

IBM i
7.3

*Database
Administration*

IBM

Note

Before using this information and the product it supports, read the information in [“Notices” on page 45.](#)

This edition applies to IBM i 7.3 (product number 5770-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

This document may contain references to Licensed Internal Code. Licensed Internal Code is Machine Code and is licensed to you under the terms of the IBM License Agreement for Machine Code.

© **Copyright International Business Machines Corporation 1998, 2015.**

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

Contents

Administration.....	1
What's new for IBM i 7.3.....	1
PDF file for Database administration.....	1
Database administration.....	1
Accessing data through client interfaces.....	1
Accessing data with Java.....	2
Accessing data with Domino.....	2
Accessing data with ODBC.....	2
Accessing data with IBM i PASE.....	2
Accessing data with IBM i Access for Windows OLE DB Provider.....	2
Accessing data with IBM i Access for Windows .Net Provider.....	2
Accessing data with Net.Data.....	3
Accessing data through a Linux partition.....	3
Accessing data using Distributed Relational Database Architecture (DRDA).....	3
Altering and managing database objects.....	3
Creating database objects.....	3
Ensuring data integrity.....	4
Importing and exporting data between systems.....	4
Working with multiple databases.....	4
Working with system-period temporal tables.....	4
System-period temporal tables.....	5
Row-begin column.....	5
Row-end column.....	5
Transaction start-ID.....	5
SYSTEM_TIME period.....	5
History tables.....	5
Creating a system-period temporal table.....	6
Additional system-period temporal table CREATE TABLE examples.....	7
Inserting data into a system-period temporal table.....	8
Updating data in a system-period temporal table.....	9
Deleting data in a system-period temporal table.....	10
Using a system-period temporal table for tracking auditing information.....	11
System-period temporal table timestamp value conflicts.....	13
Querying system-period temporal data.....	14
Specifying the time criteria in a query.....	14
Specifying the time criteria by using the CURRENT TEMPORAL SYSTEM_TIME special register.....	16
Specify the time criteria for a view.....	18
Restrictions when inserting, updating, or deleting data in a system-period temporal table.....	20
Native I/O considerations with a system-period temporal table.....	21
Maintaining the history table.....	21
Partitioning a system-period temporal table.....	21
Row and column access control with a system-period temporal table.....	22
Saving and restoring system-period temporal tables.....	22
Using catalogs to find system-period temporal table information.....	23
Temporal table restrictions.....	23
Working with triggers and constraints.....	24
Writing DB2 programs.....	24
Database backup and recovery.....	24
Distributed database administration.....	24
Queries and reports.....	25

Security.....	25
Authority Options for SQL Analysis and Tuning	25
Row and column access control (RCAC).....	30
Overview.....	31
Permissions and masks.....	32
SQL statements	33
Secure functions.....	33
Secure triggers.....	33
Administrative authority.....	33
Best practices when using permissions and masks.....	34
Creating permissions and masks.....	34
Single permission with all users.....	34
Single permission with a group profile.....	34
Single permission with a dependent table.....	34
Single permission with a UDF.....	35
Permissions for each user.....	35
Attributes of multiple permissions.....	36
Unqualified object names.....	36
Dependent objects.....	37
Ownership.....	37
Schema	37
Schema authority.....	37
Secured UDFs.....	37
ALWCPYDTA and isolation level.....	38
Restoring objects.....	38
Additional operations.....	38
Adding application profile to permissions and masks.....	38
Reclaim Storage.....	38
Query Reports.....	38
MQTs.....	39
Temporal tables.....	39
Group Profiles and QIBM_DB_SECADM.....	39
Copy File (CPYF) parameters.....	40
OmniFind Text Search Server for DB2 for i	40
Using RCAC on Multi-Formatted Logical Files.....	40
Propagation of masked data.....	41
Classic Query Engine (CQE) and SQL Query Engine (SQE).....	43
Native open and SQL query differences involving RCAC.....	43
Result set ordering.....	43
Notices.....	45
Programming interface information.....	46
Trademarks.....	46
Terms and conditions.....	47

Database administration

Db2® for IBM® i provides database administration, backup and recovery, query, and security functions.

You can also explore other database information using the main navigation tree or [Database information finder](#).

What's new for IBM i 7.3

Read about new or significantly changed information for the Database administration topic collection.

System-period Temporal Table

A [system-period temporal table](#) is a table that maintains historical versions of its rows.

How to see what's new or changed

To help you see where technical changes have been made, the information center uses:

- The [»](#) image to mark where new or changed information begins.
- The [«](#) image to mark where new or changed information ends.

In PDF files, you might see revision bars (|) in the left margin of new and changed information.

To find other information about what's new or changed this release, see the [Memo to users](#).

PDF file for Database administration

You can view and print a PDF file of this information.


To view or download the PDF version of this document, select [Database administration](#).

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the [Adobe Web site](http://get.adobe.com/reader/) (<http://get.adobe.com/reader/>) .

Database administration

Db2 for i provides various methods for setting up and managing databases.

Related concepts

[Journal management](#)

Accessing data through client interfaces

You can access Db2 for i data through client interfaces on the server, such as the Java Database Connectivity (JDBC) driver, the Open Database Connectivity (ODBC) driver, IBM i Portable Application

Solutions Environment (IBM i PASE), OLE DB Provider, .Net Provider, Net.Data®, or Distributed Relational Database Architecture™ (DRDA).

Accessing data with Java

You can access Db2 for i data in your Java™ programs by using the Java Database Connectivity (JDBC) driver that is included with the IBM Developer Kit for Java licensed program.

The driver allows you to perform the following tasks:

- Access database files.
- Access JDBC database functions with embedded Structured Query Language (SQL) for Java.
- Run SQL statements and process results.

Related concepts

[Accessing your System i5 database with the IBM Developer Kit for Java JDBC driver](#)

Accessing data with Domino

You can use IBM Lotus® Domino® for i5/OS to integrate data from Db2 for i databases and Domino databases in both directions.

To take advantage of this integration, you need to understand and manage how authorizations work between the two types of databases.

Related concepts

[Lotus Domino for i5/OS](#)

Accessing data with ODBC

You use the IBM i Access for Windows Open Database Connectivity (ODBC) driver to enable your ODBC client applications to effectively share data with each other and with the server.

Related concepts

[ODBC administration](#)

Accessing data with IBM i PASE

IBM i Portable Application Solutions Environment (IBM i PASE) is an integrated runtime environment for AIX®, UNIX, or other applications that are running on the IBM i operating system. IBM i PASE supports the Db2 for i call level interface (CLI).

Related concepts

[Database](#)

Accessing data with IBM i Access for Windows OLE DB Provider

The IBM i Access for Windows OLE DB Provider, along with the Programmer's Toolkit, facilitates the IBM i client/server application development from the Microsoft Windows client PC.

The IBM i Access for Windows OLE DB Provider gives programmers record-level access interfaces to Db2 for i database files. In addition, it provides support for SQL, data queues, programs, and commands.

Related reference

[System i Access for Windows OLE DB Provider](#)

Accessing data with IBM i Access for Windows .Net Provider

The IBM i Access for Windows .Net Provider access.

The IBM i Access for Windows .Net Provider allows access to DB2 for IBM i through the Microsoft ADO.NET interface.

Accessing data with Net.Data

Net.Data is an application that runs on a server. You can use Net.Data to easily create dynamic Web documents that are called Web macros. Web macros that are created for Net.Data have the simplicity of HTML with the functionality of CGI-BIN applications.

Net.Data makes it easy to add live data to static Web pages. Live data includes information that is stored in databases, files, applications, and system services.

Related concepts

[Net.Data applications for the HTTP Server](#)

Accessing data through a Linux partition

IBM and a variety of Linux® distributors have cooperated to integrate the Linux operating system with the reliability of the IBM i architecture.

Linux brings a new generation of Web-based applications to the IBM i product. IBM has changed the Linux PowerPC® kernel to run in a secondary logical partition and contributed the kernel back to the Linux community.

Accessing data using Distributed Relational Database Architecture (DRDA)

A *distributed relational database* consists of a set of SQL objects that are spread across interconnected computer systems. Each relational database has a relational database manager to manage the tables in its environment.

The database managers communicate and cooperate with each other in a way that allows a given database manager access to run SQL statements on a relational database on another system.

Related reference

[Distributed relational database function and SQL](#)

Altering and managing database objects

Db2 for i provides both Structured Query Language (SQL) and system methods for altering and managing database objects.

Several methods are available for working with database objects. You can use the IBM Navigator for i interface, SQL statements, or IBM i commands.

Related concepts

[System i Navigator database tasks](#)

Related reference

[Terminology: SQL versus traditional file access](#)

Creating database objects

The first step in developing your database is to create the objects that hold your data. You can create tables, views, and indexes with SQL. You can also create physical and logical files using the traditional system interface.

You can create database objects using IBM Navigator for i, SQL, or the traditional system interface.

Related concepts

[System i Navigator database tasks](#)

Related reference

[Terminology: SQL versus traditional file access](#)

Ensuring data integrity

Db2 for i provides several integrity measures, such as constraints, trigger programs, and commitment control.

Constraints, triggers, and commitment control can protect your database against inadvertent insertions, deletions, and updates. Constraints basically govern how data values can change, while triggers are automatic actions that start, or *trigger*, an event, such as an update of a specific table.

Related concepts

[Commitment control](#)

[Working with triggers and constraints](#)

You can use triggers or constraints to manage data in your database tables.

Importing and exporting data between systems

Importing data is the process of retrieving data from external sources, while *exporting data* is the process of extracting data from Db2 for i and copying data to another system.

Importing data into Db2 for i can be a one-time event or it can be an ongoing task, like weekly updates for business reporting purposes. These types of data move operations are typically accomplished through import, export, or load functions.

Related concepts

[Copying a file](#)

[Copying files](#)

[Copying source file data](#)

[Moving a file](#)

Related tasks

[Importing and exporting data](#)

[Loading and unloading data from systems other than System i](#)

Working with multiple databases

The system provides a system database (identified as *SYSBAS*) and the ability to work with one or more user databases.

User databases are implemented through the use of independent disk pools, which are set up in the disk management function of System i® Navigator. After an independent disk pool is set up, it appears as another database in the Databases folder of System i Navigator.

When you expand a system in System i Navigator and then expand **Databases**, a list of databases that you can work with is shown. To establish a connection to a database, expand the database that you want to work with.

Related concepts

[Disk management](#)

Working with system-period temporal tables

Defining a system-period temporal table allows you to maintain historical versions of the table's rows. A system-period temporal table stores current versions of your data and uses its associated history table to store prior versions of rows that were updated or deleted.

A system-period temporal table is an SQL table that can be created by using the CREATE TABLE or ALTER TABLE SQL statements. The management of history for the table works for both SQL Data Manipulation Language (DML) statements and also for native DB I/O operations. The database manager uses the SYSTEM_TIME period to preserve historical versions of each row that is affected by an update or delete operation. The database manager stores the historical versions of the rows in a history table that is associated with the system-period temporal table. An alter table to add versioning establishes the link between the system-period temporal table and its history table. By referencing only a system-period

temporal table, your queries have access to your data at the current point in time and from past points in time.

System-period temporal tables

A system-period temporal table must have a row-begin column, row-end column, transaction start-ID column, and a SYSTEM_TIME period.

The row-begin, row-end, and transaction start-ID columns must be created using the GENERATED ALWAYS AS clause. When this clause is used, DB2 for i maintains the timestamp values in these columns.

Row-begin column

The row-begin column represents the time when the data in the row became current. It is defined as a TIMESTAMP(12).

The database manager generates a value for this column by using a reading of the system clock. If multiple rows are inserted or updated within a single SQL transaction, the value for the row-begin column is only generated once at the time of the first data change statement and is used as the row-begin value for all data change operations in that transaction.

When an existing table is altered to add a row-begin column, the row-begin column is populated with the default value 0001-01-01-00.00.00.000000000000 for all existing rows.

Row-end column

The row-end column represents the time when the data in the row was no longer current.

The rows in the system-period temporal table are by definition current, so the row-end column is populated with a default value, 9999-12-30-00.00.00.000000000000. For rows in a history table, the value in the row-end column represents when the row was added to the history table. For more information on how this value is set for historical rows, see [Updating data in a system-period temporal table](#) and [Deleting data in a system-period temporal table](#).

When an existing table is altered to add a row-end column, the row-end column is populated with the default value 9999-12-30-00.00.00.000000000000 for all existing rows.

Transaction start-ID

The transaction start-ID column captures the time of the first data change operation in the transaction.

If multiple rows are inserted or updated within a single SQL transaction, then the values for the transaction start-ID column are the same for all the rows and are unique from the values generated for this column by other transactions. When the transaction start-ID column is not null, you can use it to identify all the rows in the tables that were written by the same transaction. If the transaction start-ID is defined to allow the NULL value, it is set to a value only when the transaction start-ID does not match the row-begin column's value. The difference between the transaction start-ID and the row-begin column are explained further in the [System-period temporal table timestamp value conflicts](#) topic.

SYSTEM_TIME period

The SYSTEM_TIME period indicates the time span when the version of a row is current.

The SYSTEM_TIME period is defined by a row-begin column and a row-end column.

History tables

Each system-period temporal table is tied to a history table. When a row is updated or deleted from a system-period temporal table, the database manager inserts a copy of the old row into its associated history table with an updated row-end timestamp. This storage of prior system-period temporal table data gives you the ability to retrieve data from past points in time.

The columns in the history table and system-period temporal table must have the exact same names, order, and data types. You can create a history table with the same names and definitions as the columns of the system-period temporal table by using the LIKE clause of the CREATE TABLE statement. For example:

```
CREATE TABLE hist_policy_info LIKE policy_info;
```

A history table is subject to the following rules and restrictions when versioning is enabled:

- A history table cannot explicitly be dropped. It can only implicitly be dropped when the associated system-period temporal table is dropped.
- History table columns cannot explicitly be added, dropped, or changed.
- A history table cannot be defined as parent, child, or self-referencing in a referential constraint.

Once versioning is established, a change to a system-period temporal table causes an implicit corresponding change to the history table. For example, if a system-period temporal table is altered to add a column, the same column is added to the history table.

Since deleting data in a history table might jeopardize your ability to audit the system-period temporal table data history, you should restrict access to a history table to protect its data.

Related reference

[CREATE TABLE](#)

[ALTER TABLE](#)

Creating a system-period temporal table

Creating a system-period temporal table and a corresponding history table that are in a versioning relationship results in two tables that track when data changes occur and preserves historical versions of that data.

When creating a system-period temporal table, you must:

- Define your new or existing table so it can be used as a system-period temporal table by:
 - Defining row-begin, row-end, and transaction start-ID columns for the database manager to use for maintaining historical times for each row.
 - Defining a SYSTEM_TIME period to track when a row is current.
- Create a history table to receive old rows from the system-period temporal table.
- Add versioning to establish the link between the system-period temporal table and the history table.

The example in the following section shows the creation of a table that stores policy information for the customers of an insurance company

To create a system-period temporal table

1. Create a table with row-begin, row-end, and transaction start-ID columns and a SYSTEM_TIME period.

In the following example, the POLICY_INFO table stores information about a customer's insurance coverage. The SYSTEM_TIME period columns, SYS_START and SYS_END, show when a row is current. The TS_ID column shows the time of the first data change operation in the transaction that impacted the row.

```
CREATE TABLE policy_info
( policy_id  CHAR(4) 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,
  ts_id      TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID,
  PERIOD SYSTEM_TIME (sys_start, sys_end) );
```

2. Create a matching history table. You can create a history table with the same column names and descriptions as the columns of the system-period temporal table by using the LIKE clause of the CREATE TABLE statement. For example:

```
CREATE TABLE hist_policy_info LIKE policy_info;
```

3. Add versioning to the system-period temporal table to establish a link to the history table. For example:

```
ALTER TABLE policy_info ADD VERSIONING USE HISTORY TABLE hist_policy_info;
```

Once all three steps are complete, the HIST_POLICY_INFO history table receives the old rows from the POLICY_INFO table.

The ON DELETE ADD EXTRA ROW clause can optionally be specified on the ALTER TABLE ADD VERSIONING statement. If the clause is specified, an extra row is inserted into the history table when a row is deleted from the system-period temporal table. When the additional row is written to the history table, the values for the generated expression columns, including the row-begin and row-end column, are generated. For more information on how ON DELETE ADD EXTRA ROW can be used for tracking audit information, see [Using a system-period temporal table for tracking auditing information](#).

Related reference

[CREATE TABLE](#)

[ALTER TABLE](#)

Additional system-period temporal table CREATE TABLE examples

Additional examples include hiding columns and changing an existing table into a system-period temporal table.

Hiding columns

The row-begin, row-end, and transaction start-ID columns can be defined as IMPLICITLY HIDDEN. Columns defined as IMPLICITLY HIDDEN do not appear in the select list unless explicitly referenced. Since values for the row-begin, row-end, and transaction start-ID columns are generated by the database manager, hiding them can minimize any potential impact on your existing applications.

For example:

- A SELECT * query run against a table does not return any implicitly hidden columns in the result table.
- An INSERT statement without a column list does not expect the default value to be specified for any implicitly hidden columns.

The following example creates the POLICY_INFO table with the TIMESTAMP(12) columns SYS_START, SYS_END, and TS_ID defined as implicitly hidden.

```
CREATE TABLE policy_info
( policy_id CHAR(4) NOT NULL,
  coverage INT NOT NULL,
  sys_start TIMESTAMP(12) NOT NULL
    GENERATED ALWAYS AS ROW BEGIN
    IMPLICITLY HIDDEN,
  sys_end    TIMESTAMP(12) NOT NULL
    GENERATED ALWAYS AS ROW END
    IMPLICITLY HIDDEN,
  ts_id      TIMESTAMP(12) NOT NULL
    GENERATED ALWAYS AS TRANSACTION START ID
    IMPLICITLY HIDDEN,
  PERIOD SYSTEM_TIME (sys_start, sys_end) );
```

Creating the HIST_POLICY_INFO history table using the LIKE clause of the CREATE TABLE statement results in the history table inheriting the implicitly hidden attribute from the POLICY_INFO table. If you do not use the LIKE clause when creating the history table, then any columns defined as hidden in the system-period temporal table must also be defined as hidden in the associated history table. If the column definitions do not match, you will not be able to add versioning.

Changing an existing table into a system-period temporal table

In the following example, the three timestamp columns and a SYSTEM_TIME period are added to the existing table, EMPLOYEES, preparing it to be used as a system-period temporal table.

```
ALTER TABLE employees
ADD COLUMN sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN
ADD COLUMN sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END
ADD COLUMN ts_id TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID
ADD PERIOD SYSTEM_TIME(sys_start, sys_end);
```

A history table must be created and versioning added to finish enabling the tracking of historical rows.

Inserting data into a system-period temporal table

Inserting data into a system-period temporal table is similar to inserting data into a regular table.

The database manager automatically generates the values for the row-begin and row-end timestamp columns when data is inserted into a system-period temporal table. The database manager also generates the transaction start-ID value that uniquely identifies the transaction that is inserting the row.

In this example, the following data was inserted on January 31, 2015 (2015-01-31) into the table created in the example in [Creating a system-period temporal table](#).

Insert three rows into the POLICY_INFO table.

```
INSERT INTO policy_info (policy_id, coverage)
VALUES('A123',12000);

INSERT INTO policy_info (policy_id, coverage)
VALUES('B345',18000);

INSERT INTO policy_info (policy_id, coverage)
VALUES('C567',20000);
```

The POLICY_INFO table now contains the insurance coverage data in the following table. The SYS_START, SYS_END, and TS_ID column entries were generated by the database manager.

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
B345	18000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	20000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000

The SYS_START timestamp values are the same for all three newly inserted rows because in this example the rows were inserted as part of the same transaction.

The HIST_POLICY_INFO history table remains empty because no history rows are generated by an insert.

The row-begin column, SYS_START, represents the time when the row became current. The database manager generates this value by using a reading of the system clock at the moment it executes the first data change statement in the transaction that generates the row. The database manager also generates the transaction start-ID column, TS_ID, which captures the time when execution started for a transaction that impacts the row. In most cases, the timestamp values for both these columns are the same because they result from the execution of the same transaction. If the transaction start-ID column is defined to allow the NULL value, it is NULL when the value of the transaction start-ID column is identical to the row-begin column.

When multiple transactions are updating the same row, timestamp conflicts can occur. The database manager can resolve these conflicts by adjusting the row-begin column timestamp values. In such cases, the values in the row-begin column and the transaction start-ID column differs. The [System-period temporal table timestamp value conflicts](#) topic provides more details on timestamp adjustments.

Related reference

[INSERT](#)

Updating data in a system-period temporal table

Updating data in a system-period temporal table results in rows added to the system-period temporal table's associated history table.

In addition to updating the values in rows of the system-period temporal table, the UPDATE statement causes the database manager to insert a copy of the existing row into the associated history table. The history row is generated as part of the same transaction that updates the row. If a single transaction updates the same row multiple times, only one history row is generated and that row reflects the state of the record before any changes were made by the transaction.

In the following example, a customer's insurance coverage is increased on February 28, 2016 (2016-02-28). This example uses the table created in [Creating a system-period temporal table](#) with rows that were added in [Inserting data into a system-period temporal table](#).

The following table contains the POLICY_INFO table data before the update.

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
B345	18000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	20000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000

Update the coverage for policy C567 to 25000.

```
UPDATE policy_info  
SET coverage = 25000  
WHERE policy_id = 'C567';
```

The update to policy C567 affects the system-period temporal table and its history table, causing the following things to occur:

1. The coverage value for the row with policy C567 is updated to 25000.
2. In the system-period temporal table, the database manager updates the SYS_START and TS_ID values to the timestamp of the update and the SYS_END value is unchanged.
3. The original row is inserted into the history table. The database manager updates the SYS_END value to the timestamp of the update. This row can be interpreted as the valid coverage for policy C567 from 2015-01-31-22.31.33.495925000000 to 2016-02-28-09.10.12.649592000000.

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
B345	18000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	25000	2016-02-28-09.10.12.6 49592000000	9999-12-30-00.00.00.0 00000000000	2016-02-28-09.10.12.6 49592000000

Table 4. Data in the history table, HIST_POLICY_INFO, after the UPDATE statement

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
C567	20000	2015-01-31-22.31.33.4 95925000000	2016-02-28-09.10.12.6 49592000000	2015-01-31-22.31.33.4 95925000000

If one or more updates occur while running under commitment control, and the user rolls back the transaction, then both the updates to the system-period temporal table and the inserts into the history table are rolled back.

When multiple transactions are updating the same row, timestamp conflicts can occur. When these conflicts occur, the setting for the SYSTIME_PERIOD_ADJ QAQQINI option determines whether timestamp adjustments are made or if transaction fails. The System-period temporal table timestamp value conflicts topic provides more details on timestamp adjustments. Application programmers might consider using SQLCODE or SQLSTATE values to handle potential timestamp value adjustment-related return codes from SQL update statements.

Related reference

[UPDATE](#)

Deleting data in a system-period temporal table

Deleting a row from a system-period temporal table removes the row from the table and adds one or two rows to the associated history table. The DELETE statement copies the existing row into the associated history table before the row is deleted from the system-period temporal table. The rows in the history table are added with the appropriate system timestamps.

In the following example, the owner of policy B345 decides to cancel insurance coverage. The data for the customer was deleted on September 1, 2016 (2016-09-01) from the example table created in the [Creating a system-period temporal table](#) topic and updated in the [Updating data in a system-period temporal table](#) topic.

The following table contains the POLICY_INFO table data before the delete.

Table 5. Data in the system-period temporal table, POLICY_INFO, before the DELETE statement

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
B345	18000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	25000	2016-02-28-09.10.12.6 49592000000	9999-12-30-00.00.00.0 00000000000	2016-02-28-09.10.12.6 49592000000

The following statement deletes the policy information for the policy ID B345.

```
DELETE FROM policy_info WHERE policy_id = 'B345';
```

The deletion of policy B345 affects the system-period temporal table and its history table, causing the following things to occur:

1. The original row is copied to the history table. The database manager updates the SYS_END column value to the timestamp of the first data change operation in the transaction.
2. The row where the POLICY_ID column value is B345 is deleted from the system-period temporal table.

Table 6. Data in the system-period temporal table, POLICY_INFO, after the DELETE statement

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	25000	2016-02-28-09.10.12.6 49592000000	9999-12-30-00.00.00.0 00000000000	2016-02-28-09.10.12.6 49592000000

Table 7. Data in the history table, HIST_POLICY_INFO, after the DELETE statement

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
C567	20000	2015-01-31-22.31.33.4 95925000000	2016-02-28-09.10.12.6 49592000000	2015-01-31-22.31.33.4 95925000000
B345	18000	2015-01-31-22.31.33.4 95925000000	2016-09-01-12.18.22.9 59254000000	2015-01-31-22.31.33.4 95925000000

Related reference

[DELETE](#)

Using a system-period temporal table for tracking auditing information

An audit trail of the changes that are made to the system-period temporal table can be made more informative with the addition of one or more generated expression columns.

Some examples of auditing information that can be tracked are

- when was data modified,
- who modified the data
- what SQL operation modified the data.

To track when data was modified, define the table as a system-period temporal table. To track who and what SQL statement modified the data, use a generated expression column. For more information about available generated expression columns, see [CREATE TABLE](#).

In the following example, the POLICY_INFO table contains two extra columns. AUDIT_USER tracks who modified the data by using the SESSION_USER special register. AUDIT_OP tracks the SQL operation that modified the data by using the DATA CHANGE OPERATION.

```
CREATE TABLE policy_info
(policy_id CHAR(4) NOT NULL,
coverage INT NOT NULL,
audit_user VARCHAR(128) GENERATED ALWAYS AS (SESSION_USER),
audit_op CHAR(1) GENERATED ALWAYS AS (DATA CHANGE OPERATION),
sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
ts_id TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID,
PERIOD SYSTEM_TIME (sys_start, sys_end) );

CREATE TABLE hist_policy_info LIKE policy_info;

ALTER TABLE policy_info ADD VERSIONING USE HISTORY TABLE hist_policy_info
ON DELETE ADD EXTRA ROW;
```

Each row in the system temporal table and its corresponding history table tracks whether the row was changed by an insert, update, or delete operation and the name of the user that last modified the row.

ON DELETE ADD EXTRA ROW

If the ON DELETE ADD EXTRA ROW clause is specified on the ALTER TABLE ADD VERSIONING statement, an extra row is inserted into the history table when a row is deleted from a system-period temporal table. This additional row contains information about the delete operation itself.

The following example illustrates the value of ON DELETE ADD EXTRA ROW. This example shows a series of three transactions. In June of 2014, USER1 inserts a row into the POLICY_INFO table. Then, in January of 2015, USER2 updates the row. Finally, in August of 2015, USER3 deletes the row:

```
INSERT INTO policy_info (policy_id, coverage)
VALUES ('A123',12000);

UPDATE policy_info
SET coverage = 25000
WHERE policy_id = 'A123'

DELETE FROM policy_info
WHERE policy_id = 'A123'
```

Without the ON DELETE ADD EXTRA ROW clause, this series of transactions would write only the following two entries to the history table. The first entry records the row as it was before USER2 made her update, and the second entry records the row as it was before USER3 did his delete. The delete operation records the row as it last existed in the temporal table. Without the ON DELETE ADD EXTRA ROW clause, there is no audit record of the delete operation itself.

Table 8. Data in the history table, HIST_POLICY_INFO, after the DELETE statement when ON DELETE ADD EXTRA ROW is not specified

POLICY_ID	COVERAG E	AUDIT_USE R	AUDIT_O P	SYS_START	SYS_END	TS_ID
A123	12000	USER1	I	2014-06-31-22. 31.33.495925	2015-01-31-22. 31.33.857632	2014-06-31-22. 31.33.495925
A123	25000	USER2	U	2015-01-31-22. 31.33.857632	2015-08-31-22. 31.33.634521	2015-01-31-22. 31.33.857632

With the ON DELETE ADD EXTRA ROW clause added to the ADD VERSIONING statement, an extra row is written history table. Because the row-begin and row-end values are generated for the extra row, they are the same timestamp value. This timestamp also matches the row-end value of the first row inserted for the delete. In the following table, this additional row is shown:

Table 9. Data in the history table, HIST_POLICY_INFO, after the DELETE statement when ON DELETE ADD EXTRA ROW is specified

POLICY_ID	COVERAG E	AUDIT_USE R	AUDIT_O P	SYS_START	SYS_END	TS_ID
A123	12000	USER1	I	2014-06-31-22. 31.33.495925	2015-01-31-22. 31.33.857632	2014-06-31-22. 31.33.495925
A123	25000	USER2	U	2015-01-31-22. 31.33.857632	2015-08-31-22. 31.33.634521	2015-01-31-22. 31.33.857632
A123	25000	USER3	D	2015-08-31-22. 31.33.634521	2015-08-31-22. 31.33.634521	2015-08-31-22. 31.33.634521

Related reference

[Creating auditing columns](#)

System-period temporal table timestamp value conflicts

The database manager will prevent a historical row of a system-period temporal table from being generated with a row-end timestamp that is less than its row-begin timestamp. This type of conflict can happen when multiple transactions have updated the same row of a system-period temporal table.

An example scenario is shown in the table below. The timestamps for this example have been simplified as a place holder instead of a sample system clock reading.

Table 10. Concurrent transactions on a system-period temporal table

Current Timestamp	Transaction A	Trans A SYS_STA RT value	Transaction B	Trans B SYS_STA RT value
T1	INSERT INTO policy_info (policy_id, coverage) VALUES ('S777',7000)	T1		
T2			INSERT INTO policy_info (policy_id, coverage) VALUES ('T888',8000)	T2
T3			COMMIT	T2
T4	UPDATE policy_info SET policy_id = 'X999' WHERE policy_id = 'T888'	T1		
T5	COMMIT	T1		

When multiple rows are inserted or updated in the same transaction, the row-begin column is the same for all impacted rows. The value for the row-begin comes from a reading of the system clock at the moment of the first data change in the transaction. In the table above, all rows inserted or updated by transaction A have a row-begin timestamp of T1, and the row inserted by transaction B has a row-begin timestamp of T2. At time T3, the insert of row two is committed with a row-begin timestamp of T2.

When transaction A updates the row where the POLICY_ID is equal to 'T888' at time T4, the updated row's row-begin timestamp is changed to T1 because the update occurred in transaction A. The original row is inserted into the history table, and the value of the row-end timestamp for the historical row is updated to match the updated row's row-begin timestamp value, T1. The row-begin value of the historical row is T2. This would result in a historical row with a row-end timestamp value, T1, that is earlier than its row-begin timestamp value, T2. This is not allowed.

The SYSTIME_PERIOD_ADJ QAQQINI option determines what action the system takes when a historical row has a row-end timestamp that is less than the row-begin timestamp. The QAQQINI value can be set to either *ERROR or *ADJUST, with *ERROR being the default value. When *ERROR is used, an error with SQLCODE -20528 and SQLSTATE 57062 will be returned when the row-end timestamp value is less than the row-begin timestamp value. When *ADJUST is used, an adjustment is made to the value of the row-begin timestamp and the row-end timestamp instead of returning an error. In such cases, the values in the row begin column and the transaction start-ID column will differ. For more information on the SYSTIME_PERIOD_ADJ QAQQINI option, see [Performance and query optimization](#).

Another example of when this situation may occur is when the system clock is adjusted. This could happen when changing the time to account for daylight savings time. As in the previous case, the value of the QAQQINI option would determine whether an error is returned or if the timestamp values are adjusted.

Querying system-period temporal data

Querying a system-period temporal table can return results for a specified point or period in time. The results can include current values and previous historic values.

To query a temporal table, you can specify the time criteria several ways:

- in a query explicitly using a SYSTEM_TIME period specification,
- in a query implicitly using the CURRENT TEMPORAL SYSTEM_TIME system register, or
- in a view definition.

Specifying the time criteria in a query

When querying a system-period temporal table, you can include FOR SYSTEM_TIME in the FROM clause to query the current and past state of your data.

Time periods can be specified in three ways:

- **AS OF** *value*

Includes each row for which the row-begin value is less than or equal to *value* and the row-end value is greater than *value*. Zero rows are returned if *value* is the null value.

- **FROM** *value1 TO value2*

Includes rows that exist completely within the period that is specified from *value1* up to *value2*. A row is included if the row-begin value is less than *value2* and the row-end value is greater than *value1*. Zero rows are returned if *value1* is greater than or equal to *value2* or if *value1* or *value2* is the null value.

- **BETWEEN** *value1 AND value2*

Includes rows that overlap at any point in time between *value1* and *value2*. A row is included if its row-begin value is less than or equal to *value2* and the row-end value is greater than *value1*. Zero rows are returned if *value1* is greater than *value2* or if *value1* or *value2* is the null value. If *value1* = *value2*, the expression is equivalent to AS OF *value1*.

For detailed information about time periods, see [table-reference](#) and [Queries](#).

Related reference

[SELECT](#)

Example: Time criteria in the query

These sample queries request policy information from a system-period temporal table. Each query uses a variation of the FOR SYSTEM_TIME specification.

The POLICY_INFO temporal table and its associated history table that is used in the examples contains these rows:

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	25000	2016-02-28-09.10.12.6 49592000000	9999-12-30-00.00.00.0 00000000000	2016-02-28-09.10.12.6 49592000000

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
C567	20000	2015-01-31-22.31.33.4 95925000000	2016-02-28-09.10.12.6 49592000000	2015-01-31-22.31.33.4 95925000000

Table 12. History table, HIST_POLICY_INFO (continued)

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
B345	18000	2015-01-31-22.31.33.4 95925000000	2016-09-01-12.18.22.9 59254000000	2015-01-31-22.31.33.4 95925000000

• Query with no time period specified

```
SELECT policy_id, coverage, sys_start, sys_end
FROM policy_info
WHERE policy_id = 'C567'
```

This query returns one row. The SELECT queries only the POLICY_INFO table. The history table is not queried because FOR SYSTEM_TIME was not specified.

Table 13. Result of query with no time period specified

POLICY_ID	COVERAGE	SYS_START	SYS_END
C567	25000	2016-02-28-09.10.12. 649592000000	9999-12-30-00.00.00. 000000000000

• Query with FOR SYSTEM_TIME AS OF clause specified

```
SELECT policy_id, coverage, sys_start, sys_end
FROM policy_info
FOR SYSTEM_TIME AS OF '2016-02-28-09.10.12.649592000000'
```

This query returns three rows. The SELECT queries both the POLICY_INFO and the HIST_POLICY_INFO tables. The row-begin column of the period is inclusive, while the row-end column is exclusive. The history table row with a SYS_END column value of 2016-02-28-09.10.12.649592000000 equals the AS OF value, but must be less than that value to be returned

Table 14. Result of query with a FOR SYSTEM_TIME AS OF period specified

POLICY_ID	COVERAGE	SYS_START	SYS_END
A123	12000	2015-01-31-22.31.334 959250000	9999-12-30-00.00.00. 000000000000
C567	25000	2016-02-28-09.10.12. 649592000000	9999-12-30-00.00.00. 000000000000
B345	18000	2015-01-31-22.31.33. 495925000000	2016-09-01-12.18.22. 959254000000

• Query with FOR SYSTEM_TIME FROM...TO specified.

```
SELECT policy_id, coverage, sys_start, sys_end
FROM policy_info
FOR SYSTEM_TIME FROM '0001-01-01-00.00.00.000000000000'
TO '9999-12-30-00.00.00.000000000000'
WHERE policy_id = 'C567'
```

This query returns two rows. The SELECT queries both the POLICY_INFO and the HIST_POLICY_INFO tables.

Table 15. Result of query with FOR SYSTEM_TIME FROM...TO specified.

POLICY_ID	COVERAGE	SYS_START	SYS_END
C567	25000	2016-02-28-09.10.12. 649592000000	9999-12-30-00.00.00. 000000000000

Table 15. Result of query with FOR SYSTEM_TIME FROM...TO specified. (continued)			
POLICY_ID	COVERAGE	SYS_START	SYS_END
C567	20000	2015-01-31-22.31.33. 495925000000	2016-02-28-09.10.12. 649592000000

• **Query with FOR SYSTEM_TIME BETWEEN...AND specified.**

```
SELECT policy_id, coverage
FROM policy_info
FOR SYSTEM_TIME BETWEEN '2016-02-28-09.10.12.649592000000'
AND '9999-12-30-00.00.00.000000000000'
```

This query returns three rows. The SELECT queries both the POLICY_INFO and the HIST_POLICY_INFO tables. The rows with a SYS_START column value of 2016-02-28-09.10.12.649592000000 are equal to *value1* and are returned because the begin time of a period is included. The rows with a SYS_END column value of 2016-02-28-09.10.12.649592000000 are equal to *value1* and are not returned because the end time of a period is not included.

Table 16. Result of query with FOR SYSTEM_TIME BETWEEN...AND specified.	
POLICY_ID	COVERAGE
A123	12000
C567	25000
B345	18000

Specifying the time criteria by using the CURRENT TEMPORAL SYSTEM_TIME special register

Using the CURRENT TEMPORAL SYSTEM_TIME special register to specify the AS OF time for a query can reduce or eliminate the changes that are required to run an application against different points in time.

When you have an application that you want to run against a system-period temporal table to query the state of your data for a number of different dates, you can set the date in a special register. If you need to query your data as of today, as of the end of the last quarter, or as of the same date from last year, it might not be possible to change the application to add AS OF specifications to each SQL statement. This restriction is likely the case when you are using packaged applications. To address such scenarios, you can use the CURRENT TEMPORAL SYSTEM_TIME special register to set the timestamp at the job level.

Setting the CURRENT TEMPORAL SYSTEM_TIME special register does not affect non-temporal tables. Only queries on temporal tables with versioning enabled use the time set in the special register. There is also no effect on DDL statements.

When the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, applications that issue a query against any system-period temporal table returns data as of that date or timestamp.

When the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, data modification statements like INSERT, UPDATE, and DELETE against system-period temporal tables are blocked, because if the special register is set to some time in the past, for example five years ago, then allowing data modification operations might result in changes to your historical data records. Modifying the history table is not allowed.

The CRTSQLxxx precompiler commands have a system time sensitive option that indicates whether references to system-period temporal tables in static and dynamic SQL statements are affected by the value of the CURRENT TEMPORAL SYSTEM_TIME special register. This behavior can also be specified for SQL programs, procedures, or functions by the SYSTIME option on the SET OPTION statement. The default is *SYSTIME. All SQL programs and service programs that were created before 7.3 are considered *NOSYSTIME. When the value is *SYSTIME, the special register is applied to statements that reference system-period temporal tables. When the value is *NOSYSTIME, the special register is ignored.

Related reference

CURRENT TEMPORAL SYSTEM_TIME

Example: Time criteria specified by using the CURRENT TEMPORAL SYSTEM_TIME special register

These sample queries request policy information from a system-period temporal table for a specific point in time by using the CURRENT TEMPORAL SYSTEM_TIME special register.

The POLICY_INFO temporal table and its associated history table that is used in the examples are:

Table 17. System-period temporal table, POLICY_INFO

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	25000	2016-02-28-09.10.12.6 49592000000	9999-12-30-00.00.00.0 00000000000	2016-02-28-09.10.12.6 49592000000

Table 18. History table, HIST_POLICY_INFO

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
C567	20000	2015-01-31-22.31.33.4 95925000000	2016-02-28-09.10.12.6 49592000000	2015-01-31-22.31.33.4 95925000000
B345	18000	2015-01-31-22.31.33.4 95925000000	2016-09-01-12.18.22.9 59254000000	2015-01-31-22.31.33.4 95925000000

- **Example 1:** Set the CURRENT TEMPORAL SYSTEM_TIME special register to one year before the current timestamp. Assume a current timestamp of 2016-05-17-14.45.31.434235000000.

```
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT_TIMESTAMP - 1 YEAR;
```

The following query of the system-period temporal table returns results as of one year ago.

```
SELECT policy_id, coverage FROM policy_info;
```

Since POLICY_INFO is a system-period temporal table and the CURRENT TEMPORAL SYSTEM_TIME special register is not null, the query is run as:

```
SELECT policy_id, coverage FROM policy_info  
FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME;
```

The SELECT queries both the POLICY_INFO and the HIST_POLICY_INFO tables and returns:

Table 19. Result of query with the special register set.

POLICY_ID	COVERAGE
A123	12000
C567	20000
B345	18000

- **Example 2:** Set the CURRENT TEMPORAL SYSTEM_TIME special register to a timestamp and reference a system-period temporal table in view definitions.

```
CREATE VIEW coverage AS SELECT policy_id, coverage FROM policy_info;
```

```
SET CURRENT TEMPORAL SYSTEM_TIME = TIMESTAMP('2016-02-28-09.10.12.649592000000');
```

```
SELECT * FROM coverage;
```

Since the view references POLICY_INFO, which is a system-period temporal table, and the CURRENT TEMPORAL SYSTEM_TIME is set, the table reference in the view has FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME implicitly added to it. The SELECT queries both the POLICY_INFO and the HIST_POLICY_INFO tables and returns:

Table 20. Result of query with the special register set.

POLICY_ID	COVERAGE
A123	12000
C567	25000
B345	18000

- **Example 3:** Set the CURRENT TEMPORAL SYSTEM_TIME special register and reference a system-period temporal table in a subselect.

```
CREATE VIEW customer_policy AS SELECT * FROM customer_info
  WHERE cust_policy_id IN (SELECT policy_id FROM policy_info);

SET CURRENT TEMPORAL SYSTEM_TIME = TIMESTAMP('2016-02-28-09.10.12.649592000000');

SELECT * FROM customer_policy;
```

The query in this example involves a view over a non system-period temporal table that references a system-period temporal table.

The CURRENT TEMPORAL SYSTEM TIME special register is applied to all system-period temporal tables in a view. It is ignored for any other table reference. For this query of a view, FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM TIME is implicitly added to the POLICY_INFO table reference.

- **Example 4:** Set the special register and run a query that contains a time period specification. This is an invalid query.

```
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 1 YEAR;

SELECT * FROM policy_info FOR SYSTEM_TIME AS OF TIMESTAMP('2016-02-28-09.10.12.649592000000');
```

An error is returned because there are multiple time period specifications. The special register was set to a non-null value and the query also specified a time. When the query uses a FOR SYSTEM_TIME clause, the CURRENT TEMPORAL SYSTEM_TIME special register must be the NULL value.

Specify the time criteria for a view

A FOR SYSTEM_TIME period specification that follows the name of a view in a table reference applies to all of the system-period temporal table references in the definition of that view. If the view does not access any temporal tables, the period specification has no effect on the result table of the view.

A view reference followed by a period specification must not include an external function that is not NO SQL or an SQL function that is not inline capable.

To query a view that references a temporal table, use either of the following methods:

- Specify a SYSTEM_TIME period specification following the name of a view in the FROM clause of a query.
- Use the CURRENT TEMPORAL SYSTEM_TIME special register. In this case, you must not include a period specification in the query.

Example: Time criteria in the view

These sample queries request policy information from a view built over a system-period temporal table.

The POLICY_INFO temporal table and its associated history table that is used in the examples are:

Table 21. System-period temporal table, POLICY_INFO

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
A123	12000	2015-01-31-22.31.33.4 95925000000	9999-12-30-00.00.00.0 00000000000	2015-01-31-22.31.33.4 95925000000
C567	25000	2016-02-28-09.10.12.6 49592000000	9999-12-30-00.00.00.0 00000000000	2016-02-28-09.10.12.6 49592000000

Table 22. History table, HIST_POLICY_INFO

POLICY_ID	COVERAGE	SYS_START	SYS_END	TS_ID
C567	20000	2015-01-31-22.31.33.4 95925000000	2016-02-28-09.10.12.6 49592000000	2015-01-31-22.31.33.4 95925000000
B345	18000	2015-01-31-22.31.33.4 95925000000	2016-09-01-12.18.22.9 59254000000	2015-01-31-22.31.33.4 95925000000

Example 1: Create a view COVERAGE that references the system-period temporal table POLICY_INFO. Then, query the view and specify a FOR SYSTEM_TIME period specification to return rows that were current as of one year ago. Assume a current timestamp of 2016-05-17-14.45.31.434235000000.

```
CREATE VIEW coverage AS
  SELECT policy_id, coverage
  FROM policy_info;

SELECT * FROM coverage
  FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP - 1 YEAR;
```

The SELECT queries both the POLICY_INFO and the HIST_POLICY_INFO tables and returns:

Table 23. Result of query of a view that has a period specification.

POLICY_ID	COVERAGE
A123	12000
C567	20000
B345	18000

Example 2: Create a view COVERAGE that reference the system-period temporal table POLICY_INFO and return rows that were current as of one year ago.

```
CREATE VIEW coverage AS
  SELECT policy_id, coverage
  FROM policy_info
  FOR SYSTEM_TIME AS CURRENT_TIMESTAMP - 1 YEAR;

SELECT * FROM coverage;
```

The query of this view always returns rows that were current AS OF one year ago. Assuming a current timestamp of 2016-05-17-14.45.31.434235000000, the query returns the same results as the results that were returned in example one.

Restrictions when inserting, updating, or deleting data in a system-period temporal table

An insert, update, or delete cannot be done when the insert, update, or delete would implicitly attempt to change history rows.

Restrictions for cursors and UPDATE/DELETE WHERE CURRENT OF

A FOR UPDATE cursor cannot be defined that includes a period specification on the table identified in the FROM clause. The following example returns an error:

```
DECLARE updateCursor CURSOR FOR
  SELECT coverage
  FROM policy_info FOR SYSTEM_TIME AS OF '2015-01-31-22.31.33.495925000000'
  FOR UPDATE;
```

If the FOR UPDATE clause is removed from the DECLARE CURSOR statement and a period specification is defined, the cursor becomes read-only. If an UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF is used to attempt to update or delete rows in the system-period temporal table and its associated history table, an error is returned. The following example returns an error:

```
CREATE OR REPLACE PROCEDURE updateProc
LANGUAGE SQL
BEGIN
  DECLARE val1 INTEGER;
  DECLARE updateCursor CURSOR FOR
    SELECT coverage
    FROM policy_info FOR SYSTEM_TIME AS OF '2015-01-31-22.31.33.495925000000';
  OPEN updateCursor;
  FETCH updateCursor INTO val1;
  UPDATE policy_info SET coverage = coverage + 1000
  WHERE CURRENT OF updateCursor;
  CLOSE updateCursor;
END;
```

The update returns an error because it implicitly attempts to update history rows. The SELECT explicitly queries the POLICY_INFO table and implicitly queries its associated history table, HIST_POLICY_INFO. Rows in a history table that are accessed implicitly cannot be updated.

Restrictions for inserting, updating, and deleting data in a view

Inserts, updates, and deletes are not allowed against a view that has a period specification defined. For example, the following UPDATE would return an error:

```
CREATE VIEW coverage AS
  SELECT policy_id, coverage
  FROM policy_info FOR SYSTEM_TIME AS OF TIMESTAMP('2016-02-28-09.10.12.649592000000');

UPDATE coverage SET coverage = coverage + 1000 WHERE policy_id = 'C567';
```

The update returns an error because it implicitly attempts to update history rows. However, you might create an instead of update trigger to update the system-period temporal table. For example:

```
CREATE TRIGGER coverage_update_iot INSTEAD OF UPDATE ON coverage
REFERENCING NEW AS N OLD AS O
FOR EACH ROW MODE DB2SQL
BEGIN
  UPDATE policy_info
  SET policy_id = n.policy_id, coverage = n.coverage
  WHERE policy_id = o.policy_id AND coverage = o.coverage;
END;

UPDATE coverage SET coverage = coverage + 1000 WHERE policy_id = 'C567';
```

An error is not returned because the trigger directs the update to the system-period temporal table instead of the view.

Native I/O considerations with a system-period temporal table

The native I/O processing of read, write, and update operations is, in general, similar to the corresponding SQL operations. However, there are some considerations to keep in mind.

Both SQL and native update and delete operations against a system-period temporal table causes historical rows to be added to a history table. Additionally, if the ON DELETE ADD EXTRA ROW clause is specified on the ALTER TABLE command, both SQL and native delete operations add an extra row when a record is deleted from the table. Similarly, the QAQQINI value for SYSTIME_PERIOD_ADJ has the same effect on concurrent native updates as it has on concurrent SQL update operations. For both native and SQL, the QAQQINI value causes the database to either raise an error or adjust to the row-begin and row-end timestamps. A native DB application receives a CPF503B exception, reason code 3, when the SYSTIME_PERIOD_ADJ value is set to *DEFAULT or *ERROR and a native UPDATE would have caused the value of the historical row's end time to be set to a value that is less than the historical row's begin time.

That said, several differences between native and SQL processing that are worth noting.

While native read requests work against either the temporal or the history table, they do not merge the results together. Only records from the file read are returned. For example, when the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, an SQL read of the temporal table can return rows that are drawn from both the temporal table and its corresponding history table. However, for a native read of the temporal table, the database ignores the special register and return values only from the temporal table.

When the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, SQL Data Manipulation (DML) statements fail with SQLCODE/SQLSTATE set to -20535/51046. An SQL user is unable to modify the temporal table's data. However, no similar restriction exists for a native user. Native write, update, and delete operations modify the temporal table whether the special register is set or not.

Finally, a native write or update can specify values in the I/O buffer for both the SYSTEM_TIME period generated columns and other generated columns, if they exist. However, regardless of what values are supplied in the I/O buffer, DB2 for i substitutes the correct values for any generated columns.

Maintaining the history table

Because history tables experience more inserts than deletes, your history tables can become large. Deciding how to prune your history tables to get rid of the rows that you no longer need can be a complex task.

You need to understand the value of your individual rows. Some content, like customer contracts, might be untouchable and can never be deleted. Other records, like website visitor information, can be pruned without concern. Often it is not the age of a row that determines when it can be pruned and archived, but rather it is some business logic that is the deciding factor. The following list contains some possible rules for pruning:

- Prune rows that are selected by a user-supplied query that reflects business rules.
- Prune rows older than a certain age.
- Prune history rows when more than N versions exist for that record (retain only the latest N versions).
- Prune history rows when the record is deleted from the associated system-period temporal table (when no current versions exist).

Several ways exist to periodically prune old data from a history table. One way to prune data is range partitioning on the history table. Old partitions can then be detached from the history table and archived or deleted. For more information, see [Partitioning a system-period temporal table](#).

Partitioning a system-period temporal table

A system-period temporal table can have its table data divided into multiple partitions. A history table that is associated with a system-period temporal table can also be partitioned.

When versioning is enabled, the following behaviors apply when attaching or detaching a partition to a system-period temporal table:

Attaching partitions

- A single partition, partitioned table can be attached to a system-period temporal table as a partition while versioning is enabled.
- The table attached must have the same column definitions as the system-period temporal table, including the three timestamp columns defined as ROW BEGIN, ROW END, and TRANSACTION START ID.
- The table attached does not require a SYSTEM_TIME period definition.

Detaching partitions

- A partition cannot be detached from a system-period temporal table while versioning is enabled. You can drop versioning and then detach a partition from the base table. The detached partition becomes an independent single partition, partitioned table. Detaching a partition from a history table does not require that you drop versioning.
- A detached partition retains all three timestamp columns (ROW BEGIN, ROW END, and TRANSACTION START ID), but not the SYSTEM_TIME period definition.

Related reference

[Partitioned tables](#)

Row and column access control with a system-period temporal table

Row and column access control can be defined on both a system-period temporal table and its associated history table.

Row and column access control (RCAC) is a layer of data security that controls access to a table at the row level, column level, or both. RCAC can be applied to system-period temporal tables and history tables. When RCAC is activated for a system-period temporal table, the database manager automatically activates row access control on the history table and creates a default row permission for the history table. This default row permission prevents any direct access to the history table. When the history table is protected by the default row permission, updates and deletes still generate history rows in the history table.

When a temporal query is run against a system-period temporal table, the row permissions and column masks from the system-period temporal table are also applied to the rows returned from the history table. For example, if a row permission is defined for a system-period temporal table and a query with an `FOR SYSTEM_TIME AS OF` clause is run, both current and historical rows are returned when the current or historical row satisfies both the RCAC rule from the temporal table and was current as of the time specified.

If the history table has only a default permission, you cannot query it directly. However, if a row or column rule other than the default permission is also defined on the history table, that rule is applied when the history table is accessed directly. Therefore, if you need to query the history table directly, you can create a row permission or column mask on the history table that matches the row permission or mask that was created on the system-period temporal table. When the row permission or column mask is created, you are able to query the history table directly while also controlling access to the data.

For more information about RCAC, see [Row and column access control \(RCAC\)](#).

Saving and restoring system-period temporal tables

Both the system-period temporal table and its history table must be explicitly saved. The database manager does not save both files when only one of the files is saved.

When a system-period temporal table is restored without its corresponding history table or a history table is restored without its corresponding system-period temporal table, the restored table's versioning relationship remains defined but is not established. When this happens, the table's versioning status is changed to the defined state. A CPI3231 informational message is issued when a system-period temporal table's versioning status is changed to the defined state. There are restrictions to the actions that can be performed on a system-period temporal table or a history table when the table is in defined state. Inserts,

updates, and deletes are prevented. Most DDL changes are also prevented. The only operations that are allowed are:

- ALTER TABLE ADD VERSIONING
- ALTER TABLE DROP VERSIONING
- DROP TABLE

To determine if a system-period temporal table is in the defined state, you can query the QSYS2/SYSPERIODS catalog view. To determine whether a history table is in the defined state, you can query the QSYS2/SYSHISTORYTABLES catalog view. Both views have the column, VERSIONING_STATUS, that contains an 'E' if a versioning relationship between the system-period temporal table and the history table has been established or a 'D' if a versioning relationship between the system-period temporal table and the history table has been defined but not established.

For more information, see [Saving physical files that are temporal](#) and [Restoring physical files that are temporal](#).

Using catalogs to find system-period temporal table information

There are several catalog views that contain system-period temporal table information.

- **QSYS2/SYSTABLES**

Contains a column called TEMPORAL_TYPE. When the value is set to “S” the table is a system-period temporal table. When it is set to “H” the table is a history table. The value “N” is used for all other tables.

- **QSYS2/SYSCOLUMNS**

Contains a column called HAS_DEFAULT. This column indicates the type of generated column.

- **QSYS2/SYSPERIODS**

Contains one row for each table with a system period and identifies temporal and versioning information.

- **QSYS2/SYSHISTORYTABLES**

Contains one row for each history table.

Temporal table restrictions

System-period temporal tables are subject to a number of restrictions.

Following are the restrictions for system-period temporal tables:

- Both the system-period temporal table and its history table must be in the same schema.
- To update or delete data from a versioned system-period temporal table, both the temporal table and its history file must be journaled.
- ALTER or CREATE OR REPLACE TABLE SQL statements that cause a potential loss of data are not supported on system-period temporal tables.
- The following operations are not allowed for system-period temporal tables:
 - ALTER TABLE DROP COLUMN
 - ALTER TABLE ADD GENERATED COLUMN
- The TRUNCATE statement is not supported against a system-period temporal table.
- A distributed table cannot also be a system-period temporal table.
- The following query operations are not allowed against a system-period temporal table:
 - A period specification cannot be specified within a query that references a distributed table, a table with a read trigger, uses ICU 2.6.1 sort sequence, or has more than 1000 table references.
 - A period specification cannot be specified for a native logical file.

- A period specification cannot be specified for a table or view that references a column in a CONTAINS or SCORE function.
- A period specification cannot be specified for a view where the view definition includes an external function that is not NO SQL or an SQL function that is not inline capable.

Working with triggers and constraints

You can use triggers or constraints to manage data in your database tables.

A *trigger* is a type of program that is automatically called whenever a specified action is performed on a specific table. Triggers are useful for keeping audit trails, detecting exceptional conditions, maintaining relationships in the database, and running applications and operations that coincide with the change operation.

Alternatively, generated columns can be used instead of a trigger to maintain auditing information. For more information about using generated columns for auditing, see [“Using a system-period temporal table for tracking auditing information”](#) on page 11.

A *constraint* is a restriction or limitation that you place on your database. Constraints are implemented at the table level. You can use constraints to create referential integrity in your database.

You can work with triggers and constraints using IBM Navigator for i, SQL, or the traditional system interface.

Related concepts

[System i Navigator database tasks](#)

Writing DB2 programs

Db2 for i provides various methods for writing applications that access or update data.

You can write embedded SQL programs, external functions, external procedures, Db2 for i CLI applications, and trigger programs.

Related concepts

[Embedded SQL programming](#)

[Writing a DB2 for i5/OS CLI application](#)

Related tasks

[Creating trigger programs](#)

Related reference

[Defining an external procedure](#)

[Writing UDFs as external functions](#)

Database backup and recovery

Saving your data can be time-consuming and requires discipline. However, it is crucial that you back up your data because you never know when you might need to recover it.

Related concepts

[Backup and recovery](#)

[Journal management](#)

[Recovering and restoring your database](#)

Distributed database administration

With Db2 for i, you can work with databases that are distributed across several systems.

Related concepts

[Distributed database programming](#)




Queries and reports

You can use SQL, the Open Query File (OPNQRYF) command, the Query (QQQRY) API, Open Database Connectivity (ODBC), or the IBM Query for i licensed program to create and run queries.

One of the most common tasks that you perform with your database is to retrieve information. The system provides several methods to create and run queries and reports.

You can use an SQL statement to retrieve information. This SQL statement is called a *query*. The query searches the tables stored in your database to find the answer to the question that you posed with your SQL statement. The answer is expressed as a set of rows, which is referred to as the result set. After a query has been run, you can also create a report to display the data provided in your result set.

In addition to using SQL, you can use other functions and products to create and run queries and reports. See the following information for details.

- [IBM DB2® Web Query for IBM i overview](#)
- [Query for IBM i](#) 
- [Query Management Programming](#) 
- [Query Manager Use](#) 

In addition, you can build SELECT, INSERT, UPDATE, and DELETE SQL statements in the SQL Assist window of System i Navigator.

Related concepts

[SQL programming](#)

Related tasks

[Building SQL statements with SQL Assist](#)

Related reference

[Open Query File \(OPNQRYF\) command](#)

[Query \(QQQRY\) API](#)

Security

Authorizing users to data at the system and data levels allows you to control access to your database.

Securing your database requires you to establish ownership and public authority to objects and specific authority to your applications.

Related concepts

[DRDA server access control exit programs](#)

[Granting file and data authority](#)

[Limiting access to specific fields in a database file](#)

[Security](#)

[Specifying public authority](#)

[Using database file capabilities to control I/O operations](#)

[Using logical files to secure data](#)

Authority Options for SQL Analysis and Tuning

This topic describes the authority options for SQL analysis and tuning.

Db2 for i has a rich set of commands, stored procedures, APIs and tools for analysis and tuning of the performance aspects of database applications. Previously, a system security officer would need to grant *JOBCTL user special authority to enable database analysts and database administrators to use the database tools. Since *JOBCTL authority allows a user to change many system critical settings that are unrelated to database activity, it was not an easy decision for security officers to grant this authority. In

some cases, it was an easy decision and *JOBCTL was not granted to database analysts, thus prohibiting the use of the full set of database tools.

Note: For more information about setting overrides for the QAQQINI file refer to the following link: [QAQQINI file override support](#).

Now the security officer has additional capability to authorize access to database analysis tools and the SQL Plan Cache. Db2 for i which takes advantage of the function usage capability available in the operating system. A new function usage group called QIBM_DB has been created with function IDs in the QIBM_DB group:

1. QIBM_DB_SQLADM (Database Administrator tasks)
2. QIBM_DB_SYSMON (Database Information tasks)
3. QIBM_DB_DDMDRDA (DDM & DRDA Application Server Access)
4. QIBM_DB_ZDA (Toolbox Application Server Access)
5. QIBM_DB_SECADM (Database Security Administrator)

The security officer now has flexibility to grant authorities by either; granting *JOBCTL special authority or authorizing a user or group to the IBM i Database Administrator Function through Application Administration in System i Navigator of IBM Navigator for i. The Change Function Usage (CHGFCNUSG) command, with a function ID of QIBM_DB_SQLADM, can also be used to change the list of users that are allowed to perform Database Administration operations. The function usage controls allow groups or specific users to be allowed or denied authority. The CHGFCNUSG command also provides a parameter which can be used to grant function usage authority to any user that has *ALLOBJ user special authority. (e.g. ALLOBJAUT(*USED))

The **Database Administrator** function is needed whenever a user is analyzing and viewing SQL performance data. Some of the more common functions are displaying statements from the SQL Plan Cache, analyzing SQL Performance Monitors and SQL Plan Cache Snapshots, and displaying the SQL details of a job other than your own.

The database administrator function usage is an alternative to granting *JOBCTL, but it does not replace the requirement of having the correct object authority. To enable database administrator tasks which are unrelated to performance analysis, refer to the specific task for details on the authorization requirements. For example, to allow an administrator to reorganize a table, they must have object authorities granted, which are not covered by QIBM_DB_SQLADM.

In addition to QIBM_DB_SQLADM, the Change Function Usage (CHGFCNUSG) command, with a function ID of QIBM_DB_SYSMON, can also be used to change the list of users that are allowed to perform Database Information operations.

The **Database Information** function provides much less authority than Database Administrator. The primary use is to allow a user to examine high-level database properties. For example, a user that does not have *JOBCTL or QIBM_DB_SQLADM, could be allowed to view the SQL Plan Cache properties if granted authority to QIBM_DB_SYSMON.

To work with QIBM_DB database group function usage from System i Navigator, follow these steps:

1. Launch Application Administration as shown in figure 1.
2. Expand the 'IBM i' and 'Database' folders under the Host Applications tab as shown in figure 2.
3. Customize the Database Administrator (QIBM_DB_SQLADM) function usage as shown in figure 3.

In this example, the security officer determined that they wanted to set up a group called Dbagroup that would contain all the users that they wanted to give this level of authority. And they explicitly wanted to deny access to SIfuser. Now the security officer has one convenient and easily monitored place to view and authorize users to these functions.

Figure 1. Launch Application Administration.

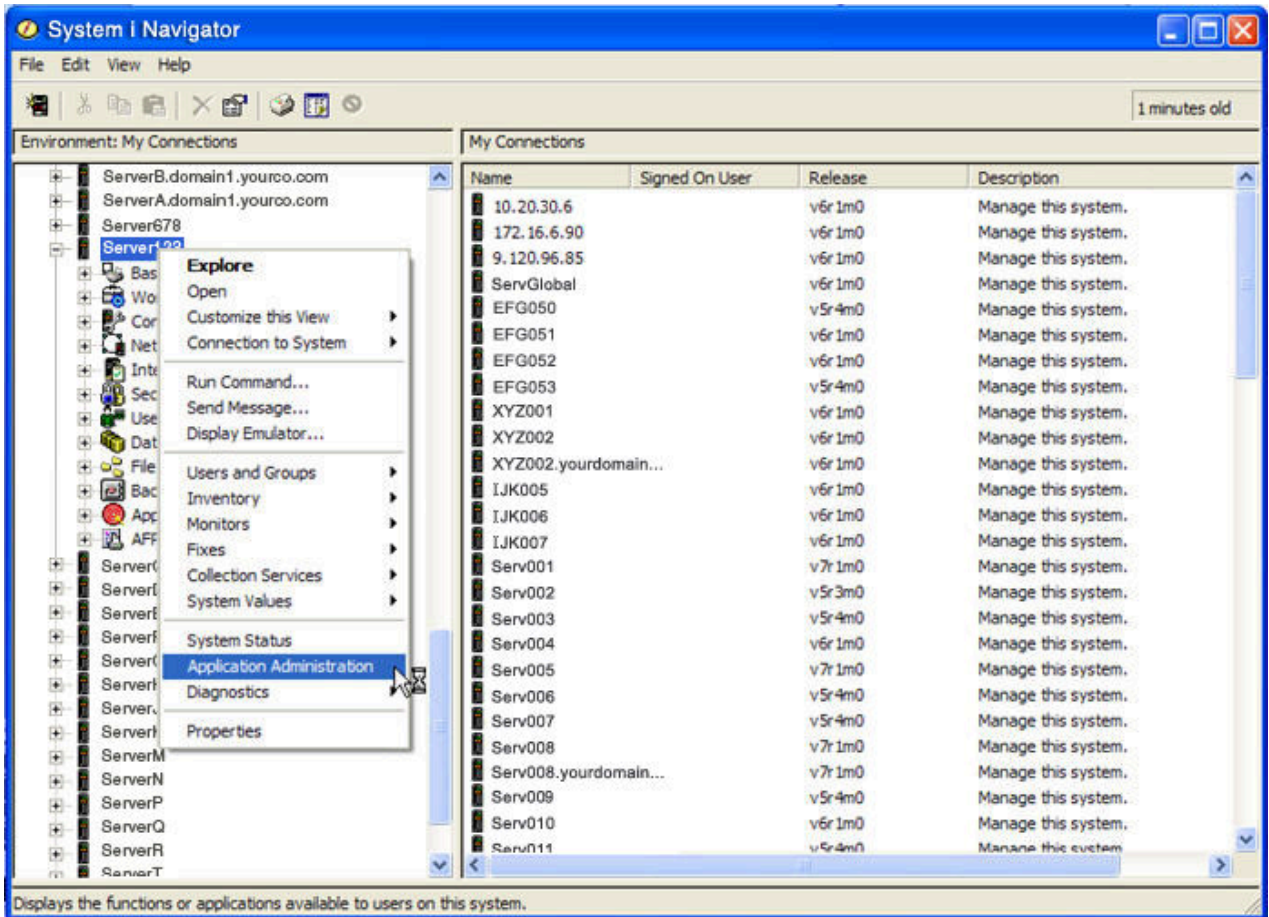


Figure 2. Expand the Database group

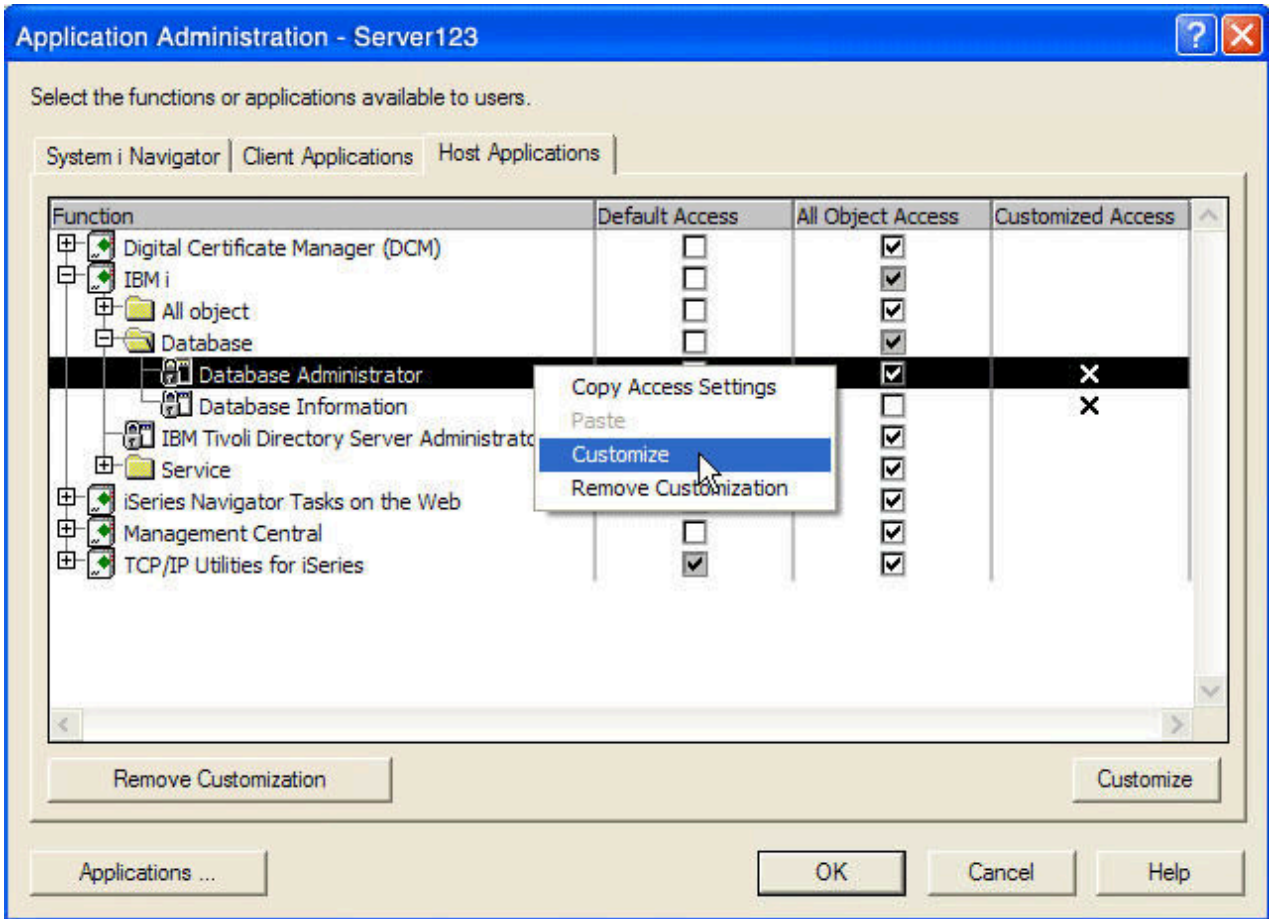


Figure 3. Change the QIBM_DB_SQLADM function usage settings

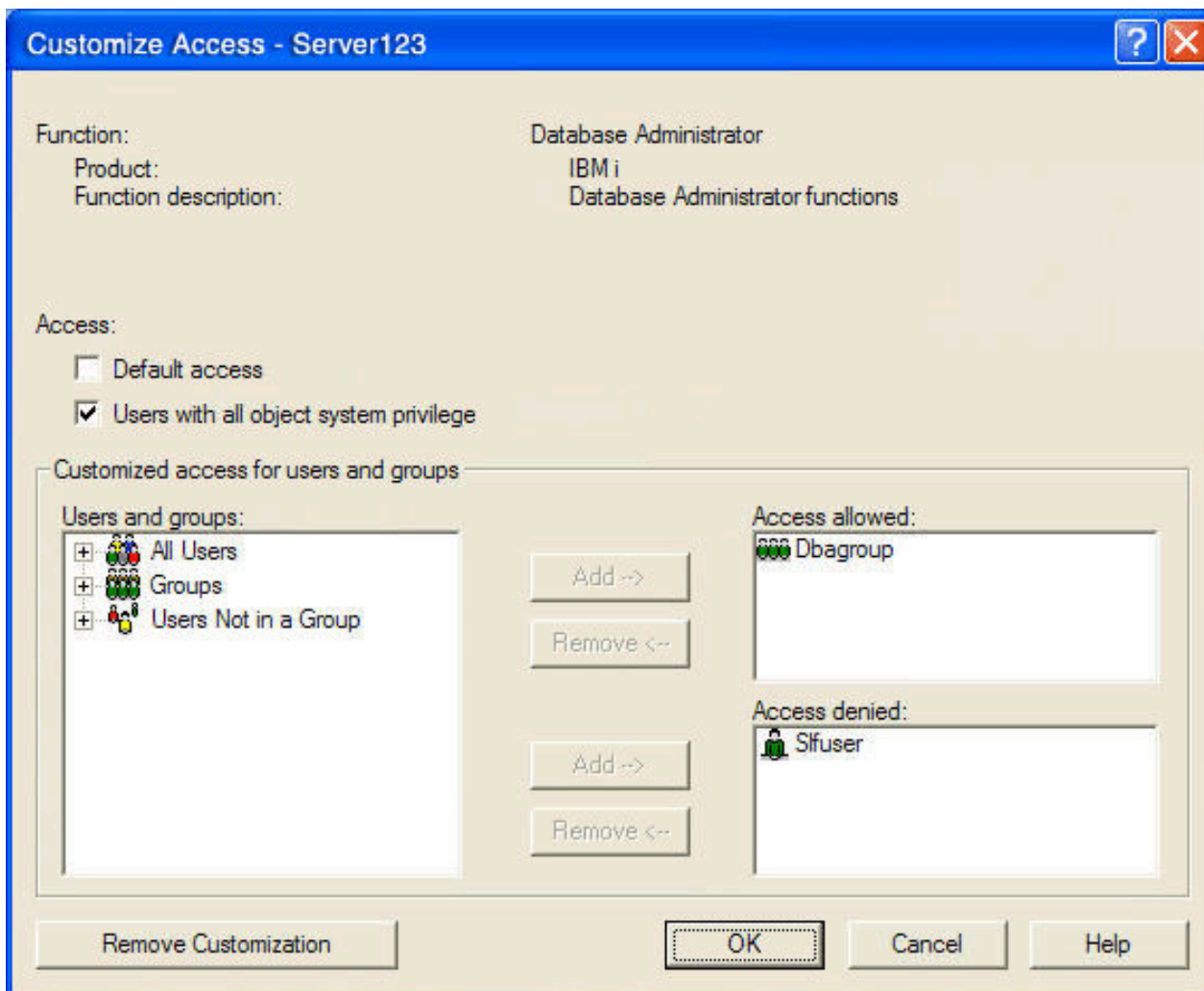


Table 1 describes some of the authorization changes related to DB2 commands, Stored Procedures, and APIs.

Table 24. Authorization requirements for Database performance and analysis

User Action	*JOBCTL	QIBM_DB_SQLADM	QIBM_DB_SYSMON	No Authority
SET CURRENT DEGREE (SQL statement)	Allowed	Allowed	Not Allowed	Not Allowed
CHGQRYA command targeting a different user's job	Allowed	Allowed	Not Allowed	Not Allowed
STRDBMON or ENDDBMON commands targeting a different user's job	Allowed	Allowed	Not Allowed	Not Allowed
STRDBMON or ENDDBMON commands targeting a job that matches the current user	Allowed	Allowed	Allowed	Allowed
QUSRJOBI() API format 900 or System i Navigator's SQL Details for Job	Allowed	Allowed	Allowed	Not Allowed
DUMP PLAN CACHE PROPERTIES procedure	Allowed	Allowed	Allowed	Not Allowed

Table 24. Authorization requirements for Database performance and analysis (continued)

User Action	*JOBCTL	QIBM_DB_SQLADM	QIBM_DB_SYSMON	No Authority
Visual Explain within Run SQL Scripts	Allowed	Allowed	Allowed	Allowed
Visual Explain outside of Run SQL Scripts	Allowed	Allowed	Not Allowed	Not Allowed
ANALYZE PLAN CACHE procedure	Allowed	Allowed	Not Allowed	Not Allowed
DUMP PLAN CACHE procedure	Allowed	Allowed	Not Allowed	Not Allowed
MODIFY PLAN CACHE procedure	Allowed	Allowed	Not Allowed	Not Allowed
MODIFY PLAN CACHE PROPERTIES procedure	Allowed	Allowed	Not Allowed	Not Allowed
CHANGE PLAN CACHE SIZE procedure	Allowed	Allowed	Not Allowed	Not Allowed
START PLAN CACHE EVENT MONITOR procedure	Allowed	Allowed	Not Allowed	Not Allowed
END PLAN CACHE EVENT MONITOR procedure	Allowed	Allowed	Not Allowed	Not Allowed
END ALL PLAN CACHE EVENT MONITORS procedure	Allowed	Allowed	Not Allowed	Not Allowed

Row and column access control (RCAC)

Row and column access control (RCAC) provide a data-centric alternative to achieve data security.

RCAC places access control at the table level around the data itself. SQL rules that are created on rows and columns are the basis of the implementation of this capability.

RCAC terms

- Base table - The table (physical file) the permission or mask is added to.
- Dependent object - Any object (file, schema, function, or other object) the permission or mask references.
- QIBM_DB_SECADM – The function usage identifier the user must be authorized to in order to manipulate all actions that are related to permissions and masks.
- Row and Column Access Control (RCAC) – Access control is the ability to control the access to data by using permissions and masks.
- Permission - A row permission defines a row access control rule for rows of a table.
- Mask - A column mask defines a column access control rule for a specific column in a table.
- RULETEXT – The expression to be used by the permission or mask.
- 5770-SS1 IBM Advanced Data Security for i (Option 47) – Product that needs to be ordered and installed to be able to:
 - create row permissions.
 - create column masks.
 - execute database access over objects that have active RCAC.

Overview

IBM Advanced Data Security for i introduces RCAC as an extra layer of data security.

RCAC provides access control to a table at the row level, column level, or both. RCAC can be used to complement the table privileges model. To comply with various government regulations, you might implement procedures and methods to ensure that information is adequately protected. Individuals in your organization are permitted access to only the subset of data that is required to perform their job tasks. For example, government regulations in your area might state that a doctor is authorized to view the medical records of their own patients, but not of other patients. The same regulations might also state that, unless a patient gives their consent, a healthcare provider is not permitted access to patient personal information, such as the patients home phone number. You can use RCAC to ensure that your users only have access to the data that is required for their work. For example, RCAC can filter patient information and data to include only that data, which a particular doctor is authorized to view.

Other patients do not exist as far as the doctor is concerned. Similarly, when a patient service representative queries the patient table at the same hospital, they are able to view the patient name and telephone number columns, but the medical history column is masked for them. If data is masked, a NULL or an alternate value is displayed instead of the actual medical history. RCAC has the following advantages:

1. No database user is inherently exempted from the RCAC rules. Even high-level authorities such as users with all object authority (special authority (such as *ALLOBJ)) authority are not exempt from these rules. Only users with QIBM_DB_SECADM authority can manage RCAC within a database. Therefore you can use RCAC to prevent users with all object authority from freely accessing all data in a database.
2. Table data is protected regardless of how a table is accessed. Applications, improvised query tools and report generation tools are all subject to the access control rules. The enforcement is data-centric.
3. No application changes are required to take advantage of this additional layer of data security. RCAC is established and defined in a way that is not apparent to existing applications. However RCAC represents an important shift in paradigm in the sense that it is no longer what is being asked but rather who is asking. Even though two users can execute what appears to be identical queries, when row permission predicates are added to the query, those two users might observe a different result set. This behavior is the exact intent of the solution. It means that application designers and DBAs must be conscious that queries do not see the whole picture in terms of the data in the table unless granted RCAC authorization.
4. Prior to RCAC controls for data-centric data protection, DB2 for i users would protect the data through the creation of several to many SQL views or Select-omit logical files. While this technique of relying upon a view/logical file to limit data achieves the goal, it creates several problems:
 - a. Applications had to be coded to work with specialized views, instead of a common object.
 - b. In large installations, the number of views which exist for this purpose quickly grows to a large number, resulting in additional object management considerations like Save/Restore.
 - c. The security officer has to spend time adjusting authorizations to many objects.
 - d. For select-omit logical files, DB2 for i has to spend processing cycles to keep each select-omit logical file up to date as the underlying object(s) change.

Besides achieving the benefits of innately secure data when deploying RCAC, DB2 for i customers can retire the many views which exist solely to protect data.

IBM Advanced Data Security for i

IBM Advanced Data Security for i is an installable option that is used to manage security policies by enforcing RCAC with permissions and masks.

If IBM Advanced Data Security for i, is not installed, see Installing, upgrading, or deleting IBM i/OS® and related software for information about installing extra licensed programs. To install IBM Advanced Data Security for i, use option 47 in the list of installable options for the operating system.

Tables which contain enabled RCAC permissions or masks can be restored regardless of whether the IBM Advanced Data Security for i is installed. However if the option is not installed, permissions and masks cannot be created and tables, views, or indexes cannot be accessed which contain active permissions or masks.

Separation of duties

Separation of duties helps businesses comply with industry regulations or organizational requirements and simplifies the management of authorities. Separation of duties is commonly used to prevent fraudulent activities or errors by a single person. It provides the ability for administrative functions to be divided across individuals without overlapping responsibilities, so that one user does not possess unlimited authority, such as with *ALLOBJ authority.

For example, assume that a business has assigned the duty to manage security on IBM i to Theresa. Prior to release IBM i 7.2, in order to grant privileges, Theresa had to have the same privileges Theresa was granting. Thus, in order to grant *USE privileges to the PAYROLL table, Theresa had to have *OBJMGT and *USE authority (or a higher level of authority such as *ALLOBJ). This requirement allowed Theresa to access data in the PAYROLL table even though Theresa's job description was only to manage security.

In IBM i 7.2, the function usage, QIBM_DB_SECADM, provides a user with the ability to grant authority, revoke authority, change ownership, or change primary group. This is done without giving access to the object or, in the case of a database table, to the data that is in the table or allowing other operations on the table. QIBM_DB_SECADM function usage can only be granted by a user with *SECADM special authority and can be given to a user or a group.

QIBM_DB_SECADM is also responsible for administering RCAC. RCAC restricts which rows a user is allowed to access in a table and whether a user is allowed to see information in certain columns of a table.

The best practice is that the RCAC administrator has QIBM_DB_SECADM function usage and absolutely no data privileges. The RCAC administrator can deploy and maintain the RCAC constructs and would be unable to grant themselves unauthorized access to data.

Permissions and masks

RCAC is a model in which a security administrator manages privacy and security policies.

RCAC permits all users to access the same table, as opposed to alternative views of a table. RCAC, however, restricts access to the data in the table based on individual user permissions or rules as specified by a policy that is associated with the table. There are two sets of rules. One set of rules operates on rows (permissions) and the other on columns (masks). In order to create permissions and masks the IBM Advanced Data Security for i must be installed.

Row permission

- A row permission defines a row access control rule for a specific table.
- A row access control rule is an SQL search condition that describes what set of rows a user can access.
- The definition of each row permission may reference the user or group in the search condition. If multiple row permissions are defined for a table and row access control is activated, the search condition in each row permission is connected by the logical OR operator to form the row access control search condition. This row access control search condition is applied whenever the table is accessed. It acts as a filter to the table before any other user-specified operations, such as predicates and ordering are processed. It acts like the WITH CHECK OPTION clause of a view to ensure that a row to be inserted or updated conforms to the definitions of the row permissions in an INSERT, UPDATE, or MERGE statement.

Column mask

- A column mask defines a column access control rule for a specific column in a table.
- A column access control rule is an SQL CASE expression that describes what column values a user is permitted to see and under what conditions.

- The definition of each column mask may reference the user or group in the search conditions in the CASE WHEN clause. While multiple columns in a table may have column masks, only one column mask can be created for a single column. When column access control is activated for the table, the CASE expression in the column mask definition is applied to the output column to determine the masked values that are returned to an application. The application of column masks affects the final output only. It does not impact the operations, such as predicates and ordering in an SQL statement.

RCAC can be activated for a table before or after row permissions or column masks are created for the table. If row permissions or column masks exist, activating row and column access control simply makes the permissions or masks become effective. If row permissions do not yet exist, activating row access control for a table means that Db2 for i generates a default row permission that prevents any access to the data in the table.

SQL statements

The SQL create, alter, and drop statements support the implementation of RCAC with permissions and masks.

- [Create Permission](#)
- [Alter Permission](#)
- [Drop Permission](#)
- [Create Mask](#)
- [Alter Mask](#)
- [Drop Mask](#)
- [Alter Function](#)
- [Alter Trigger](#)
- [Alter Table](#)

Authorization

The authorization ID of the SQL statement must be authorized to the Database Security Administrator function of IBM i. See [Administrative authority](#).

Secure functions

Functions must be defined as secure before they can be called within RCAC definitions.

The SECURED attribute is required if the UDF is referenced in the definition of a row permission or column mask because the UDF will have access to data prior to the application of RCAC. The SECURED attribute is also required for a UDF that is invoked in an SQL statement when the function arguments reference columns that are activated with column access control.

Secure triggers

Triggers defined on a table with RCAC activated must be secure.

The SECURED attribute is required for a trigger when the associated table has RCAC activated or the associated view whose underlying table is activated with RCAC. If a trigger exists but is not secure, RCAC cannot be activated for the associated table.

Administrative authority

Authorization to the Database Security Administrator function of IBM i can be assigned through Application Administration in IBM Navigator for i.

The Change Function Usage Information (CHGFCNUSG) command, with a function ID of QIBM_DB_SECADM, can be used to change the list of authorized users.

Best practices when using permissions and masks

Permissions and masks can be created for a table in a number of different implementations. This section will explain some of the implementations that can be used to create permissions and masks.

Creating permissions and masks

A number of considerations need to be determined to decide the best way to create permissions or masks.

Creating permissions or masks when row or column access control is active

The job creating the permission or mask obtains an exclusive lock on the base table. If row or column access control is active, the base table is not allowed to be read until the creating of the permission or mask is complete. The applications reading the base table need to be ended until the permissions or masks are added.

Creating permissions or masks when row or column access control is not active

The job creating the permission or mask obtains an exclusive lock on the base table. However, the base table is allowed to be read until the row or column access control is activated. The applications reading the base table do not have to be ended.

Single permission with all users

Example 1: Using a single permission with all the users defined in the permission.

```
CREATE SCHEMA MY_LIB
CREATE TABLE MY_LIB.PERMISSION_TABLE (COLUMN1 INT)
CREATE PERMISSION MY_LIB.PERM1 ON MY_LIB.PERMISSION_TABLE FOR ROWS WHERE
    VERIFY_GROUP_FOR_USER(CURRENT_USER, 'USER1', 'USER2', 'USER3') = 1
    ENFORCED FOR ALL ACCESS ENABLE

ALTER TABLE MY_LIB.PERMISSION_TABLE ACTIVATE ROW ACCESS CONTROL
/*****
/* Sign on as USER1 */
/*****/
INSERT INTO MY_LIB.PERMISSION_TABLE VALUES(1) /* Allowed. */
```

The advantage of a single permission is the best query performance for applications. The disadvantage is adding another user, the permission has to be dropped and created to add the new user.

Single permission with a group profile

Example 2: Using a single permission with all the users defined in a group profile in the permission.

```
CREATE SCHEMA MY_LIB
CREATE TABLE MY_LIB.PERMISSION_TABLE (COLUMN1 INT)
CREATE PERMISSION MY_LIB.PERM1 ON MY_LIB.PERMISSION_TABLE
    AS P_GROUP FOR ROWS WHERE
    VERIFY_GROUP_FOR_USER(SESSION_USER, 'PERM_GROUP') = 1
    ENFORCED FOR ALL ACCESS ENABLE
ALTER TABLE MY_LIB.PERMISSION_TABLE ACTIVATE ROW ACCESS CONTROL
/*****/
/* Sign on as USER1 which is a member of the user group PERM_GROUP */
/*****/
INSERT INTO MY_LIB.PERMISSION_TABLE VALUES(1) /* Allowed. */
```

The advantage of a single permission checking a group profile means the permission does not have to change adding another user. The disadvantage for every query of the base table, the VERIFY_GROUP_FOR_USER function is checked.

Single permission with a dependent table

Example 3: Using a single permission with the users defined in a dependent table.

```
CREATE SCHEMA MY_LIB
```

```

CREATE SCHEMA RCAC_DEPENDENT
CREATE TABLE MY_LIB.PERMISSION_TABLE (COLUMN1 INT)
CREATE TABLE RCAC_DEPENDENT.USERS (USERNAME CHAR (10))
INSERT INTO RCAC_DEPENDENT.USERS
VALUES('USER1      '), ('USER2      '), ('USER3      ')
CREATE TABLE MY_LIB.PERMISSION_TABLE (FIELD1 INT)
CREATE PERMISSION MY_LIB.PERM1 ON MY_LIB.PERMISSION_TABLE
FOR ROWS WHERE
CURRENT_USER IN (SELECT USERNAME FROM RCAC_DEPENDENT.USERS)
ENFORCED FOR ALL ACCESS ENABLE
ALTER TABLE MY_LIB.PERMISSION_TABLE ACTIVATE ROW ACCESS CONTROL
/*****
/* Sign on as USER1
/*****
INSERT INTO MY_LIB.PERMISSION_TABLE VALUES(1) /* Allowed.

```

The advantage of a single permission checking a dependent table is that when adding another user, the permission does not have to change. The disadvantage is the performance consideration of querying the dependent table.

Single permission with a UDF

Example 4: Using a single permission with a User Defined Function (UDF).

```

CREATE SCHEMA RCAC_DEPENDENT
CREATE SCHEMA MY_LIB
CREATE TABLE MY_LIB.PERMISSION_TABLE (COLUMN1 INT)
CREATE OR REPLACE FUNCTION RCAC_DEPENDENT.UDF_PERMISSION
(
)
RETURNS CHAR(10)
LANGUAGE SQL
MODIFIES SQL DATA
NO EXTERNAL ACTION
DETERMINISTIC
NOT FENCED
SECURED
BEGIN
DECLARE ALLOWS CHAR(10);
IF (CURRENT_USER = 'USER1') THEN
SET ALLOWS = 'ALLOWED';
ELSE
SET ALLOWS = 'DISALLOWED'; END IF;
RETURN ALLOWS;
END

CREATE PERMISSION MY_LIB.PERMISSION_USER
ON MY_LIB.PERMISSION_TABLE
FOR ROWS WHERE
RCAC_DEPENDENT.UDF_PERMISSION() = 'ALLOWED'
ENFORCED FOR ALL ACCESS ENABLE

ALTER TABLE MY_LIB.PERMISSION_TABLE ACTIVATE ROW ACCESS CONTROL

```

The advantage of a single permission checking a UDF is adding another user, the permission does not have to change. The disadvantage appears when the UDF changed. During the next open of the table with the permission, verification must be done to allow the new UDF to be used with the permission. The verification causes the permission or mask to be regenerated once for the table.

Permissions for each user

Example 5: Using multiple permissions, a permission for each user.

```

CREATE SCHEMA MY_LIB
CREATE TABLE MY_LIB.PERMISSION_TABLE (COLUMN1 INT)

CREATE PERMISSION MY_LIB.P1 ON MY_LIB.PERMISSION_TABLE
FOR ROWS WHERE
CURRENT_USER = 'USER1'
ENFORCED FOR ALL ACCESS ENABLE

CREATE PERMISSION MY_LIB.P2 ON MY_LIB.PERMISSION_TABLE
FOR ROWS WHERE

```

```

CURRENT_USER = 'USER2      '
ENFORCED FOR ALL ACCESS ENABLE

CREATE PERMISSION MY_LIB.P3 ON MY_LIB.PERMISSION_TABLE
FOR ROWS WHERE
CURRENT_USER = 'USER3      '
ENFORCED FOR ALL ACCESS ENABLE

ALTER TABLE MY_LIB.PERMISSION_TABLE ACTIVATE ROW ACCESS CONTROL

```

The advantage of multiple permissions is the ease of use of having individual permissions. The disadvantage is having to add another user, a new permission has to be added. The new permission causes a regeneration of the composite permission used for the table.

Attributes of multiple permissions

The attributes for each permission of the base table need to be the same.

The attributes need to be the same because when the permission is executed, the data (rows of the base table) is checked for each permission. For example, take the case where one permission is using a *PERIOD as the decimal point and another permission is using a *COMMA. The permissions are different because the type of decimal point that is expected by each permission is not the same. The following attributes can change the execution of the permission:

- DATFMT, TIMFMT, DATSEP, TIMSEP DECMPT
- SRTSEQ and LANGID
- DECFLTRND
- Decimal point and DECRESULT

If the attributes listed are not the same for each permission, an unexpected result may be returned.

Unqualified object names

Unqualified object names in the RULETEXT become schema qualified during the creation of the permissions or masks.

For example, creating permissions or masks in a test environment cause the object names to become qualified with the test schema name. Therefore, it is best to qualify the schema name to avoid confusion of the schema name.

```

CREATE SCHEMA MY_LIB
CREATE SCHEMA RCAC_LIB
CREATE TABLE MY_LIB.PERMISSION_TABLE (COLUMN1 INT)
CREATE TABLE RCAC_LIB.DEPENDENT_TABLE (COLUMN1 INT)
SET SCHEMA RCAC_LIB
CREATE PERMISSION MY_LIB.PERMISSION_USE
ON MY_LIB.PERMISSION_TABLE FOR ROWS
WHERE
COLUMN1 IN (SELECT COLUMN1 FROM DEPENDENT_TABLE)
ENFORCED FOR ALL ACCESS ENABLE

/*****
/* The select statement will show the RULETEXT as being qualified. */
/*****
SELECT CHAR(RULETEXT,200) FROM QSYS2.SYSCONTROL
WHERE SCHEMA = 'MY_LIB'

/*****
/* The RULETEXT is now qualified. */
/*****
PERMISSION_TABLE.COLUMN1 IN
(SELECT RCAC_LIB.DEPENDENT_TABLE.COLUMN1 FROM RCAC_LIB.DEPENDENT_TABLE)

```


Dependent objects

A number of considerations must be determined to decide how to handle dependent objects of the permissions and masks.

Ownership

Dependent objects of a permission or mask should be owned by the user profile with the QIBM_DB_SECADM functional authority and no object management authority should be granted to other users.

This restricts the possibility of the dependent object being manipulated by an authorized user to change a permission or mask to allow unintended access to data.

Schema

All dependent tables or views of a permission or mask should be created in a different schema than the schema of the base table.

If the user executes a Create Duplicate Object (CRTDUPOBJ), or Restore (RSTOBJ) of the base table to a new schema, the schema names of the dependent objects are not changed. By keeping the dependent tables and views in a different schema after the CRTDUPOBJ or RSTOBJ of the base table, the newly created base table references the same dependent objects as the original base table.

If the dependent objects of the permissions and masks are in the same schema, if the user duplicates the schema, the duplicated permissions and masks reference the objects of the original schema. Therefore, when cloning a schema and the objects within, the best practice is to use the Generate SQL feature within IBM i Navigator. By de-selecting the "Schema Qualify Objects" option, the resulting SQL script will no longer contain schema qualified references within the permissions and masks. The user can precede execution of the SQL script with a SET SCHEMA statement specifying the target schema.

Schema authority

The schema that contains the dependent objects should not allow object management authority to users.

By not granting object management authority to users, the dependent objects will not be allowed to be manipulated by users.

Secured UDFs

An SQL user-defined function (UDF) used in the RULETEXT of a permission or mask must be marked as SECURE.

This same rule applies for any function that may be invoked with a masked column specified as an argument. The SECURE attribute is stored in the *PGM or *SRVPGM executable that is called when the UDF is invoked. When the *PGM/*SRVPGM for a SECURED SQL function is restored, the SECURE attribute that is associated with the function may be lost unless one of the following is true:

- The user doing the restore is authorized to the QIBM_DB_SECADM function.
- The user doing the restore has *SAVSYS special authority.
- The user named QSECOFR is doing the restore.

Old Program Model (OPM) programs cannot be used for functions (UDFs) defined in permissions or masks. This is because the system cannot verify the program during other database operations such as restore or rename.

When creating a UDTF or UDF, the default is FENCED, meaning the UDTF or UDF is executed in a secondary thread. Certain SQL special registers like CURRENT USER may not behave as expected when referenced in a FENCED UDF. Therefore, when UDTFs or UDFs are used in the RCAC text, use NOT FENCED.

ALWCPYDTA and isolation level

The expressions in the RULETEXT of the permission or mask runs with the same ALWCPYDTA and isolation level attributes when opening a base table, index, or view with an active permission or mask.

For native opens the ALWCPYDTA attribute is *NO. This prevents temporary copies of the data from being used to execute the permission or mask expressions. If the permission or mask requires a temporary copy of the data, it is recommended that the corresponding expressions be moved to a secure UDF that runs with an ALWCPYDTA attribute of *YES or *OPTIMIZE. The RULETEXT of the permission, or mask could then be changed to reference the UDF instead of the expression that needed a temporary copy of the data.

Restoring objects

Restoring a different version of a dependent object of the base table can impact the existing permissions and masks.

The process to verify the dependent objects for permissions and masks is done the first time the base table is opened and not during the restore process.

Therefore, after restoring the dependent objects for the permissions or masks, the system administrator should include in the process a simple open operation of the base table. This allows the verification to be completed and avoid verification at application run time.

It is important to ensure that the proper dependent objects of the permissions and masks are restored when restoring the base tables with the permissions and masks.

Additional operations

A number of considerations must be reviewed creating permissions or masks for a table.

Adding application profile to permissions and masks

Some existing applications might need to add the profile of the job running the application to the permissions and masks of the base table.

Some examples of these applications would be the Data Propagator, High Availability (HA) software, and similar applications. If the application profile is not added to the permissions and masks, the permissions and masks are enforced and the application may use partial rows and masked data.

Reclaim Storage

After completing a Reclaim Storage (RCLSTG), any data spaces that are orphaned and found by the reclaim storage operation are added to the QRCL library as a table.

Since these data spaces could be the result of a base table that had RCAC, the data spaces that are now tables in QRCL do not have any RCAC. After the RCLSTG completes, the system administrator needs to query the tables in QRCL and handle (copy the data and delete the table) the tables that need to be protected with RCAC.

Query Reports

This recommendation applies to query report writer functions such as Query for i or DB2 for i Query Manager.

When using a web query report writer function, it is recommended, for consistent results, that a sort is also applied to any column that is used for report break processing. With the application of column masks, the sorting is done on a column before masks are applied, but the break processing that is done by the report writer function may be done using masked values. As a result, inconsistent break groupings and different summary values may be seen when running a query report after masks are defined on the based table.

MQTs

When populating or refreshing an MQT, it does not account for any predicates or expressions from masks or permissions on dependent tables.

When the MQT is used for optimization in a query, the underlying row permissions and column masks are built into the query that uses the MQT. In order for the MQT to be used for optimization, the MQT must include any columns that are used by the masks or permissions.

In the following example, the MQT TOTALSALES cannot be used by any query that includes CreditCardNum because CustID is used by the mask for CreditCardNum but it is not in the select list from the MQT.

```
CREATE SCHEMA MY_LIB
CREATE TABLE MY_LIB.SALES(CustID INT,
                          CreditCardNum VARCHAR(12),
                          Amount DEC(6,2))

CREATE MASK MY_LIB.CCN_MASK ON SALES FOR COLUMN CreditCardNum
RETURN
CASE
  WHEN (CustID < 10) THEN CreditCardNum
  ELSE 'b*****' || SUBSTR(CreditCardNum, 9, 4)
END
ENABLE;

CREATE TABLE MY_LIB.TOTALSALES
AS (SELECT CreditCardNum AS SCCN, SUM(Amount) AS SSUM
    FROM SALES
    GROUP BY CreditCardNum)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY USER

SELECT CreditCardNum, Sum(Amount)
FROM MY_LIB.SALES
GROUP BY CreditCardNum
```

Temporal tables

A system-period temporal table can have a row permission or a column mask.

For more information about temporal tables and RCAC, see [“Row and column access control with a system-period temporal table”](#) on page 22.

Group Profiles and QIBM_DB_SECADM

Authorization IDs that are authorized to the QIBM_DB_SECADM function should not be added to a group profile.

Such an authorization ID can transfer ownership or grant privileges for an object to any authorization ID other than itself. However, the authorized ID still can transfer or grant to the group of which the authorized ID is a member.

Users who have the necessary authorities to delete, move, copy, rename, or replace the *PGM/*SRVPGM objects are unable to do those operations when the *PGM/*SRVPGM object corresponds to a SECURE FUNCTION and the user is authorized to the QIBM_DB_SECADM function. A user that is allowed to use the QIBM_DB_SECADM function can use the Create SQL ILE CL commands (CRTSQLCBLI, CRTSQLCI, CRTSQLCCPPI, or CRTSQLRPGI) or any of the Create Bound Program CL commands (CRTBNDC, CRTBNDCBL, CRTBNDCCL, CRTBNDCPP, CRTBNDRPG) to replace a *PGM/*SRVPGM associated with a SECURE FUNCTION.

After the object is created, the object can be copied to the QRPLOBJ library. The QRPLOBJ copy of the SECURE FUNCTION can be copied or moved to another library, but will not be allowed to be used as a SECURE FUNCTION unless the program is renamed, moved, copied, or saved/restored by a user with QIBM_DB_SECADM authority. Remember, a user without QIBM_DB_SECADM authority is allowed to delete, move, or copy the object in QRPLOBJ, but is not allowed to delete it from the library to which it was moved or copied.

Copy File (CPYF) parameters

The Copy File (CPYF) command can compare returned values from the FROMFILE TOKEY, INCCHAR and INCREL parameters.

If a mask is defined for the column that is used by any of these parameters, the mask value is returned from the FROMFILE and used by the parameter that could result in unexpected results.

OmniFind Text Search Server for DB2 for i

The OmniFind Text Search Server for DB2 for i (5733-OMF) version 1.3 or higher allows customers to create a text search index over a column of a table that is protected by RCAC.

After a text search index is created, the CONTAINS and SCORE built-in SQL functions can be used to perform full text searches over the indexed column. Customers should be aware of the following considerations when creating a text search index over a column that is protected by RCAC.

- A text search server performs the task of indexing and searching documents; the indexed data is stored outside of DB2 as stream files in the integrated file system. Because the indexed data is stored outside of DB2, users that have access to the text search server could possibly reconstruct sensitive documents from the index.
- Data is exchanged with the text search server using network protocols that are not encrypted, digital certificates are not verified.
- A text search index requires that the base table contain one or more identifying columns that are a primary key, unique index, or ROWID. The identifying column is used to identify a specific row when interacting with the text search server or an administrator; the values are stored in the staging table, and may be returned from administrative procedures. When a text search index is created over a table that is protected by RCAC, the identifying column should contain a generated value, such as a ROWID or an identity column. This allows individual rows to be identified using non-sensitive information. For more information, please refer to the [OmniFind Text Search Server for DB2 for i](#).

Using RCAC on Multi-Formatted Logical Files

A multiple format logical file contains either more than one record format or has more than one file that is specified on the PFILE keyword (DDS) of a logical file.

In order to open a logical file where the logical file has more than one file that is specified on the PFILE keyword, the following criteria must be met:

1. Each permission or mask on the same based on physical file must have a unique correlation name.
2. Since permissions and mask names in the same library must be unique and cannot use the mask or permission name for determining a match between two tables. Instead, the match is using the correlation name. The correlation name that is used for the “same” permission or mask that is applied to multiple based on physical files must be the same for each file.
3. The RULETEXT for a matching permission or mask must be the same. In cases where no correlation name is specified on the permission or mask, the RULETEXT is normalized to use the table name as the correlation name. Therefore, the only way to force RULETEXT to be the same between two permissions and masks is to use the same explicit correlation name.
4. Each matching mask or permission between tables must be defined with the same parser options:
 - Date/time format and separator
 - SRTSEQ and LANGID
 - DECFLTRND
 - Decimal point and DECRESULT
 - CCSID of RULETEXT
5. RCAC for every based on physical file must be in the same active, or deactive state.
6. Each mask or permission must be in the same ENABLED/DISABLED state as its match on the other based on physical files.

In this example LF1 is based on PF1, PF2 and PF3. and each definition uses the correlation name PERM1 so that the SQL checking code can identify them as being equivalent.

```
CREATE SCHEMA MY_LIB
SET SCHEMA MY_LIB
CREATE TABLE MY_LIB.PF1 (COLUMN1 INT)
CREATE TABLE MY_LIB.PF2 (COLUMN1 INT)
CREATE TABLE MY_LIB.PF3 (COLUMN1 INT)

DDS for LF1
FMT LF .....A.....T.Name+++++.Len++TdP.....Functions+++++
      R RECORD1                PFILE(PF1 PF2 PF3)
      COLUMN1
      K COLUMN1

ADDLIBLE MY_LIB
CRTLF FILE(MY_LIB/LF1) SRCFILE(MY_LIB/QDSSRC)

CREATE PERMISSION PF1_P1 ON MY_LIB.PF1 AS PERM1 FOR ROWS WHERE
CURRENT_USER = 'USER3' ENFORCED FOR ALL ACCESS

CREATE PERMISSION PF2_P2 ON MY_LIB.PF2 AS PERM1 FOR ROWS WHERE
CURRENT_USER = 'USER3' ENFORCED FOR ALL ACCESS

CREATE PERMISSION PF3_P3 ON MY_LIB.PF3 AS PERM1 FOR ROWS WHERE
CURRENT_USER = 'USER3' ENFORCED FOR ALL ACCESS

CREATE MASK PF1_M1 ON MY_LIB.PF1 AS MASK1
FOR COLUMN COLUMN1 RETURN
CASE WHEN COLUMN1 > 55000 THEN 0 END

CREATE MASK PF2_M2 ON MY_LIB.PF2 AS MASK1
FOR COLUMN COLUMN1 RETURN
CASE WHEN COLUMN1 > 55000 THEN 0 END

CREATE MASK PF3_M3 ON MY_LIB.PF3 AS MASK1
FOR COLUMN COLUMN1 RETURN
CASE WHEN COLUMN1 > 55000 THEN 0 END

ALTER TABLE PF1 ACTIVATE ROW ACCESS CONTROL
ALTER TABLE PF2 ACTIVATE ROW ACCESS CONTROL
ALTER TABLE PF3 ACTIVATE ROW ACCESS CONTROL

ALTER TABLE PF1 ACTIVATE COLUMN ACCESS CONTROL
ALTER TABLE PF2 ACTIVATE COLUMN ACCESS CONTROL
ALTER TABLE PF3 ACTIVATE COLUMN ACCESS CONTROL
```

Propagation of masked data

Performing an insert or update operation into a base table with active column access control, the operation may fail because the data is the masked data.

This can happen when the data to be inserted or updated contains the masked value, and the masked data was selected from a table with active column access control and the select was done in the same SQL statement. As an example, assume that both TABLE1 and TABLE2 have active column access control and for the insert, selecting from TABLE2 would return the masked data. The following statement would return an error:

```
INSERT INTO TABLE1 SELECT * FROM TABLE2
```

The statement would fail with SQ20478 – Row or column access control is not valid.

However, assume for this example, TABLE1 and TABLE2 contain two columns, NAME and SSN. For the user doing the INSERT, the mask is defined to return the string 'XXX-XX-nnnn' when querying TABLE2.

```
SELECT NAME, SSN INTO :name, :ssn FROM TABLE2;
INSERT INTO TABLE1 VALUES(:name, :ssn);
```

This same type of problem can also occur if the user is running a native database application. A READ from TABLE2 followed by a WRITE into TABLE1 could result in masked data that is written to the file. Or in

the case of an update, even if the SSN column is not intended to change on the UPDATE, the record being updated contains the masked value for the SSN column and the SSN column changes.

Two solutions to prevent masked data are provided:

1. BEFORE trigger.
2. CHECK constraint.

Before Trigger Solution

The trigger solution checks the new data that is written into a column and conditionally sets the column to the current value, or sets it to the DEFAULT.

This is an example of a before insert/update trigger for preventing masked data:

```
CREATE SCHEMA MY_LIB
CREATE TABLE MY_LIB.EMP_INFO
    (COL1_name CHAR(10) WITH DEFAULT 'DEFAULT',
    COL2_ssn CHAR(11) WITH DEFAULT 'DEFAULT')

/*****
/* Create a mask to give COL2_ssn for DBMGR, but for any other user */
/* mask the column. This table will contain a trigger to ensure the */
/* column can never contain a masked value. */
*****/

CREATE MASK MASK_SSN ON MY_LIB.EMP_INFO
FOR COLUMN COL2_ssn
RETURN
CASE
    WHEN VERIFY_GROUP_FOR_USER(SESSION_USER, 'DBMGR') = 1
    THEN COL2_ssn
    ELSE 'XXX-XX-' || SUBSTR(COL2_ssn,8,4)
END
ENABLE

ALTER TABLE MY_LIB.EMP_INFO ACTIVATE COLUMN ACCESS CONTROL

CREATE TRIGGER PREVENT_MASK_SSN BEFORE INSERT OR UPDATE ON MY_LIB.EMP_INFO
REFERENCING NEW ROW AS N OLD ROW AS O
FOR EACH ROW MODE DB2ROW
SECURED
WHEN (SUBSTR(N.COL2_ssn,1,7) = 'XXX-XX-')
BEGIN
    IF INSERTING THEN SET N.COL2_ssn = DEFAULT;
    ELSEIF UPDATING THEN SET N.COL2_ssn = O.COL2_ssn;
    END IF;
END
```

Attempting an insert or update operation causes the before trigger to be executed and ensure the correct data into column COL2_ssn.

Check Constraint Solution

The check constraint-based solution provides new SQL syntax to allow the specification of an action to perform when a violation of the check constraint's check-condition occurs instead of returning a hard error. However, if the check-condition continues to fail after the action is taken, a hard error will be returned and the SQL statement fails with the existing constraint failure, (SQLSTATE=23513, SQLCODE=-545).

A check constraint with the on-violation-clause is allowed on both the CREATE TABLE and ALTER TABLE statements.

In the following example, the mask is defined to return a value of 'XXX-XX-nnnn' for any query that is not done by a user profile in the 'DBMGR' group. The constraint checks that the column SSN does not have the masked value.

```
CREATE SCHEMA MY_LIB
SET SCHEMA MY_LIB
CREATE TABLE MY_LIB.EMP_INFO
```

```

(COL1_name CHAR(10) WITH DEFAULT 'DEFAULT',
COL2_ssn CHAR(11) WITH DEFAULT 'DEFAULT')


CREATE MASK MASK_ssn ON MY_LIB.EMP_INFO
FOR COLUMN COL2_ssn RETURN
CASE
WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'DBMGR' ) = 1
THEN COL2_ssn
ELSE 'XXX-XX-' || SUBSTR(COL2_ssn,8,4)
END
ENABLE

/* Check constraint for the update and insert.*/
ALTER TABLE MY_LIB.EMP_INFO
ADD CONSTRAINT MASK_ssn_preserve
CHECK(SUBSTR(COL2_ssn,1,7)<>'XXX-XX-')
ON UPDATE VIOLATION PRESERVE COL2_ssn
ON INSERT VIOLATION SET COL2_ssn = 'DEFAULT'

```

Classic Query Engine (CQE) and SQL Query Engine (SQE)

The section explains the native open and query processing differences between CQE and SQE.

For more information about general differences between CQE and SQE see the article [Classic Query Engine \(CQE\) and SQL Query Engine \(SQE\) Differences](#) .

Native open and SQL query differences involving RCAC

Some files with RCAC are not allowed to be accessed.

An attempt to use the native environment to open a file with active RCAC involving any of the following is not allowed:

- A logical file with multiple formats if the open attempt is for more than one format.
- A distributed file.
- A file with read triggers.
- A program described file.
- A file or query that specifies an ICU 2.6.1 sort sequence.

An attempt to use SQL to query a table with active RCAC involving any of the following is not allowed:

- A distributed file.
- A file with read triggers.
- A file or query that specifies an ICU 2.6.1 sort sequence.

Result set ordering

SQE implementation may result in a different result set ordering for WRKQRY, RUNQRY, or OPNQRYF.

When a query is performed without explicitly specifying that the results be returned in a specific order, both the SQE and CQE optimizers will choose whatever plan will perform the best. This means that both SQE and CQE may or may not return the results in a keyed file order. Since CQE has far less advanced capability than SQE, it is more likely to return the results in a keyed order and SQE is less likely to return the results in a keyed order. Hence, if a query is specified with WRKQRY, RUNQRY, or OPNQRYF and the row ordering is important, explicitly specify the key field(s) and key field ordering.

Notices

This information was developed for products and services offered in the U.S.A.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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 Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 the names and addresses used by an actual business enterprise 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 follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Programming interface information

This Database administration publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks 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](http://www.ibm.com/legal/copytrade.shtml)" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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 of Oracle, Inc. in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

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 works 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.

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.



Product Number: 5770-SS1