Row and Column Access Control Support in IBM DB2 for i

- Implement roles and separation of duties
- Leverage row permissions on the database
- Protect columns by defining column masks

Jim Bainbridge
Hernando Bedoya
Rob Bestgen
Mike Cain
Dan Cruikshank
Jim Denton
Doug Mack
Tom McKinley
Kent Milligan

Redpaper

ibm.com/redbooks

Contents

DB2 for i Center of Excellence

Expert help to achieve your business requirements

Highlights

- Enhance the performance of your database operations
- Earn greater return on IT projects through modernization of database and applications
- Rely on IBM expert consulting, skills sharing and renown services
- Take advantage of access to a worldwide source of expertise

We build confident, satisfied clients

No one else has the vast consulting experiences, skills sharing and renown service offerings to do what we can do for you.

Because no one else is IBM.

With combined experiences and direct access to development groups, we're the experts in IBM DB2® for i. The DB2 for i Center of Excellence (CoE) can help you achieve—perhaps reexamine and exceed—your business requirements and gain more confidence and satisfaction in IBM product data management products and solutions.

Who we are, some of what we do

Global CoE engagements cover topics including:

- Database performance and scalability
- Advanced SQL knowledge and skills transfer
- Business intelligence and analytics
- DB2 Web Query
- Query/400 modernization for better reporting and analysis capabilities
- Database modernization and re-engineering
- Data-centric architecture and design
- Extremely large database and overcoming limits to growth
- ISV education and enablement

Preface

This IBM® Redpaper™ publication provides information about the IBM i 7.2 feature of IBM DB2® for i Row and Column Access Control (RCAC). It offers a broad description of the function and advantages of controlling access to data in a comprehensive and transparent way. This publication helps you understand the capabilities of RCAC and provides examples of defining, creating, and implementing the row permissions and column masks in a relational database environment.

This paper is intended for database engineers, data-centric application developers, and security officers who want to design and implement RCAC as a part of their data control and governance policy. A solid background in IBM i object level security, DB2 for i relational database concepts, and SQL is assumed.

Authors

This paper was produced by the IBM DB2 for i Center of Excellence team in partnership with the International Technical Support Organization (ITSO), Rochester, Minnesota US.

Jim Bainbridge is a senior DB2 consultant on the DB2 for i Center of Excellence team in the IBM Lab Services and Training organization. His primary role is training and implementation services for IBM DB2 Web Query for i and business analytics. Jim began his career with IBM 30 years ago in the IBM Rochester Development Lab, where he developed cooperative processing products that paired IBM PCs with IBM S/36 and AS/.400 systems. In the years since, Jim has held numerous technical roles, including independent software vendors technical support on a broad range of IBM technologies and products, and supporting customers in the IBM Executive Briefing Center and IBM Project Office.

Hernando Bedoya is a Senior IT Specialist at STG Lab Services and Training in Rochester, Minnesota. He writes extensively and teaches IBM classes worldwide in all areas of DB2 for i. Before joining STG Lab Services, he worked in the ITSO for nine years writing multiple IBM Redbooks® publications. He also worked for IBM Colombia as an IBM AS/400® IT Specialist doing presales support for the Andean countries. He has 28 years of experience in the computing field and has taught database classes in Colombian universities. He holds a Master's degree in Computer Science from EAFIT, Colombia. His areas of expertise are database technology, performance, and data warehousing. Hernando can be contacted at hbedoya@us.ibm.com.

1

Securing and protecting IBM DB2 data

Recent news headlines are filled with reports of data breaches and cyber-attacks impacting global businesses of all sizes. The Identity Theft Resource Center[1] reports that almost 5000 data breaches have occurred since 2005, exposing over 600 million records of data. The financial cost of these data breaches is skyrocketing. Studies from the Ponemon Institute[2] revealed that the average cost of a data breach increased in 2013 by 15% globally and resulted in a brand equity loss of $9.4 million per attack. The average cost that is incurred for each lost record containing sensitive information increased more than 9% to $145 per record.

Businesses must make a serious effort to secure their data and recognize that securing information assets is a cost of doing business. In many parts of the world and in many industries, securing the data is required by law and subject to audits. Data security is no longer an option; it is a requirement.

This chapter describes how you can secure and protect data in DB2 for i. The following topics are covered in this chapter:

- Security fundamentals
- Current state of IBM i security
- DB2 for i security controls

[1] http://www.idtheftcenter.org
[2] http://www.ponemon.org/

1.1 Security fundamentals

Before reviewing database security techniques, there are two fundamental steps in securing information assets that must be described:

- First, and most important, is the definition of a company's *security policy*. Without a security policy, there is no definition of what are acceptable practices for using, accessing, and storing information by who, what, when, where, and how. A security policy should minimally address three things: confidentiality, integrity, and availability.

  The monitoring and assessment of adherence to the security policy determines whether your security strategy is working. Often, IBM security consultants are asked to perform security assessments for companies without regard to the security policy. Although these assessments can be useful for observing how the system is defined and how data is being accessed, they cannot determine the level of security without a security policy. Without a security policy, it really is not an assessment as much as it is a baseline for monitoring the changes in the security settings that are captured.

  A security policy is what defines whether the system and its settings are secure (or not).

- The second fundamental in securing data assets is the use of *resource security*. If implemented properly, resource security prevents data breaches from both internal and external intrusions. Resource security controls are closely tied to the part of the security policy that defines who should have access to what information resources. A hacker might be good enough to get through your company firewalls and sift his way through to your system, but if they do not have explicit access to your database, the hacker cannot compromise your information assets.

With your eyes now open to the importance of securing information assets, the rest of this chapter reviews the methods that are available for securing database resources on IBM i.

1.2 Current state of IBM i security

Because of the inherently secure nature of IBM i, many clients rely on the default system settings to protect their business data that is stored in DB2 for i. In most cases, this means no data protection because the default setting for the Create default public authority (QCRTAUT) system value is *CHANGE.

Even more disturbing is that many IBM i clients remain in this state, despite the news headlines and the significant costs that are involved with databases being compromised. This default security configuration makes it quite challenging to implement basic security policies. A tighter implementation is required if you really want to protect one of your company's most valuable assets, which is the data.

Traditionally, IBM i applications have employed menu-based security to counteract this default configuration that gives all users access to the data. The theory is that data is protected by the menu options controlling what database operations that the user can perform. This approach is ineffective, even if the user profile is restricted from running interactive commands. The reason is that in today's connected world there are a multitude of interfaces into the system, from web browsers to PC clients, that bypass application menus. If there are no object-level controls, users of these newer interfaces have an open door to your data.

Many businesses are trying to limit data access to a need-to-know basis. This security goal means that users should be given access only to the minimum set of data that is required to perform their job. Often, users with object-level access are given access to row and column values that are beyond what their business task requires because that object-level security provides an all-or-nothing solution. For example, object-level controls allow a manager to access data about all employees. Most security policies limit a manager to accessing data only for the employees that they manage.

1.3.1 Existing row and column control

Some IBM i clients have tried augmenting the all-or-nothing object-level security with SQL views (or logical files) and application logic, as shown in Figure 1-2. However, application-based logic is easy to bypass with all of the different data access interfaces that are provided by the IBM i operating system, such as Open Database Connectivity (ODBC) and System i Navigator.

Using SQL views to limit access to a subset of the data in a table also has its own set of challenges. First, there is the complexity of managing all of the SQL view objects that are used for securing data access. Second, scaling a view-based security solution can be difficult as the amount of data grows and the number of users increases.

Even if you are willing to live with these performance and management issues, a user with *ALLOBJ access still can directly access all of the data in the underlying DB2 table and easily bypass the security controls that are built into an SQL view.

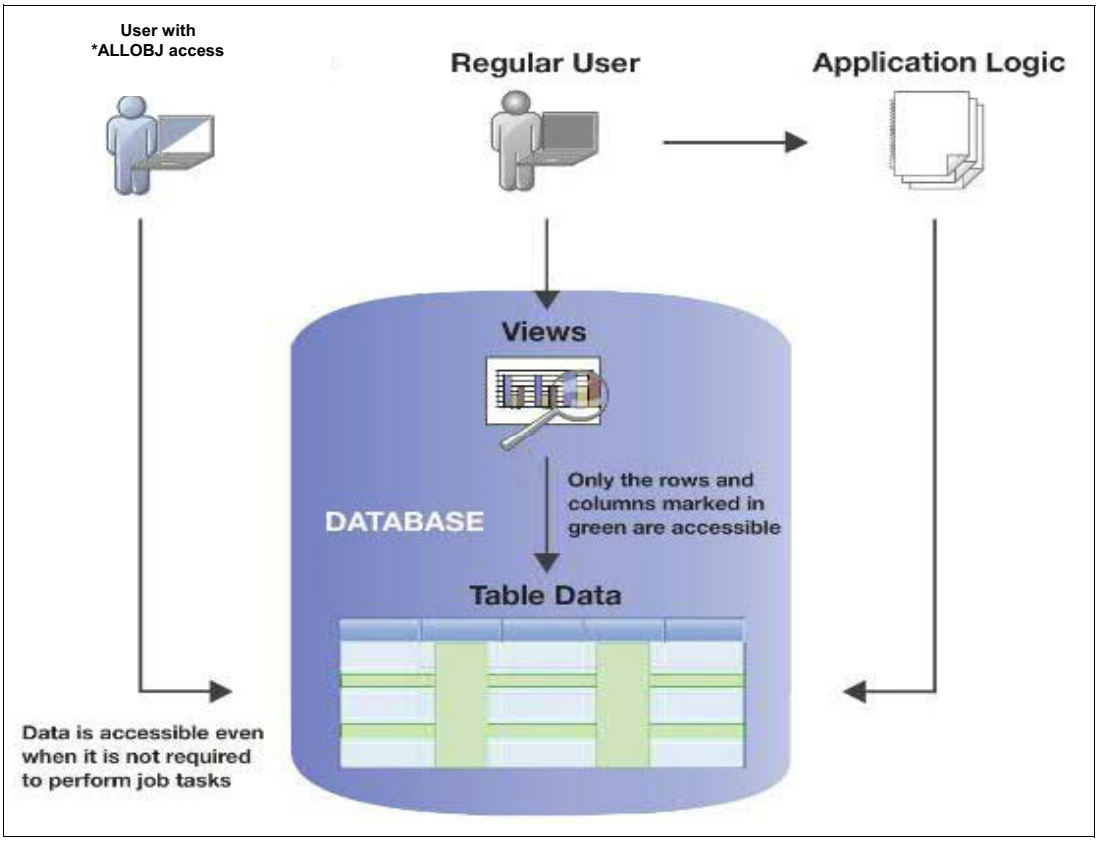

Figure 1-2 Existing row and column controls

2.1.6 Change Function Usage CL command

The following CL commands can be used to work with, display, or change function usage IDs:

- Work Function Usage (**WRKFCNUSG**)
- Change Function Usage (**CHGFCNUSG**)
- Display Function Usage (**DSPFCNUSG**)

For example, the following **CHGFCNUSG** command shows granting authorization to user HBEDOYA to administer and manage RCAC rules:

```
CHGFCNUSG FCNID(QIBM_DB_SECADM) USER(HBEDOYA) USAGE(*ALLOWED)
```

2.1.7 Verifying function usage IDs for RCAC with the FUNCTION_USAGE view

The FUNCTION_USAGE view contains function usage configuration details. Table 2-1 describes the columns in the FUNCTION_USAGE view.

Table 2-1 FUNCTION_USAGE view

Column name	Data type	Description
FUNCTION_ID	VARCHAR(30)	ID of the function.
USER_NAME	VARCHAR(10)	Name of the user profile that has a usage setting for this function.
USAGE	VARCHAR(7)	Usage setting: ► ALLOWED: The user profile is allowed to use the function. ► DENIED: The user profile is not allowed to use the function.
USER_TYPE	VARCHAR(5)	Type of user profile: ► USER: The user profile is a user. ► GROUP: The user profile is a group.

To discover who has authorization to define and manage RCAC, you can use the query that is shown in Example 2-1.

Example 2-1 Query to determine who has authority to define and manage RCAC

```
SELECT     function_id,
           user_name,
           usage,
           user_type
FROM       function_usage
WHERE      function_id='QIBM_DB_SECADM'
ORDER BY   user_name;
```

2.2 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 the *ALLOBJ authority.

For example, assume that a business has assigned the duty to manage security on IBM i to Theresa. Before release IBM i 7.2, to grant privileges, Theresa had to have the same privileges Theresa was granting to others. Therefore, 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 the data in the PAYROLL table even though Theresa's job description was only to manage its security.

In IBM i 7.2, the QIBM_DB_SECADM function usage grants authorities, revokes authorities, changes ownership, or changes the primary group 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 be granted only by a user with *SECADM special authority and can be given to a user or a group.

QIBM_DB_SECADM also is responsible for administering RCAC, which 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.

A preferred practice is that the RCAC administrator has the QIBM_DB_SECADM function usage ID, but absolutely no other data privileges. The result is that the RCAC administrator can deploy and maintain the RCAC constructs, but cannot grant themselves unauthorized access to data itself.

Table 2-2 shows a comparison of the different function usage IDs and *JOBCTL authority to the different CL commands and DB2 for i tools.

*Table 2-2 Comparison of the different function usage IDs and *JOBCTL authority*

User action	*JOBCTL	QIBM_DB_SECADM	QIBM_DB_SQLADM	QIBM_DB_SYSMON	No Authority
SET CURRENT DEGREE (SQL statement)	X		X		
CHGQRYA command targeting a different user's job	X		X		
STRDBMON or **ENDDBMON** commands targeting a different user's job	X		X		
STRDBMON or **ENDDBMON** commands targeting a job that matches the current user	X		X	X	X
QUSRJOBI() API format 900 or System i Navigator's SQL Details for Job	X		X	X	
Visual Explain within Run SQL scripts	X		X	X	X
Visual Explain outside of Run SQL scripts	X		X		
ANALYZE PLAN CACHE procedure	X		X		
DUMP PLAN CACHE procedure	X		X		
MODIFY PLAN CACHE procedure	X		X		
MODIFY PLAN CACHE PROPERTIES procedure (currently does not check authority)	X		X		
CHANGE PLAN CACHE SIZE procedure (currently does not check authority)	X		X		

The SQL **CREATE PERMISSION** statement that is shown in Figure 3-1 is used to define and initially enable or disable the row access rules.

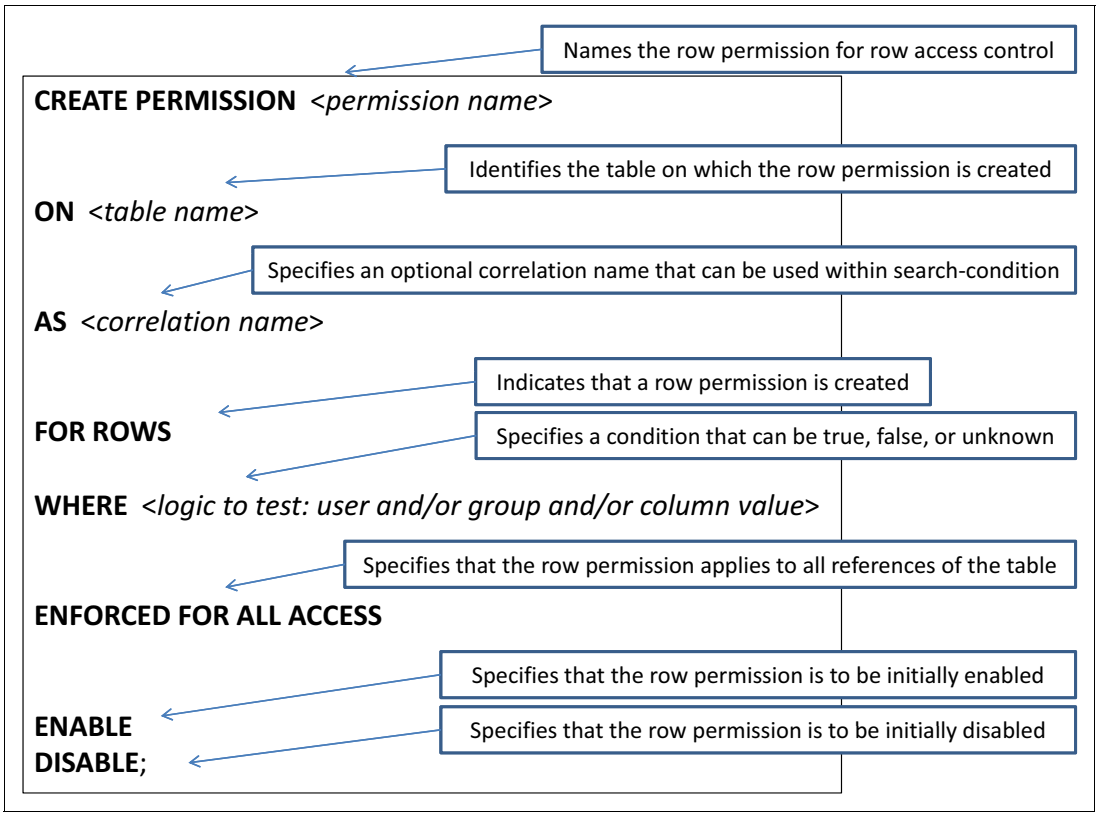

Figure 3-1 CREATE PERMISSION SQL statement

Column mask

A column mask is a database object that manifests a column value access control rule for a specific column in a specific table. It uses a CASE expression that describes what you see when you access the column. For example, a teller can see only the last four digits of a tax identification number.

Table 3-1 summarizes these special registers and their values.

Table 3-1 Special registers and their corresponding values

Special register	Corresponding value
USER or SESSION_USER	The effective user of the thread excluding adopted authority.
CURRENT_USER	The effective user of the thread including adopted authority. When no adopted authority is present, this has the same value as USER.
SYSTEM_USER	The authorization ID that initiated the connection.

Figure 3-5 shows the difference in the special register values when an adopted authority is used:

- A user connects to the server using the user profile ALICE.
- USER and CURRENT USER initially have the same value of ALICE.
- ALICE calls an SQL procedure that is named proc1, which is owned by user profile JOE and was created to adopt JOE's authority when it is called.
- While the procedure is running, the special register USER still contains the value of ALICE because it excludes any adopted authority. The special register CURRENT USER contains the value of JOE because it includes any adopted authority.
- When proc1 ends, the session reverts to its original state with both USER and CURRENT USER having the value of ALICE.

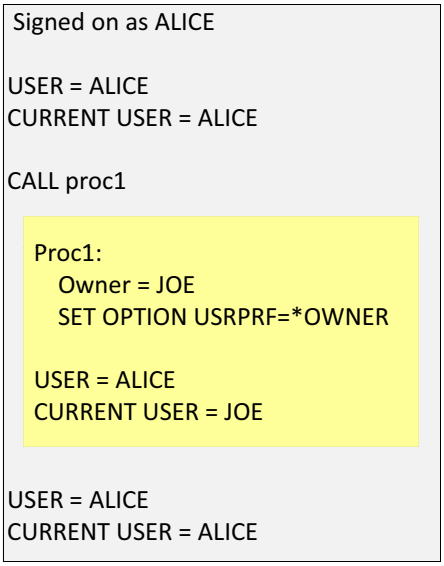

Figure 3-5 Special registers and adopted authority

3.2.2 Built-in global variables

Built-in global variables are provided with the database manager and are used in SQL statements to retrieve scalar values that are associated with the variables.

IBM DB2 for i supports nine different built-in global variables that are read only and maintained by the system. These global variables can be used to identify attributes of the database connection and used as part of the RCAC logic.

Table 3-2 lists the nine built-in global variables.

Table 3-2 Built-in global variables

Global variable	Type	Description
CLIENT_HOST	VARCHAR(255)	Host name of the current client as returned by the system
CLIENT_IPADDR	VARCHAR(128)	IP address of the current client as returned by the system
CLIENT_PORT	INTEGER	Port used by the current client to communicate with the server
PACKAGE_NAME	VARCHAR(128)	Name of the currently running package
PACKAGE_SCHEMA	VARCHAR(128)	Schema name of the currently running package
PACKAGE_VERSION	VARCHAR(64)	Version identifier of the currently running package
ROUTINE_SCHEMA	VARCHAR(128)	Schema name of the currently running routine
ROUTINE_SPECIFIC_NAME	VARCHAR(128)	Name of the currently running routine
ROUTINE_TYPE	CHAR(1)	Type of the currently running routine

3.3 VERIFY_GROUP_FOR_USER function

The VERIFY_GROUP_FOR_USER function was added in IBM i 7.2. Although it is primarily intended for use with RCAC permissions and masks, it can be used in other SQL statements. The first parameter must be one of these three special registers: SESSION_USER, USER, or CURRENT_USER. The second and subsequent parameters are a list of user or group profiles. Each of these values must be 1 - 10 characters in length. These values are not validated for their existence, which means that you can specify the names of user profiles that do not exist without receiving any kind of error.

If a special register value is in the list of user profiles or it is a member of a group profile included in the list, the function returns a long integer value of 1. Otherwise, it returns a value of 0. It never returns the null value.

Here is an example of using the VERIFY_GROUP_FOR_USER function:

1. There are user profiles for MGR, JANE, JUDY, and TONY.
2. The user profile JANE specifies a group profile of MGR.
3. If a user is connected to the server using user profile JANE, all of the following function invocations return a value of 1:

```
VERIFY_GROUP_FOR_USER (CURRENT_USER, 'MGR')
VERIFY_GROUP_FOR_USER (CURRENT_USER, 'JANE', 'MGR')
VERIFY_GROUP_FOR_USER (CURRENT_USER, 'JANE', 'MGR', 'STEVE')
```

The following function invocation returns a value of 0:

```
VERIFY_GROUP_FOR_USER (CURRENT_USER, 'JUDY', 'TONY')
```

```
RETURN
CASE
     WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'HR', 'EMP' ) = 1
     THEN EMPLOYEES . DATE_OF_BIRTH

     WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'MGR' ) = 1
     AND SESSION_USER = EMPLOYEES . USER_ID
     THEN EMPLOYEES . DATE_OF_BIRTH

     WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'MGR' ) = 1
     AND SESSION_USER <> EMPLOYEES . USER_ID
     THEN ( 9999 || '-' ||  MONTH ( EMPLOYEES . DATE_OF_BIRTH ) || '-'     ||
           DAY (EMPLOYEES.DATE_OF_BIRTH ))
     ELSE NULL
 END
 ENABLE  ;
```

2. The other column to mask in this example is the TAX_ID information. In this example, the rules to enforce include the following ones:
   - Human Resources can see the unmasked TAX_ID of the employees.
   - Employees can see only their own unmasked TAX_ID.
   - Managers see a masked version of TAX_ID with the first five characters replaced with the X character (for example, XXX-XX-1234).
   - Any other person sees the entire TAX_ID as masked, for example, XXX-XX-XXXX.

   To implement this column mask, run the SQL statement that is shown in Example 3-9.

Example 3-9 Creating a mask on the TAX_ID column

```
CREATE MASK   HR_SCHEMA.MASK_TAX_ID_ON_EMPLOYEES
ON            HR_SCHEMA.EMPLOYEES AS EMPLOYEES
FOR COLUMN    TAX_ID
RETURN
CASE
       WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'HR' ) = 1
       THEN EMPLOYEES . TAX_ID

       WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'MGR' ) = 1
       AND SESSION_USER = EMPLOYEES . USER_ID
       THEN EMPLOYEES . TAX_ID

       WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'MGR' ) = 1
       AND SESSION_USER <> EMPLOYEES . USER_ID
       THEN ( 'XXX-XX-' CONCAT QSYS2 . SUBSTR ( EMPLOYEES . TAX_ID , 8 , 4 ) )

       WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'EMP' ) = 1
       THEN EMPLOYEES . TAX_ID

       ELSE 'XXX-XX-XXXX'
END
ENABLE  ;
```

3. Figure 3-10 shows the masks that are created in the HR_SCHEMA.

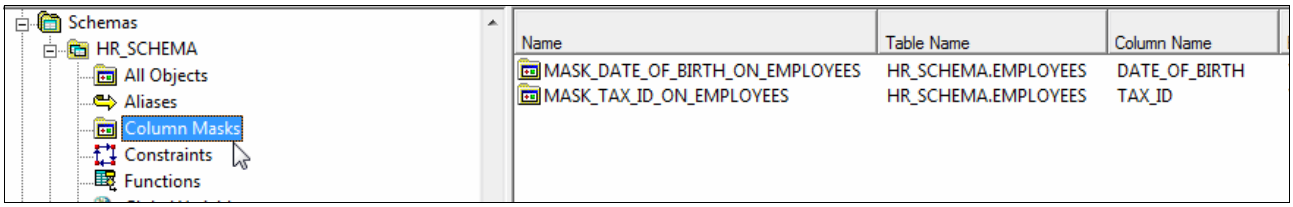

Figure 3-10 Column masks shown in System i Navigator

3.6.6 Activating RCAC

Now that you have created the row permission and the two column masks, RCAC must be activated. The row permission and the two column masks are enabled (last clause in the scripts), but now you must activate RCAC on the table. To do so, complete the following steps:

1. Run the SQL statements that are shown in Example 3-10.

Example 3-10 Activating RCAC on the EMPLOYEES table

```
/*   Active Row Access Control (permissions)  */
/*   Active Column Access Control (masks)     */
ALTER TABLE HR_SCHEMA.EMPLOYEES
ACTIVATE ROW ACCESS CONTROL
ACTIVATE COLUMN ACCESS CONTROL;
```

2. Look at the definition of the EMPLOYEE table, as shown in Figure 3-11. To do this, from the main navigation pane of System i Navigator, click **Schemas** → **HR_SCHEMA** → **Tables**, right-click the **EMPLOYEES** table, and click **Definition**.

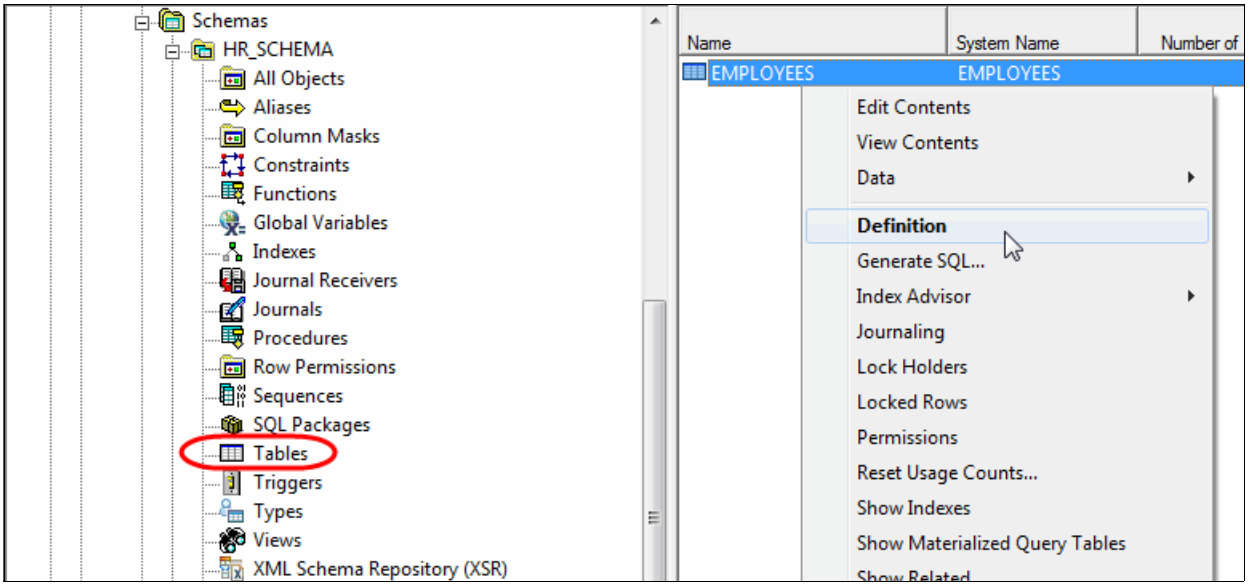

Figure 3-11 Selecting the EMPLOYEES table from System i Navigator

2. Figure 4-68 shows the Visual Explain of the same SQL statement, but with RCAC enabled. It is clear that the implementation of the SQL statement is more complex because the row permission rule becomes part of the **WHERE** clause.

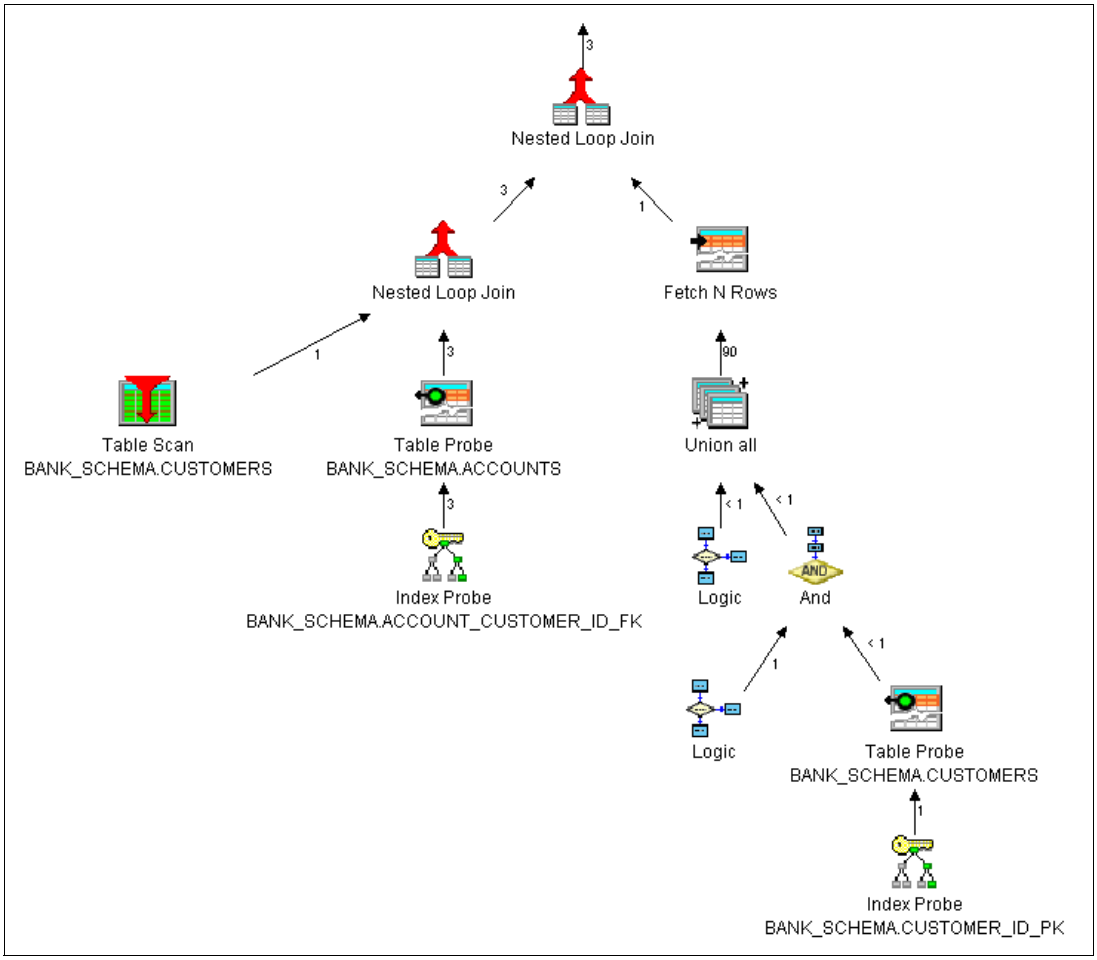

Figure 4-68 Visual Explain with RCAC enabled

3. Compare the advised indexes that are provided by the Optimizer without RCAC and with RCAC enabled. Figure 4-69 shows the index advice for the SQL statement without RCAC enabled. The index being advised is for the ORDER BY clause.

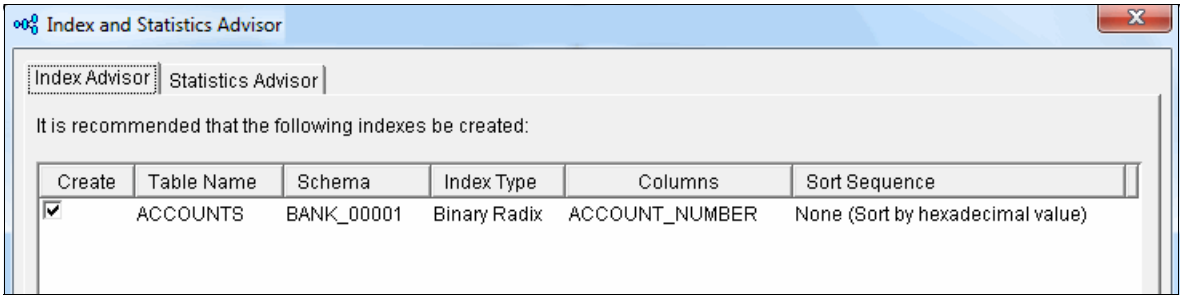

Figure 4-69 Index advice with no RCAC

```
THEN C . CUSTOMER_TAX_ID
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'TELLER' ) = 1
THEN ( 'XXX-XX-' CONCAT QSYS2 . SUBSTR ( C . CUSTOMER_TAX_ID , 8 , 4 ) )
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
THEN C . CUSTOMER_TAX_ID
ELSE 'XXX-XX-XXXX'
END
   ENABLE  ;

CREATE MASK BANK_SCHEMA.MASK_DRIVERS_LICENSE_ON_CUSTOMERS ON BANK_SCHEMA.CUSTOMERS AS C
   FOR COLUMN CUSTOMER_DRIVERS_LICENSE_NUMBER
   RETURN  CASE
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'ADMIN' ) = 1
THEN C . CUSTOMER_DRIVERS_LICENSE_NUMBER
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'TELLER' ) = 1
THEN C . CUSTOMER_DRIVERS_LICENSE_NUMBER
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
THEN C . CUSTOMER_DRIVERS_LICENSE_NUMBER
ELSE '*************'
END
   ENABLE  ;

CREATE MASK BANK_SCHEMA.MASK_LOGIN_ID_ON_CUSTOMERS ON BANK_SCHEMA.CUSTOMERS AS C
   FOR COLUMN CUSTOMER_LOGIN_ID
   RETURN  CASE
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'ADMIN' ) = 1
THEN C . CUSTOMER_LOGIN_ID
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
THEN C . CUSTOMER_LOGIN_ID
ELSE '*****'
END
   ENABLE  ;

CREATE MASK BANK_SCHEMA.MASK_SECURITY_QUESTION_ON_CUSTOMERS ON BANK_SCHEMA.CUSTOMERS AS C
   FOR COLUMN CUSTOMER_SECURITY_QUESTION
   RETURN  CASE
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'ADMIN' ) = 1
THEN C . CUSTOMER_SECURITY_QUESTION
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
THEN C . CUSTOMER_SECURITY_QUESTION
ELSE '*****'
END
   ENABLE  ;

CREATE MASK BANK_SCHEMA.MASK_SECURITY_QUESTION_ANSWER_ON_CUSTOMERS ON BANK_SCHEMA.CUSTOMERS AS C
   FOR COLUMN CUSTOMER_SECURITY_QUESTION_ANSWER
   RETURN  CASE
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'ADMIN' ) = 1
THEN C . CUSTOMER_SECURITY_QUESTION_ANSWER
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
THEN C . CUSTOMER_SECURITY_QUESTION_ANSWER
ELSE '*****'
END
   ENABLE  ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
   ACTIVATE ROW ACCESS CONTROL
   ACTIVATE COLUMN ACCESS CONTROL ;
```

Row and Column Access Control Support in IBM DB2 for i

Implement roles and separation of duties

Leverage row permissions on the database

Protect columns by defining column masks

This IBM Redpaper publication provides information about the IBM i 7.2 feature of IBM DB2 for i Row and Column Access Control (RCAC). It offers a broad description of the function and advantages of controlling access to data in a comprehensive and transparent way. This publication helps you understand the capabilities of RCAC and provides examples of defining, creating, and implementing the row permissions and column masks in a relational database environment.

This paper is intended for database engineers, data-centric application developers, and security officers who want to design and implement RCAC as a part of their data control and governance policy. A solid background in IBM i object level security, DB2 for i relational database concepts, and SQL is assumed.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

REDP-5110-00