
 - DELETE statement [208](#)
 - UPDATE statement [228](#)
- column masks
 - creating [108](#)
- COMPRESS NO
 - clause of CREATE TABLE statement [175](#)
- COMPRESS NO clause
 - CREATE INDEX statement [97](#)
- COMPRESS YES
 - clause of CREATE TABLE statement [175](#)
- COMPRESS YES clause
 - CREATE INDEX statement [97](#)
- configuration
 - verifying [16](#)
- configuring
 - Db2
 - configuring [13](#)
 - keyring [12](#)
 - keystore [12](#)
 - roadmap [5](#)
 - setup user ID [9](#)
 - SQL Data Insights
 - configuring [13](#)
 - SSL certificate [12](#)
 - user authentication [12](#)

- configuring (*continued*)
 - z/OS system environment [9](#)
- connection
 - creating [21](#)
- CONSTRAINT
 - clause of CREATE TABLE statement [160](#)
- CONSTRAINT clause
 - CREATE TABLE statement [142](#), [156](#), [157](#)
- CONTAINS SQL clause
 - CREATE FUNCTION (inlined SQL scalar) statement [68](#)
- COPY
 - clause of CREATE INDEX statement [101](#)
- CREATE FUNCTION (inlined SQL scalar) statement
 - description [62](#)
 - example [70](#)
- CREATE FUNCTION (sourced) statement
 - description [51](#)
 - example [61](#)
- CREATE FUNCTION (SQL table) statement
 - description [71](#)
 - examples [78](#)
- CREATE INDEX statement
 - description [79](#)
 - example [107](#)
- CREATE MASK statement
 - description [108](#)
 - examples [115](#)
- CREATE PERMISSION statement
 - description [117](#)
 - examples [123](#)
- CREATE TABLE statement
 - description [124](#)
 - example [190](#)
 - materialized query table [124](#)
- CREATE VIEW statement
 - description [193](#)
 - example [199](#)
- creating
 - connection [21](#)
 - started task
 - Spark [31](#)
 - SQL DI [30](#)
- CURRENT TEMPORAL BUSINESS_TIME special register
 - assigning a value [216](#)
- CURRENT TEMPORAL SYSTEM_TIME special register
 - assigning a value [217](#)
- cursor
 - closing
 - error in UPDATE [231](#)

D

- DATA CAPTURE clause
 - CREATE TABLE statement [173](#)
- data type
 - CREATE TABLE statement [138](#)
- DATE
 - data type
 - CREATE TABLE statement [138](#)
- DBCLOB (double-byte character large object)
 - data type [138](#)
- DECFLOAT
 - data type

- DECFLOAT (*continued*)
 - data type (*continued*)
 - CREATE TABLE statement [138](#)
- DECIMAL
 - data type
 - CREATE TABLE statement [138](#)
- declare default element namespace clause
 - CREATE INDEX statement [89](#)
- declare namespace clause
 - CREATE INDEX statement [89](#)
- DEFER
 - clause of CREATE INDEX statement [99](#)
- DEFINE clause
 - CREATE INDEX statement [96](#)
- DEFINITION ONLY clause
 - CREATE TABLE statement [189](#)
- DELETE
 - statement
 - description [200](#)
 - example [215](#)
- delete rules [210](#)
- deleting
 - rows from a table [200](#)
- DESC clause
 - CREATE INDEX statement [87](#)
 - CREATE TABLE statement [170](#)
- DETERMINISTIC clause
 - CREATE FUNCTION (inlined SQL scalar) statement [67](#)
- disability [vi](#)
- distinct type
 - CREATE TABLE statement [142](#)
- DOUBLE data type
 - CREATE TABLE statement [138](#)
- DOUBLE PRECISION data type
 - CREATE TABLE statement [138](#)
- DSNTIP81 installation panel [41](#)
- DSSIZE
 - clause of CREATE INDEX statement [99](#)
 - clause of CREATE TABLE statement [176](#)

E

- edit routine
 - named in CREATE TABLE statement [172](#)
 - specified by EDITPROC option [172](#)
- EDITPROC clause
 - CREATE TABLE statement [172](#)
- ENABLE QUERY OPTIMIZATION clause
 - CREATE TABLE statement [160](#)
- enabling
 - AI object [22](#), [26](#)
 - AI query [22](#), [26](#)
- ENDING AT clause
 - CREATE INDEX statement [98](#)
 - CREATE TABLE statement [171](#)
- ENFORCED clause
 - CREATE TABLE statement [160](#)
- ERASE clause
 - CREATE INDEX statement [94](#)
- error
 - during update [231](#)
- EXCLUDING COLUMN DEFAULTS clause
 - CREATE TABLE statement [164](#)
- EXCLUDING IDENTITY COLUMN ATTRIBUTES clause

EXCLUDING IDENTITY COLUMN ATTRIBUTES clause (*continued*) **H**

CREATE TABLE statement [163](#)

EXCLUDING ROW CHANGE TIMESTAMP COLUMN

ATTRIBUTES clause

CREATE TABLE statement [163](#)

exit routine

named in CREATE TABLE statement [153](#)

EXTERNAL ACTION clause

CREATE FUNCTION (inlined SQL scalar) statement [67](#)

F

field procedure

named in CREATE TABLE statement [153](#)

FIELDPROC clause

CREATE TABLE statement [153](#)

FINAL TABLE clause

FROM clause [245](#)

FLOAT

data type

CREATE TABLE statement [138](#)

FOR

clause of CREATE TABLE statement [138](#)

FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
clause

CREATE TABLE statement [148](#)

FOR ROW n OF ROWSET clause

DELETE statement [209](#)

UPDATE statement [230](#)

FOREIGN KEY clause

CREATE TABLE statement [157](#)

free space

index [95](#)

FREEPAGE

clause of CREATE INDEX statement
description [95](#)

FROM clause

DELETE statement [204](#)

fullselect

CREATE VIEW statement [196](#)

functions

scalar

AI_ANALOGY [43](#)

AI_COMMONALITY [45](#)

AI_SEMANTIC_CLUSTER [49](#)

AI_SIMILARITY [47](#)

G

GBPCACHE clause

CREATE INDEX statement [96](#)

GENERATE KEY USING clause

CREATE INDEX statement [89](#)

GENERATED clause

CREATE TABLE statement [147](#)

GRAPHIC

data type

CREATE TABLE statement [138](#)

GROUP BY clause

cannot join view [198](#)

hardware requirements

planning [6](#), [8](#)

I

identity column

CREATE TABLE statement [150](#)

IMPLICITLY HIDDEN clause

CREATE TABLE statement [154](#)

IN

clause of CREATE TABLE statement [168](#), [169](#)

INCLUDE clause

DELETE statement [207](#)

UPDATE statement [228](#)

INCLUDING COLUMN DEFAULTS clause

CREATE TABLE statement [164](#)

INCLUDING IDENTITY COLUMN ATTRIBUTES clause

CREATE TABLE statement [163](#)

INCLUDING ROW CHANGE TIMESTAMP COLUMN
ATTRIBUTES clause

CREATE TABLE statement [163](#)

INCLUSIVE clause

CREATE INDEX statement [99](#)

CREATE TABLE statement [171](#)

index

creating with CREATE INDEX statement [79](#)

partitioning [98](#)

SQL DI table [33](#)

INDEX clause

CREATE INDEX statement [85](#)

INLINE LENGTH clause

CREATE TABLE statement [154](#)

installation

verifying [16](#)

installing

roadmap [5](#)

SQL Data Insights [14](#)

INTEGER

data type

CREATE TABLE statement [138](#)

integrated catalog facility

CREATE INDEX statement [95](#)

isolation level

control by SQL statement

DELETE statement [210](#)

UPDATE statement [231](#)

isolation-clause

DELETE statement [210](#)

UPDATE statement [231](#)

K

key

length

partitioning index [98](#), [228](#)

primary

defining on a single column [143](#)

key-expression clause

CREATE INDEX statement [86](#)

keyring

configuring [12](#)

- keyring-based keystore
 - configuring [12](#)
- keystore
 - configuring [12](#)

L

- LANGUAGE SQL clause
 - CREATE FUNCTION (inlined SQL scalar) statement [66](#)
- LIKE clause
 - CREATE TABLE statement [161](#)
- links
 - non-IBM Web sites [264](#)
- lock
 - during update [231](#)
- LOGGED
 - clause of CREATE TABLE statement [174](#)
- LONG VARCHAR data type [180](#)
- LONG VARGRAPHIC data type [180](#)

M

- materialized-query-definition
 - CREATE TABLE statement [166](#)
- MAX AI DATA CACHING field of DSNTIP81 [41](#)
- MAXVALUE
 - clause of CREATE TABLE statement [151](#)
- MEMBER CLUSTER
 - clause of CREATE TABLE statement [176](#)
- MINVALUE
 - clause of CREATE TABLE statement [151](#)
- model
 - viewing [24](#)
- modifying
 - settings [29](#)
 - SQL DI [29](#)
- MXAIDTCACH subsystem parameter [41](#)

N

- nested table expressions [243](#)
- network requirements
 - planning [6, 8](#)
- NO ACTION delete rule
 - CREATE TABLE statement [159](#)
- NO EXTERNAL ACTION clause
 - CREATE FUNCTION (inlined SQL scalar) statement [67](#)
- NO MAXVALUE
 - clause of CREATE TABLE statement [151](#)
- NO MINVALUE
 - clause of CREATE TABLE statement [151](#)
- NO ORDER
 - clause of CREATE TABLE statement [153](#)
- NOCACHE clause
 - CREATE TABLE statement [189](#)
- NOCYCLE clause
 - CREATE TABLE statement [189](#)
- NOMAXVALUE clause
 - CREATE TABLE statement [189](#)
- NOMINVALUE clause
 - CREATE TABLE statement [189](#)
- NOORDER clause

- NOORDER clause (*continued*)
 - CREATE TABLE statement [189](#)
- NOT CLUSTER clause
 - CREATE INDEX statement [92](#)
- NOT DETERMINISTIC clause
 - CREATE FUNCTION (inlined SQL scalar) statement [67](#)
- NOT ENFORCED clause
 - CREATE TABLE statement [160](#)
- NOT LOGGED
 - clause of CREATE TABLE statement [174](#)
- NOT NULL clause
 - CREATE TABLE statement
 - description [142](#)
- NOT PADDED clause
 - CREATE INDEX statement [92](#)
- NOT VOLATILE
 - clause of CREATE TABLE statement [174](#)
- NULLS LAST clause
 - CREATE TABLE statement [170](#)
- NUMERIC data type
 - CREATE TABLE statement [138](#)

O

- OBID
 - clause of CREATE TABLE statement [173](#)
- object
 - adding [22](#)
- OLD TABLE clause
 - FROM clause [245](#)
- ON clause
 - CREATE INDEX statement [85](#)
- ON DELETE clause
 - CREATE TABLE statement [159](#)
- ORDER
 - clause of CREATE TABLE statement [153](#)
- overview
 - SQL Data Insights [1](#)

P

- PADDED clause
 - CREATE INDEX statement [92](#)
- PAGENUM
 - clause of CREATE TABLE statement [176](#)
- PARAMETER CCSID clause
 - CREATE FUNCTION (inlined SQL scalar) statement [67](#)
 - CREATE FUNCTION statement [56](#)
- PART clause
 - CREATE INDEX statement [106](#)
 - CREATE TABLE statement [189](#)
- PARTITION
 - clause of CREATE INDEX statement [98](#)
- PARTITION BY RANGE
 - clause of CREATE INDEX statement [97](#)
- PARTITION BY RANGE clause
 - CREATE TABLE statement [170](#)
- PARTITION BY SIZE clause
 - CREATE TABLE statement [169](#)
- PARTITION clause
 - CREATE TABLE statement [170](#)
- partition-by-clause
 - CREATE TABLE statement [169](#)

- PARTITIONED clause
 - CREATE INDEX statement [92](#)
- PCTFREE
 - clause of CREATE INDEX statement [95](#)
- PIECESIZE clause
 - CREATE INDEX statement [100](#)
- planning
 - hardware [6, 8](#)
 - installation [6, 8](#)
 - network [6, 8](#)
 - roadmap [5](#)
 - software [6, 8](#)
 - system capacity [6, 8](#)
- port
 - planning [6, 8](#)
- PRIMARY KEY clause
 - CREATE TABLE statement [143, 156](#)
- PRIQTY clause
 - CREATE INDEX statement [93](#)

Q

- QUERYNO clause
 - DELETE statement [210](#)
 - UPDATE statement [231](#)

R

- RACF
 - keyring [12](#)
 - keystore [12](#)
- read-only
 - view [198](#)
- READS SQL DATA clause
 - CREATE FUNCTION (inlined SQL scalar) statement [68](#)
- REAL data type
 - CREATE TABLE statement [138](#)
- referential constraint
 - CREATE TABLE statement [157](#)
- Remote Recovery Data Facility (RRDF) [173](#)
- RESTRICT
 - delete rule
 - CREATE TABLE statement [159](#)
- RETURN-statement clause
 - CREATE FUNCTION (inlined SQL scalar) statement [69](#)
- RETURNS clause
 - CREATE FUNCTION (inlined SQL scalar) statement [66](#)
- RETURNS clause of CREATE FUNCTION statement [55](#)
- roadmap
 - configuring [5](#)
 - installing [5](#)
 - planning [5](#)
- row
 - deleting [200](#)
 - updating [219](#)
- row change timestamp column
 - CREATE TABLE statement [148](#)
- row ID
 - data type [138](#)
- row permission
 - creating [117](#)
- ROWID
 - data type

- ROWID (*continued*)
 - data type (*continued*)
 - CREATE TABLE statement [138](#)
- RRDF (Remote Recovery Data Facility)
 - creating a table for [173](#)
- running
 - AI query [25](#)
 - started task
 - Spark [31](#)
 - SQL DI [30](#)

S

- search condition
 - DELETE statement [208](#)
 - UPDATE statement [229](#)
- SECQTY clause
 - CREATE INDEX statement [94](#)
- SET clause
 - DELETE statement [208](#)
- SET clause of UPDATE statement [228](#)
- SET CURRENT TEMPORAL BUSINESS_TIME statement
 - description [216](#)
- SET CURRENT TEMPORAL SYSTEM_TIME statement
 - description [217](#)
- SET NULL delete rule
 - CREATE TABLE statement [159](#)
- settings
 - modifying [29](#)
- setup user ID
 - configuring [9](#)
- shortcut keys
 - keyboard [vi](#)
- SKIP LOCKED DATA clause
 - DELETE statement [210](#)
 - UPDATE statement [231](#)
- software requirements
 - planning [6, 8](#)
- SOURCE clause of CREATE FUNCTION statement [56](#)
- Spark
 - started task [31](#)
- SPECIFIC clause
 - CREATE FUNCTION (inlined SQL scalar) statement [66](#)
 - CREATE FUNCTION statement [56](#)
- SQL Data Insights
 - administering [29](#)
 - configuring [14](#)
 - installing [14](#)
 - overview [1](#)
 - upgrading [19](#)
- SQL DI
 - started task [30](#)
- SQL DI table
 - index [33](#)
 - tablespace [33](#)
- SQL statements
 - CREATE FUNCTION (inlined SQL scalar) [62](#)
 - CREATE FUNCTION (sourced) [51](#)
 - CREATE FUNCTION (SQL table) [71](#)
 - CREATE INDEX [79](#)
 - CREATE MASK [108](#)
 - CREATE PERMISSION [117](#)
 - CREATE TABLE [124](#)
 - CREATE VIEW [193](#)

SQL statements (*continued*)

DELETE

description [200](#)

example [215](#)

SET CURRENT TEMPORAL BUSINESS_TIME [216](#)

SET CURRENT TEMPORAL SYSTEM_TIME [217](#)

UPDATE

description [219](#)

example [237](#)

SQLCA (SQL communication area)

entry changed by UPDATE [231](#)

SSL certificate

configuring [12](#)

started task

creating [30](#), [31](#)

running [30](#), [31](#)

Spark [31](#)

SQL DI [30](#)

STATIC DISPATCH clause

CREATE FUNCTION (inlined SQL scalar) statement [68](#)

CREATE FUNCTION statement [77](#)

STOGROUP

clause of CREATE INDEX statement [93](#), [95](#)

SYSAIDB.SYSAICOLUMNCENTERSSQL DI table [33](#)

SYSAIDB.SYSAICOLUMNCONFIGSQL DI table [33](#)

SYSAIDB.SYSAICONFIGURATIONSSQL DI table [33](#)

SYSAIDB.SYSAIMODELSSQL DI table [33](#)

SYSAIDB.SYSAIOBJECTSSQL DI table [33](#)

SYSAIDB.SYSAITRAININGJOBSSQL DI table [33](#)

system capacity

planning [6](#), [8](#)

T

table

creating

CREATE TABLE statement [124](#)

table check constraint

defining

CREATE TABLE statement [160](#)

deleting rows [211](#)

updating rows [232](#)

table function reference [243](#)

TABLE LIKE clause

CREATE FUNCTION statement [55](#)

table locator variable [243](#)

table space

creating

implicitly [168](#)

tablespaceSQL DI table [33](#)

TIME

data type

CREATE TABLE statement [138](#)

TIMESTAMP

data type

CREATE TABLE statement [138](#)

TRACKMOD NO

clause of CREATE TABLE statement [176](#)

TRACKMOD YES

clause of CREATE TABLE statement [176](#)

U

UNIQUE clause

CREATE INDEX statement [85](#)

CREATE TABLE statement [143](#), [156](#)

UPDATE

statement

description [219](#)

example [237](#)

update rule [231](#)

updating

rows in a table [219](#)

upgrading

SQL Data Insights [19](#)

user authentication

configuring [12](#)

user-defined function

creating with CREATE FUNCTION (inlined SQL scalar)

statement [62](#)

creating with CREATE FUNCTION statement [51](#), [71](#)

USING clause

CREATE INDEX statement [93](#), [94](#)

USING TYPE DEFAULTS clause

CREATE TABLE statement [164](#)

V

validation routine

VALIDPROC clause [172](#)

VALIDPROC clause

CREATE TABLE statement [172](#)

VALUES clause

CREATE INDEX statement [106](#)

CREATE TABLE statement [189](#)

VARCHAR

data type

CREATE TABLE statement [138](#)

VARGRAPHIC

data type

CREATE TABLE statement [138](#)

VCAT

USING clause

CREATE INDEX statement [93](#), [95](#)

verifying

configuration [16](#)

installation [16](#)

view

creating

CREATE VIEW statement [193](#)

using

read-only [198](#)

VIEW clause

CREATE VIEW statement [193](#)

viewing

AI query [24](#)

model [24](#)

VOLATILE

clause of CREATE TABLE statement [174](#)

VSAM (virtual storage access method)

catalog [95](#)

W

WHERE clause

DELETE statement [208](#)

UPDATE statement [229](#)

WHERE CURRENT OF clause

DELETE statement [209](#)

UPDATE statement [230](#)

WITH CHECK OPTION clause of CREATE VIEW statement
[196](#)

WITH common-table-expression clause of CREATE VIEW
statement [196](#)

X

XML

data type

CREATE TABLE statement [138](#)

XML pattern expression clause

CREATE INDEX statement [89](#)

XMLPATTERN clause

CREATE INDEX statement [89](#)

XMLSCHEMA

data type

CREATE TABLE statement [138](#)



Product Number: 5698-DB2
5698-DBV