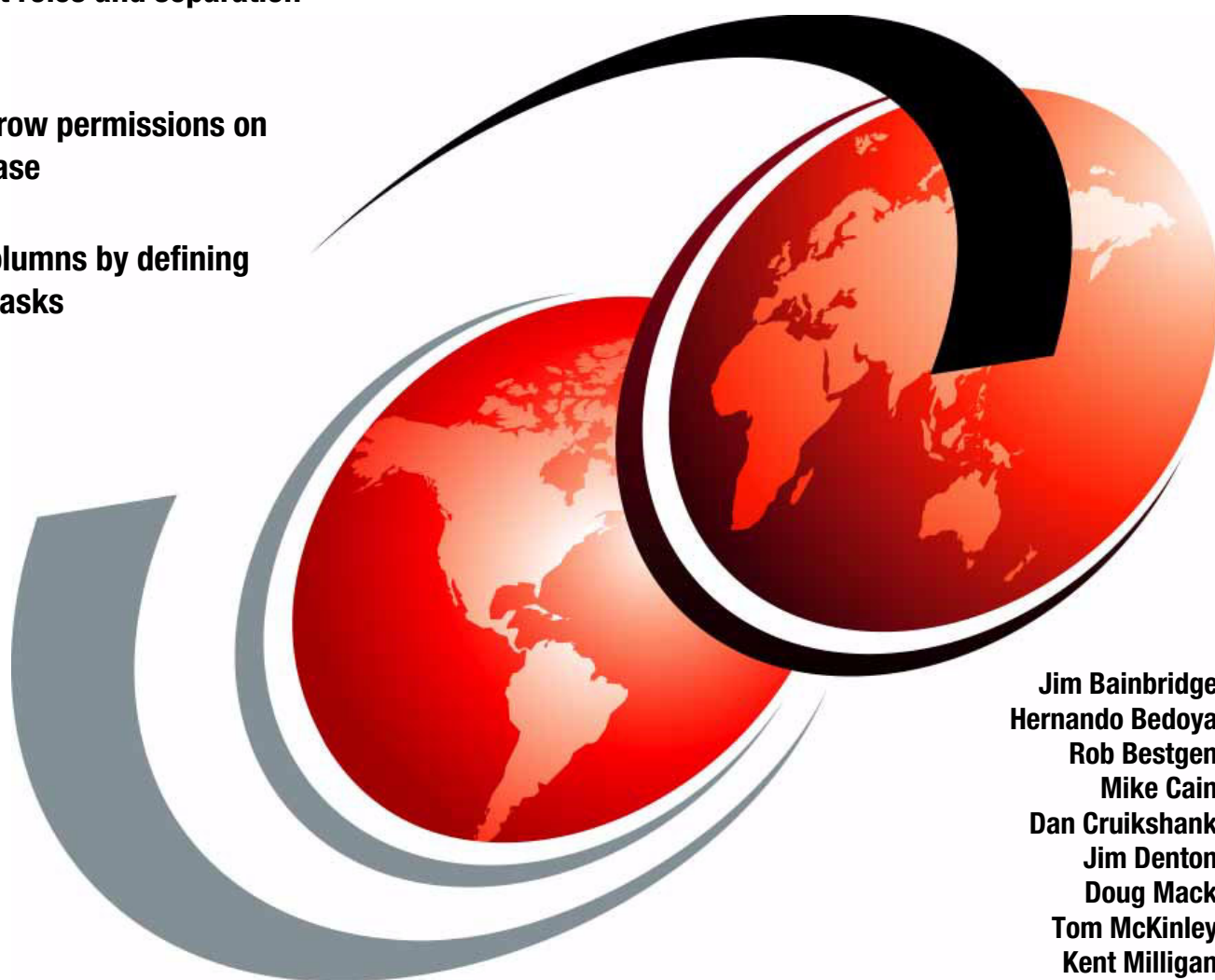


Row and Column Access Control Support in IBM DB2 for i

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



International Technical Support Organization

**Row and Column Access Control Support in IBM DB2
for i**

November 2014

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (November 2014)

This edition applies to Version 7, Release 2 of IBM i (product number 5770-SS1).

© Copyright International Business Machines Corporation 2014. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
DB2 for i Center of Excellence	ix
Preface	xi
Authors	xi
Now you can become a published author, too!	xiii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiv
Chapter 1. Securing and protecting IBM DB2 data	1
1.1 Security fundamentals	2
1.2 Current state of IBM i security	2
1.3 DB2 for i security controls	3
1.3.1 Existing row and column control	4
1.3.2 New controls: Row and Column Access Control	5
Chapter 2. Roles and separation of duties	7
2.1 Roles	8
2.1.1 DDM and DRDA application server access: QIBM_DB_DDMDRDA	8
2.1.2 Toolbox application server access: QIBM_DB_ZDA	8
2.1.3 Database Administrator function: QIBM_DB_SQLADM	9
2.1.4 Database Information function: QIBM_DB_SYSMON	9
2.1.5 Security Administrator function: QIBM_DB_SECADM	9
2.1.6 Change Function Usage CL command	10
2.1.7 Verifying function usage IDs for RCAC with the FUNCTION_USAGE view	10
2.2 Separation of duties	10
Chapter 3. Row and Column Access Control	13
3.1 Explanation of RCAC and the concept of access control	14
3.1.1 Row permission and column mask definitions	14
3.1.2 Enabling and activating RCAC	16
3.2 Special registers and built-in global variables	18
3.2.1 Special registers	18
3.2.2 Built-in global variables	19
3.3 VERIFY_GROUP_FOR_USER function	20
3.4 Establishing and controlling accessibility by using the RCAC rule text	21
3.5 SELECT, INSERT, and UPDATE behavior with RCAC	22
3.6 Human resources example	22
3.6.1 Assigning the QIBM_DB_SECADM function ID to the consultants	23
3.6.2 Creating group profiles for the users and their roles	23
3.6.3 Demonstrating data access without RCAC	24
3.6.4 Defining and creating row permissions	25
3.6.5 Defining and creating column masks	26
3.6.6 Activating RCAC	28
3.6.7 Demonstrating data access with RCAC	29
3.6.8 Demonstrating data access with a view and RCAC	32

Chapter 4. Implementing Row and Column Access Control: Banking example	37
4.1 Business requirements for the RCAC banking scenario	38
4.2 Description of the users roles and responsibilities	39
4.3 Implementation of RCAC	42
4.3.1 Reviewing the tables that are used in this example	42
4.3.2 Assigning function ID QIBM_DB_SECADM to the Database Engineers group	47
4.3.3 Creating group profiles for the users and their roles	50
4.3.4 Creating the CUSTOMER_LOGIN_ID global variable	52
4.3.5 Defining and creating row permissions	54
4.3.6 Defining and creating column masks	58
4.3.7 Restricting the inserting and updating of masked data	60
4.3.8 Activating row and column access control	63
4.3.9 Reviewing row permissions.	64
4.3.10 Demonstrating data access with RCAC	66
4.3.11 Query implementation with RCAC activated.	75
Chapter 5. RCAC and non-SQL interfaces	79
5.1 Unsupported interfaces	80
5.2 Native query result differences	80
5.3 Accidental updates with masked values	81
5.4 System CL commands considerations	82
5.4.1 Create Duplicate Object (CRTDUPOBJ) command	82
5.4.2 Copy File (CPYF) command	82
5.4.3 Copy Library (CPYLIB) command.	83
Chapter 6. Additional considerations	85
6.1 Timing of column masking	86
6.2 RCAC effects on data movement	88
6.2.1 Effects when RCAC is defined on the source table	88
6.2.2 Effects when RCAC is defined on the target table	89
6.2.3 Effects when RCAC is defined on both source and target tables	90
6.3 RCAC effects on joins	91
6.3.1 Inner joins	92
6.3.2 Outer joins.	94
6.3.3 Exception joins	96
6.4 Monitoring, analyzing, and debugging with RCAC	97
6.4.1 Query monitoring and analysis tools	97
6.4.2 Index advisor.	99
6.4.3 Metadata using catalogs	100
6.5 Views, materialized query tables, and query rewrite with RCAC	102
6.5.1 Views	102
6.5.2 Materialized query tables	103
6.5.3 Query rewrite	105
6.6 RCAC effects on performance and scalability.	105
6.7 Exclusive lock to implement RCAC (availability issues)	107
6.8 Avoiding propagation of masked data	108
6.8.1 Check constraint solution	108
6.8.2 Before trigger solution.	109
6.9 Triggers and functions (SECURED)	109
6.9.1 Triggers.	109
6.9.2 Functions	110
6.10 RCAC is only one part of the solution	111
Chapter 7. Row and Column Access Control management	113

7.1 Managing row permissions and column masks.	114
7.1.1 Source management.	114
7.1.2 Modifying definitions	114
7.1.3 Turning on and off.	114
7.1.4 Regenerating	114
7.2 Managing tables with row permissions and column masks.	115
7.2.1 Save and restore.	115
7.2.2 Table migration	116
7.3 Monitoring and auditing function usage	117
Chapter 8. Designing and planning for success	119
8.1 Implementing RCAC with good design and proper planning	120
8.2 DB2 for i Center of Excellence	120
Appendix A. Database definitions for the RCAC banking example	121
Related publications	127
Other publications	127
Online resources	127
Help from IBM	128

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.

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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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.

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.


Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AS/400®
DB2®
DRDA®

IBM®
Power Systems™
Redbooks®

Redpaper™
Redbooks (logo) ®
System i®

The following terms are trademarks of other companies:

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



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

DB2 for i Center of Excellence

Expert help to achieve your business requirements

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



What you can expect

Depending on the engagement, our team of consultants offer:

- Briefings, consulting and guidance on demand
- Illumination of the DB2 for i capabilities and leadership to exploit them
- Analysis and remediation of performance and scalability issues caused by inefficient database design and implementation
- Configuration of systems, operating system and products to fully leverage database capabilities

Key client benefits

Gain greater database and application performance within your current environment. Achieve greater productivity in the development and maintenance of database and applications using modern techniques. Architect and design data structures to accommodate and benefit from business analytics (BA) tools and processes.

For more information

Pricing depends on the scope of work. Learn more about the DB2 for i Center of Excellence and other related products and services. Contact stgls@us.ibm.com or visit:

ibm.com/systems/services/labservices



© Copyright IBM Corporation 2013

IBM Corporation
Route 100
Somers, NY 10589

Produced in the United States of America
March 2013

IBM, the IBM logo, ibm.com, DB2 and Power Systems are 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" at www.ibm.com/legal/copytrade.shtml.

This document is current as of the initial date of publication and may be changed by IBM at any time.

Not all offerings are available in every country in which IBM operates.



Please Recycle

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.



Rob Bestgen is a member of the DB2 for i Center of Excellence team helping customers use the capabilities of DB2 for i. In addition, Rob is the chief architect of the DB2 SQL Query Engine (SQE) for DB2 for i and is the product development manager for DB2 Web Query for i.



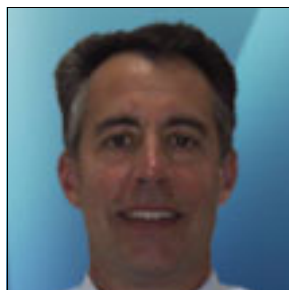
Mike Cain is a Senior Technical Staff Member within the IBM Systems and Technology Group. He is also the founder and team leader of the DB2 for i Center of Excellence in Rochester, Minnesota US. Before his current position, he worked as an IBM AS/400 Systems Engineer and technical consultant. Before joining IBM in 1988, Mike worked as a System/38 programmer and data processing manager for a property and casualty insurance company. Mike has 26 years of experience with IBM, engaging clients and Business Partners around the world. In addition to assisting clients, he uses his knowledge and experience to influence the IBM solution, development, and support processes.



Dan Cruikshank has been an IT Professional since 1972. He has consulted on a number of different project areas since joining IBM Rochester in 1988. Since 1993, Dan was focused primarily on resolving IBM System i® application and database performance issues at several IBM customer accounts. Since 1998, Dan has been one of the primary instructors for the Database Optimization Workshop. Most recently, Dan is a member of the DB2 for i Center of Excellence team with IBM Rochester Lab Services.



Jim Denton is a senior consultant at the IBM DB2 for i Center of Excellence, where his responsibilities include both teaching courses and hands on consulting. Jim specializes in SQL performance, data-centric programming, and database modernization. Jim started his IBM career in 1981 as an S/38 operating system programmer. Before joining the consulting team, his key assignments included 10 years as a systems performance specialist, five years as the lead “JDE on i” analyst, three years as a consultant at the IBM Benchmark and Briefing Center in Montpellier France, and a total of 11 years as an operating system developer, including five years designing and implementing enhancements to DB2 for i.



Doug Mack is a DB2 for i and Business Intelligence Consultant in the IBM Power Systems™ Lab Services organization. Doug's 30+ year career with IBM spans many roles, including product development, technical sales support, Business Intelligence Sales Specialist, and DB2 for i Product Marketing Manager. Doug is a featured speaker at User Group conferences and meetings, IBM Technical Conferences, and Executive Briefings.



Tom McKinley is an IBM Lab Services Consultant working on DB2 for IBM i in Rochester MN. His main focus is complex query performance that is associated with Business Intelligence running on Very Large Databases. He worked as a developer or performance analyst in the DB area from 1986 until 2006. Some of his major pieces of work include the Symmetric Multiple processing capabilities of DB2 for IBM i and Large Object Data types. In addition, he was on the original team that designed and built the SQL Query Engine. Before his database work, he worked on Licensed Internal Code for System 34 and System 36.



Kent Milligan is a senior DB2 consultant on the DB2 for i Center of Excellence team within the IBM Lab Services and Training organization. His primary responsibility is helping software developers use the latest DB2 technologies and port applications from other databases to DB2 for i. After graduating from the University of Iowa, Kent spent the first eight years of his IBM career as a member of the DB2 development team in Rochester.

Thanks to the following people for their contributions to this project:

Debra Landon

International Technical Support Organization, Rochester Center

Craig Aldrich, Mark Anderson, Theresa Euler, Scott Forstie, Chad Olstad
IBM Rochester Development

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



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 Center¹ 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 Institute² 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.

Some clients using this default configuration have toughened their database security with exit-point solutions from third-party vendors. IBM i exit points allow a user-written program to be called every time that a particular interface (for example, FTP) is used or an event occurs (for example, a profile is created). Security tools that are based on these exit points increase the level of security on a system by locking down interfaces that are not under the control of menu-based or application authority. In addition, exit-point solutions allow clients to implement more granular security controls, such as allowing users access only to the database during certain hours of the day.

Although exit-point solutions can provide great benefits, they are not an alternative to object-level control of your databases. Exit-point solutions help secure interfaces, but they do not completely protect the data that is stored in your DB2 objects. Exit points do not exist for every data access interface on the system. Therefore, if an application starts using an unprotected interface, the only thing protecting your data is object-level access control. When your security implementation totally relies on exit points, then it is also important to track any new data interfaces that appear as IBM delivers new releases and products to ensure that your exit-point solution provides coverage for those new interfaces.

An exit-point solution is a good option for databases with security holes that are caused by a reliance on the default security setup or menu-based control. However, your security work should not stop there. Instead, you must continue to work on a complete database security solution by controlling data access at the object level.

1.3 DB2 for i security controls

As described in 1.2, “Current state of IBM i security” on page 2, object-level controls on your DB2 objects are a critical success factor in securing your business data. Although database object-level security is a strong security feature, some clients have found that object-level security does not have the granularity that is required to adhere to regulatory or compliance policies. A user that is granted object-level access to a DB2 table has the authority to view all of the rows and values in that table.

As shown in Figure 1-1, it is an all-or-nothing access to the rows of a table.

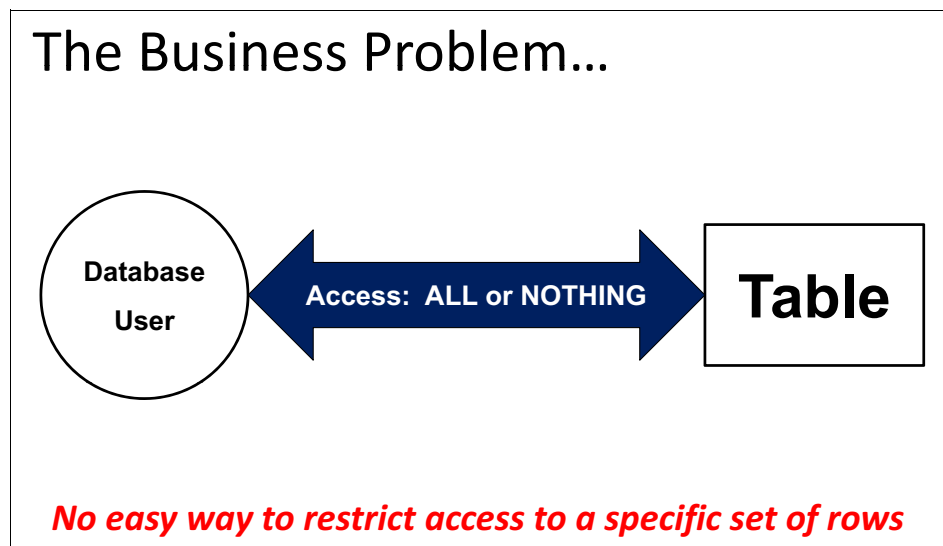


Figure 1-1 All-or-nothing access to the rows of a table

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.

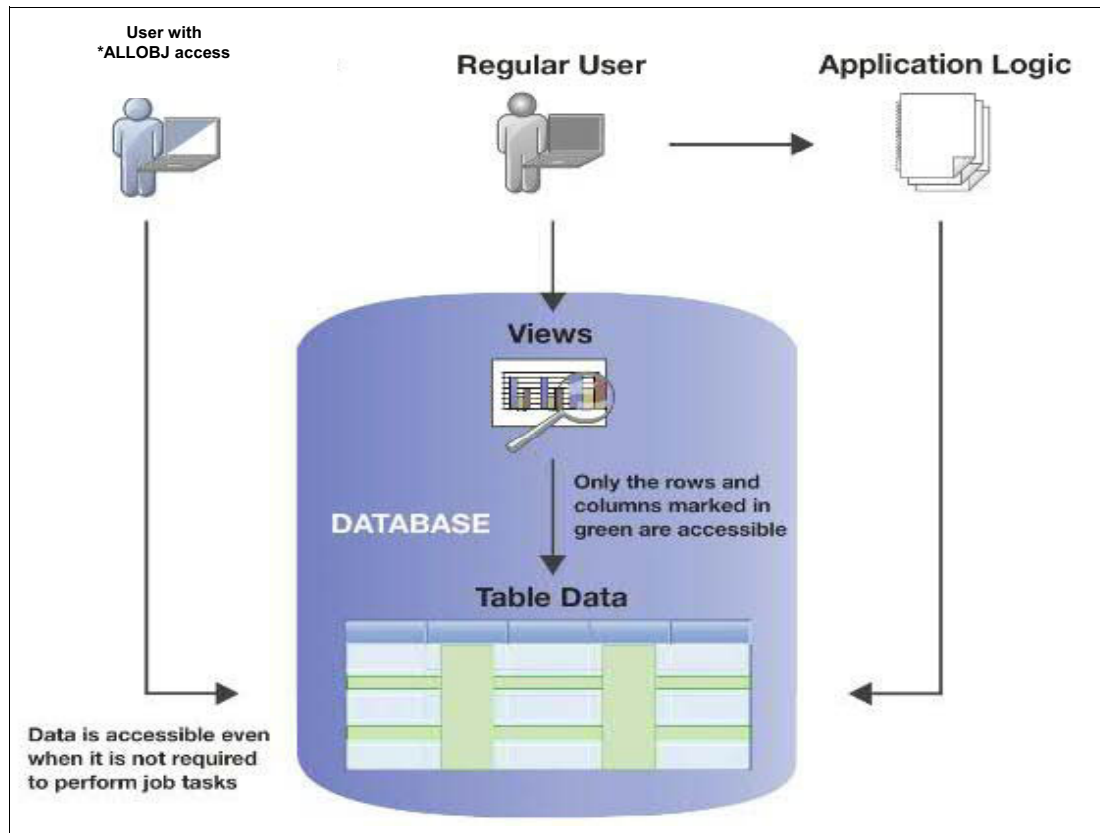


Figure 1-2 Existing row and column controls

1.3.2 New controls: Row and Column Access Control

Based on the challenges that are associated with the existing technology available for controlling row and column access at a more granular level, IBM delivered new security support in the IBM i 7.2 release; this support is known as Row and Column Access Control (RCAC).

The new DB2 RCAC support provides a method for controlling data access across all interfaces and all types of users with a data-centric solution. Moving security processing to the database layer makes it easier to build controls that meet your compliance policies. The RCAC support provides an additional layer of security that complements object-level authorizations to limit data access to a need-to-know basis. Therefore, it is critical that you first have a sound object-level security implementation in place.



Roles and separation of duties

One of the primary objectives of row and column access control (RCAC) is to create data security policies that control and govern user access to data and limit the data access of DB2 designers and administrators to only the minimum that is required to do their jobs.

To accomplish these tasks, RCAC engineers devised a set of functional roles that, as a group, implement effectively data access requirements and also limit the span of control of each role so that each role is given only the authorities that are needed to perform its specific set of tasks.

This chapter describes the concepts of roles and separation of duties on DB2 for i and covers the following topics:

- ▶ Roles
- ▶ Separation of duties

2.1 Roles

Traditionally, data access roles are defined in a binary way, where access to the data is either not permitted or access to the data is permitted. A full access capability can also be instantiated by the *ALLOBJ special authority, either explicitly or implicitly, for the security officer. If you hold the role of security officer, or have all *ALLOBJ special authority, you have access to all the data, with no exceptions. Unfortunately, this might not meet the organization's requirements for limiting access to data or separation of duties.

To assist with defining roles and the separation of duties with appropriate authority, IBM i provides *function usage IDs*. A function usage ID implements granular security controls rather than granting users powerful special authorities, such as all object, job control, or service.

Roles are divided among the following DB2 functions and their corresponding function usage IDs:

- ▶ DDM and IBM DRDA® application server access: QIBM_DB_DDMDRDA
- ▶ Toolbox application server access: QIBM_DB_ZDA
- ▶ Database Administrator function: QIBM_DB_SQLADM
- ▶ Database Information function: QIBM_DB_SYSMON
- ▶ Security Administrator function: QIBM_DB_SECADM

2.1.1 DDM and DRDA application server access: QIBM_DB_DDMDRDA

The QIBM_DB_DDMDRDA function usage ID restricts access to the DDM and DRDA application server (QRWTSRVR). This function usage ID provides an easy alternative (rather than writing an exit program) to control access to DDM and DRDA from the server side. The function usage IDs ship with the default authority of *ALLOWED. The security officer can easily deny access to specific users or groups.

This is an alternative to a User Exit Program approach. No coding is required, it is easy to change, and it is auditable.

2.1.2 Toolbox application server access: QIBM_DB_ZDA

The QIBM_DB_ZDA function usage ID restricts access to the optimized server that handles DB2 requests from clients (QZDASOINIT and QZDASSINIT). Server access is used by the ODBC, OLE DB, and .NET providers that ship with IBM i Access for Windows and JDBC Toolbox, Run SQL scripts, and other parts of System i Navigator and Navigator for i Web console.

This function usage ID provides an easy alternative (rather than writing an exit program) to control access to these functions from the server side. The function usage IDs ship with the default authority of *ALLOWED. The security officer can easily deny access to specific users or groups.

This is an alternative to a User Exit Program approach. No coding is required, it is easy to change, and it is auditable.

2.1.3 Database Administrator function: QIBM_DB_SQLADM

The Database Administrator function (QIBM_DB_SQLADM) is needed whenever a user is analyzing and viewing SQL performance data. Some of the more common database administrator functions include 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 provides an alternative to granting *JOBCTL, but simply having the Database Administrator authorization does not carry with it all the needed object authorities for every administration task. The default behavior is to deny authorization.

To perform database administrator tasks that are not related to performance analysis, you must refer to the details of the task to determine its specific authorization requirements. For example, to allow a database administrator to reorganize a table, the DBA must have additional object authorities to the table that are not covered by QIBM_DB_SQLADM.

Granting QIBM_DB_SQLADM function usage

Only the security administrator (*SECADM) is allowed to change the list of users that can perform Database Administration functions.

2.1.4 Database Information function: QIBM_DB_SYSMON

The Database Information function (QIBM_DB_SYSMON) provides much less authority than Database Administrator function. Its primary use allows a user to examine high-level database properties.

For example, a user that does not have *JOBCTL or QIBM_DB_SQLADM can still view the SQL Plan Cache properties if granted authority to QIBM_DB_SYSMON. Without granting this authority, the default behavior is to deny authorization.

Granting QIBM_DB_SYSMON function usage

Only the security administrator (*SECADM) is allowed to change the list of users that can perform Database Information functions.

2.1.5 Security Administrator function: QIBM_DB_SECADM

The Security Administrator function (QIBM_DB_SECADM) 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.

Only those users with the QIBM_DB_SECADM function can administer and manage RCAC rules. RCAC can be used to prevent even users with *ALLOBJ authority from freely accessing all the data in a protected database. These users are excluded from data access unless they are specifically authorized by RCAC. Without granting this authority, the default behavior is to deny authorization.

Granting QIBM_DB_SECADM function usage

Only QSECOFR or a user with *SECADM special authority can grant the QIBM_DB_SECADM function usage to a user or group.

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: <ul style="list-style-type: none">▶ 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: <ul style="list-style-type: none">▶ 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		

User action	*JOBCTL	QIBM_DB_SECADM	QIBM_DB_SQLADM	QIBM_DB_SYSMON	No Authority
START PLAN CACHE EVENT MONITOR procedure	X		X		
END PLAN CACHE EVENT MONITOR procedure	X		X		
END ALL PLAN CACHE EVENT MONITORS procedure	X		X		
Work with RCAC row permissions (Create, modify, or delete)		X			
Work with RCAC column masks (Create, modify, or delete)		X			
Change Object Owner (CHGOBJOWN) CL command		X			
Change Object Primary Group (CHGOBJPGP) CL command		X			
Grant Object Authority (GRTOBJAUT) CL command		X			
Revoke Object Authority (RVKOBJAUT) CL command		X			
Edit Object Authority (EDTOBJAUT) CL command		X			
Display Object Authority (DSPOBJAUT) CL command		X			
Work with Objects (WRKOBJ) CL command		X			
Work with Libraries (WRKLIB) CL command		X			
Add Authorization List Entry (ADDAUTLE) CL command		X			
Change Authorization List Entry (CHGAUTLE) CL command		X			
Remove Authorization List Entry (RMVAUTLE) CL command		X			
Retrieve Authorization List Entry (RTVAUTLE) CL command		X			
Display Authorization List (DSPAUTL) CL command		X			
Display Authorization List Objects (DSPAUTOBJ) CL command		X			
Edit Authorization List (EDTAUTL) CL command		X			
Work with Authorization Lists (WRKAUTL) CL command		X			



Row and Column Access Control

This chapter describes what Row and Column Access Control (RCAC) is, its components, and then illustrates RCAC with a simple example.

The following topics are covered in this chapter:

- ▶ Explanation of RCAC and the concept of access control
- ▶ Special registers and built-in global variables
- ▶ `VERIFY_GROUP_FOR_USER` function
- ▶ Establishing and controlling accessibility by using the RCAC rule text
- ▶ `SELECT`, `INSERT`, and `UPDATE` behavior with RCAC
- ▶ Human resources example

3.1 Explanation of RCAC and the concept of access control

RCAC limits data access to those users who have a business “need to know”. RCAC makes it easy to set up a rich and robust security policy that is based on roles and responsibilities. RCAC functionality is made available through the optional, no charge feature called “IBM Advanced Data Security for i”, also known as option 47 of IBM i 7.2.

In DB2 for i, RCAC is implemented using two different approaches that address the shortcomings of traditional control methods and mechanisms:

- ▶ Row permissions
- ▶ Column masks

Another benefit of RCAC is that no database user is automatically exempt from the control. Users with *ALLOBJ authority can no longer freely access all of the data in the database unless they have the appropriate permission to do so. The ability to manage row permissions and column masks rests with the database security administrator. The RCAC definitions, enablement, and activation are controlled by SQL statements.

Row permissions and column masks require virtually no application changes. RCAC is based on specific rules that are transparent to existing applications and SQL interfaces. Enforcement of your security policy does not depend on how applications or tools access the data.

RCAC also facilitates multi-tenancy, which means that several independent customers or business units can share a single database table without being aware of one another. The RCAC row permission ensures each user sees only the rows they are entitled to view because the enforcement is handled by DB2 and not the application logic.

Label-based access control (LBAC): RCAC and LBAC are not the same thing. LBAC is a security model that is primarily intended for government applications. LBAC requires that data and users be classified with a fixed set of rules that are implemented. RCAC is a general-purpose security model that is primarily intended for commercial customers. You can use RCAC to create your own security rules, which in turn allows for more flexibility.

3.1.1 Row permission and column mask definitions

The following sections define row permission and column masks.

Row permission

A row permission is a database object that manifests a row access control rule for a specific table. It is essentially a search condition that describes which rows you can access. For example, a manager can see only the rows that represent his or her employees.

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.

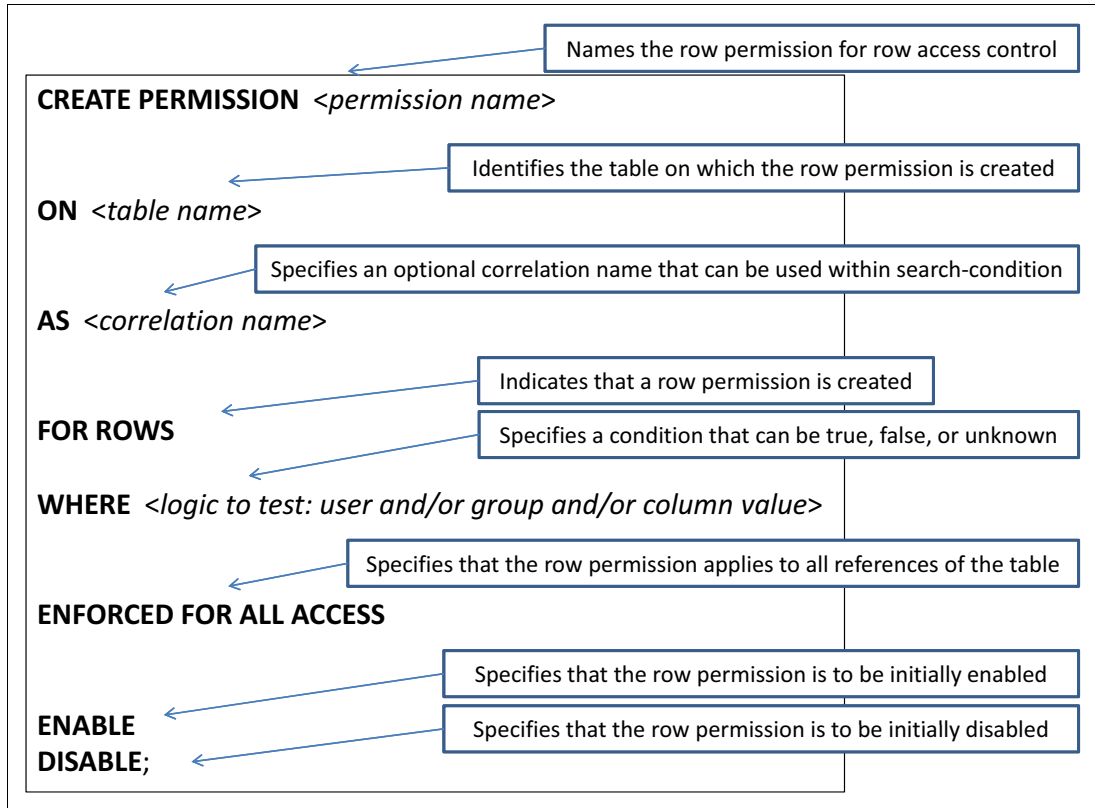


Figure 3-1 *CREATE PERMISSION SQL statement*

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.

Column masks replace the need to create and use views to implement access control. The SQL `CREATE MASK` statement that is shown in Figure 3-2 is used to define and initially enable or disable the column value access rules.

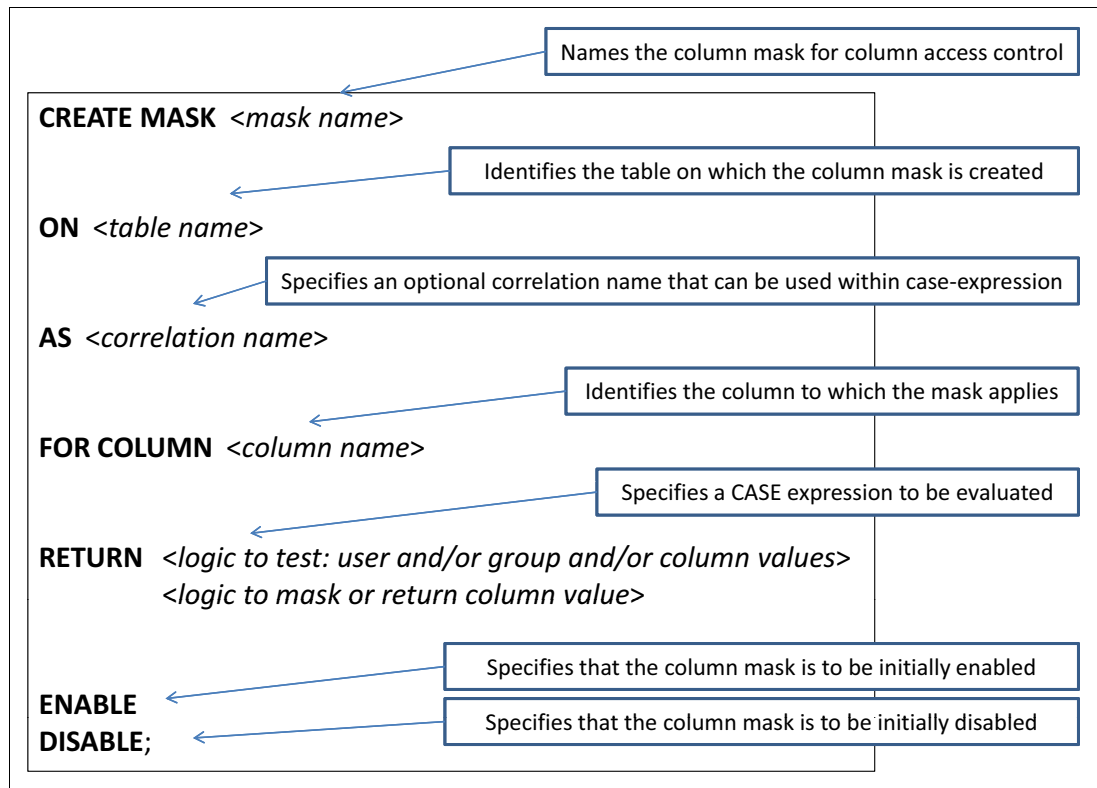


Figure 3-2 `CREATE MASK` SQL statement

3.1.2 Enabling and activating RCAC

You can enable, disable, or regenerate row permissions and column masks by using the SQL `ALTER PERMISSION` statement and the SQL `ALTER MASK` statement, as shown in Figure 3-3 on page 17.

Enabling and disabling effectively turns on or off the logic that is contained in the row permission or column mask. Regenerating causes the row permission or column mask to be regenerated. The row permission definition in the catalog is used and existing dependencies and authorizations, if any, are retained. The row permission definition is reevaluated as though the row permission were being created. Any user-defined functions (UDFs) that are referenced in the row permission must be resolved to the same secure UDFs as were resolved during the original row permission or column mask creation. The regenerate option can be used to ensure that the RCAC logic is intact and still valid before any user attempts to access the table.

Note: An exclusive lock is required on the table object to perform the alter operation. All open cursors must be closed.

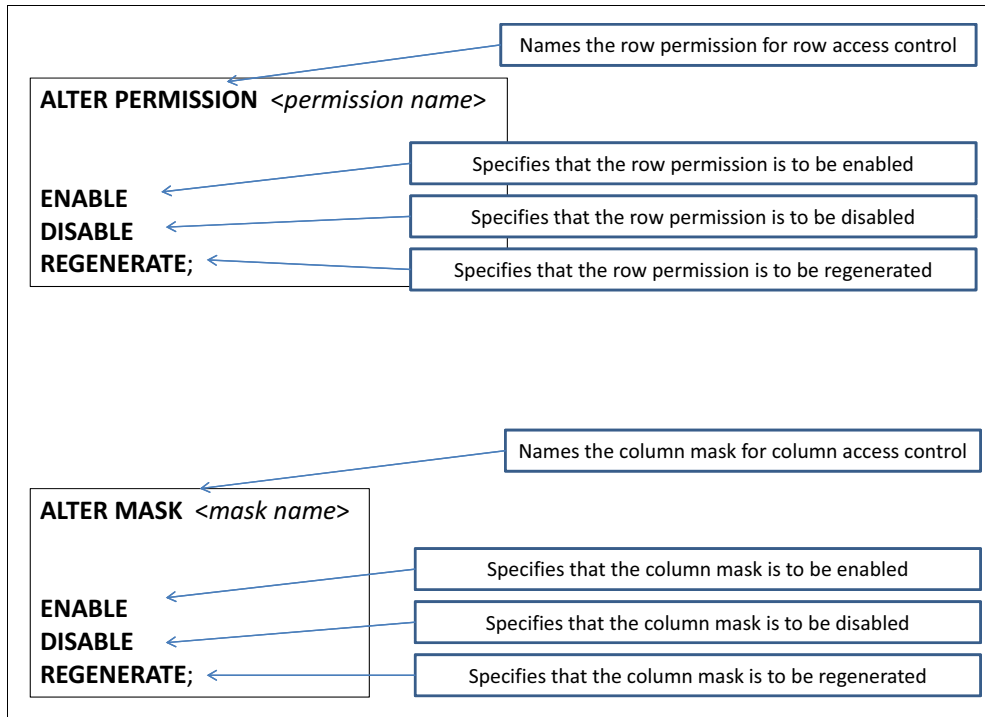


Figure 3-3 ALTER PERMISSION and ALTER MASK SQL statements

You can activate and deactivate RCAC for new or existing tables by using the SQL **ALTER TABLE** statement (Figure 3-4). The **ACTIVATE** or **DEACTIVATE** clause must be the option that is specified in the statement. No other alterations are permitted at the same time. The activating and deactivating effectively turns on or off all RCAC processing for the table. Only enabled row permissions and column masks take effect when activating RCAC.

Note: An exclusive lock is required on the table object to perform the alter operation. All open cursors must be closed.

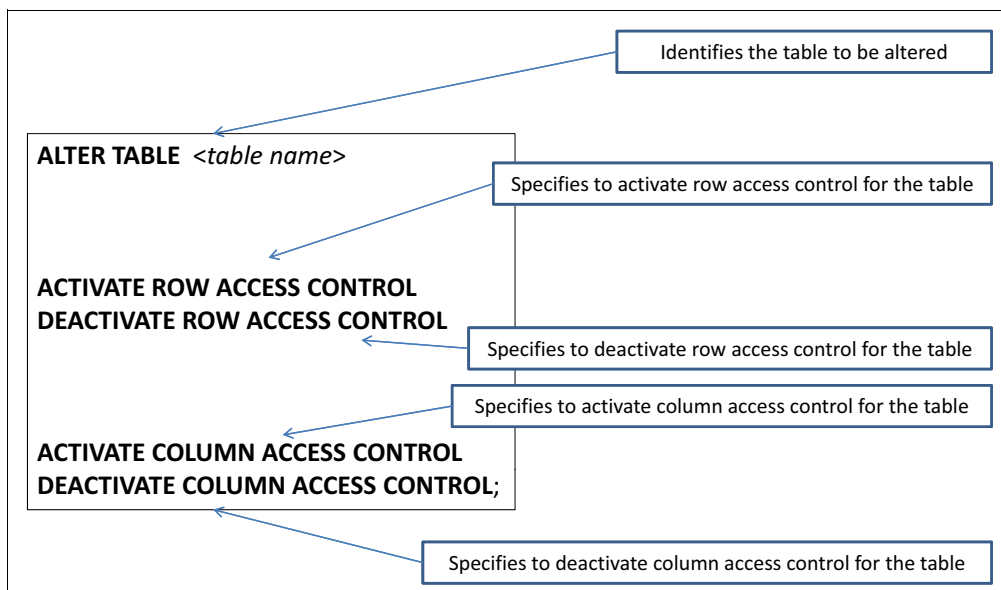


Figure 3-4 ALTER TABLE SQL statement

When row access control is activated on a table, a default permission is established for that table. The name of this permission is QIBM_DEFAULT_ <table-name>_<schema-name>. This default permission contains a simple piece of logic (0=1) which is never true. The default permission effectively denies access to every user unless there is a permission defined that allows access explicitly. If row access control is activated on a table, and there is no permission that is defined, no one has permission to any rows. All queries against the table produce an empty set.

It is possible to define, create, and enable multiple permissions on a table. Logically, all of the permissions are ORed together to form a comprehensive test of the user's ability to access the data. A column can have only one mask that is defined over it. From an implementation standpoint, it does not matter if you create the column masks first or the row permissions first.

Note: If a user does not have permission to access the row, the column mask logic is not invoked.

3.2 Special registers and built-in global variables

This section describes how you can use special registers and built-in global variables to implement RCAC.

3.2.1 Special registers

A special register is a storage area that is defined for an application process by DB2 and is used to store information that can be referenced in SQL statements. A reference to a special register is a reference to a value that is provided by the current server.

IBM DB2 for i supports four different special registers that can be used to identify what user profiles are relevant to determining object authorities in the current connection to the server. SQL uses the term *runtime authorization ID*, which corresponds to a user profile on DB2 for i. Here are the four special registers:

- ▶ *USER* is the runtime user profile that determines the object authorities for the current connection to the server. It has a data type of VARCHAR(18). This value can be changed by the SQL statement **SET SESSION AUTHORIZATION**.
- ▶ *SESSION_USER* is the same as the USER register, except that it has a data type of VARCHAR(128).
- ▶ *CURRENT_USER* was added in IBM i 7.2 and is similar to the USER register, but it has one important difference in that it also reports adopted authority. High-level language programs and SQL routines such as functions, procedures, and triggers can optionally be created to run using either the caller's or the owner's user profile to determine data authorities. For example, an SQL procedure can be created to run under the owner's authority by specifying **SET OPTION USRPRF=*OWNER**. This special register can also be referenced as CURRENT_USER. It has a data type of VARCHAR(128).
- ▶ *SYSTEM_USER* is the user profile that initiates the connection to the server. It is not used by RCAC, but is included here for completeness. Many jobs, including the QZDASOINIT prestared jobs, initially connect to the server with a default user profile and then change to use some other user profile. SYSTEM_USER reports this value, typically QUSER for a QZDASOINIT job. It has a data type of VARCHAR(128).

In addition to these four special registers, any of the DB2 special registers can be referenced as part of the rule text.

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.

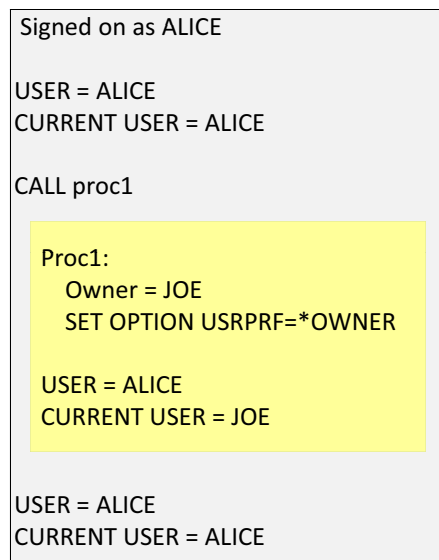


Figure 3-5 Special registers and adopted authority

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')
```

3.4 Establishing and controlling accessibility by using the RCAC rule text

When defining a row permission or column mask, the “magic” of establishing and controlling accessibility comes from the *rule text*. The rule text represents the search criteria and logic that is implemented by the database engine.

In the case of a row permission, the rule text is the “test” of whether the user can access the row. If the test result is true, the row can be accessed. If the test result is false, the row essentially does not exist for the user. From a set-at-a-time perspective, the permission defines which rows can be part of the query result set, and which rows cannot.

In the case of a column mask, the rule text is both the test of whether the user can see the actual column value, and it is the masking logic if the user cannot have access to actual column value.

For a simple example of implementing row permissions and column masks, see 3.6, “Human resources example” on page 22.

In general, almost any set-based, relational logic is valid. For the row permission, the search condition follows the same rules that are used by the search condition in a WHERE clause.

For the column mask, the logic follows the same rules as the CASE expression. The result data type, length, null attribute, and CCSID of the CASE expression must be compatible with the data type of the column. If the column does not allow the null value, the result of the CASE expression cannot be the NULL value. The application or interface making the data access request is expecting that all of the column attributes and values are consistent with the original definition, regardless of any masking.

For more information about what is permitted, see the “Database programming” topic of the IBM i 7.2 Knowledge Center, found at:

http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzahg/rzahgdbp.htm?lang=en

One of the first tasks in either the row permission or the column mask logic is to determine who the user is, and whether they have access to the data. Elegant methods to establish the identity and attributes of the user can be employed by using the special registers, global variables, and the VERIFY function. After the user's identity is established, it is a simple matter of allowing or disallowing access by using true or false testing. The examples that are included in this paper demonstrate some of the more common and obvious techniques.

More sophisticated methods can employ existential, day of year / time of day, and relational comparisons with set operations. For example, you can use a date master or date dimension table to determine whether the current date is a normal business day. If the current date is a valid business day, then access is allowed. If the current date is not a business day (for example a weekend day or holiday), access is denied. This test can be accomplished by performing a lookup using a subquery, such as the one that is shown in Example 3-1.

Example 3-1 Subquery that is used as part of the rule

```
CURRENT_DATE IN (SELECT  D.DATE_KEY
                    FROM    DATE_MASTER D
                    WHERE   D.BUSINESS_DAY = 'Y')
```

Given that joins and subqueries can be used to perform set-based operations against existing data that is housed in other objects, almost any relational test can be constructed. If the data in the objects is manipulated over time, the RCAC test logic (and user query results) can be changed without modifying the actual row permission or column mask. This includes moving a user from one group to another or changing a column value that is used to allow or disallow access. For example, if Saturday is now a valid business day, only the BUSINESS_DAY value in the DATE_MASTER must be updated, not the permission logic. This technique can potentially avoid downtime because of the exclusive lock that is needed on the table when adding or changing RCAC definitions.

3.5 SELECT, INSERT, and UPDATE behavior with RCAC

RCAC provides a database-centric approach to determining which rows can be accessed and what column values can be seen by a specific user. Given that the control is handled by DB2 internally, every data manipulation statement is under the influence of RCAC, with no exceptions. When accessing the table, the **SELECT** statements, searched **UPDATE** statements, and searched **DELETE** statements implicitly and transparently contain the row permission and the column mask rule text. This means that the data set can be logically restricted and reduced on a user by user basis.

Furthermore, DB2 prevents an **INSERT** statement from inserting a row or an **UPDATE** statement from modifying a row such that the current user cannot be permitted to access it. You cannot create a situation in which the data you inserted or changed is no longer accessible to you.

For more information and considerations about data movement in an RCAC environment, see Chapter 6, “Additional considerations” on page 85.

Note: DB2 does not provide any indication back to the user that the data set requested was restricted or reduced by RCAC. This is by design, as it helps minimize any changes to the applications accessing the data.

3.6 Human resources example

This section illustrates with a simple example the usage of RCAC on a typical Human Resources application (schema). In this sample Human Resources schema, there is an important table that is called EMPLOYEES that contains all the information that is related to the employees of the company. Among the information that normally is stored in the EMPLOYEES table, there is some sensitive information that must be hidden from certain users:

- ▶ Tax_Id information
- ▶ YEAR of the birth date of the employee (hiding the age of the employee)

In this example, there are four different types of users:

- ▶ Employees
- ▶ Managers
- ▶ Human Resources Manager
- ▶ Consultant/IT Database Engineer (In this example, this person is an external consultant that is not an employee of the company.)

The following sections describe step-by-step what is needed to be done to implement RCAC in this environment.

3.6.1 Assigning the QIBM_DB_SECADM function ID to the consultants

The consultant must have authority to implement RCAC, so you must use one of the function IDs that are provided in DB2 for i (see 2.1.5, “Security Administrator function: QIBM_DB_SECADM” on page 9). Complete the following steps:

1. Run the Change Functional Usage (**CHGFCNUSG**) CL commands that are shown in Example 3-2. These commands must be run by someone that has the *SECOFR authority.

Example 3-2 Function ID required to implement RCAC

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

2. There is a way to discover which user profiles have authorization to implement RCAC. This can be done by running the SQL statement that is shown in Example 3-3.

Example 3-3 Verifying what user profiles have authorization to implement RCAC

```
SELECT      function_id,
            user_name,
            usage,
            user_type
FROM        qsys2.function_usage
WHERE       function_id ='QIBM_DB_SECADM'
ORDER BY   user_name;
```

3. The result of the SQL statement is shown in Figure 3-6. In this example, either MCAIN or HBEDOYA can implement RCAC in the Human Resources database.

FUNCTION_ID	USER_NAME	USAGE	USER_TYPE
QIBM_DB_SECADM	HBEDOYA	ALLOWED	USER
QIBM_DB_SECADM	MCAIN	ALLOWED	USER

Figure 3-6 Result of the function ID query

3.6.2 Creating group profiles for the users and their roles

Assuming that all the employees have a valid user profile, the next step is to create group profiles to group the employees. Complete the following steps:

1. In this example, there are three group profiles:
 - HR (Human Resource personnel)
 - MGR (Managers)
 - EMP (Employees)

These are created by creating user profiles with no password. Example 3-4 shows the Create User Profile (**CRTUSRPRF**) CL commands that you use to create these group profiles.

Example 3-4 Creating group profiles

```
CRTUSRPRF USRPRF(EMP) PASSWORD() TEXT('Employees Group')
CRTUSRPRF USRPRF(MGR) PASSWORD() TEXT('Managers Group')
CRTUSRPRF USRPRF(HR) PASSWORD() TEXT('Human Resources Group')
```

2. You now must assign users to a group profile. Employees go in to the EMP group profile, Managers go into the MGR group profile, and Human Resource employees go into the HR group profile. For simplicity, this example selects one employee (DSSMITH), one manager (TQSPENSER), and one HR analyst (VGLUCCHESS).

Note: Neither of the consultants (MCAIN and HBEDOYA) belong to any group profile.

3.6.3 Demonstrating data access without RCAC

Before implementing RCAC, run some simple SQL statements to demonstrate data access without RCAC. Complete the following steps:

1. The first SQL statement, which is shown in Example 3-5, basically counts the total number of rows in the EMPLOYEES table.

Example 3-5 Counting the number of employees

```
SELECT COUNT(*) as ROW_COUNT FROM HR_SCHEMA.EMPLOYEES;
```

The result of this query is shown in Figure 3-7, which is the total number of employees of the company.

ROW_COUNT
42

Figure 3-7 Number of employees

2. Run a second SQL statement (shown in Example 3-6) that lists the employees. If you have read access to the table, you see all the rows no matter who you are.

Example 3-6 Displaying the information of the Employees

```
SELECT      EMPLOYEE_ID,  
           LAST_NAME,  
           JOB_DESCRIPTION,  
           DATE_OF_BIRTH,  
           TAX_ID,  
           USER_ID,  
           MANAGER_OF_EMPLOYEE  
FROM        HR_SCHEMA.EMPLOYEES
```

The result of this query is shown in Figure 3-8.

EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
000010	HAAS	MANAGER	1933-08-24	123-45-3978	CIHAAS	MLTHOMPSON
200011	HEMMINGER	HRREP	1933-08-14	123-45-3979	DJHEMMINGE	CIHAAS
200121	ORLANDO	HRREP	1942-10-18	123-45-2168	GORLANDO	CIHAAS
000120	O'CONNELL	HRREP	1942-10-18	123-45-2167	SOCONNELL	CIHAAS
000110	LUCCHESI	HRREP	1929-11-05	123-45-3490	VGLUCCHESS	CIHAAS
000020	THOMPSON	BIGBOSS	1948-02-02	123-45-3476	MLTHOMPSON	MLTHOMPSON
000130	QUINTANA	ANALYST	1925-09-15	123-45-4578	DMQUINTANA	SAKWAN
000140	NICHOLLS	ANALYST	1946-01-19	123-45-1793	HANICHOLLS	SAKWAN
200141	NATZ	ANALYST	1946-01-19	123-45-1794	KNNATZ	SAKWAN
000030	KWAN	MANAGER	1941-05-11	123-45-4738	SAKWAN	MLTHOMPSON
000150	ADAMSON	DESIGNER	1947-05-17	123-45-4510	BADAMSON	IFSTERN
000200	BROWN	DESIGNER	1941-05-29	123-45-4501	DBROWN	IFSTERN
000160	PIANKA	DESIGNER	1955-04-12	123-45-3782	ERPIANKA	IFSTERN
000060	STERN	MANAGER	1945-07-07	123-45-6423	IFSTERN	MLTHOMPSON
000190	WALKER	DESIGNER	1952-06-25	123-45-2986	JHWALKER	IFSTERN
000220	LUTZ	DESIGNER	1948-03-19	123-45-0672	JKLUTZ	IFSTERN
200171	YAMAMOTO	DESIGNER	1951-01-05	123-45-2891	KYAMAMOTO	IFSTERN
000170	YOSHIMURA	DESIGNER	1951-01-05	123-45-2890	MJYOSHIMUR	IFSTERN
000180	SCOUTTEN	DESIGNER	1949-02-21	123-45-1682	MSSCOUTTEN	IFSTERN
200221	JOHN	DESIGNER	1948-03-19	123-45-6730	RKJOHN	IFSTERN
000210	JONES	DESIGNER	1953-02-23	123-45-0942	WTJONES	IFSTERN
000250	SMITH	CLERK	1939-11-12	123-45-0961	DSSMITH	EDPULASKI
000070	PULASKI	MANAGER	1953-05-26	123-45-7831	EDPULASKI	MLTHOMPSON
000230	JEFFERSON	CLERK	1935-05-30	123-45-2094	JJJEFFERSO	EDPULASKI
000270	PEREZ	CLERK	1953-05-26	123-45-9001	MLPEREZ	EDPULASKI
200241	MONTEVERDE	CLERK	1954-03-31	123-45-3781	RMMONTEVER	EDPULASKI
000240	MARINO	CLERK	1954-03-31	123-45-3780	SMMARINO	EDPULASKI
000260	JOHNSON	CLERK	1936-10-05	123-45-8953	SPJOHNSON	EDPULASKI
000050	GEYER	MANAGER	1925-09-15	123-45-6789	JBGEYER	MLTHOMPSON
000280	SCHNEIDER	OPERATOR	1936-03-28	123-45-8997	ERSCHNEIDE	EWHENDERSO
200281	SCHWARTZ	OPERATOR	1936-03-28	123-45-8998	ERSCHWARTZ	EWHENDERSO
000090	HENDERSON	MANAGER	1941-05-15	123-45-5498	EWHENDERSO	MLTHOMPSON
000290	PARKER	OPERATOR	1946-07-09	123-45-4502	JRPARKER	EWHENDERSO
000310	SETRIGHT	OPERATOR	1931-04-21	123-45-3332	MFSETRIGHT	EWHENDERSO
200311	SPRINGER	OPERATOR	1931-04-21	123-45-3333	MFSPRINGER	EWHENDERSO
000300	SMITH	OPERATOR	1936-10-27	123-45-2095	PXSMITH	EWHENDERSO
200331	WONG	FIELDREP	1941-07-18	123-45-2104	HWONG	TQSPENSER
000340	GOUNOT	FIELDREP	1926-05-17	123-45-5698	JRGOUNOT	TQSPENSER
200341	ALONZO	FIELDREP	1926-05-17	123-45-5699	RRALONZO	TQSPENSER
000320	MEHTA	FIELDREP	1932-08-11	123-45-9990	RVMEHTA	TQSPENSER
000100	SPENSER	MANAGER	1956-12-18	123-45-0972	TQSPENSER	MLTHOMPSON
000330	LEE	FIELDREP	1941-07-18	123-45-2103	WLEE	TQSPENSER

Figure 3-8 List of employees without RCAC enabled

3.6.4 Defining and creating row permissions

Implement RCAC on the EMPLOYEES table by completing the following steps:

1. Start by defining a row permission. In this example, the rules to enforce include the following ones:
 - Human Resources employees can see all the rows.
 - Managers can see only information for the employees that they manage.
 - Employees can see only their own information.
 - Consultants are not allowed to see any rows in the table.

To implement this row permission, run the SQL statement that is shown in Example 3-7.

Example 3-7 Creating a permission for the EMPLOYEE table

```

CREATE PERMISSION  HR_SCHEMA.PERMISSION1_ON_EMPLOYEES
ON                HR_SCHEMA.EMPLOYEES AS EMPLOYEES
FOR ROWS
WHERE            ( VERIFY_GROUP_FOR_USER ( SESSION_USER , 'HR' ) = 1 )
OR              ( VERIFY_GROUP_FOR_USER ( SESSION_USER , 'MGR' ) = 1
AND            ( EMPLOYEES . MANAGER_OF_EMPLOYEE = SESSION_USER
                OR EMPLOYEES . USER_ID = SESSION_USER ) )
OR              ( VERIFY_GROUP_FOR_USER ( SESSION_USER , 'EMP' ) = 1
AND            EMPLOYEES . USER_ID = SESSION_USER )
ENFORCED FOR ALL ACCESS
ENABLE ;

```

2. Look at the definition of the table and see the permissions, as shown in Figure 3-9. QIBM_DEFAULT_EMPLOYEE_HR_SCHEMA is the default permission, as described in 3.1.2, “Enabling and activating RCAC” on page 16.

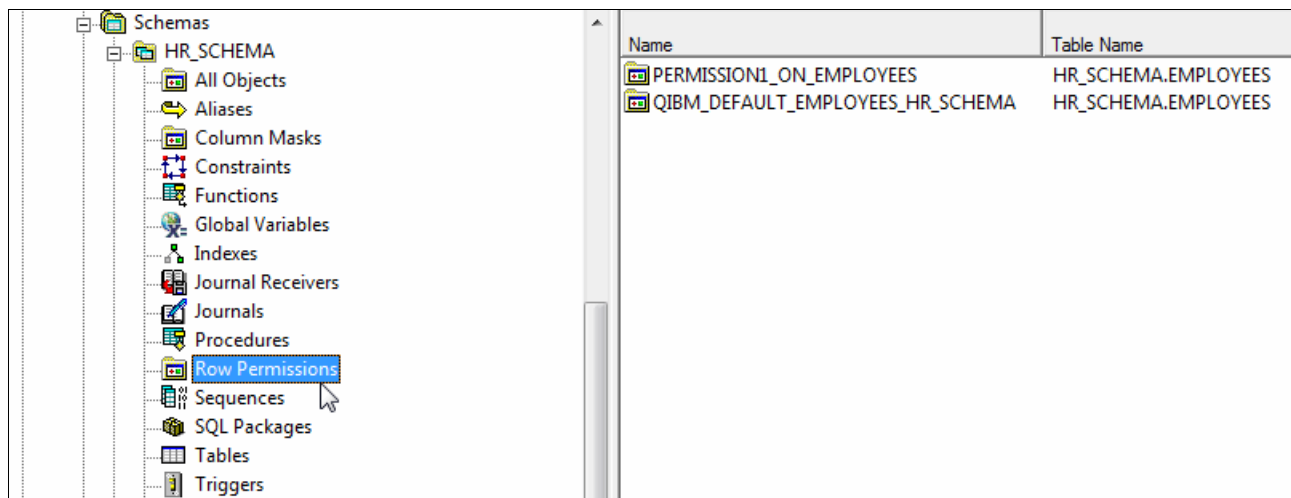


Figure 3-9 Row permissions that are shown in System i Navigator

3.6.5 Defining and creating column masks

Define the different masks for the columns that are sensitive by completing the following steps:

1. Start with the DAY_OF_BIRTH column. In this example, the rules to enforce include the following ones:
 - Human Resources can see the entire date of birth of the employees.
 - Employees can see only their own date of birth.
 - Managers can see the date of birth of their employees masked with YEAR being 9999.

To implement this column mask, run the SQL statement that is shown in Example 3-8.

Example 3-8 Creation of a mask on the DATE_OF_BIRTH column

```

CREATE MASK  HR_SCHEMA.MASK_DATE_OF_BIRTH_ON_EMPLOYEES
ON          HR_SCHEMA.EMPLOYEES AS EMPLOYEES
FOR COLUMN  DATE_OF_BIRTH

```

```

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.

Name	Table Name	Column Name
MASK_DATE_OF_BIRTH_ON_EMPLOYEES	HR_SCHEMA.EMPLOYEES	DATE_OF_BIRTH
MASK_TAX_ID_ON_EMPLOYEES	HR_SCHEMA.EMPLOYEES	TAX_ID

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**.

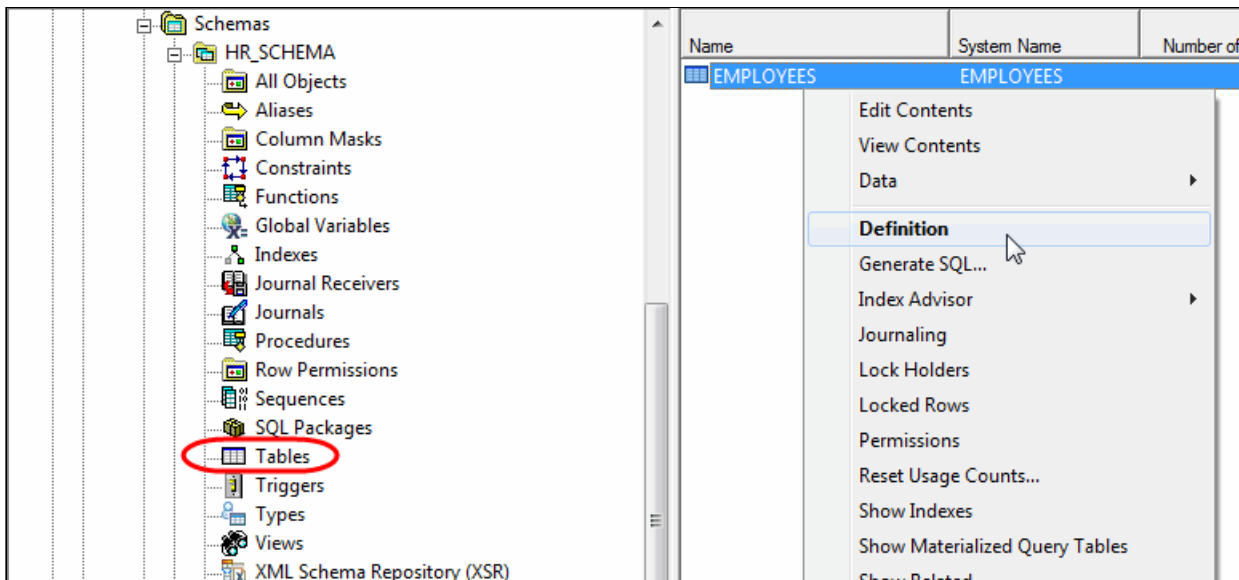


Figure 3-11 Selecting the EMPLOYEES table from System i Navigator

- The EMPLOYEES table definition is displayed, as shown in Figure 3-12. Note that the Row access control and Column access control options are checked.

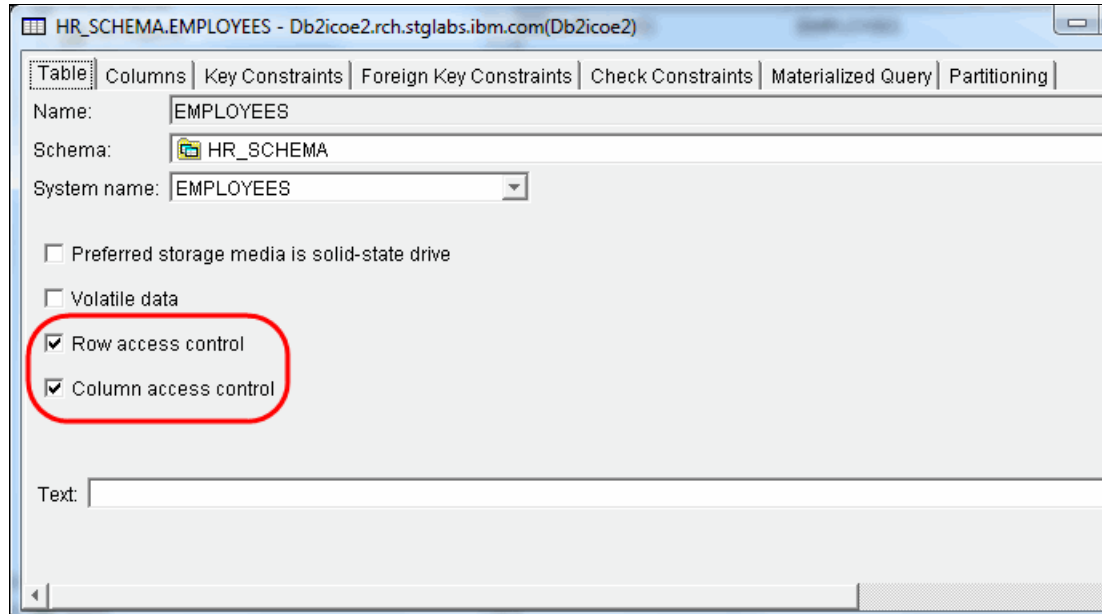


Figure 3-12 RCAC enabled on the EMPLOYEES table

3.6.7 Demonstrating data access with RCAC

You are now ready to start testing RCAC with the four different users. Complete the following steps:

- The first SQL statement that is shown in Example 3-11 illustrates the EMPLOYEE count. You know that there are 42 rows from the query that was run before RCAC was put in place (see 3.6.3, “Demonstrating data access without RCAC” on page 24).

Example 3-11 EMPLOYEES count

```
SELECT COUNT(*) as ROW_COUNT FROM HR_SCHEMA.EMPLOYEES;
```

- The result of the query for a user that belongs to the HR group profile is shown in Figure 3-13. This user can see all the 42 rows (employees).

ROW_COUNT
42

Figure 3-13 Count of EMPLOYEES by HR

- The result of the same query for a user who is logged on as TQSPENSER (Manager) is shown in Figure 3-14. TQSPENSER has five employees in his department and he can also see his own row, which is why the count is 6.

ROW_COUNT
6

Figure 3-14 Count of EMPLOYEES by a manager

4. The result of the same query that is run by an employee (DSSMITH) gives the result that is shown in Figure 3-15. Each employee can see only his or her own data (row).

ROW_COUNT
1

Figure 3-15 Count of EMPLOYEES by an employee

5. The result of the same query that is run by the Consultant/DBE gives the result that is shown in Figure 3-16. The consultants/DBE can manage and implement RCAC, but they do not see any rows at all.

ROW_COUNT
0

Figure 3-16 Count of EMPLOYEES by a consultant

Does the result make sense? Yes, it does because RCAC is enabled.

6. Run queries against the EMPLOYEES table. The query that is used in this example runs and tests with the four different user profiles and is the same query that was run in 3.6.3, “Demonstrating data access without RCAC” on page 24. It is shown in Example 3-12.

Example 3-12 SELECT statement to test with the different users

```
SELECT EMPLOYEE_ID,  
       LAST_NAME,  
       JOB_DESCRIPTION,  
       DATE_OF_BIRTH,  
       TAX_ID,  
       USER_ID,  
       MANAGER_OF_EMPLOYEE  
FROM   HR_SCHEMA.EMPLOYEES
```

7. Figure 3-17 shows the results of the query for a Human Resources (VGLUCCHES) user profile. The user can see all the rows and all the columns.

EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
000010	HAAS	MANAGER	1933-08-24	123-45-3978	CIHAAS	MLTHOMPSON
200011	HEMMINGER	HRREP	1933-08-14	123-45-3979	DJHEMMINGE	CIHAAS
200121	ORLANDO	HRREP	1942-10-18	123-45-2168	GORLANDO	CIHAAS
000120	O'CONNELL	HRREP	1942-10-18	123-45-2167	SOCONNELL	CIHAAS
000110	LUCCHESI	HRREP	1929-11-05	123-45-3490	VGLUCCHES	CIHAAS
000020	THOMPSON	BIGBOSS	1948-02-02	123-45-3476	MLTHOMPSON	MLTHOMPSON
000130	QUINTANA	ANALYST	1925-09-15	123-45-4578	DMQUINTANA	SAKWAN
000140	NICHOLLS	ANALYST	1946-01-19	123-45-1793	HANICHOLLS	SAKWAN
200141	NATZ	ANALYST	1946-01-19	123-45-1794	KNNATZ	SAKWAN
000030	KWAN	MANAGER	1941-05-11	123-45-4738	SAKWAN	MLTHOMPSON
000150	ADAMSON	DESIGNER	1947-05-17	123-45-4510	BADAMSON	IFSTERN
000200	BROWN	DESIGNER	1941-05-29	123-45-4501	DBROWN	IFSTERN
000160	PIANKA	DESIGNER	1955-04-12	123-45-3782	ERPIANKA	IFSTERN
000060	STERN	MANAGER	1945-07-07	123-45-6423	IFSTERN	MLTHOMPSON
000190	WALKER	DESIGNER	1952-06-25	123-45-2986	JHWALKER	IFSTERN
000220	LUTZ	DESIGNER	1948-03-19	123-45-0672	JKLUTZ	IFSTERN
200171	YAMAMOTO	DESIGNER	1951-01-05	123-45-2891	KYAMAMOTO	IFSTERN
000170	YOSHIMURA	DESIGNER	1951-01-05	123-45-2890	MJYOSHIMUR	IFSTERN
000180	SCOUTTEN	DESIGNER	1949-02-21	123-45-1682	MSSCOUTTEN	IFSTERN
200221	JOHN	DESIGNER	1948-03-19	123-45-6730	RKJOHN	IFSTERN
000210	JONES	DESIGNER	1953-02-23	123-45-0942	WTJONES	IFSTERN
000250	SMITH	CLERK	1939-11-12	123-45-0961	DSSMITH	EDPULASKI
000070	PULASKI	MANAGER	1953-05-26	123-45-7831	EDPULASKI	MLTHOMPSON
000230	JEFFERSON	CLERK	1935-05-30	123-45-2094	JJJEFFERSO	EDPULASKI
000270	PEREZ	CLERK	1953-05-26	123-45-9001	MLPEREZ	EDPULASKI
200241	MONTEVERDE	CLERK	1954-03-31	123-45-3781	RMMONTEVER	EDPULASKI
000240	MARINO	CLERK	1954-03-31	123-45-3780	SMMARINO	EDPULASKI
000260	JOHNSON	CLERK	1936-10-05	123-45-8953	SPJOHNSON	EDPULASKI
000050	GEYER	MANAGER	1925-09-15	123-45-6789	JBGEYER	MLTHOMPSON
000280	SCHNEIDER	OPERATOR	1936-03-28	123-45-8997	ERSCHNEIDE	EWHENDERSO
200281	SCHWARTZ	OPERATOR	1936-03-28	123-45-8998	ERSCHWARTZ	EWHENDERSO
000090	HENDERSON	MANAGER	1941-05-15	123-45-5498	EWHENDERSO	MLTHOMPSON
000290	PARKER	OPERATOR	1946-07-09	123-45-4502	JRPARKER	EWHENDERSO
000310	SETRIGHT	OPERATOR	1931-04-21	123-45-3332	MFSETRIGHT	EWHENDERSO
200311	SPRINGER	OPERATOR	1931-04-21	123-45-3333	MFSPRINGER	EWHENDERSO
000300	SMITH	OPERATOR	1936-10-27	123-45-2095	PXSMITH	EWHENDERSO
200331	WONG	FIELDREP	1941-07-18	123-45-2104	HWONG	TQSPENSER
000340	GOUNOT	FIELDREP	1926-05-17	123-45-5698	JRGOUNOT	TQSPENSER
200341	ALONZO	FIELDREP	1926-05-17	123-45-5699	RRALONZO	TQSPENSER
000320	MEHTA	FIELDREP	1932-08-11	123-45-9990	RVMEHTA	TQSPENSER
000100	SPENSER	MANAGER	1956-12-18	123-45-0972	TQSPENSER	MLTHOMPSON
000330	LEE	FIELDREP	1941-07-18	123-45-2103	WLEE	TQSPENSER

Figure 3-17 SQL statement result by Human Resources user profile

8. Figure 3-18 shows the results of the same query for the Manager (TQSPENSER). Notice the masking of the DATE_OF_BIRTH and TAX_ID columns.

EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
200331	WONG	FIELDREP	9999-07-18	XXX-XX-2104	HWONG	TQSPENSER
000340	GOUNOT	FIELDREP	9999-05-17	XXX-XX-5698	JRGOUNOT	TQSPENSER
200341	ALONZO	FIELDREP	9999-05-17	XXX-XX-5699	RRALONZO	TQSPENSER
000320	MEHTA	FIELDREP	9999-08-11	XXX-XX-9990	RVMEHTA	TQSPENSER
000100	SPENSER	MANAGER	1956-12-18	123-45-0972	TQSPENSER	MLTHOMPSON
000330	LEE	FIELDREP	9999-07-18	XXX-XX-2103	WLEE	TQSPENSER

Figure 3-18 SQL statement result by Manager profile

9. Figure 3-19 shows the results of the same query for an employee (DSSMITH). The employee can only see only his own data with no masking at all.

EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
000250	SMITH	CLERK	1939-11-12	123-45-0961	DSSMITH	EDPULASKI

Figure 3-19 SQL statement result by an employee profile

10. Figure 3-20 shows the results of the same query for the Consultant/DBE, who is not one of the company's employees.

EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
Empty Set						

Figure 3-20 SQL statement result by Consultant/DBE profile

3.6.8 Demonstrating data access with a view and RCAC

This section covers data access with a view and RCAC. Complete the following steps:

1. The EMPLOYEES table has a column that is called On_Leave_Flag (Figure 3-21 on page 33) indicating that the employee is on Leave of Absence. For this purpose, a view is created that lists only the employees that are on leave.

EMPLOYEE_ID	LAST_NAME	USER_ID	MANAGER_OF_EMPLOYEE	ON_LEAVE_FLAG
000010	HAAS	CIHAAS	MLTHOMPSON	-
200011	HEMMINGER	DJHEMMINGE	CIHAAS	-
200121	ORLANDO	GORLANDO	CIHAAS	-
000120	O'CONNELL	SOCONNELL	CIHAAS	-
000110	LUCCHESI	VGLUCCHES	CIHAAS	-
000020	THOMPSON	MLTHOMPSON	MLTHOMPSON	-
000130	QUINTANA	DMQUINTANA	SAKWAN	-
000140	NICHOLLS	HANICHOLLS	SAKWAN	-
200141	NATZ	KNNATZ	SAKWAN	-
000030	KWAN	SAKWAN	MLTHOMPSON	-
000150	ADAMSON	BADAMSON	IFSTERN	-
000200	BROWN	DBROWN	IFSTERN	-
000160	PIANKA	ERPIANKA	IFSTERN	-
000060	STERN	IFSTERN	MLTHOMPSON	-
000190	WALKER	JHWALKER	IFSTERN	-
000220	LUTZ	JKLUTZ	IFSTERN	-
200171	YAMAMOTO	KYAMAMOTO	IFSTERN	-
000170	YOSHIMURA	MJYOSHIMUR	IFSTERN	-
000180	SCOUTTEN	MSSCOUTTEN	IFSTERN	-
200221	JOHN	RKJOHN	IFSTERN	-
000210	JONES	WTJONES	IFSTERN	-
000250	SMITH	DSSMITH	EDPULASKI	-
000070	PULASKI	EDPULASKI	MLTHOMPSON	-
000230	JEFFERSON	JJJEFFERSO	EDPULASKI	-
000270	PEREZ	MLPEREZ	EDPULASKI	-
200241	MONTEVERDE	RMMONTEVER	EDPULASKI	-
000240	MARINO	SMMARINO	EDPULASKI	-
000260	JOHNSON	SPJOHNSON	EDPULASKI	Y
000050	GEYER	JBGEYER	MLTHOMPSON	-
000280	SCHNEIDER	ERSCHNEIDE	EWHENDERSO	-
200281	SCHWARTZ	ERSCHWARTZ	EWHENDERSO	-
000090	HENDERSON	EWHENDERSO	MLTHOMPSON	-
000290	PARKER	JRPARKER	EWHENDERSO	-
000310	SETRIGHT	MFSETRIGHT	EWHENDERSO	-
200311	SPRINGER	MFSRINGER	EWHENDERSO	-
000300	SMITH	PXSMITH	EWHENDERSO	-
200331	WONG	HWONG	TQSPENSER	-
000340	GOUNOT	JRGOUNOT	TQSPENSER	-
200341	ALONZO	RRALONZO	TQSPENSER	-
000320	MEHTA	RVMEHTA	TQSPENSER	-
000100	SPENSER	TQSPENSER	MLTHOMPSON	-
000330	LEE	WLEE	TQSPENSER	Y

Figure 3-21 Employees on leave

2. Example 3-13 shows the definition of the view.

Example 3-13 View of employees on leave

```
CREATE VIEW HR_SCHEMA.EMPLOYEES_ON_LEAVE (EMPLOYEE_ID,
FIRST_NAME,
MIDDLE_INITIAL,
LAST_NAME,
WORK_DEPARTMENT,
PHONE_EXTENSION,
JOB_DESCRIPTION,
DATE_OF_BIRTH,
```

```

TAX_ID,
USER_ID,
MANAGER_OF_EMPLOYEE,
ON_LEAVE_FLAG )
AS
SELECT EMPLOYEE_ID,
       FIRST_NAME ,
       MIDDLE_INITIAL,
       LAST_NAME ,
       WORK_DEPARTMENT,
       PHONE_EXTENSION,
       JOB_DESCRIPTION,
       DATE_OF_BIRTH,
       TAX_ID,
       USER_ID,
       MANAGER_OF_EMPLOYEE,
       ON_LEAVE_FLAG
FROM   HR_SCHEMA.EMPLOYEES
WHERE  ON_LEAVE_FLAG = 'Y';

```

- Use the view to query the data and see who is on leave. The SQL statement that is used is shown in Example 3-14:

Example 3-14 SQL statement for employees on leave

```

SELECT EMPLOYEE_ID,
       LAST_NAME,
       JOB_DESCRIPTION,
       DATE_OF_BIRTH,
       TAX_ID,
       USER_ID,
       MANAGER_OF_EMPLOYEE
FROM   HR_SCHEMA.EMPLOYEES_ON_LEAVE;

```

- Start with the Human Resources person (VGLUCCHES) and see what is the result of the previous query. He sees the two employees that are on leave and no masking is done over the DATE_OF_BIRTH and TAX_ID columns. The results of the query are shown in Figure 3-22.

EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
000260	JOHNSON	CLERK	1936-10-05	123-45-8953	SPJOHNSON	EDPULASKI
000330	LEE	FIELDREP	1941-07-18	123-45-2103	WLEE	TQSPENSER

Figure 3-22 Employees on leave - Human Resources user

- Figure 3-23 shows what the Manager (TQSPENSER) gets when he runs the same query over the view. He sees only the employees that are on leave that are managed by him. In this example, it is one employee. The columns are masked, which confirms that RCAC is applied to the view as well.


EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
000330	LEE	FIELDREP	9999-07-18	XXX-XX-2103	WLEE	TQSPENSER

Figure 3-23 Employee on leave - Manager of Field Reps user

6. Figure 3-24 shows what the employee (DSSMITH) gets when he runs the same query over the view. The employee gets an empty set or he gets only himself if he is on leave.

EMPLOYEE_ID	LAST_NAME	JOB_DESCRIPTION	DATE_OF_BIRTH	TAX_ID	USER_ID	MANAGER_OF_EMPLOYEE
Empty Set						

Figure 3-24 Employees on leave - employee user



Implementing Row and Column Access Control: Banking example

This chapter illustrates the Row and Column Access Control (RCAC) concepts using a banking example. Appendix A, “Database definitions for the RCAC banking example” on page 121 provides a script that you can use to create all the database definitions or DDLs to re-create this RCAC example.

The following topics are covered in this chapter:

- ▶ Business requirements for the RCAC banking scenario
- ▶ Description of the users roles and responsibilities
- ▶ Implementation of RCAC

4.1 Business requirements for the RCAC banking scenario

As part of a new internet banking project, the *Bank* decides to raise the level of data access control on the following three tables that are involved in the new customer-facing application:

- ▶ CUSTOMERS
- ▶ ACCOUNTS
- ▶ TRANSACTIONS

RCAC will be used to restrict access to the rows in these three tables by using permissions, and to restrict column values by using masks. The default position is that no user can access the rows in the tables. From there, specific bank employees are allowed access only to the rows for their job responsibilities. In addition, columns containing personal or sensitive data are masked appropriately. Bank customers are allowed access to only their rows and column values.

In this example, it is assumed that the Bank employees have access to the tables when working on the premises only. Employee access to data is provided by programs and tools using standard DB2 interfaces, such as embedded SQL, ODBC, JDBC, and CLI. The database connection authentication for these interfaces uses the employee's personal and unique IBM i user profile. Operating in their professional role, employees do not have access to bank data through the Internet.

Bank customers have access to their accounts and transactions by using a new web application. Each customer has unique credentials for logging in to the application. The authentication of the customer is handled by the web server. After the customer is authenticated, the web server establishes a connection to DB2 for data access. This connection uses a common IBM i user profile that is known as WEBUSER. This user profile is secured and is used only by the web application. No Bank employee has access to the WEBUSER profile, and no customer has an IBM i user profile.

The customer's identity is passed to DB2 by using a global variable. The global variable is secured and can be accessed only by the WEBUSER. The web application sets the CUSTOMER_LOGIN_ID variable to the customer's login value. This value is compared to the customer's login value that is found in the CUSTOMER_LOGIN_ID column of the CUSTOMERS table.

Applications that do not use the web interface do not have to be changed because the global variable is NULL by default.

A diagram of the internet banking architecture is shown in Figure 4-1:

- ▶ The row permission and column masks for the CUSTOMERS table are based on the group of which the user profile is part. If the user is a customer, their specific login ID also is tested.
- ▶ The row permission and column mask for the ACCOUNTS table are based on the CUSTOMERS table permission rules. A subquery is used to connect the accounts (child) with the customer (parent).
- ▶ The row permission for the TRANSACTIONS table is based on the ACCOUNTS table permission rules and the CUSTOMERS table permission rules. A subquery is used to connect the transactions (child) with the account (parent) and the account (child) with the customer (parent).

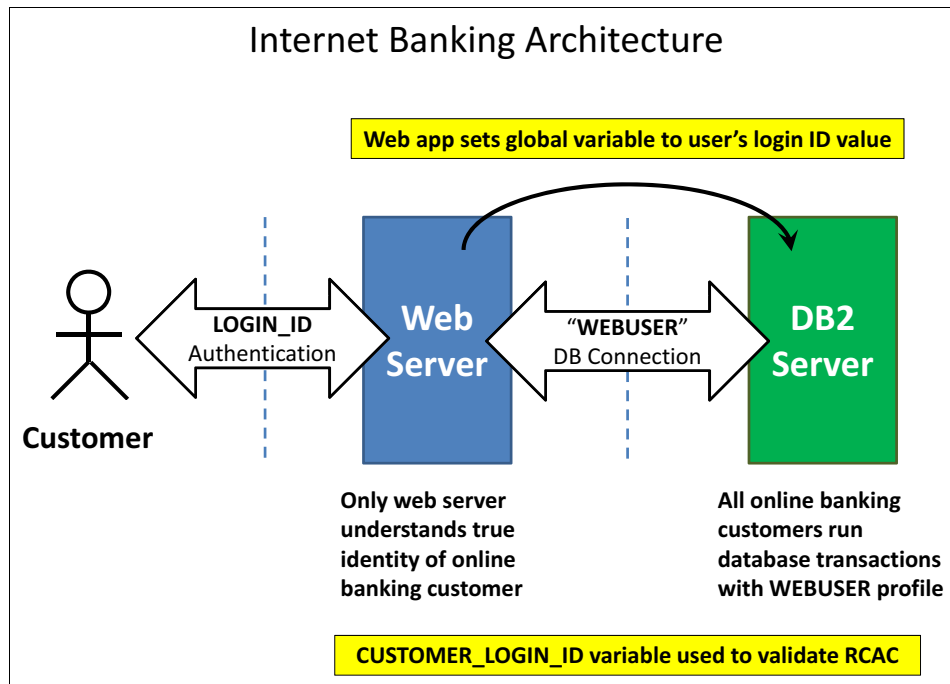


Figure 4-1 Internet banking example

4.2 Description of the users roles and responsibilities

During the requirements gathering phase, the following groups of users are identified and codified:

- ▶ SECURITY: Security officer and security administrators
- ▶ DBE: Database engineers
- ▶ ADMIN: Bank business administrators
- ▶ TELLER: Bank tellers
- ▶ CUSTOMER: Bank customers using the internet
- ▶ PUBLIC: Anyone not already in a group

Based on their respective roles and responsibilities, the users (that is, a group) are controlled by row permissions and column masks. The chart that is shown in Figure 4-2 shows the rules for row and column access in this example.

	CUSTOMERS		ACCOUNTS		TRANSACTIONS	
	Row Permissions	Column Masking	Row Permissions	Column Masking	Row Permissions	Column Masking
SECURITY	No Rows	Yes	No Rows	Yes	No Rows	No
DBE	All Rows	Yes	All Rows	Yes	All Rows	No
ADMIN	All Rows	No	All Rows	No	All Rows	No
TELLER	All Rows	Yes	All Rows	No	All Rows	No
CUSTOMER	Own Rows	No	Own Rows	No	Own Rows	No
PUBLIC	No Rows	Yes	No Rows	Yes	No Rows	No

Figure 4-2 Rules for row and column access

The chart that is shown in Figure 4-3 shows the column access that is allowed by group and lists the column masks by table.

		CUSTOMERS	ACCOUNTS
	Row Permissions	Column Masking	Column Masking
SECURITY	No Rows	CUSTOMER_DRIVERS_LICENSE_NUMBER CUSTOMER_EMAIL CUSTOMER_LOGIN_ID CUSTOMER_SECURITY_QUESTION CUSTOMER_SECURITY_QUESTION_ANSWER CUSTOMER_TAX_ID	ACCOUNT_NUMBER
DBE	All Rows	CUSTOMER_DRIVERS_LICENSE_NUMBER CUSTOMER_EMAIL CUSTOMER_LOGIN_ID CUSTOMER_SECURITY_QUESTION CUSTOMER_SECURITY_QUESTION_ANSWER CUSTOMER_TAX_ID	ACCOUNT_NUMBER
ADMIN	All Rows	None	None
TELLER	All Rows	CUSTOMER_EMAIL CUSTOMER_LOGIN_ID CUSTOMER_SECURITY_QUESTION CUSTOMER_SECURITY_QUESTION_ANSWER CUSTOMER_TAX_ID	None
CUSTOMER	Own Rows	None	None
PUBLIC	No Rows	CUSTOMER_DRIVERS_LICENSE_NUMBER CUSTOMER_EMAIL CUSTOMER_LOGIN_ID CUSTOMER_SECURITY_QUESTION CUSTOMER_SECURITY_QUESTION_ANSWER CUSTOMER_TAX_ID	ACCOUNT_NUMBER

Figure 4-3 Column masks

For the demonstration and testing of RCAC in this example, the following users interact with the database. Furthermore, the column masking rules are developed independently of the row permissions. If a person does not have permission to access the row, the column mask processing does not occur.

- ▶ Hernando Bedoya is a DB2 for i database engineer with the user profile of HBEDOYA. He is part of the DBE group.
- ▶ Mike Cain is a DB2 for i database engineer with the user profile of MCAIN. He is part of the DBE group.
- ▶ Veronica G. Lucchess is a bank account administrator with the user profile of VGLUCCHESS. She is part of the ADMIN group.
- ▶ Tom Q. Spenser is a bank teller with the user profile of TQSPENSER. He is part of the TELLER group.
- ▶ The IT security officer has the user profile of SECURITY. She is not part of any group.
- ▶ The online banking web application uses the user profile WEBUSER. This profile is part of the CUSTOMER group. Any future customer-facing applications can also use this group if needed.
- ▶ Adam O. Olsen is a bank customer with a web application login ID of KLD72CQR8JG.

4.3 Implementation of RCAC

Figure 4-4 shows the data model of the banking scenario that is used in this example.

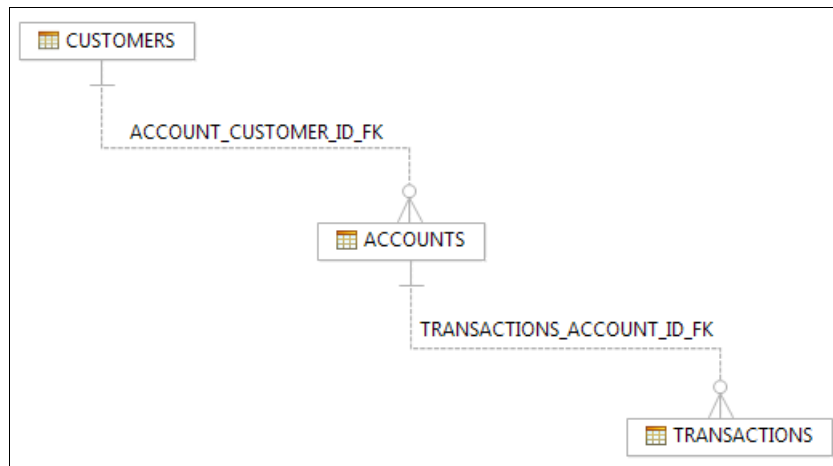


Figure 4-4 Data model of the banking scenario

This section covers the following steps:

- ▶ Reviewing the tables that are used in this example
- ▶ Assigning function ID QIBM_DB_SECADM to the Database Engineers group
- ▶ Creating group profiles for the users and their roles
- ▶ Creating the CUSTOMER_LOGIN_ID global variable
- ▶ Defining and creating row permissions
- ▶ Defining and creating column masks
- ▶ Restricting the inserting and updating of masked data
- ▶ Activating row and column access control
- ▶ Reviewing row permissions
- ▶ Demonstrating data access with RCAC
- ▶ Query implementation with RCAC activated

4.3.1 Reviewing the tables that are used in this example

This section reviews the tables that are used in this example. As shown in Figure 4-5, there are three main tables that are involved in the data model: CUSTOMERS, ACCOUNTS, and TRANSACTIONS. There are 90 customers.

Name	System Name	Number of Rows	Number of Deleted Rows
ACCOUNTS	ACCOUNTS	100	0
CUSTOMERS	CUSTOMERS	90	0
TRANSACTIONS	TRANS	529	0

Figure 4-5 Tables that are used in the banking example

Note: Appendix A, “Database definitions for the RCAC banking example” on page 121 provides a script that you can use to create all the database definitions or DDLs to re-create this RCAC example.

To review the attributes of each table that is used in this banking example, complete the following steps:

1. Review the columns of each the tables through System i Navigator. Expand **Database** → **named Database** → **Schemas** → **BANK_SCHEMA** → **Tables**.
2. Right-click the CUSTOMERS table and select **Definition**. Figure 4-6 shows the attributes for the CUSTOMERS table. The Row access control and Column access control options are not selected, which indicates that the table does not have RCAC implemented.

Figure 4-6 CUSTOMERS table attributes

3. Click the **Columns** tab to see the columns of the CUSTOMERS table, as shown in Figure 4-7.

Column Name	Data Type	Length	Nullable	Implicitly Hidden	Default Value
CUSTOMER_ID	INTEGER		No		
CUSTOMER_NAME	VARCHAR	30	No		No default
CUSTOMER_ADDRESS	VARCHAR	30	No		No default
CUSTOMER_CITY	VARCHAR	30	No		No default
CUSTOMER_STATE	CHARACTER	2	No		No default
CUSTOMER_PHONE	CHARACTER	10	No		No default
CUSTOMER_EMAIL	VARCHAR	30	No		No default
CUSTOMER_TAX_ID	CHARACTER	11	No		No default
CUSTOMER_DRIVERS_LICENSE_NUMBER	CHARACTER	13	Yes		Null
CUSTOMER_LOGIN_ID	VARCHAR	30	Yes		Null
CUSTOMER_SECURITY_QUESTION	VARCHAR	100	Yes		Null
CUSTOMER_SECURITY_QUESTION_ANSWER	VARCHAR	100	Yes		Null
INSERT_TIMESTAMP	TIMESTAMP	6	No	Yes	Current timestamp
UPDATE_TIMESTAMP	TIMESTAMP	6	No	Yes	

Figure 4-7 Column definitions of the CUSTOMERS table

- Click the **Key Constraints**, **Foreign Key Constraints**, and **Check Constraints** tabs to review the key, foreign, and check constraints on the CUSTOMERS table, as shown in Figure 4-8. There are no Foreign Key Constraints or Check Constraints on the CUSTOMERS table.

Name	Type	Key Columns
CUSTOMER_ID_PK	Primary key	CUSTOMER_ID
CUSTOMER_LOGIN_ID_UK	Unique key	CUSTOMER_LOGIN_ID

Name	Key Columns	Parent Table	Parent Key Constraint

Name	Check Condition

Figure 4-8 Reviewing the constraints on the CUSTOMERS table

- Review the definition of the ACCOUNTS table. The definition of the ACCOUNTS table is shown in Figure 4-9. RCAC has not been defined for this table yet.

Name: ACCOUNTS
 Schema: BANK_SCHEMA
 System name: ACCOUNTS

Preferred storage media is solid-state drive
 Volatile data
 Row access control
 Column access control

Text:

Figure 4-9 ACCOUNTS table attributes

6. Click the **Columns** tab to see the columns of the ACCOUNTS table, as shown in Figure 4-10.

Column Name	Data Type	Length	Nullable	Implicitly Hidden	Default Value
ACCOUNT_ID	INTEGER		No		
CUSTOMER_ID	INTEGER		No		No default
ACCOUNT_NUMBER	VARCHAR	50	No		No default
ACCOUNT_NAME	CHARACTER	12	No		No default
ACCOUNT_DATE_OPENED	DATE		Yes		Current date
ACCOUNT_DATE_CLOSED	DATE		Yes		Null
ACCOUNT_CURRENT_BALANCE	DECIMAL	11,2	No		0
INSERT_TIMESTAMP	TIMESTAMP	6	No	Yes	Current timestamp
UPDATE_TIMESTAMP	TIMESTAMP	6	No	Yes	

Figure 4-10 Column definitions of the ACCOUNTS table

7. Click the **Key Constraints**, **Foreign Key Constraints**, and **Check Constraints** tabs to review the key, foreign, and check constraints on the ACCOUNTS table, as shown in Figure 4-11. There is one Foreign Key Constraint and no Check Constraints on the ACCOUNTS table.

Name	Type	Key Columns
ACCOUNT_ID_PK	Primary key	ACCOUNT_ID

Name	Key Columns	Parent Table	Parent Key Constraint	Parent Key Columns
ACCOUNT_CUSTOMER_ID_FK	CUSTOMER_ID	BANK_SCHEM...	CUSTOMER_ID_PK	CUSTOMER_ID

Name	Check Condition
------	-----------------

Figure 4-11 Reviewing the constraints on the ACCOUNTS table

8. Review the definition of the TRANSACTIONS table. The definition of the TRANSACTIONS table is shown in Figure 4-12. RCAC is not defined for this table yet.

Figure 4-12 TRANSACTIONS table attributes

9. Click the **Columns** tab to see the columns of the TRANSACTIONS table, as shown in Figure 4-13.

Column Name	Data Type	Length	Nullable	Implicitly Hidden	Default Value
TRANSACTION_ID	INTEGER		No		
ACCOUNT_ID	INTEGER		No		No default
TRANSACTION_TYPE	CHARACTER	1	No		No default
TRANSACTION_DATE	DATE		No		Current date
TRANSACTION_TIME	TIME		No		Current time
TRANSACTION_AMOUNT	DECIMAL	11,2	No		No default
INSERT_TIMESTAMP	TIMESTAMP	6	No	Yes	Current timestamp
UPDATE_TIMESTAMP	TIMESTAMP	6	No	Yes	

Figure 4-13 Column definitions of the TRANSACTIONS table

10. Click the **Key Constraints**, **Foreign Key Constraints**, and **Check Constraints** tabs to review the key, foreign, and check constraints on the TRANSACTIONS table, as shown in Figure 4-14. There is one Foreign Key Constraint and one Check Constraint on the TRANSACTIONS table.

Figure 4-14 Reviewing the constraints on the TRANSACTIONS table

Now that you have reviewed the database model for this example, the following sections describe the steps that are required to implement RCAC in this banking scenario.

4.3.2 Assigning function ID QIBM_DB_SECADM to the Database Engineers group

The first step is to assign the appropriate function usage ID to the Database Engineers (DBEs) that will be implementing RCAC. For a description of function usage IDs, see 2.1, “Roles” on page 8. In this example, the DBEs are users MCAIN and HBEDOYA.

Complete the following steps:

1. Right-click the database connection and select **Application Administration**, as shown in Figure 4-15.

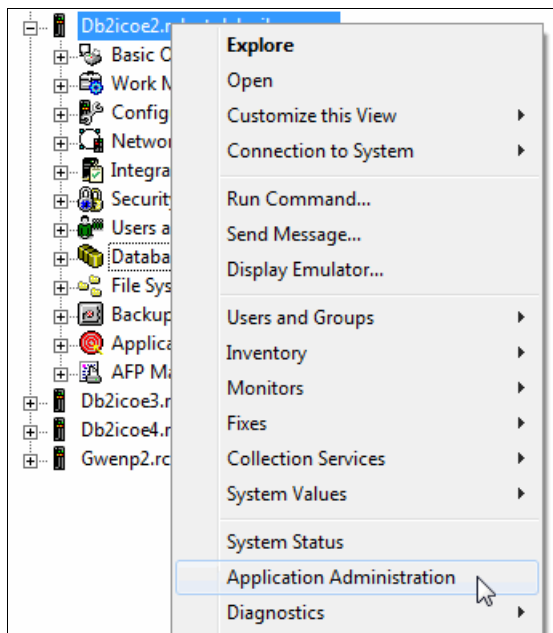


Figure 4-15 Application administration

2. The Application Administration window opens, as shown in Figure 4-16. Click **IBM i** → **Database** and select the function usage ID of **Database Security Administrator**.

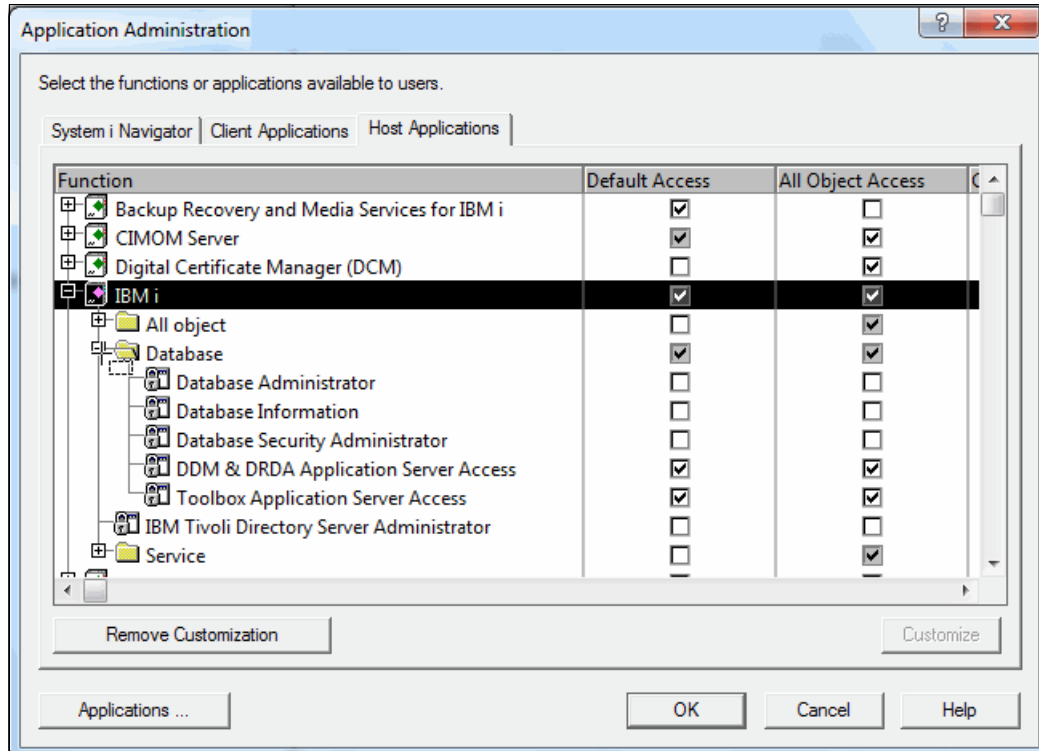


Figure 4-16 Application administration for IBM i

3. Click **Customize** for the function usage ID of Database Security Administrator, as shown in Figure 4-17.

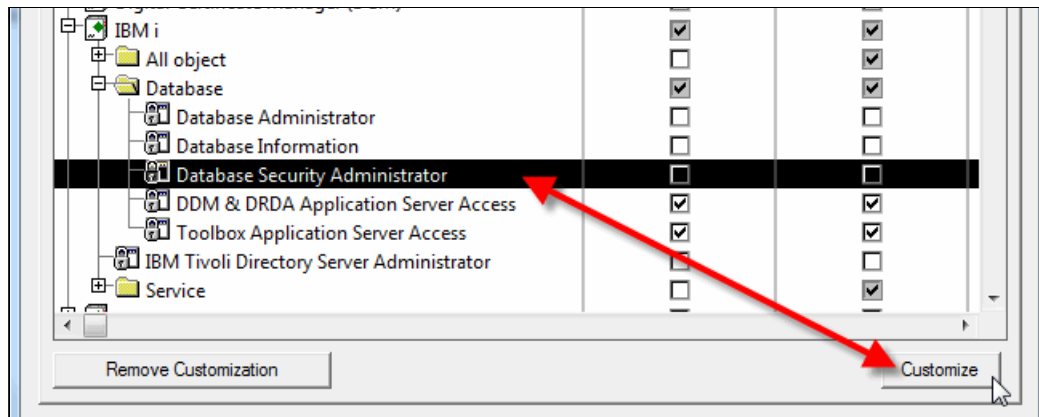


Figure 4-17 Customizing the Database Security Administrator function usage ID

- The Customize Access window opens, as shown in Figure 4-18. Click the users that need to implement RCAC. For this example, HBEDOYA and MCAIN are selected. Click **Add** and then click **OK**.

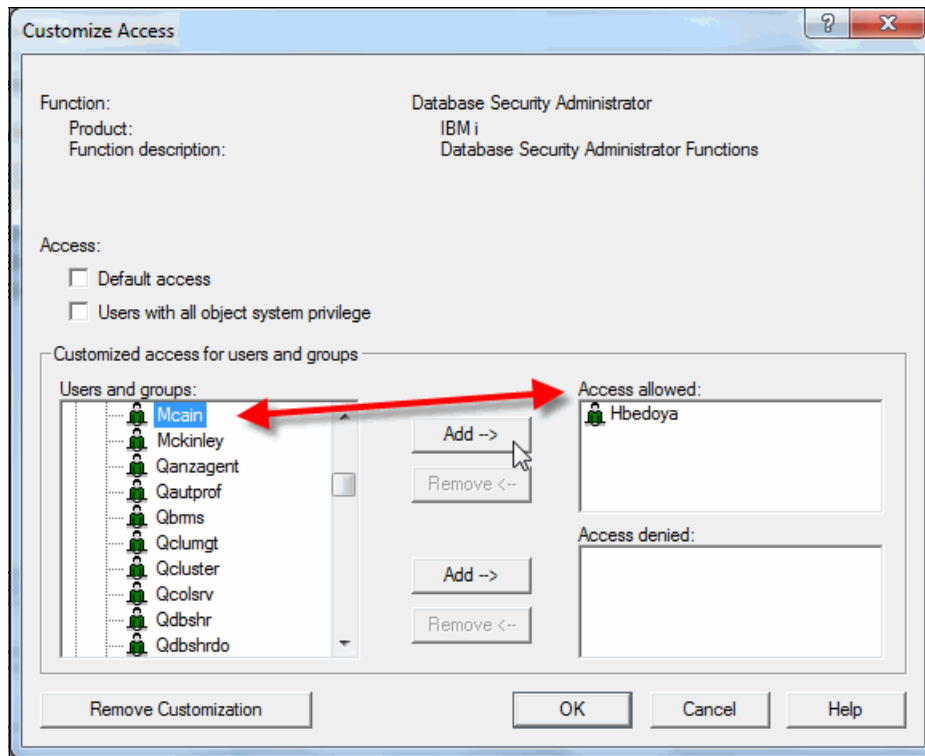


Figure 4-18 Customize Access window

- The Application Administrator window opens again. The function usage ID of Database Security Administrator now has an X in the Customized Access column, as shown in Figure 4-19.

Function	Default Access	All Object Access	Customized Access
IBM i	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
All object	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Database	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Database Administrator	<input type="checkbox"/>	<input type="checkbox"/>	
Database Information	<input type="checkbox"/>	<input type="checkbox"/>	
Database Security Administrator	<input type="checkbox"/>	<input type="checkbox"/>	X
DDM & DRDA Application Server Access	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Toolbox Application Server Access	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Figure 4-19 Function usage ID Database Security Administrator customized

- Run an SQL query that shows which user profiles are enabled to define RCAC. The SQL query is shown in Figure 4-20.

```

/* Verify Functional IDs for Database Security */

SELECT      function_id,
            user_name,
            usage,
            user_type
FROM        qsys2.function_usage
WHERE       function_id = 'QIBM_DB_SECADM'
ORDER BY   user_name;

```

FUNCTION_ID	USER_NAME	USAGE	USER_TYPE
QIBM_DB_SECADM	HBEDOYA	ALLOWED	USER
QIBM_DB_SECADM	MCAIN	ALLOWED	USER

Figure 4-20 Query to display user profiles with function usage ID for RCAC

4.3.3 Creating group profiles for the users and their roles

The next step is to create the different group profiles (ADMIN, CUSTOMER, TELLER, and DBE) and assign the different user profiles to the different group profiles. For a description of the different groups and users for this example, see 4.2, “Description of the users roles and responsibilities” on page 39.

Complete the following steps:

- On the main navigation pane of System i Navigator, right-click **Groups** and select **New Group**, as shown in Figure 4-21.

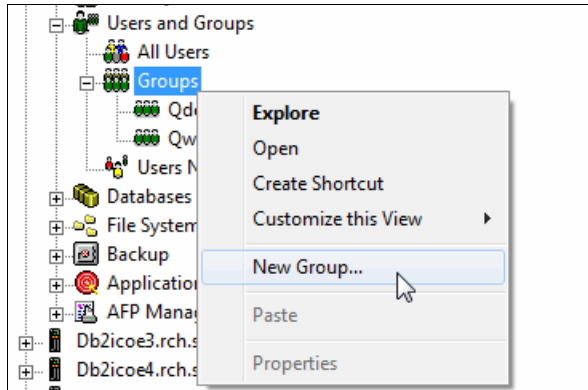


Figure 4-21 Creating group profiles

2. The New Group window opens, as shown in Figure 4-22. For each new group, enter the Group name (ADMIN, CUSTOMER, TELLER, and DBE) and add the user profiles that are associated to this group by selecting the user profile and clicking **Add**.

Figure 4-22 shows adding user TQSPENCER to the TELLER group profile.

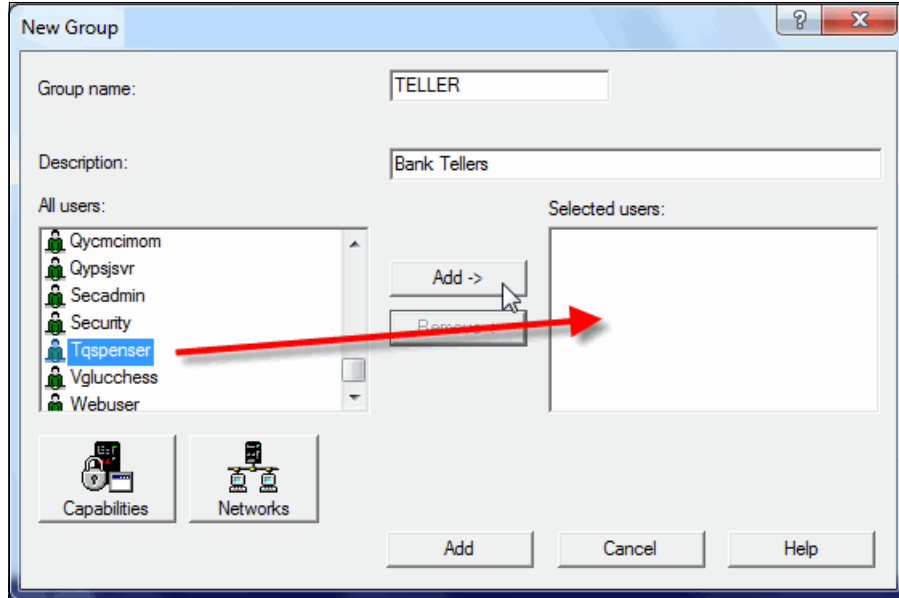


Figure 4-22 Creating group profiles and adding users

3. After you create all the group profiles, you should see them listed in System i Navigator under **Users and Groups** → **Groups**, as shown in Figure 4-23.

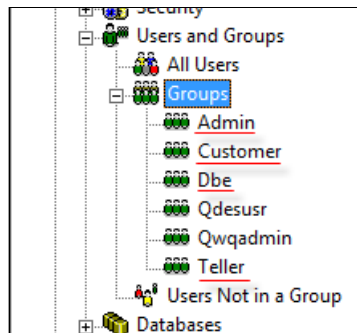


Figure 4-23 Newly created group profiles

4.3.4 Creating the CUSTOMER_LOGIN_ID global variable

In this step, you create a global variable that is used to capture the Customer_Login_ID information, which is required to validate the permissions. For more information about global variables, see 3.2.2, “Built-in global variables” on page 19.

Complete the following steps:

1. From System i Navigator, under the schema Bank_Schema, right-click **Global Variable** and select **New** → **Global Variable**, as shown in Figure 4-24.

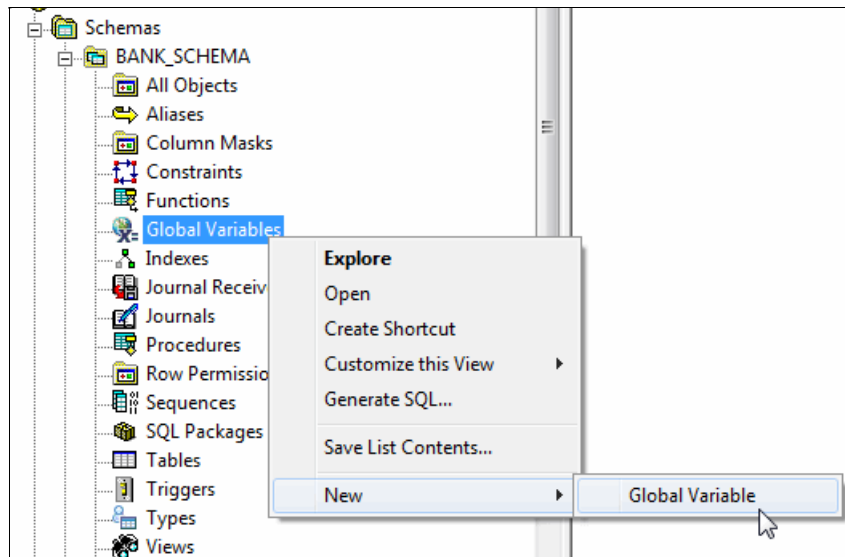


Figure 4-24 Creating a global variable

2. The New Global Variable window opens, as shown in Figure 4-25. Enter the global variable name of CUSTOMER_LOGIN_ID, select the data type of VARCHAR, and leave the default value of NULL. This default value ensures that users that do not use the web interface do not have permission to access the data. Click **OK**.

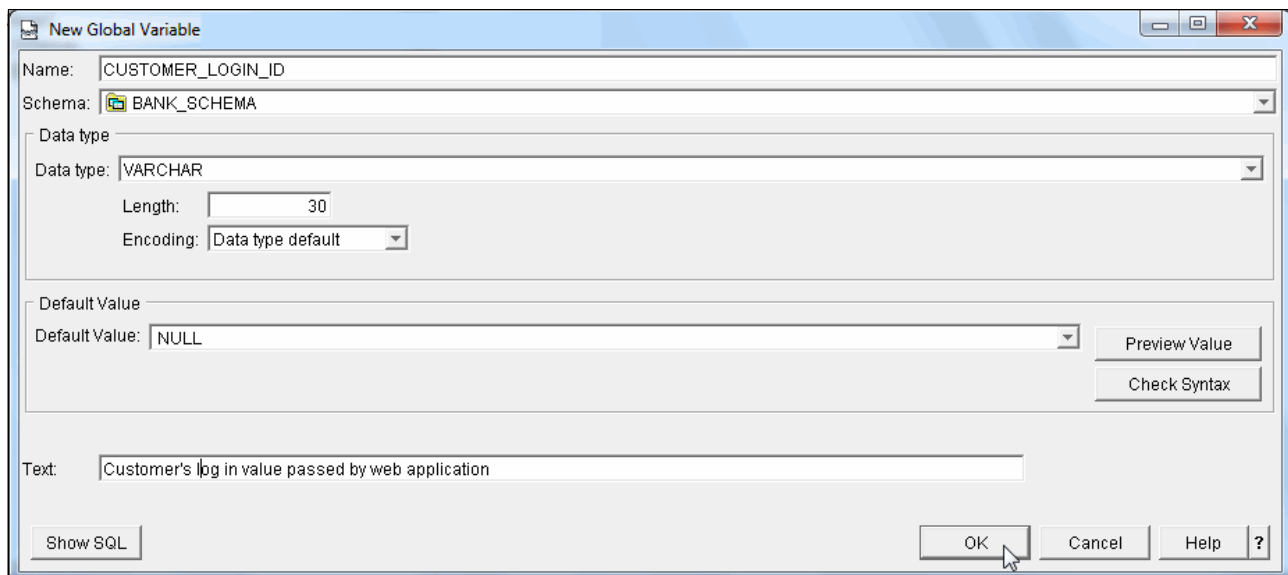


Figure 4-25 Creating a global variable called CUSTOMER_LOGIN_ID

- Now that the global variable is created, assign permissions to the variable so that it can be set by the program. Right-click the `CUSTOMER_LOGIN_ID` global variable and select **Permissions**, as shown in Figure 4-26.

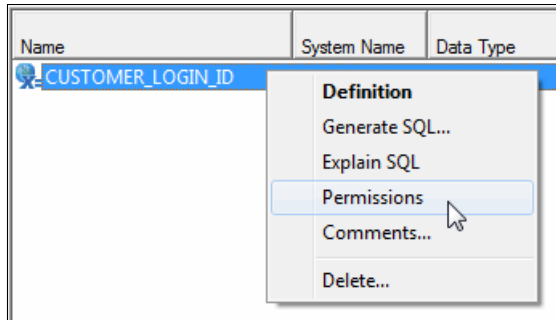


Figure 4-26 Setting permissions on the `CUSTOMER_LOGIN_ID` global variable

- The Permissions window opens, as shown in Figure 4-27. Select **Change** authority for `Webuser` so that the application can set this global variable.

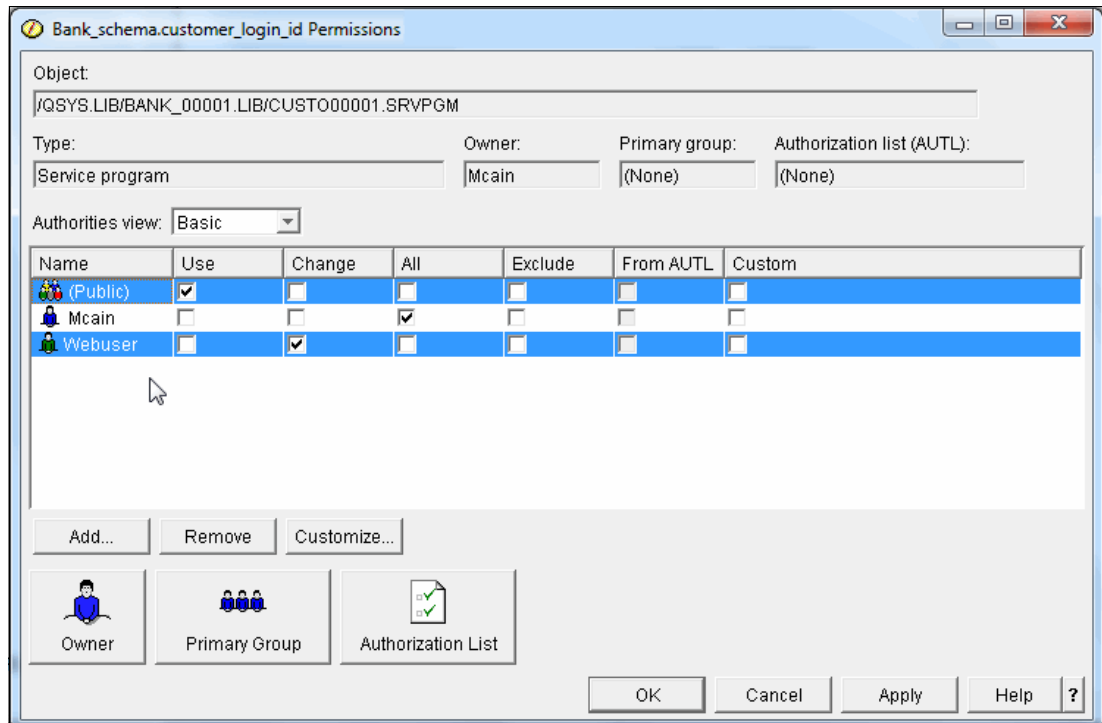


Figure 4-27 Setting change permissions for `Webuser` on the `CUSTOMER_LOGIN_ID` global variable

4.3.5 Defining and creating row permissions

You are now ready to define the row permissions of the tables. Complete the following steps:

1. From the navigation pane of System i Navigator, click **Schemas** → **BANK_SCHEMA**, right-click **Row Permissions**, and select **New** → **Row Permission**, as shown in Figure 4-28.

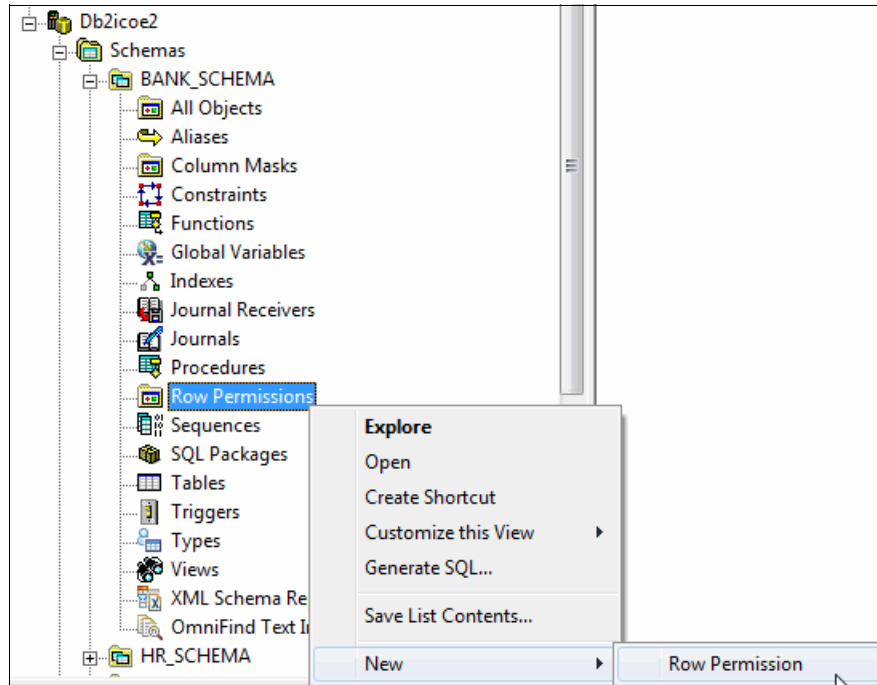


Figure 4-28 Selecting new row permissions

2. The New Row Permission window opens, as shown in Figure 4-29. Enter the information regarding the row permissions on the CUSTOMERS table. This row permission defines what is established in the following policy:

- User profiles that belong to DBE, ADMIN, and TELLER group profiles can see all the rows.
- User profiles that belong to the CUSTOMERS group profile (that is, the WEBUSER user) can see only the rows that match their customer login ID. The login ID value representing the online banking user is passed from the web application to the database by using the global variable CUSTOMER_LOGIN_ID. The permission rule uses a subquery to check whether the global variable matches the CUSTOMER_LOGIN_ID column value in the CUSTOMERS table.
- Any other user profile cannot see any rows at all.

Select the **Enabled** option. Click **OK**.

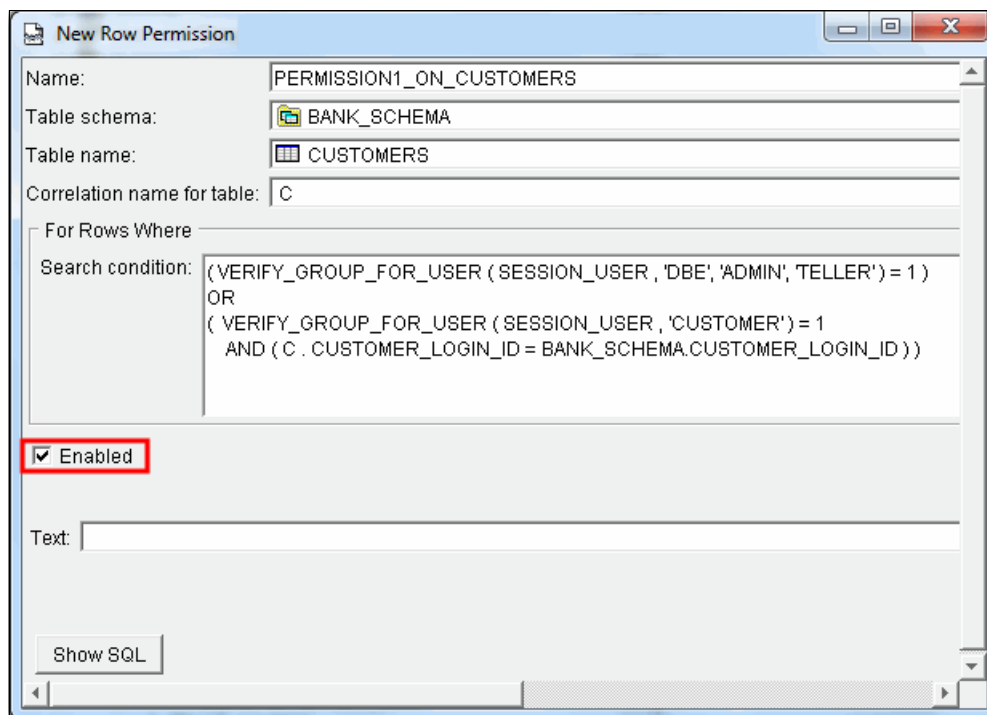


Figure 4-29 New row permissions on the CUSTOMERS table

3. Define the row permissions for the ACCOUNTS table. The New Row Permission window opens, as shown in Figure 4-30. Enter the information regarding the row permissions on the ACCOUNTS table. This row permission defines what is established in the following policy:

- User profiles that belong to DBE, ADMIN and TELLER group profiles can see all the rows.
- User profiles that belong to the CUSTOMERS group profile (that is, the WEBUSER user) can see only the rows that match their customer login ID. The login ID value representing the online banking user is passed from the web application to the database by using the global variable CUSTOMER_LOGIN_ID. The permission rule uses a subquery to check whether the global variable matches the CUSTOMER_LOGIN_ID column value in the CUSTOMERS table.
- Any other user profile cannot see any rows at all.

Select the **Enabled** option. Click **OK**.

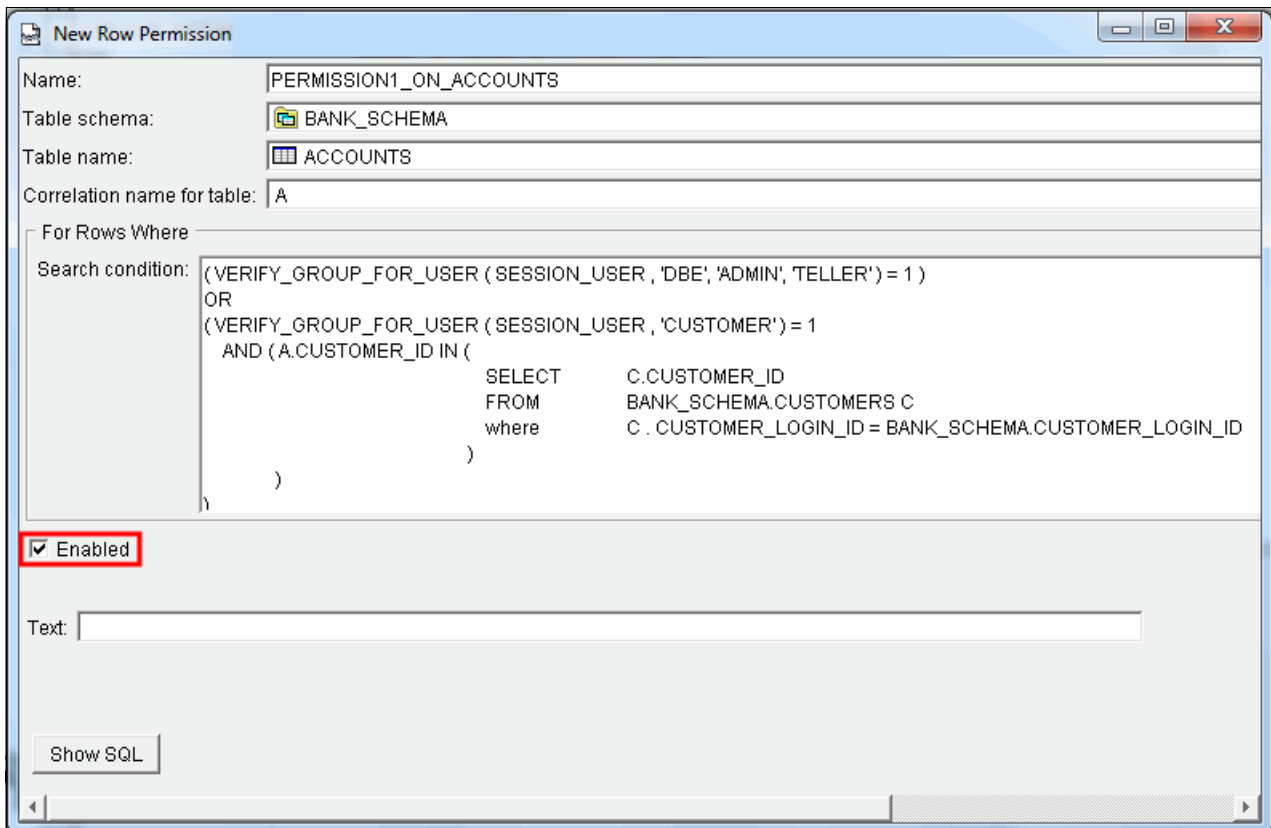


Figure 4-30 New row permissions on the ACCOUNTS table

4. Define the row permissions on the TRANSACTIONS table. The New Row Permission window opens, as shown in Figure 4-31. Enter the information regarding the row permissions on the TRANSACTIONS table. This row permission defines what is established in the following policy:
 - User profiles that belong to DBE, ADMIN, and TELLER group profiles can see all of the rows.
 - User profiles that belong to the CUSTOMERS group profile (that is, the WEBUSER user) can see only the rows that match their customer login ID. The login ID value representing the online banking user is passed from the web application to the database by using the global variable CUSTOMER_LOGIN_ID. The permission rule uses a subquery to check whether the global variable matches the CUSTOMER_LOGIN_ID column value in the CUSTOMERS table.

Note: You must join back to ACCOUNTS and then to CUSTOMERS by using a subquery to check whether the global variable matches CUSTOMER_LOGIN_ID. Also, if the row permission or column mask rule text references another table with RCAC defined, the RCAC for the referenced table is ignored.

- Any other user profile cannot see any rows at all.
- Select the **Enabled** option. Click **OK**.

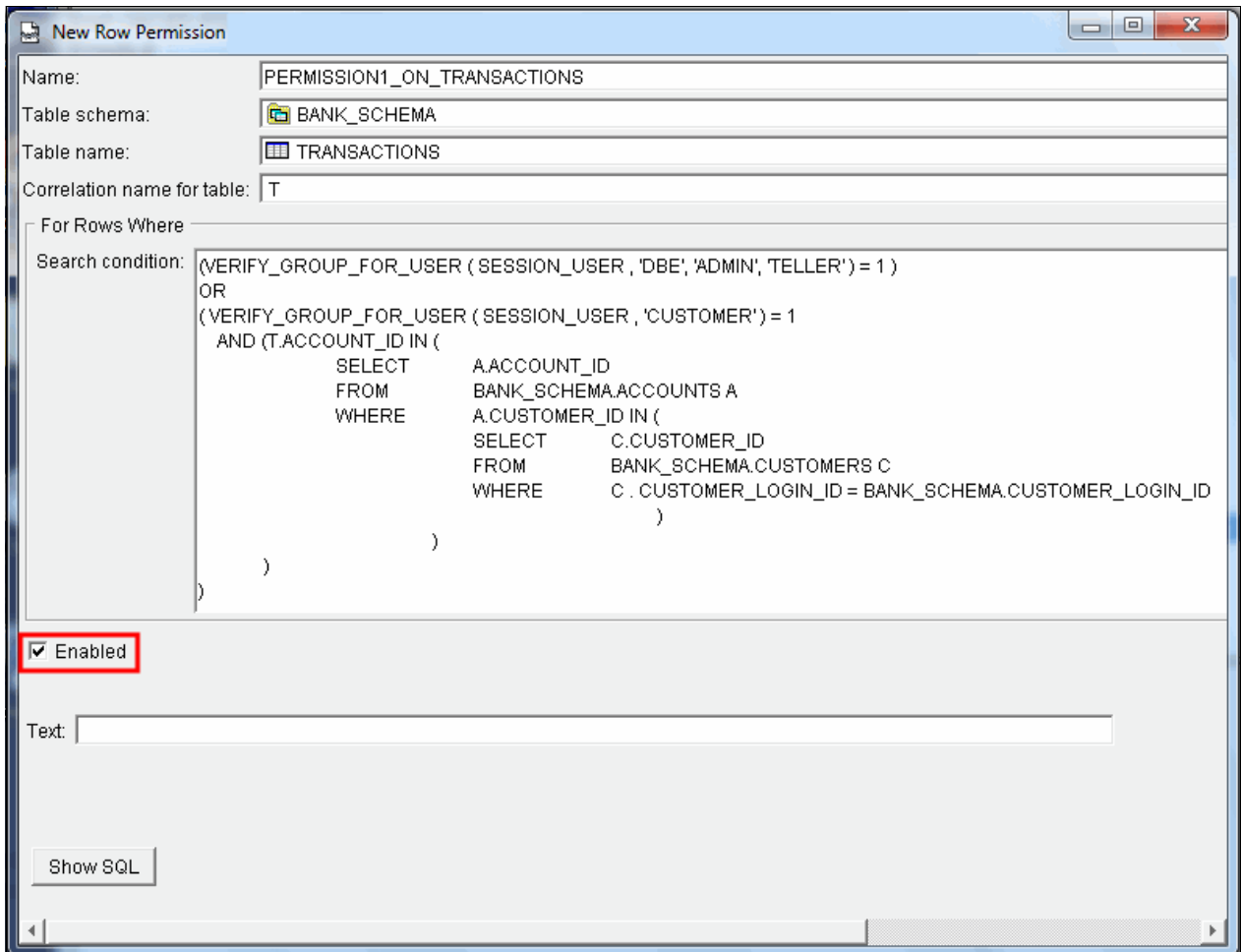


Figure 4-31 New row permissions on the TRANSACTIONS table

- To verify that the row permissions are enabled, from System i Navigator, click **Row Permissions**, as shown in Figure 4-32. The three row permissions are created and enabled.

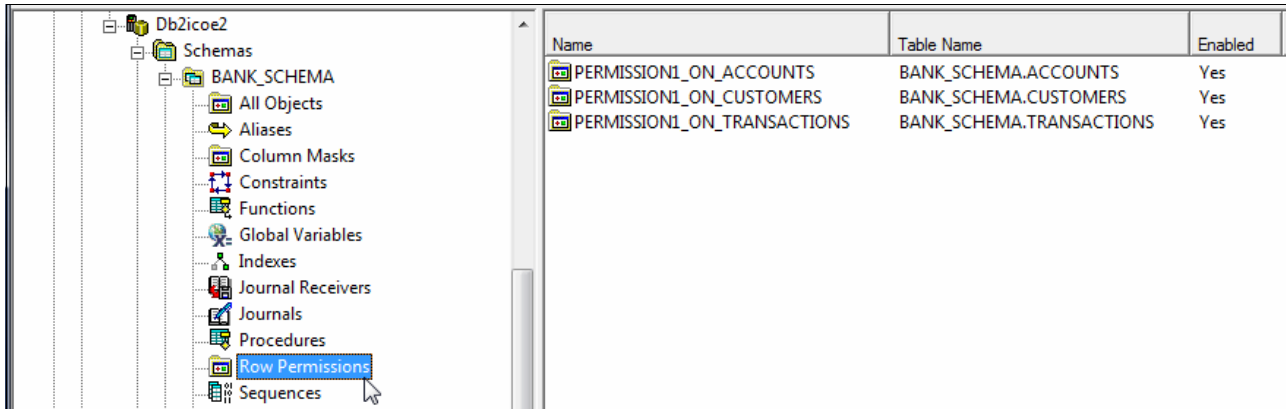


Figure 4-32 List of row permissions on BANK_SCHEMA

4.3.6 Defining and creating column masks

This section defines the masks on the columns. Complete the following steps:

- From the main navigation pane of System i Navigator, click **Schemas** → **BANK_SCHEMA**, right-click **Column Masks**, and select **New** → **Column Mask**, as shown in Figure 4-33.

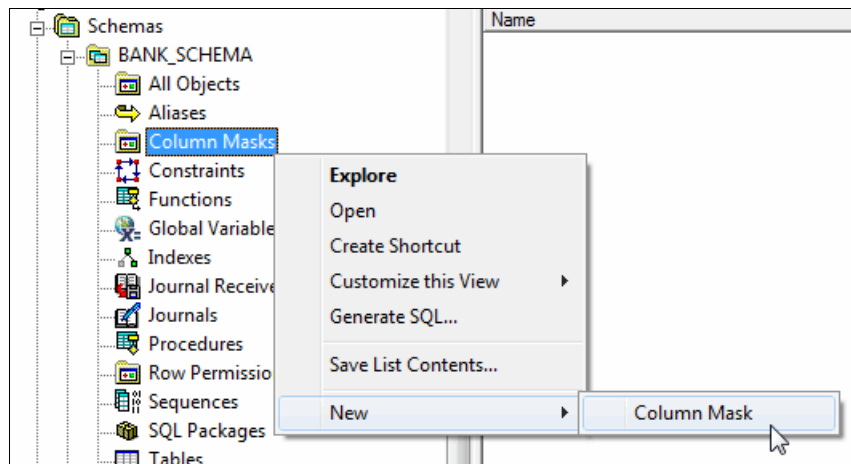


Figure 4-33 Creating a column mask

2. In the New Column Mask window, which is shown in Figure 4-34, enter the following information:
 - Select the CUSTOMERS table on which to create the column mask.
 - Select the Column to mask; in this example, it is CUSTOMER_EMAIL.
 - Define the masking logic depending on the rules that you want to enforce. In this example, either the ADMIN or CUSTOMER group profiles can see the entire email address; otherwise, it is masked to ****@****.
- Select the **Enabled** option. Click **OK**.

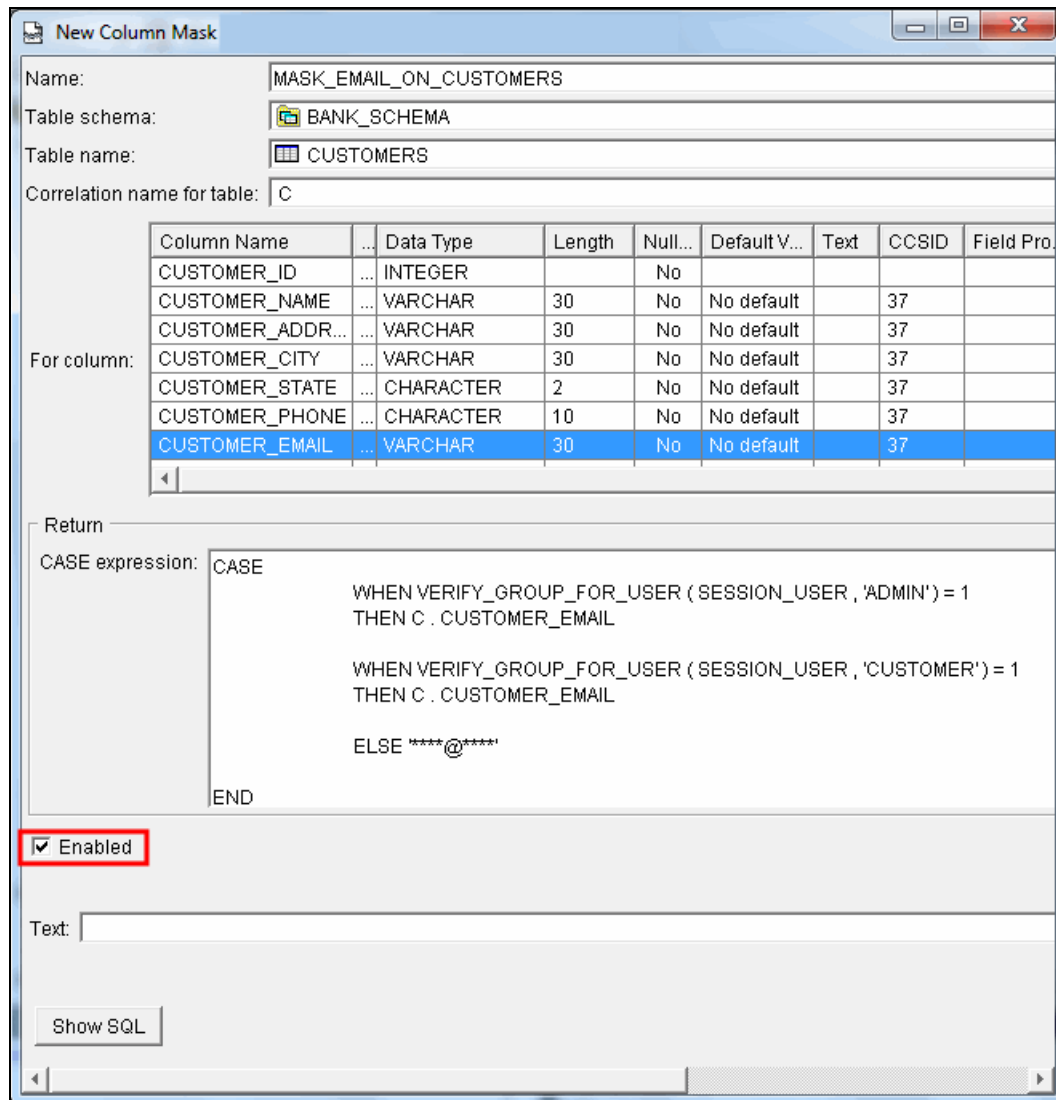


Figure 4-34 Defining a column mask on the CUSTOMERS table

3. Repeat steps 1 on page 58 and 2 to create column masks for the following columns:
 - MASK_DRIVERS_LICENSE_ON_CUSTOMERS
 - MASK_LOGIN_ID_ON_CUSTOMERS
 - MASK_SECURITY_QUESTION_ANSWER_ON_CUSTOMERS
 - MASK_ACCOUNT_NUMBER_ON_ACCOUNTS
 - MASK_SECURITY_QUESTION_ON_CUSTOMERS
 - MASK_TAX_ID_ON_CUSTOMERS

- To verify that the column masks are enabled, from System i Navigator, click **Column Masks**, as shown in Figure 4-35. The seven column masks are created and enabled.

Name	Table Name	Column Name	Enabled
MASK_ACCOUNT_NUMBER_ON_ACCOUNTS	BANK_SCHEMA.ACCOUNTS	ACCOUNT_NUMBER	Yes
MASK_DRIVERS_LICENSE_ON_CUSTOMERS	BANK_SCHEMA.CUSTOMERS	CUSTOMER_DRIVERS_LICENSE_NUMBER	Yes
MASK_EMAIL_ON_CUSTOMERS	BANK_SCHEMA.CUSTOMERS	CUSTOMER_EMAIL	Yes
MASK_LOGIN_ID_ON_CUSTOMERS	BANK_SCHEMA.CUSTOMERS	CUSTOMER_LOGIN_ID	Yes
MASK_SECURITY_QUESTION_ANSWER_ON_CUSTOMERS	BANK_SCHEMA.CUSTOMERS	CUSTOMER_SECURITY_QUESTION_ANSWER	Yes
MASK_SECURITY_QUESTION_ON_CUSTOMERS	BANK_SCHEMA.CUSTOMERS	CUSTOMER_SECURITY_QUESTION	Yes
MASK_TAX_ID_ON_CUSTOMERS	BANK_SCHEMA.CUSTOMERS	CUSTOMER_TAX_ID	Yes

Figure 4-35 List of column masks on BANK_SCHEMA

4.3.7 Restricting the inserting and updating of masked data

This step defines the check constraints that support the column masks to make sure that on INSERTS or UPDATES, data is not written with a masked value. For more information about the propagation of masked data, see 6.8, “Avoiding propagation of masked data” on page 108.

Complete the following steps:

- Create a check constraint on the column CUSTOMER_EMAIL in the CUSTOMERS table. From the navigation pane of System i Navigator, right-click the **CUSTOMERS** table and select **Definition**, as shown Figure 4-36

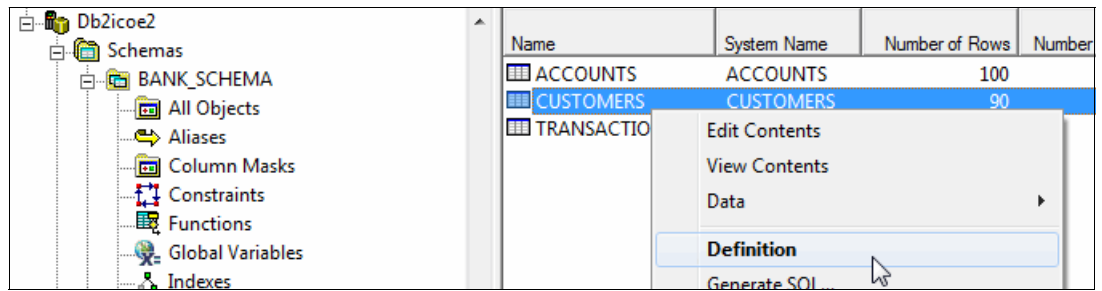


Figure 4-36 Definition of the CUSTOMERS table

- From the CUSTOMERS definition window, click the **Check Constraints** tab and click **Add**, as shown in Figure 4-37.

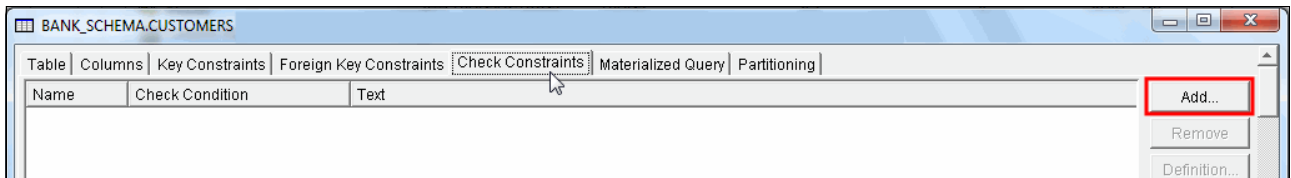


Figure 4-37 Adding a check constraint

3. The New Check Constraint window opens, as shown in Figure 4-38. Complete the following steps:
 - a. Select the **CUSTOMER_EMAIL** column.
 - b. Enter the check constraint condition. In this example, specify CUSTOMER_EMAIL to be different from ****@****, which is the mask value.
 - c. Select the **On update violation, preserve column value** option and click **OK**.

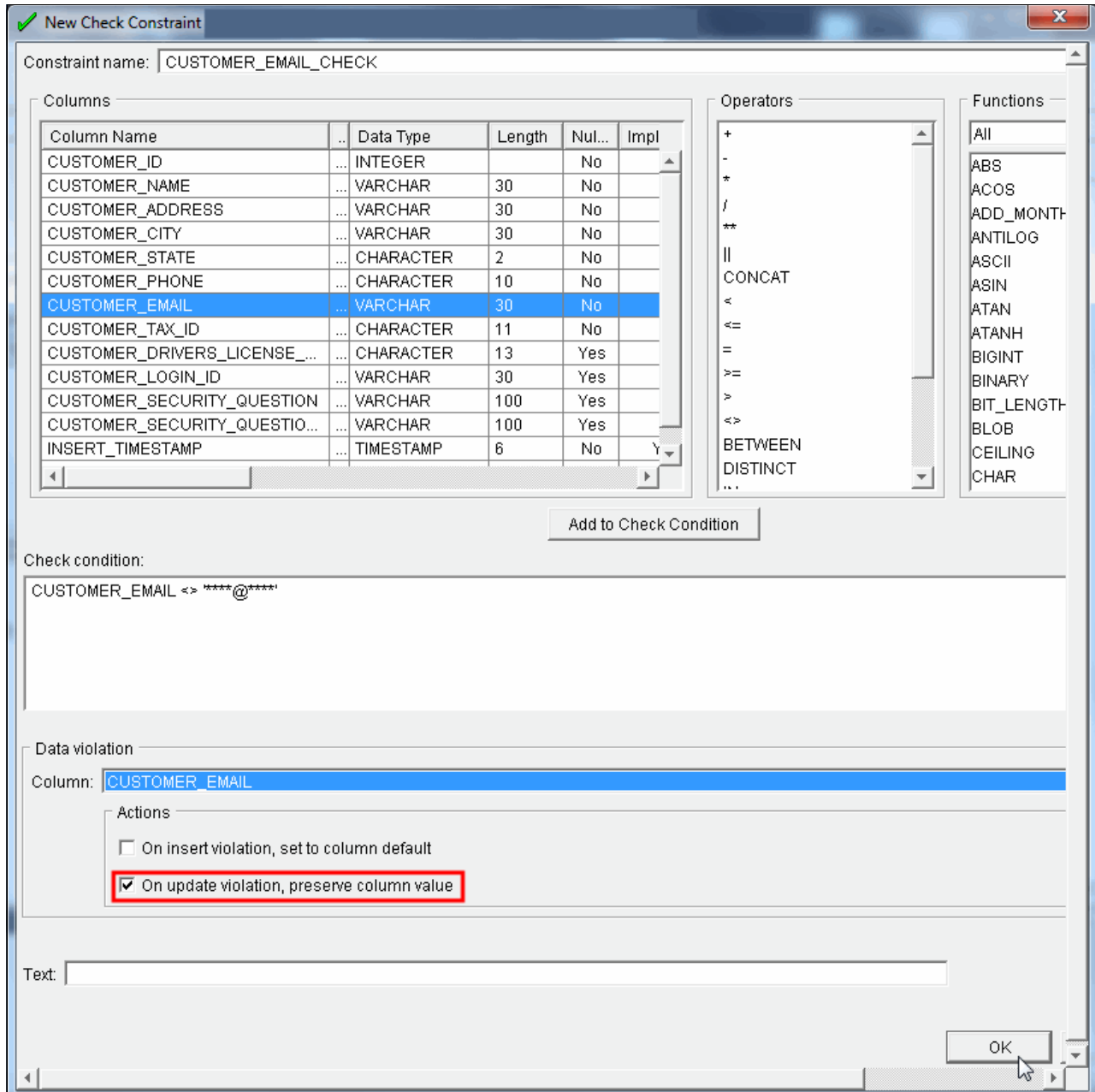


Figure 4-38 Specifying a new check constraint on the CUSTOMERS table

4. Figure 4-39 shows that there is now a check constraint on the CUSTOMERS table that prevents any masked data from being updated to the CUSTOMER_EMAIL column.

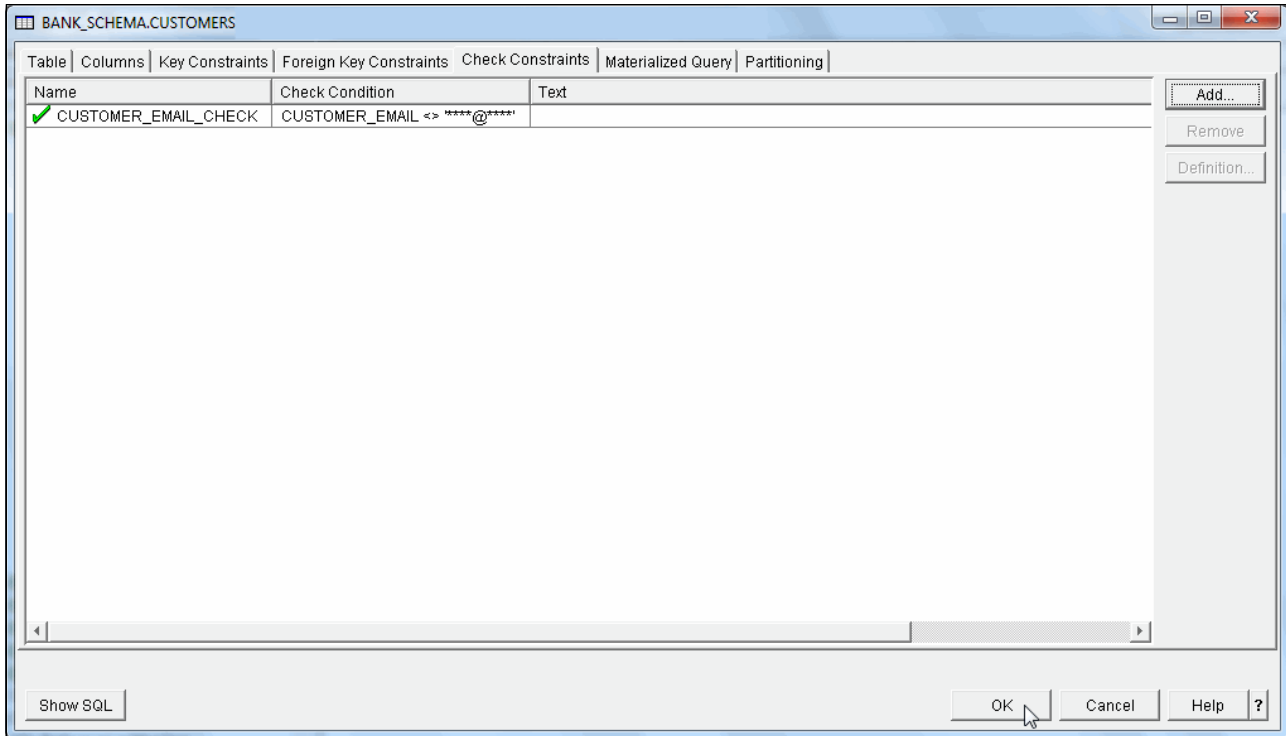


Figure 4-39 Check constraint on the CUSTOMERS table

5. Create all the other check constraints that are associated to each of the masks on the CUSTOMERS table. After this is done, these constraints should look like the ones that are shown in Figure 4-40.

Name	Type	Table Name	Enabled
✓ CUSTOMER_EMAIL_CHECK	Check Constraint	CUSTOMERS	Yes
✓ CUSTOMER_TAX_ID_CHECK	Check Constraint	CUSTOMERS	Yes
✓ CUSTOMER_LOGIN_ID_CHECK	Check Constraint	CUSTOMERS	Yes
✓ CUSTOMER_SECURITY_QUESTION_CHECK	Check Constraint	CUSTOMERS	Yes
✓ CUSTOMER_SECURITY_QUESTION_ANSWER_CHECK	Check Constraint	CUSTOMERS	Yes
✓ CUSTOMER_DRIVERS_LICENSE_CHECK	Check Constraint	CUSTOMERS	Yes
✓ ACCOUNT_NUMBER_CHECK	Check Constraint	ACCOUNTS	Yes
ACCOUNT_CUSTOMER_ID_FK	Foreign Key Constraint	ACCOUNTS	Yes
TRANSACTIONS_ACCOUNT_ID_FK	Foreign Key Constraint	TRANSACTIONS	Yes
ACCOUNT_ID_PK	Primary Key Constraint	ACCOUNTS	Yes
TRANSACTION_ID_PK	Primary Key Constraint	TRANSACTIONS	Yes
CUSTOMER_ID_PK	Primary Key Constraint	CUSTOMERS	Yes
CUSTOMER_LOGIN_ID_UK	Unique Key Constraint	CUSTOMERS	Yes

Figure 4-40 List of check constraints on the CUSTOMERS table

4.3.8 Activating row and column access control

You are now ready to activate RCAC on all three tables in this example. Complete the following steps:

1. Start by enabling RCAC on the CUSTOMERS table. From System i Navigator, right-click the **CUSTOMERS** table and select **Definition**. As shown in Figure 4-41, make sure that you select **Row access control** and **Column access control**. Click **OK**.

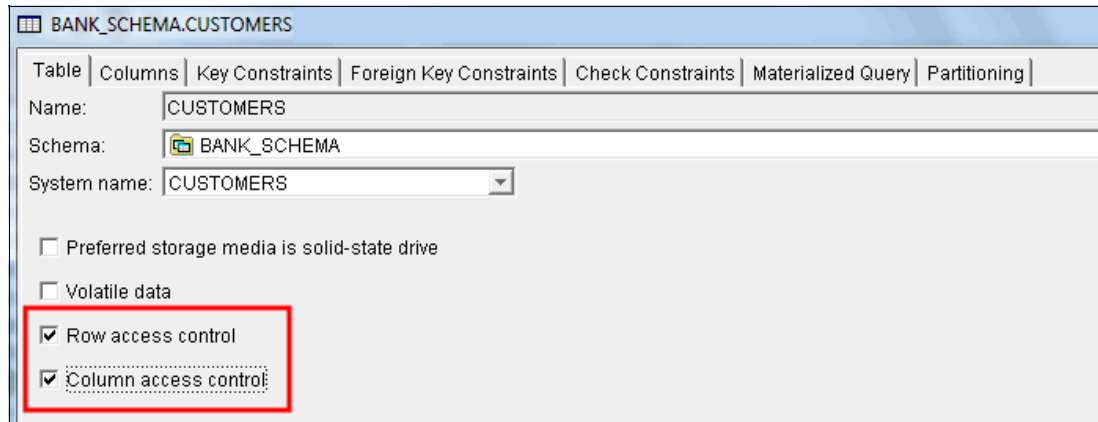


Figure 4-41 Enabling RCAC on the CUSTOMERS table

2. Enable RCAC on the ACCOUNTS table. Right-click the **ACCOUNTS** table and select **Definition**. As shown Figure 4-42, make sure that you select **Row access control** and **Column access control**. Click **OK**.

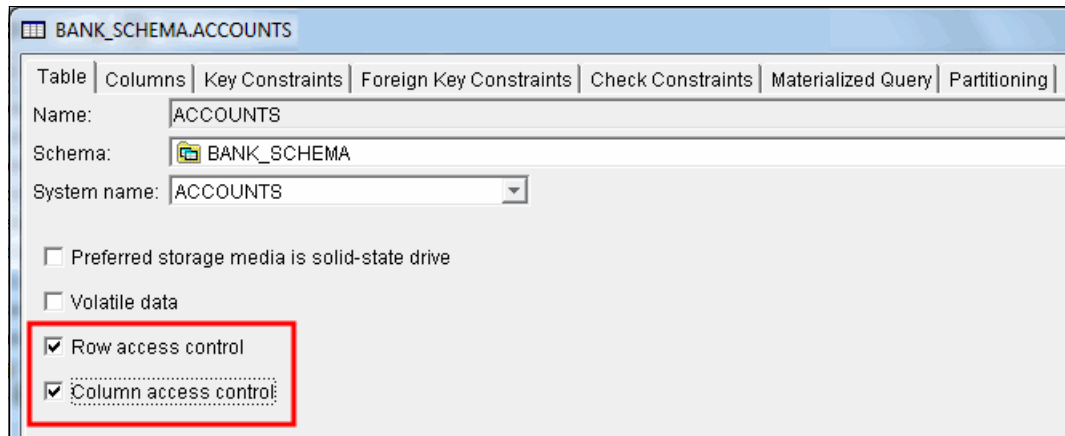


Figure 4-42 Enabling RCAC on ACCOUNTS

3. Enable RCAC on the TRANSACTIONS table. Right-click the **TRANSACTIONS** table and select **Definition**. As shown in Figure 4-43, make sure that you select **Row access control**. Click **OK**.

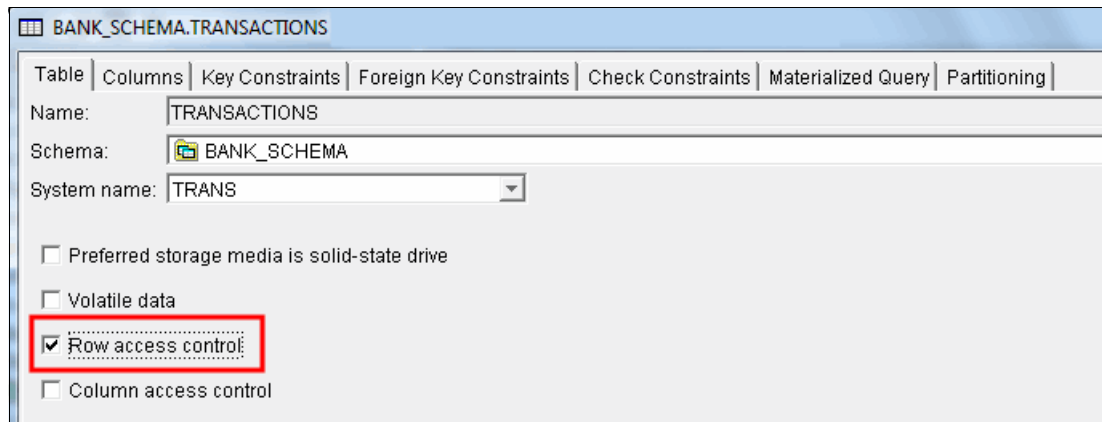


Figure 4-43 Enabling RCAC on TRANSACTIONS

4.3.9 Reviewing row permissions

This section displays all the row permissions after enabling RCAC. Complete the following steps:

1. From System i Navigator, click **Row Permissions**, as shown in Figure 4-44. Three additional Row Permissions are added (QIBM_DEFAULT*). There is one per each row permission.

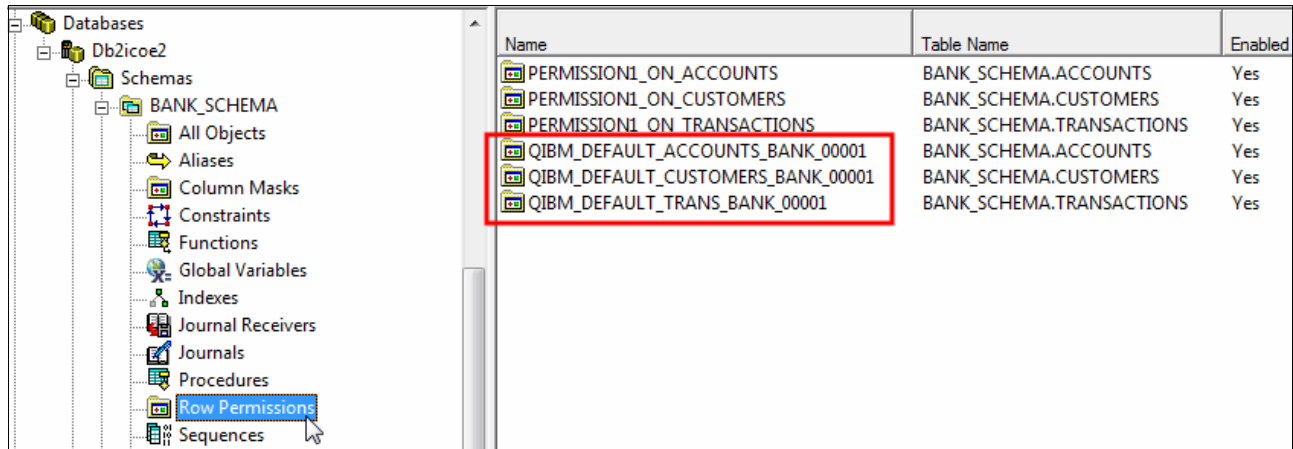
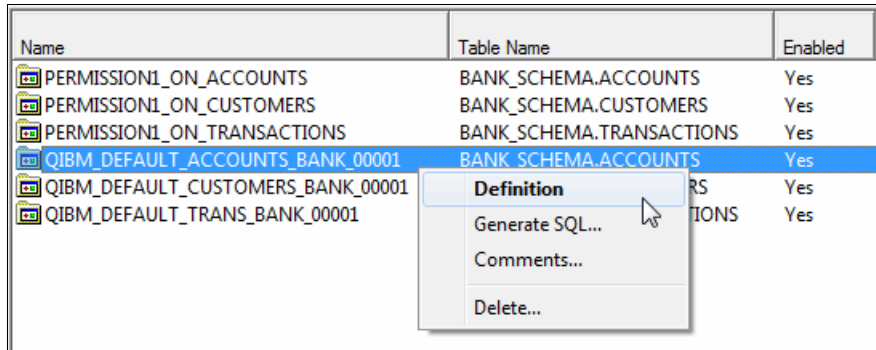


Figure 4-44 Row permissions after enabling RCAC

2. Look at one of the row permission definitions by right-clicking it and selecting **Definition**, as shown in Figure 4-45.



Name	Table Name	Enabled
PERMISSION1_ON_ACCOUNTS	BANK_SCHEMA.ACCOUNTS	Yes
PERMISSION1_ON_CUSTOMERS	BANK_SCHEMA.CUSTOMERS	Yes
PERMISSION1_ON_TRANSACTIONS	BANK_SCHEMA.TRANSACTIONS	Yes
QIBM_DEFAULT_ACCOUNTS_BANK_00001	BANK_SCHEMA.ACCOUNTS	Yes
QIBM_DEFAULT_CUSTOMERS_BANK_00001	BANK_SCHEMA.CUSTOMERS	Yes
QIBM_DEFAULT_TRANS_BANK_00001	BANK_SCHEMA.TRANSACTIONS	Yes

The context menu for the selected row permission (QIBM_DEFAULT_ACCOUNTS_BANK_00001) includes the following options:

- Definition
- Generate SQL...
- Comments...
- Delete...

Figure 4-45 Selecting row permission definition

3. A window opens, as shown in Figure 4-46. Take note of the nonsensical search condition (0=1) of the QIBM_DEFAULT row permission. This permission is ORed with all of the others and it ensures that if someone does not meet any of the criteria from the row permission then this condition is tested, and because it is false the access is denied.

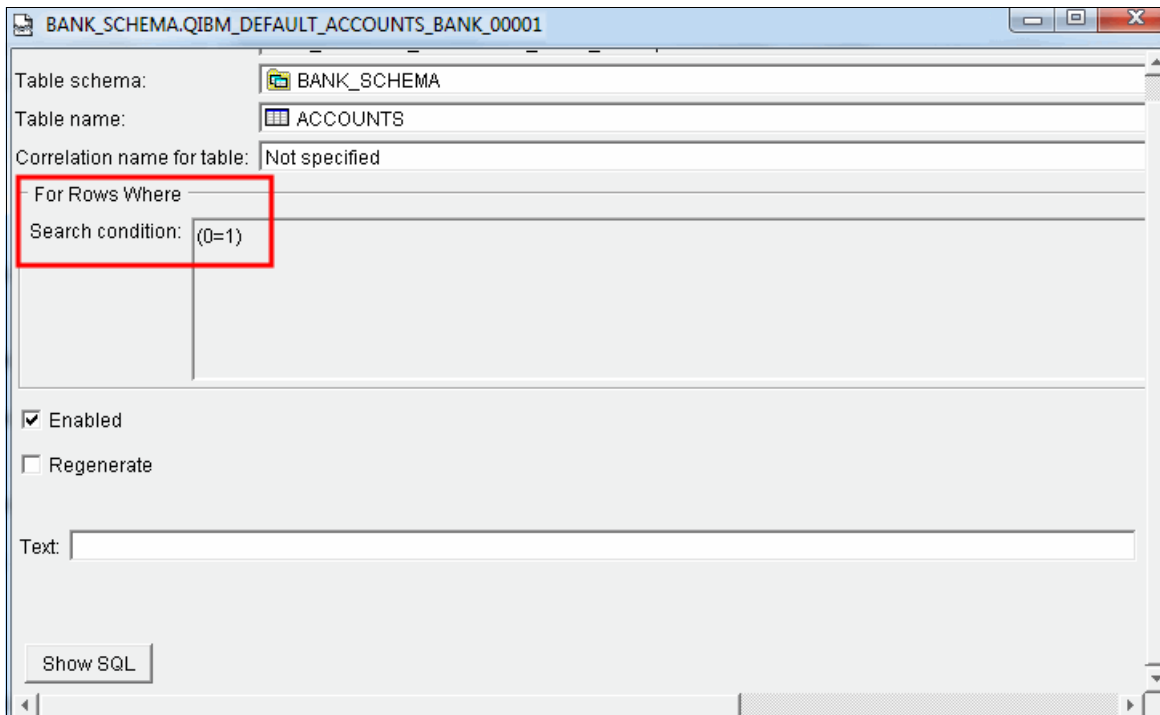


Figure 4-46 Search condition of the QIBM_DEFAULT row permission

4.3.10 Demonstrating data access with RCAC

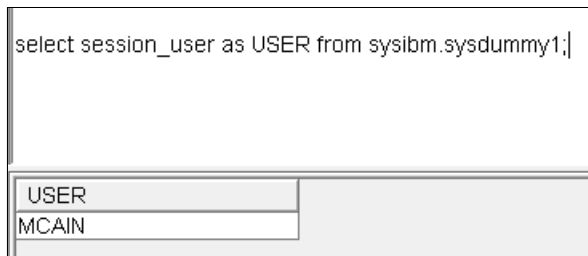
You are now ready to test the RCAC definitions. Run the following SQL statements with each type of user (DBE, SECURITY, TELLER, ADMIN, and WEBUSER):

- ▶ A **SELECT** statement that returns the SESSION_USER.
- ▶ A **SELECT** statement that counts the customers from the CUSTOMER table. There are 90 customers in the CUSTOMER table.
- ▶ A simple **SELECT** statement that returns the following output from the CUSTOMERS table ordered by customer_name:
 - customer_id
 - customer_name
 - customer_email
 - customer_tax_id
 - customer_drivers_license_number

Data access for a DBE user with RCAC

To test a DBE (MCAIN) user, complete the following steps:

1. Confirm that the user is the user of the session by running the first SQL statement, as shown in Figure 4-47. In this example, MCAIN is the DBE user.

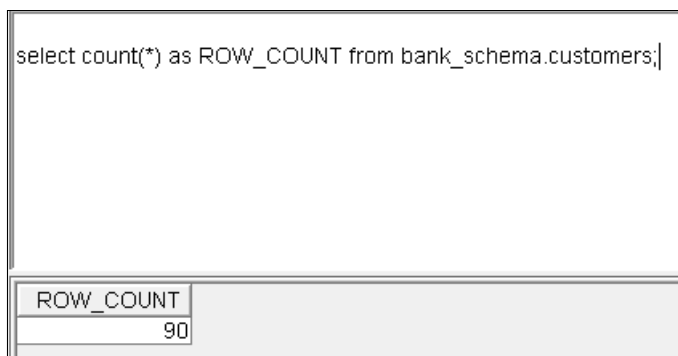


```
select session_user as USER from sysibm.sysdummy1;
```

USER
MCAIN

Figure 4-47 DBE session user

2. The number of rows that the DBE user MCAIN can see is shown in Figure 4-48.



```
select count(*) as ROW_COUNT from bank_schema.customers;
```

ROW_COUNT
90

Figure 4-48 Number of rows that DBE user can see in the CUSTOMERS table

- The result of the third SQL statement is shown in Figure 4-49. Note the masked columns. User MCAIN can see all the rows in the CUSTOMERS table, but there are some columns where the result is masked.

```

select      customer_id,
           customer_name,
           customer_email,
           customer_tax_id,
           customer_drivers_license_number
from        bank_schema.customers
order by    customer_name;

```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_TAX_ID	CUSTOMER_DRIVERS_LICENSE_NUMBER
1	Adam O. Olsen	****@****	XXX-XX-XXXX	*****
2	Aimee Compton	****@****	XXX-XX-XXXX	*****
3	Amery W. Wright	****@****	XXX-XX-XXXX	*****
4	Anika S. Holmes	****@****	XXX-XX-XXXX	*****
5	Ann Norton	****@****	XXX-XX-XXXX	*****
6	Ashely R. Collins	****@****	XXX-XX-XXXX	*****
7	Aspen Frost	****@****	XXX-XX-XXXX	*****
8	Athena R. Burt	****@****	XXX-XX-XXXX	*****
9	Aurelia Hebert	****@****	XXX-XX-XXXX	*****
10	Barry X. Molina	****@****	XXX-XX-XXXX	*****
11	Beck B. Thomas	****@****	XXX-XX-XXXX	*****
12	Benedict Y. Nolan	****@****	XXX-XX-XXXX	*****
13	Bernard K. Bowen	****@****	XXX-XX-XXXX	*****
14	Caldwell Floyd	****@****	XXX-XX-XXXX	*****
15	Carson Curtis	****@****	XXX-XX-XXXX	*****
16	Chloe U. Madden	****@****	XXX-XX-XXXX	*****
17	Ciaran Atkins	****@****	XXX-XX-XXXX	*****
18	Colton Molina	****@****	XXX-XX-XXXX	*****
19	Conan W. Maynard	****@****	XXX-XX-XXXX	*****
20	Demetrius R. Barnes	****@****	XXX-XX-XXXX	*****
21	Drake Preston	****@****	XXX-XX-XXXX	*****
22	Echo Pacheco	****@****	XXX-XX-XXXX	*****
23	Edan Henry	****@****	XXX-XX-XXXX	*****
24	Elvis Grant	****@****	XXX-XX-XXXX	*****
25	Emily M. Carpenter	****@****	XXX-XX-XXXX	*****
26	Evelyn Y. Velez	****@****	XXX-XX-XXXX	*****

Figure 4-49 SQL statement that is run by the DBE user with masked columns

Data access for SECURITY user with RCAC

To test a SECURITY user, complete the following steps:

- Confirm that the user is the user of the session by running the first SQL statement, as shown in Figure 4-50. In this example, SECURITY is the security officer.

```

select session_user as USER from sysibm.sysdummy1;

```

USER
SECURITY

Figure 4-50 SECURITY session user

2. The number of rows in the CUSTOMERS table that the security officer can see is shown in Figure 4-51. The security officer cannot see any data at all.

```
select count(*) as ROW_COUNT from bank_schema.customers;
```

ROW_COUNT
0

Figure 4-51 Number of rows that the security officer can see in the CUSTOMERS table

3. The result of the third SQL statement is shown in Figure 4-52. Note the empty set that is returned to the security officer.

```
select      customer_id,
           customer_name,
           customer_email,
           customer_tax_id,
           customer_drivers_license_number
from        bank_schema.customers
order by   customer_name;
```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_TAX_ID	CUSTOMER_DRIVERS_LICENSE_NUMBER
Empty Set				

Figure 4-52 SQL statement that is run by the SECURITY user - no results

Data access for TELLER user with RCAC

To test a Teller (TQSPENCER) user, complete the following steps:

1. Confirm that the TELLER user is the user of the session by running the first SQL statement, as shown in Figure 4-53. In this example, TQSPENCER is a TELLER user.

```
select session_user as USER from sysibm.sysdummy1;
```

USER
TQSPENCER

Figure 4-53 TELLER session user

- The number of rows in the CUSTOMERS table that the TELLER user can see is shown in Figure 4-54. The TELLER user can see all the rows.

```
select count(*) as ROW_COUNT from bank_schema.customers;
```

ROW_COUNT
90

Figure 4-54 Number of rows that the TELLER user can see in the CUSTOMERS table

- The result of the third SQL statement is shown in Figure 4-55. Note the masked columns. The TELLER user, TQSPENSER, can see all the rows, but there are some columns where the result is masked.

```
select customer_id,
customer_name,
customer_email,
customer_tax_id,
customer_drivers_license_number
from bank_schema.customers
order by customer_name;
```

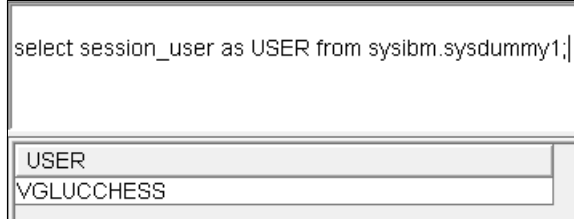
CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_TAX_ID	CUSTOMER_D
1	Adam O. Olsen	****@****	XXX-XX-5327	Q58013011205
2	Aimee Compton	****@****	XXX-XX-8075	R10041455951
3	Amery W. Wright	****@****	XXX-XX-5275	W6863444901
4	Anika S. Holmes	****@****	XXX-XX-8863	W3738994896
5	Ann Norton	****@****	XXX-XX-9303	B13020051925
6	Ashely R. Collins	****@****	XXX-XX-4047	Y37565756164
7	Aspen Frost	****@****	XXX-XX-0596	Y45436122522
8	Athena R. Burt	****@****	XXX-XX-6877	E71527862628
9	Aurelia Hebert	****@****	XXX-XX-0744	A66926929321
10	Barry X. Molina	****@****	XXX-XX-1603	A79416938734
11	Beck B. Thomas	****@****	XXX-XX-6624	X80729251958
12	Benedict Y. Nolan	****@****	XXX-XX-9761	A09840028524
13	Bernard K. Bowen	****@****	XXX-XX-0702	Q13970237937
14	Caldwell Floyd	****@****	XXX-XX-1099	Y62788281375
15	Carson Curtis	****@****	XXX-XX-5070	V07571751347
16	Chloe U. Madden	****@****	XXX-XX-0282	V86122273009
17	Ciaran Atkins	****@****	XXX-XX-8993	G16501583927
18	Colton Molina	****@****	XXX-XX-0370	R34062628388
19	Conan W. Maynard	****@****	XXX-XX-8181	G02134089968
20	Demetrius R. Barnes	****@****	XXX-XX-1341	U22499911780
21	Drake Preston	****@****	XXX-XX-2231	Y90980776375
22	Echo Pacheco	****@****	XXX-XX-5142	A00960253436
23	Edan Henry	****@****	XXX-XX-8560	A78977129664
24	Elvis Grant	****@****	XXX-XX-1605	C31658926178

Figure 4-55 SQL statement that is run by the TELLER user with masked columns

Data access for ADMIN user with RCAC

To test an ADMIN (VGLUCCHESS) user, complete the following steps:

1. Confirm that the ADMIN user is the user of the session by running the first SQL statement, as shown in Figure 4-56. In this example, VGLUCCHESS is an ADMIN user.

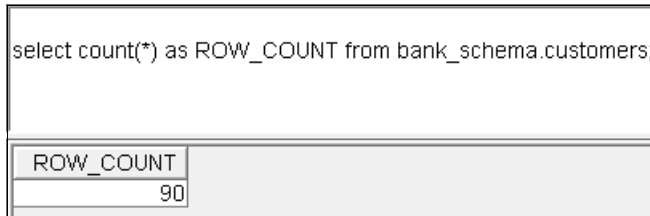


```
select session_user as USER from sysibm.sysdummy1;
```

USER
VGLUCCHESS

Figure 4-56 ADMIN session user

2. The number of rows that the ADMIN user can see is shown in Figure 4-57. The ADMIN user can see all the rows.



```
select count(*) as ROW_COUNT from bank_schema.customers;
```

ROW_COUNT
90

Figure 4-57 Number of rows that the ADMIN can see in the CUSTOMERS table

- The result of the third SQL statement is shown in Figure 4-58. There are no masked columns.

```

select      customer_id,
           customer_name,
           customer_email,
           customer_tax_id,
           customer_drivers_license_number
from        bank_schema.customers
order by    customer_name;

```

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_TAX_ID	CUSTOMER_DRIV
1	Adam O. Olsen	Adam_O_Olsen@gmail.com	451-30-5327	Q580130112053
2	Aimee Compton	Aimee_Compton@mail.net	146-92-8075	R100414559519
3	Amery W. Wright	Amery_W_Wright@mail.net	536-77-5275	W686344490177
4	Anika S. Holmes	Anika_S_Holmes@mail.net	416-22-8863	W373899489631
5	Ann Norton	Ann_Norton@mail.net	770-27-9303	B130200519256
6	Ashely R. Collins	Ashely_R_Collins@gmail.com	377-14-4047	Y375657561646
7	Aspen Frost	Aspen@mail.net	964-99-0596	Y454361225229
8	Athena R. Burt	Athen@mail.net	723-23-6877	E715278626289
9	Aurelia Hebert	Aurelia_Hebert@gmail.com	202-89-0744	A669269293219
10	Barry X. Molina	Barry@mail.net	490-90-1603	A794169387344
11	Beck B. Thomas	Beck_B_Thomas@mail.net	554-57-6624	X807292519585
12	Benedict Y. Nolan	Benedict_Y_Nolan@gmail.com	640-52-9761	A098400285247
13	Bernard K. Bowen	Bernard_K_Bowen@mail.net	229-22-0702	Q139702379378
14	Caldwell Floyd	Caldw@mail.net	332-77-1099	Y627882813752
15	Carson Curtis	Carson_Curtis@gmail.com	157-58-5070	V075717513479
16	Chloe U. Madden	Chloe_U_Madden@gmail.com	799-58-0282	V861222730096
17	Ciaran Atkins	Ciaran_Atkins@gmail.com	801-17-8993	G165015839276
18	Colton Molina	Colto@mail.net	141-95-0370	R340626283882
19	Conan W. Maynard	Conan_W_Maynard@gmail.com	798-23-8181	G021340899699
20	Demetrius R. Barnes	Demetrius_R_Barnes@gmail.com	350-11-1341	U224999117805
21	Drake Preston	Drake_Preston@gmail.com	930-45-2231	Y909807763757
22	Echo Pacheco	Echo_Pacheco@gmail.com	578-38-5142	A009602534360
23	Edan Henry	Edan_Henry@gmail.com	774-89-8560	A789771296645

Figure 4-58 SQL statement that is run by the ADMIN user - no masked columns

Data access for WEBUSER user with RCAC

To test a CUSTOMERS (WEBUSER) user that accesses the database by using the web application, complete the following steps:

- Confirm that the user is the user of the session by running the first SQL statement, as shown in Figure 4-59. In this example, WEBUSER is a CUSTOMER user.

```

select session_user as USER from sysibm.sysdummy1;

```

USER
WEBUSER

Figure 4-59 WEBUSER session user

2. A global variable (CUSTOMER_LOGIN_ID) is set by the web application and then is used to check the row permissions. Figure 4-60 shows setting the global variable by using the customer login ID.

```
set bank_schema.customer_login_id = 'KLD72CQR8JG';
```

← set by web application


```
select bank_schema.customer_login_id as CUSTOMER_LOGIN_ID from SYSIBM.SYSDUMMY1;
```

CUSTOMER_LOGIN_ID
KLD72CQR8JG

Figure 4-60 Setting the global variable CUSTOMER_LOGIN_ID

3. Verify that the global variable was set with the correct value by clicking the **Global Variable** tab, as shown in Figure 4-61.

```
set bank_schema.customer_login_id = 'KLD72CQR8JG';
```

Name	Value
 BANK_SCHEMA.CUSTOMER_LOGIN_ID	'KLD72CQR8JG'

Messages **Global Variables**

Figure 4-61 Viewing the global variable value

4. The number of rows that the WEBUSER can see is shown in Figure 4-62. This user can see only the one row that belongs to his web-based user ID.

```
select count(*) as ROW_COUNT from bank_schema.customers;
```

ROW_COUNT
1

Figure 4-62 Number of rows that the WEBUSER can see in the CUSTOMERS table

5. The result of the third SQL statement is shown in Figure 4-63. There are no masked columns, and the user can see only one row, which is the user's own row.

<pre> select customer_id, customer_name, customer_email, customer_tax_id, customer_drivers_license_number from bank_schema.customers order by customer_name; </pre>				
CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_TAX_ID	CUSTOMER_DRIVERS_LICENSE_NUMBER
1	Adam O. Olsen	Adam_O_Olsen@gmail.com	451-30-5327	Q580130112053

Figure 4-63 SQL statement that is run by WEBUSER - no masked columns

Other examples of data access with RCAC

To run an SQL statement that lists all the accounts and current balance by customer, complete the following steps:

1. Run the SQL statement that is shown in Figure 4-64 using the WEBUSER user profile. The SQL statement has no WHERE clause, but the WEBUSER can see only his accounts.

<pre> -- Obtain list of accounts and current balance by customer select c.customer_name, a.account_number, a.account_current_balance from bank_schema.customers c inner join bank_schema.accounts a on (c.customer_id = a.customer_id) order by a.account_number; </pre>			
CUSTOMER_NAME	ACCOUNT_NUMBER	ACCOUNT_CURRENT_BALANCE	
Adam O. Olsen	ES6297776393893751651945	2188.10	
Adam O. Olsen	MD6094870133007960377717	3178.88	
Adam O. Olsen	RO25ZOOZ9584143366592904	3104.36	

Figure 4-64 List of accounts and current balance by customer using the WEBUSER user profile

2. Figure 4-65 shows running a more complex SQL statement that calculates transaction total by account for year and quarter. Run this statement using the WEBUSER profile. The SQL statement has no **WHERE** clause, but the WEBUSER user can see only his transactions.

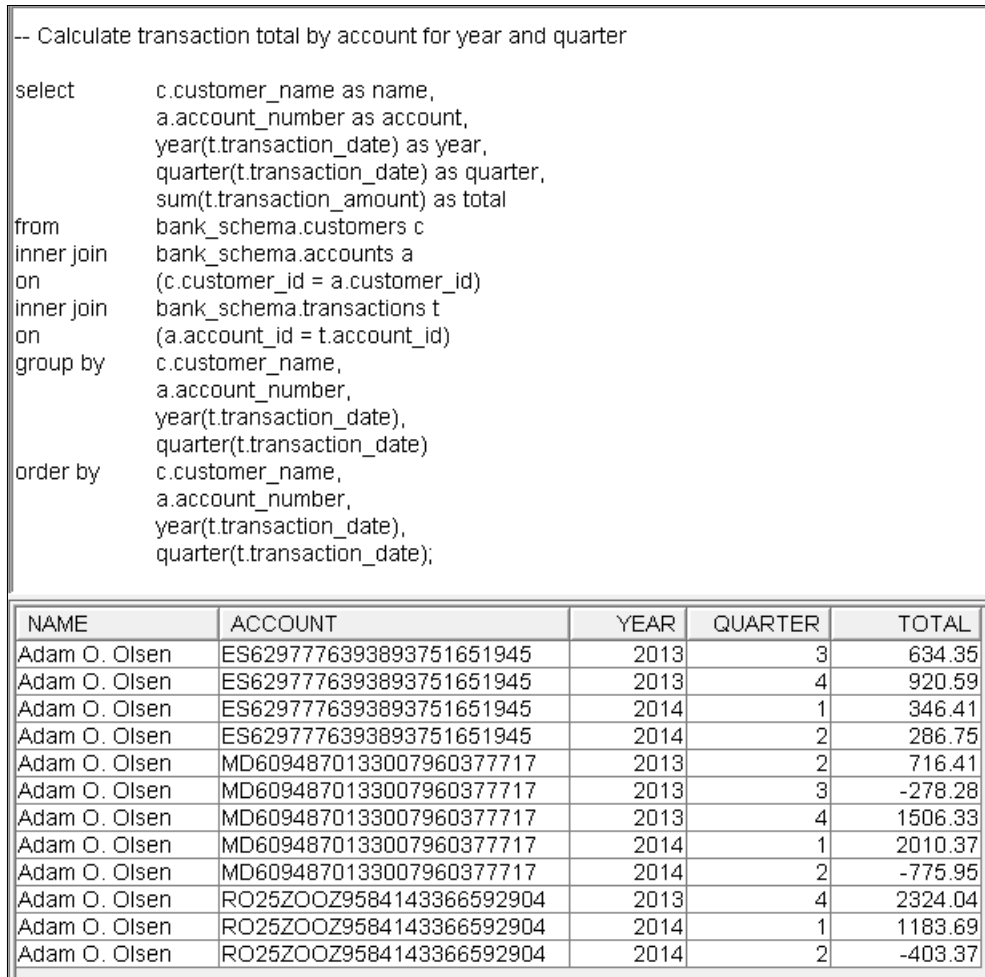


Figure 4-65 Calculate transaction total by account for year and quarter using the WEBUSER profile

- Run the same SQL statement that lists the accounts and current balance by customer, but use a TELLER user profile. The result of this SQL statement is shown in Figure 4-66. The TELLER user can see all the rows in the CUSTOMERS table.

```

-- Obtain list of accounts and current balance by customer

select      c.customer_name,
            a.account_number,
            a.account_current_balance
from        bank_schema.customers c
inner join  bank_schema.accounts a
on         (c.customer_id = a.customer_id)
order by   a.account_number;

```

CUSTOMER_NAME	ACCOUNT_NUMBER	ACCOUNT_CURRENT_BALANCE
Bernard K. Bowen	AD0463839023149964101565	0.00
Evelyn Y. Velez	AD5500285175441299225701	0.00
Barry X. Molina	AD8076223565008050426825	4509.64
Evelyn Y. Velez	AL58222676098746569207726336	0.00
Emily M. Carpenter	AL65576346522724125983621540	0.00
Griffith K. Houston	AT025553951060191435	0.00
Edan Henry	AT229233541853170130	0.00
Athena R. Burt	AZ33284179664581738937389969	3406.87
Fritz Mendoza	AZ40294326545209152260806426	0.00
Beck B. Thomas	AZ96552558233406004227451634	4356.33
Griffith K. Houston	BA611667038494848539	0.00
Amery W. Wright	BG06SIIIE89215785130980	3368.47
Benedict Y. Nolan	BG52PXTQ88241475028069	0.00
Amery W. Wright	BH51276149127304731856	1948.63
Griffin T. Nieves	CH9472267026271632989	0.00
Farrah Gray	CR7896153784330877615	0.00
Ciaran Atkins	CR9383751672140421147	0.00
Athena R. Burt	CZ2483268865184555754259	2396.09
Beck B. Thomas	CZ2763286613152842021101	2791.32
Griffin T. Nieves	CZ6117397346176186517859	0.00
Grady F. Curry	DE98756732171571929510	0.00
Evelyn Y. Velez	DK0486118222853611	0.00
Ferris Livingston	DK1538983647512967	0.00
Walter P. Helms	DK2000500066400071	1000.00

Figure 4-66 List of accounts and current balance by customer using a TELLER user profile

4.3.11 Query implementation with RCAC activated

This section looks at some other interesting information that is related to RCAC by comparing the access plans of the same SQL statement without RCAC and with RCAC. This example uses Visual Explain and runs an SQL statement that lists the accounts and current balance by customer.

Complete the following steps:

1. Figure 4-67 shows the SQL statement in Visual Explain ran with no RCAC. The implementation of the SQL statement is a two-way join, which is exactly what the SQL statement is doing.

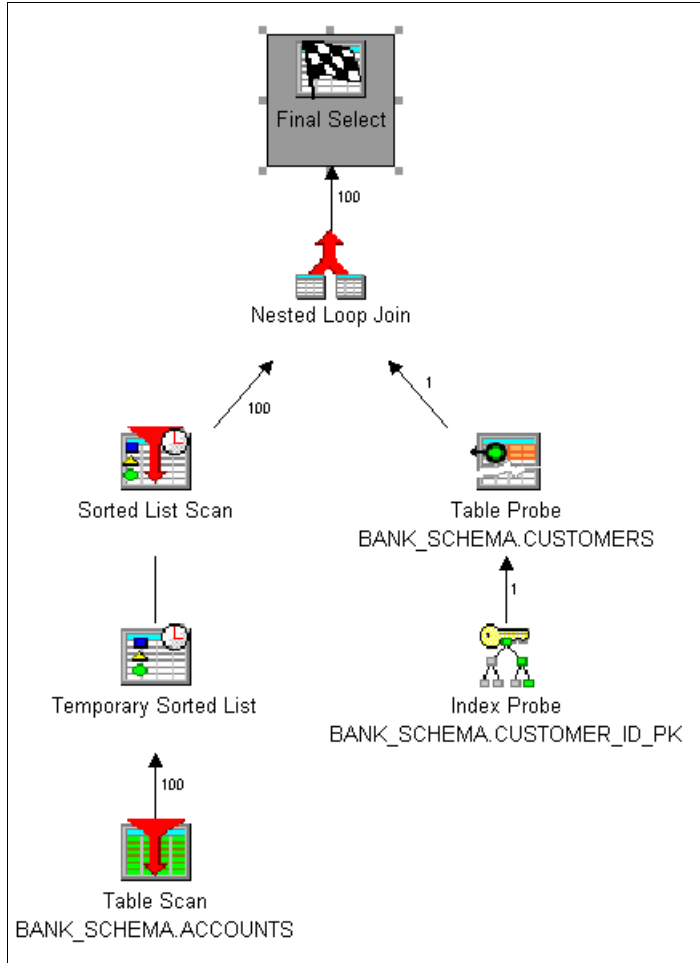


Figure 4-67 Visual Explain with no RCAC enabled

- 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.

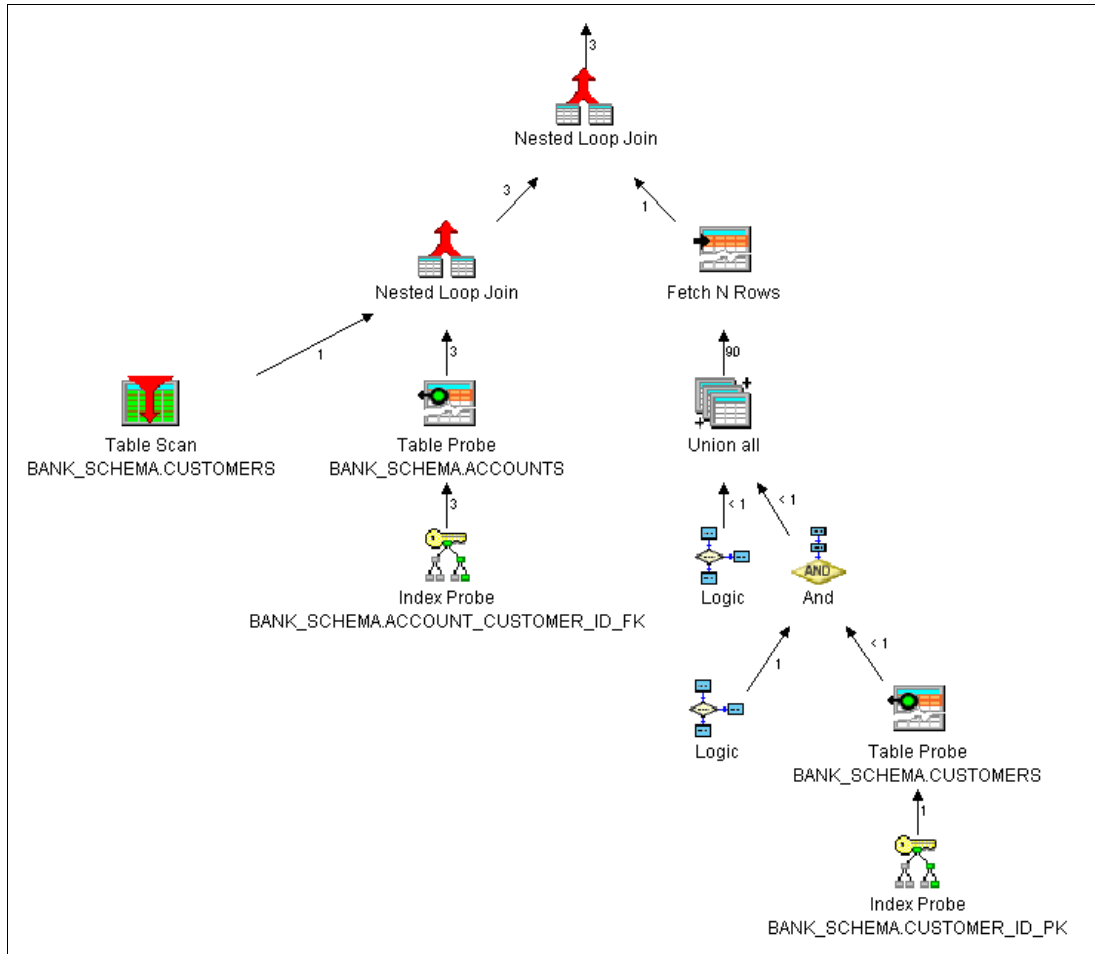


Figure 4-68 Visual Explain with RCAC enabled

- 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.

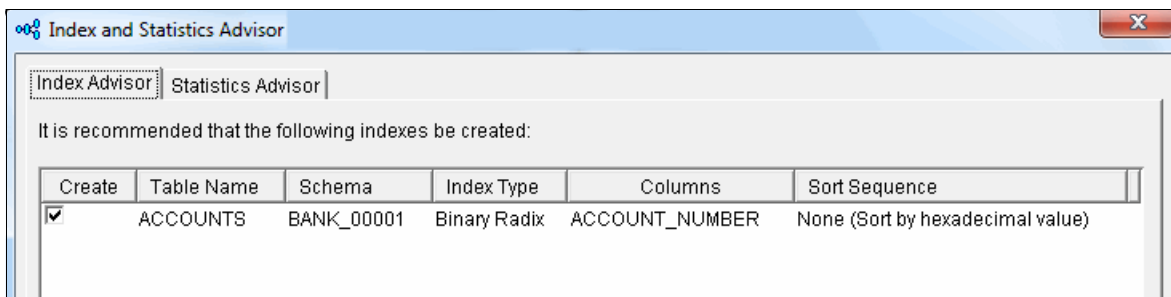
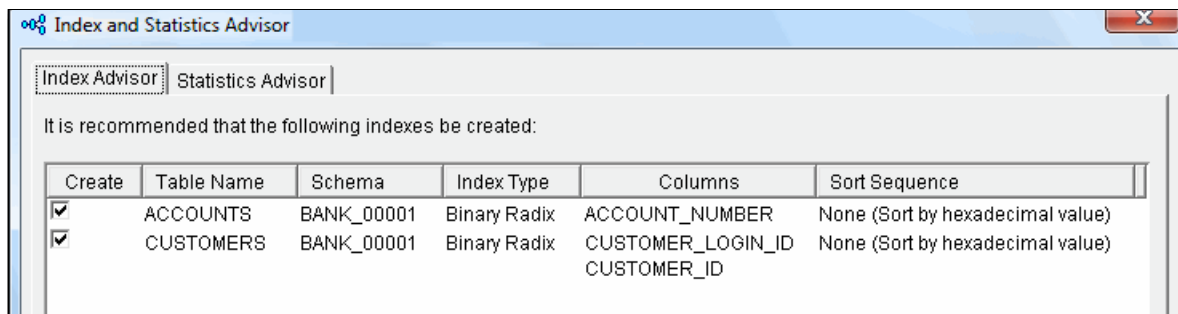


Figure 4-69 Index advice with no RCAC

4. Now, look at the advised indexes with RCAC enabled. As shown in Figure 4-70, there is an additional index being advised, which is basically for the row permission rule. For more information, see 6.4.2, “Index advisor” on page 99.



The screenshot shows a window titled "Index and Statistics Advisor" with two tabs: "Index Advisor" (selected) and "Statistics Advisor". Below the tabs, it states "It is recommended that the following indexes be created:". A table lists the recommended indexes with columns for "Create", "Table Name", "Schema", "Index Type", "Columns", and "Sort Sequence".

Create	Table Name	Schema	Index Type	Columns	Sort Sequence
<input checked="" type="checkbox"/>	ACCOUNTS	BANK_00001	Binary Radix	ACCOUNT_NUMBER	None (Sort by hexadecimal value)
<input checked="" type="checkbox"/>	CUSTOMERS	BANK_00001	Binary Radix	CUSTOMER_LOGIN_ID CUSTOMER_ID	None (Sort by hexadecimal value)

Figure 4-70 Index advice with RCAC enabled



RCAC and non-SQL interfaces

A benefit of Row and Column Access Control (RCAC) is that its security controls are enforced across all the interfaces that access DB2 for i because the security rules are defined and enforced at the database level. The examples that are shown in this paper focus on SQL-based access, but row permissions and column masks also are enforced for non-SQL interfaces, such as native record-level access in RPG and COBOL programs and CL commands, such as Display Physical File Member (DSPPFM) and Copy File (CPYF).

This consistent enforcement across all interfaces is a good thing, but there are some nuances and restrictions as a result of applying an SQL-based technology such as RCAC to non-SQL interfaces. These considerations are described in this chapter.

The following topics are covered in this chapter in this chapter:

- ▶ Unsupported interfaces
- ▶ Native query result differences
- ▶ Accidental updates with masked values
- ▶ System CL commands considerations

5.1 Unsupported interfaces

It is not possible to create a row permission or column mask on a distributed table or a program-described file.

After a row permission or column mask is added to a table, there are some data access requests that no longer work. An attempt to open or query a table with activated RCAC controls involving any of the following scenarios is rejected with the CPD43A4 error message:

- ▶ A logical file with multiple formats if the open attempt requests more than one format.
- ▶ A table or query that specifies an ICU 2.6.1 sort sequence.
- ▶ A table with read triggers.

This unsupported interface error occurs when a table with RCAC controls is accessed, not when the RCAC control is created and activated.

For example, assume that there is a physical file, PF1, which is referenced by a single format logical file (LFS) and a multi-format logical file (LFM). A row permission is successfully created and activated for PF1. Any application that accesses PF1 directly or LFS continues to work without any issues. However, any application that opens LFM with multiple formats receives an error on the open attempt after the row permission is activated for PF1.

Important: This potential runtime error places a heavy emphasis on a comprehensive testing plan to ensure that all programs are tested. If testing uncovers an unsupported interface, then you must investigate whether the application can be rewritten to use a data access interface that is supported by RCAC.

5.2 Native query result differences

The SQL Query Engine (SQE) is the only engine that is enhanced by IBM to enforce RCAC controls on query requests. In order for native query requests to work with RCAC, these native query requests are now processed by SQE instead of the Classic Query Engine (CQE). Native query requests can consist of the following items:

- ▶ Query/400
- ▶ QQQQRY API
- ▶ Open Query File (**OPNQRYF**) command
- ▶ Run Query (**RUNQRY**) command
- ▶ Native open (RPG, COBOL, OPNDBF, and so on) of an SQL view

Legacy queries that have been running without any issues for many years and over many IBM i releases are now processed by a different query engine. As a result, the runtime behavior and results that are returned can be different for native query requests with RCAC enabled. The **OPNQRYF** command and Query/400 run with SQE by default.

The following list documents some of the query output differences that can occur when native query requests are processed by CQE:

- ▶ Different ordering in the result set
- ▶ Different values for null columns or columns with errors
- ▶ Suppression of some mapping error messages
- ▶ Loss of RRN positioning capabilities
- ▶ Duplicate key processing behavior differences
- ▶ Missing key feedback

For a list of the differences and additional details, see the *IBM i Memo to Users Version 7.2*, found at:

http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzahg/rzahgmtu.htm

In addition, the performance of a native query with SQE can be different. It is possible that a new index or keyed logical file might need to be created to improve the performance.

Important: Based on the potential impacts of query result set and performance differences, you should perform extensive functional testing and performance benchmarking of applications and reports that use native query interfaces.

5.3 Accidental updates with masked values

The masked values that are returned by a column mask can potentially cause the original data value to be accidentally overwritten, especially with applications using native record-level access.

For example, consider a table containing three columns of first name, last name, and tax ID that is read by an RPG program. The user running the program is not authorized to see the tax ID value, so a masked value (****3333) is written into the program's record buffer, as shown Figure 5-1.

In this example, the application reads the data for an update to correct the misspelling of the last name. The last name value is changed to Smith in the buffer. Now, a WRITE request is issued by the program, which uses the contents of the record buffer to update the row in the underlying DB2 table. Unfortunately, the record buffer still contains a masked value for the tax ID, so the tax ID value in the table is accidentally set to the masked value.

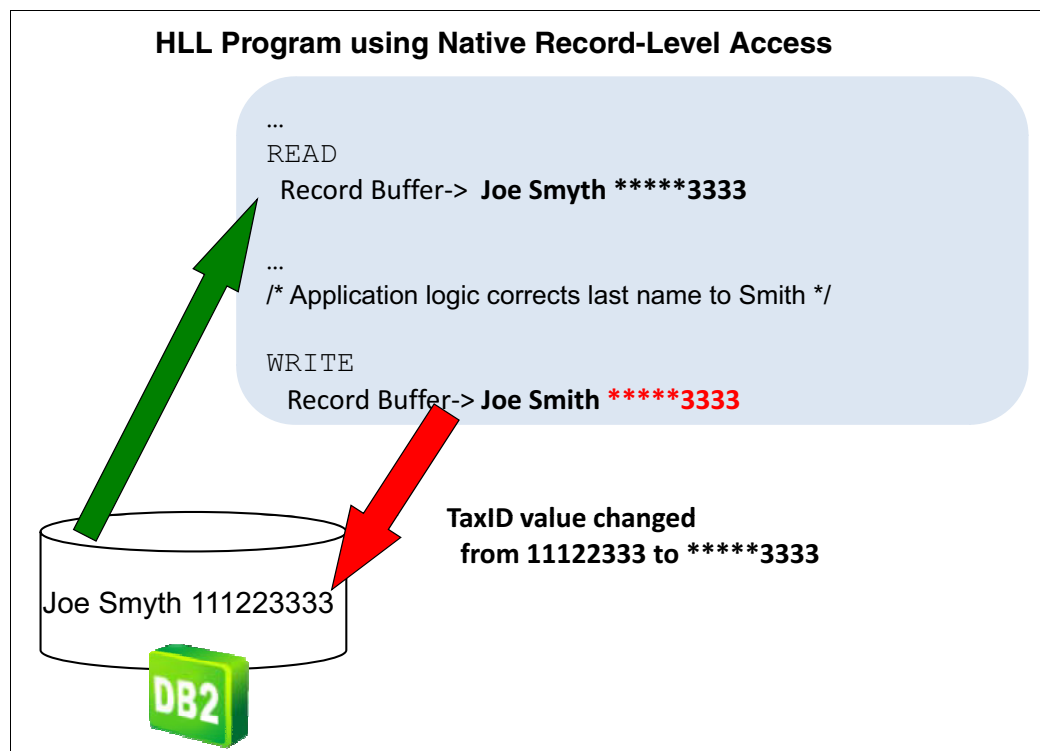


Figure 5-1 Accidental update with masked values scenario

Obviously, careful planning and testing should be exercised to avoid accidental updates with masked values.

DB2 for i also enhanced its check constraint support in the IBM i 7.2 release with a new **ON UPDATE** clause that allows the existing value to be preserved when a masked value is detected by a check constraint. Details about how to employ this new check constraint support can be found in 6.8.1, “Check constraint solution” on page 108.

5.4 System CL commands considerations

As stated earlier, RCAC controls are enforced on all data access interfaces. This enforcement is not limited to programmatic interfaces; it also includes system CL commands that read and insert data, such as the Create Duplicate Object (**CRTDUPOBJ**) and Start DFU (**STRDFU**) CL commands. This section documents the behavior of the Create Duplicate Object (**CRTDUPOBJ**), Copy File (**CPYF**), and Copy Library (**CPYLIB**) CL commands with RCAC.

5.4.1 Create Duplicate Object (CRTDUPOBJ) command

The **CRTDUPOBJ** command is enhanced with a new Access Control (**ACCCTL**) parameter in the IBM i 7.2 release to copy RCAC controls to the new object being created. Row permissions and column masks are copied to the new object by default because the default value for the **ACCCTL** parameter is *ALL.

If the invoker of the **CRTDUPOBJ** command asks for data to be copied with a value of *YES for the **DATA** parameter, the value of the **ACCCTL** parameter must be *ALL. If not, the command invocation receives an error.

When data is copied to the duplicated object with the **DATA** parameter, all rows and unmasked column values are copied into the new object, even if the command invoker is not authorized to view all rows or certain column values. This behavior occurs because the RCAC controls also are copied to the new object. The copied RCAC controls enforce that only authorized users are allowed to view row and column values in the newly duplicated object.

5.4.2 Copy File (CPYF) command

The **CPYF** command copies only data, so there is no new parameter to copy RCAC controls to the target table. Therefore, if **CPYF** is used to create a target table, there are no RCAC controls placed on the target table.

When RCAC controls are in place on the source table, the **CPYF** command is limited to reading rows and column values that are based on the invoker of the **CPYF** command. If a user is authorized to see all rows and column values, then all rows and unmasked column values are copied to the target table (assuming no RCAC controls are on the target table). If a user without full access runs the **CPYF** command, the **CPYF** command can copy only a subset of the rows into the target table. In addition, if that user can view only masked column values, then masked values are copied into the target table. This also applies to the Copy to Import File (**CPYTOIMPF**) command.

If the target table has RCAC controls defined and activated, then the **CPYF** command is allowed only to add or replace rows in the target table based on the RCAC controls. If **CPYF** tries to add a row to the target table that the command invoker is not allowed to view according to the target RCAC controls, then an error is received.

5.4.3 Copy Library (CPYLIB) command

The **CPYLIB** command is enhanced with the same Access Control (**ACCCTL**) parameter as the **CRTDUPOBJ** command in the IBM i 7.2 release (see 5.4.1, “Create Duplicate Object (CRTDUPOBJ) command” on page 82). Row permissions and column masks are copied to the new object in the new library by default because the default value for the **ACCCTL** parameter is *ALL.



Additional considerations

This chapter covers additional considerations that must be taken into account when implementing Row and Column Access Control (RCAC), including the following functions:

- ▶ Timing of column masking
- ▶ Data movement
- ▶ Joins
- ▶ Views
- ▶ Materialized query tables
- ▶ Index advisor
- ▶ Monitoring, analysis, and debugging
- ▶ Performance and scalability

The following topics are covered in this chapter:

- ▶ Timing of column masking
- ▶ RCAC effects on data movement
- ▶ RCAC effects on joins
- ▶ Monitoring, analyzing, and debugging with RCAC
- ▶ Views, materialized query tables, and query rewrite with RCAC
- ▶ RCAC effects on performance and scalability
- ▶ Exclusive lock to implement RCAC (availability issues)
- ▶ Avoiding propagation of masked data
- ▶ Triggers and functions (SECURED)
- ▶ RCAC is only one part of the solution

6.1 Timing of column masking

An important design and implementation consideration is the fact that RCAC column masking occurs after all of the query processing is complete, which means that the query results are not at all based on the masked values. Any local selection, joining, grouping, or ordering operations are based on the unmasked column values. Only the final result set is the target of the masking.

An example of this situation is shown in Figure 6-1. However, note that aggregate functions (a form of grouping) are based on masked values.

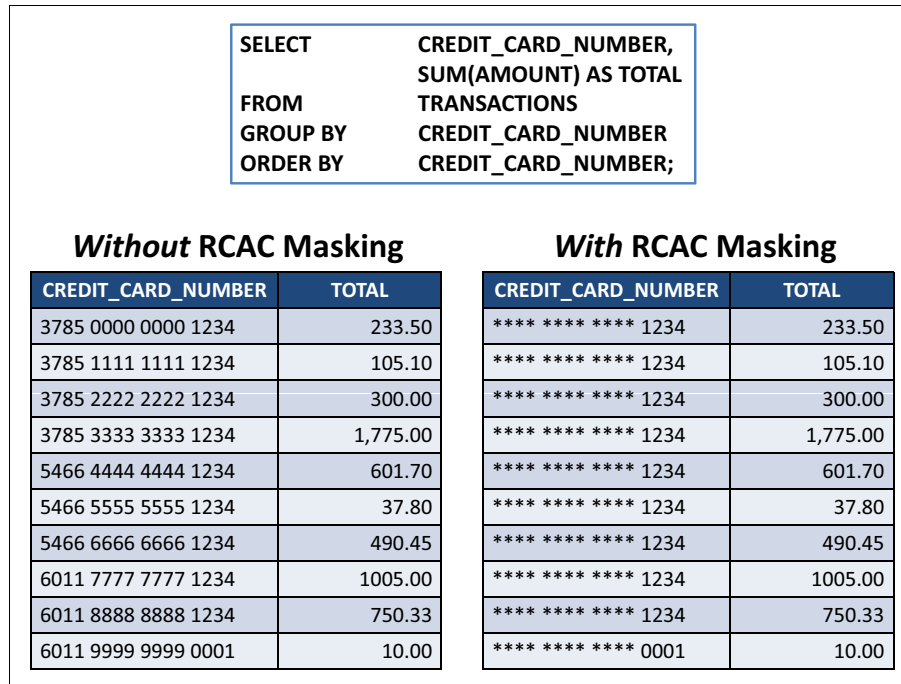


Figure 6-1 Timing of column masking

Conversely, field procedure masking causes the column values to be changed (that is, masked) and stored in the row. When the table is queried and the masked columns are referenced, the masked data is used for any local selection, joining, grouping, or ordering operations. This situation can have a profound effect on the query's final result set and not just on the column values that are returned. Field procedure masking occurs when the column values are read from disk before any query processing. RCAC masking occurs when the column values are returned to the application after query processing. This difference in behavior is shown in Figure 6-2.

Note: Column masks can influence an SQL **INSERT** or **UPDATE**. For example, you cannot insert or update a table with column access control activated with masked data generated from an expression within the same statement that is based on a column with a column mask.

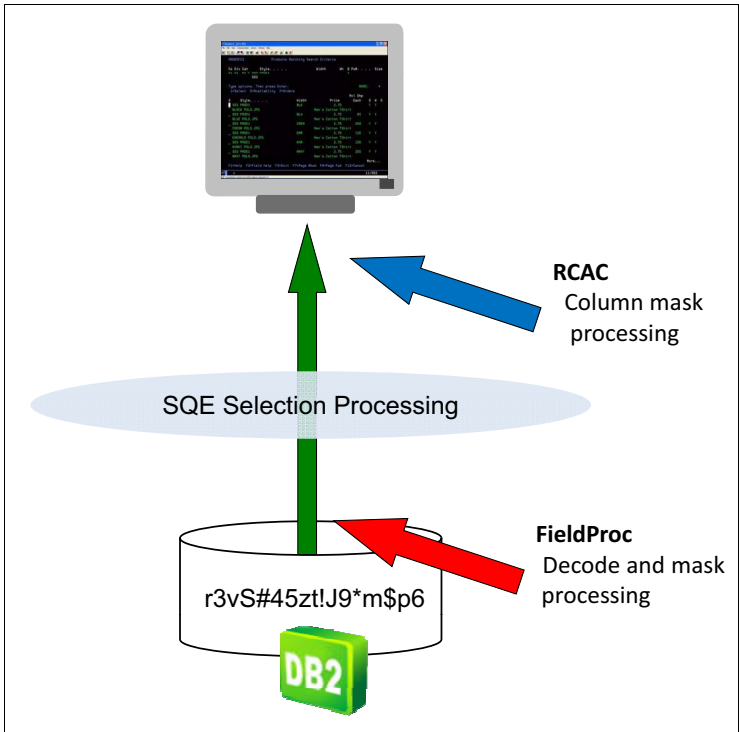


Figure 6-2 Masking differences between Fieldproc and RCAC

6.2 RCAC effects on data movement

As described earlier and shown in Figure 6-3, RCAC is applied pervasively regardless of the data access programming interface, SQL statement, or IBM i command. The effects of RCAC on data movement scenarios can be profound and possibly problematic. It is important to understand these effects and make the appropriate adjustments to avoid incorrect results or data loss.

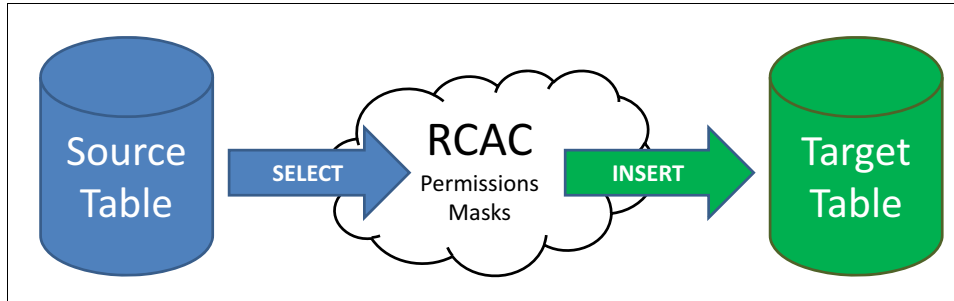


Figure 6-3 RCAC and data movement

The “user” that is running the data movement application or process, whether it be a high availability (HA) scenario, an extract, transform, load (ETL) scenario, or just copying data from one file or table to another one, must have permission to all the source rows without masking, and not be restricted from putting rows into the target. Allowing the data movement application or process to bypass the RCAC rules must be based on a clear and concise understanding of the organization's object security and data access policy. Proper design, implementation, and testing are critical success factors when applying RCAC.

Important: RCAC is applied to the table or physical file access. It is not applied to the journal receiver access. Any and all database transactions are represented in the journal regardless of RCAC row permissions and column masks. This makes it essential that IBM i security is used to ensure that only authorized personnel have access to the journaled data.

This section covers in detail the following three examples:

- ▶ Effects when RCAC is defined on the source table
- ▶ Effects when RCAC is defined on the target table
- ▶ Effects when RCAC is defined on both source and target tables

6.2.1 Effects when RCAC is defined on the source table

Example 6-1 shows a simple example that illustrates the effect of RCAC as defined on the source table.

Example 6-1 INSERT INTO TARGET statement

```
INSERT INTO TARGET (SELECT * FROM SOURCE);
```

For example, given a “source” table with a row permission defined as NAME <> 'CAIN' and a column mask that is defined to project the value 999.99 for AMOUNT, the **SELECT** statement produces a result set that has the RCAC rules applied. This reduced and modified result set is inserted into the “target” table even though the query is defined as returning all rows and all columns. Instead of seven rows that are selected from the source, only three rows are returned and placed into the target, as shown in Figure 6-4.

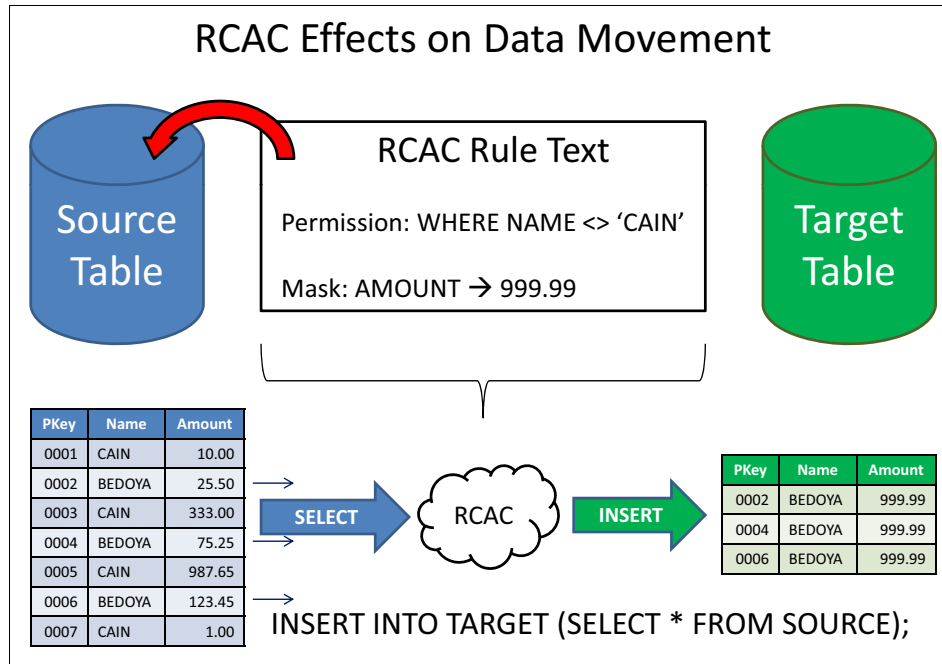


Figure 6-4 RCAC effects on data movement from SOURCE

6.2.2 Effects when RCAC is defined on the target table

Example 6-2 shows a simple example that illustrates the effect of RCAC as defined on the target table.

Example 6-2 INSERT INTO TARGET statement

```
INSERT INTO TARGET (SELECT * FROM SOURCE);
```

Given a “target” table with a row permission defined as NAME <> 'CAIN' and a column mask that is defined to project the value 999.99 for AMOUNT, the **SELECT** statement produces a result set that represents all the rows and columns. The seven row result set is inserted into the “target”, and the RCAC row permission causes an error to be returned, as shown in Figure 6-5. The source rows where NAME = 'CAIN' do not satisfy the target table's permission, and therefore cannot be inserted. In other words, you are inserting data that you cannot read.

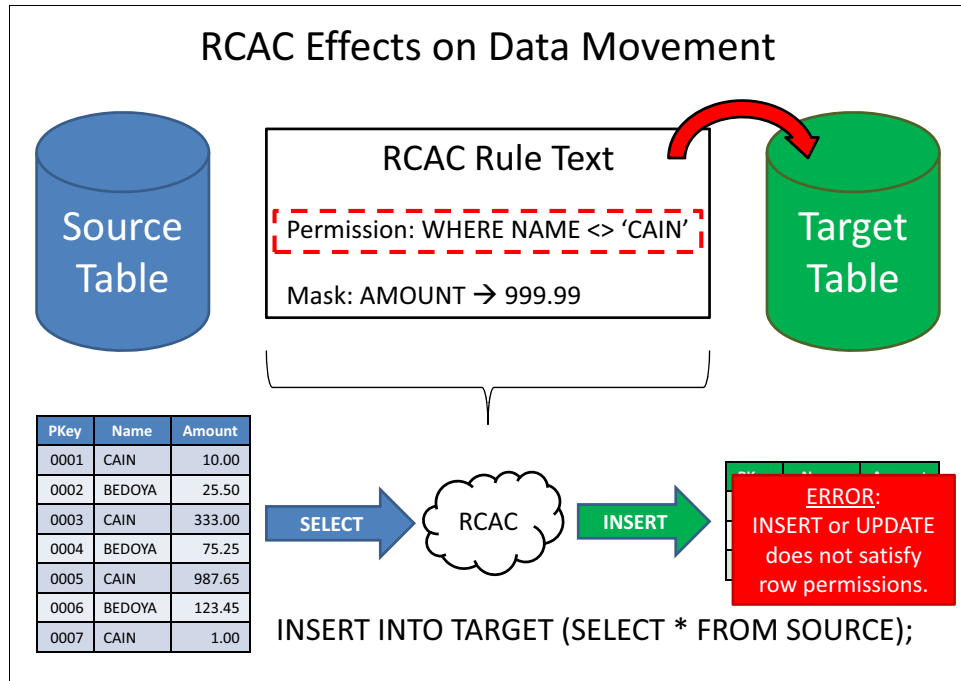


Figure 6-5 RCAC effects on data movement on TARGET

6.2.3 Effects when RCAC is defined on both source and target tables

Example 6-3 shows a simple example that illustrates the effect of RCAC as defined on both the source and the target tables.

Example 6-3 INSERT INTO TARGET statement

```
INSERT INTO TARGET (SELECT * FROM SOURCE);
```

Given a “source” table and a “target” table with a row permission defined as NAME <> 'CAIN' and a column mask that is defined to project the value 999.99 for AMOUNT, the **SELECT** statement produces a result set that has the RCAC rules applied. This reduced and modified result set is inserted into the “target” table even though the query is defined as returning all rows and all columns. Instead of seven rows that are selected from the source, only three rows are returned.

Although the source rows where NAME <> 'CAIN' do satisfy the target table's permission, the AMOUNT column value of 999.99 represents masked data and therefore cannot be inserted. An error is returned indicating the failure, as shown in Figure 6-6. In this scenario, DB2 is protecting against an overt attempt to insert masked data.

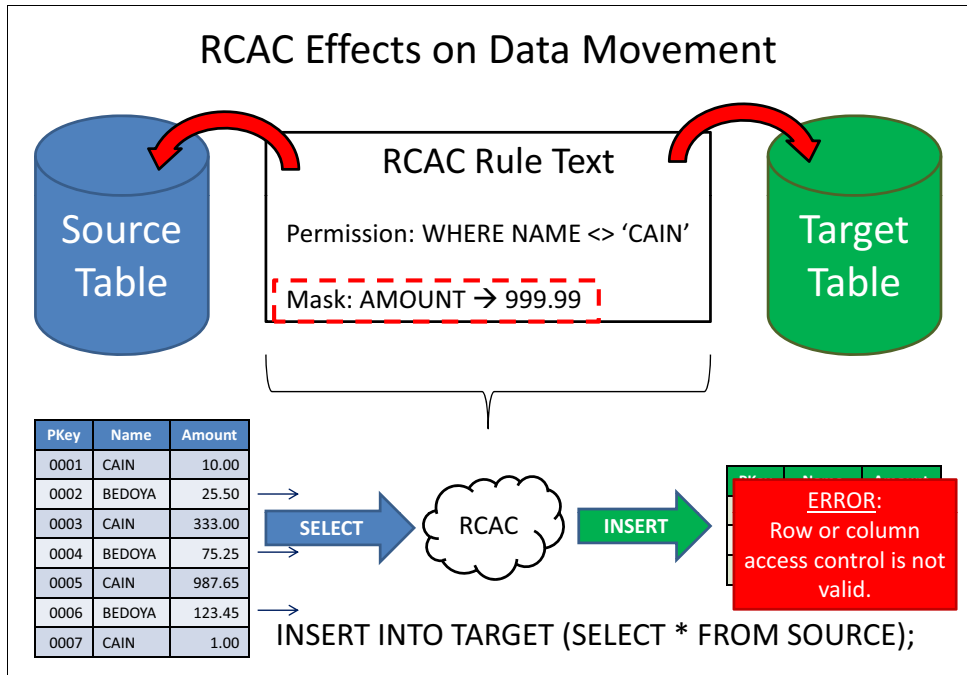


Figure 6-6 RCAC effects on data movement on SOURCE and TARGET

6.3 RCAC effects on joins

As mentioned previously, a fundamental concept of row permission is that it defines a logical subset of rows that a user or group of users is permitted to access and use. This subset becomes the new basis of any query against the table that has RCAC enabled.

Note: Thinking of the row permission as defining a virtual set of rows that can be operated on is the secret to understanding the effect of RCAC on any join operation.

As shown in Figure 6-7, there are two different sets, set A and set B. However, set B has a row permission that subsets the rows that a user can see.

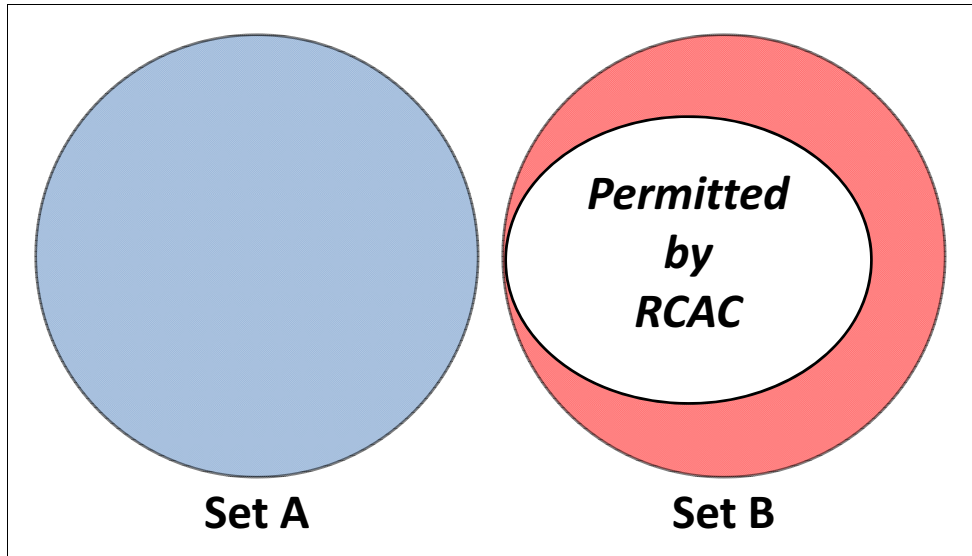


Figure 6-7 Set A and set B with row permissions

6.3.1 Inner joins

Inner join defines the intersection of two data sets. For a row to be returned from the inner join query, it must appear in both sets, as shown in Figure 6-8.

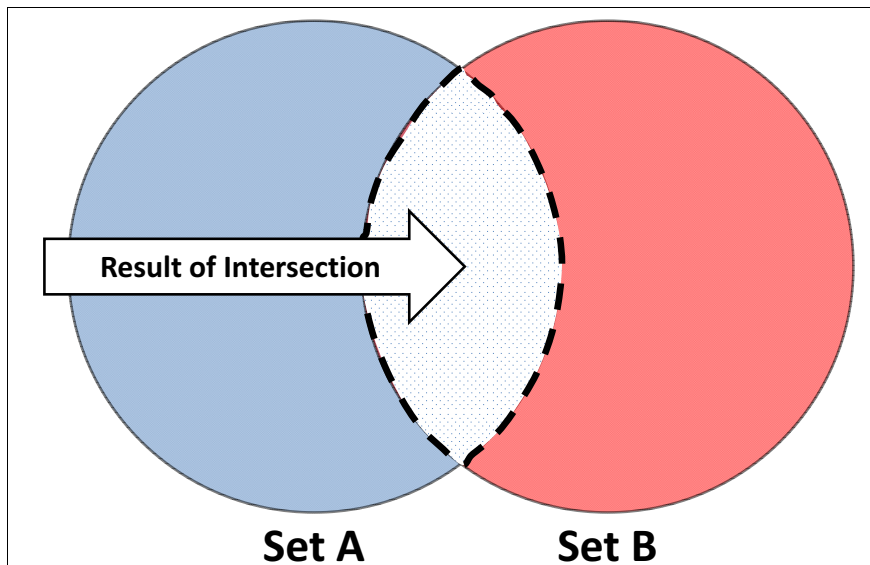


Figure 6-8 Inner join without RCAC permission

Given that row permission serves to eliminate logically rows from one or more sets, the result set from an inner join (and a subquery) can be different when RCAC is applied. RCAC can reduce the number of rows that are permitted to be accessed by the join, as shown in Figure 6-9.

Effect of column masks on inner joins: Because column masks are applied after the query final results are determined, the masked value has no effect on the join processing and corresponding query result set.

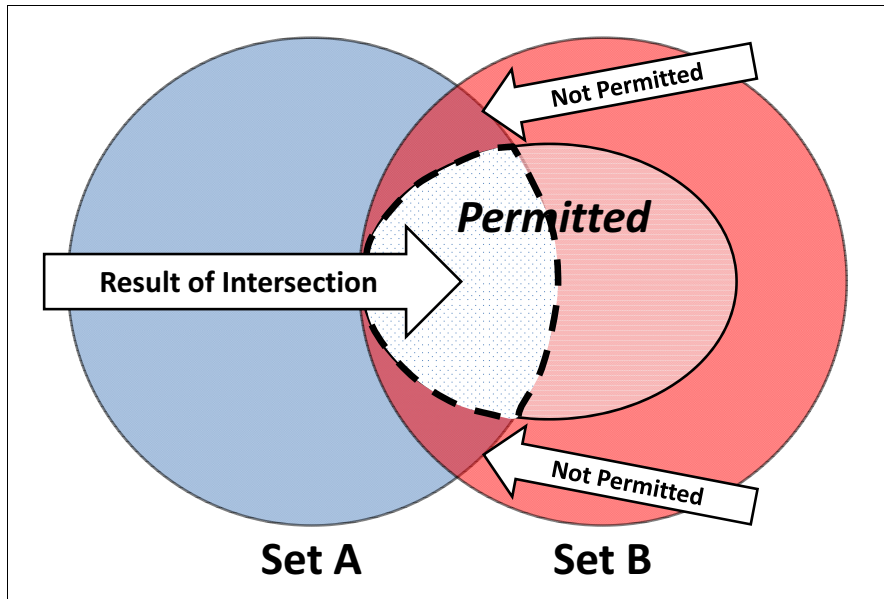


Figure 6-9 Inner join with RCAC permission

6.3.2 Outer joins

Outer joins preserve one or both sides of two data sets. A row can be returned from the outer join query if it appears in the primary set (LEFT, RIGHT, or both in the case of FULL), as shown in Figure 6-10. Column values from the secondary set are returned if the row has a match in the primary set. Otherwise, NULL is returned for the column value by default.

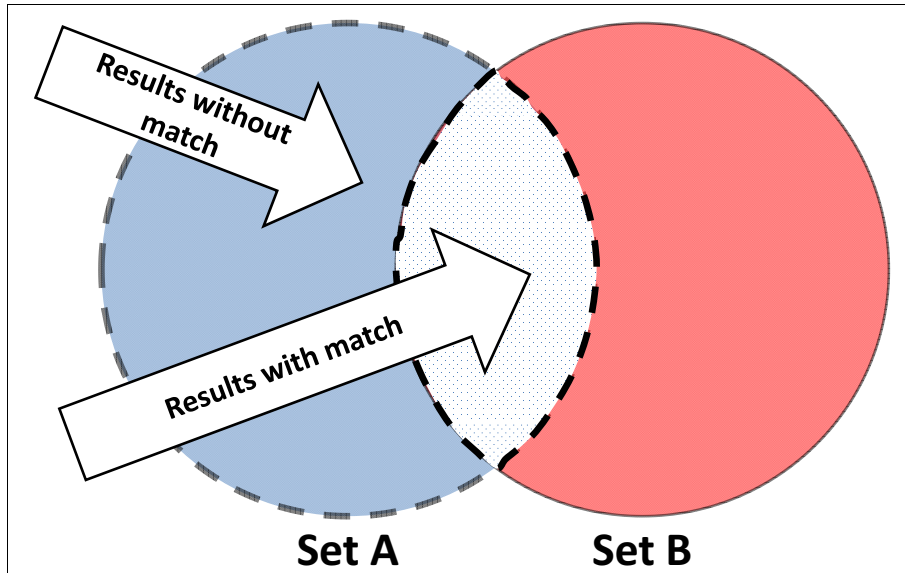


Figure 6-10 Outer join without RCAC permission

Given that row permission serves to eliminate logically rows from one or more sets, more column values that are returned from the secondary table in outer join can be NULL when RCAC is applied, as shown in Figure 6-11.

Effect of column masks on inner joins: Because column masks are applied after the query final results are determined, the masked value has no effect on the join processing and corresponding query result set.

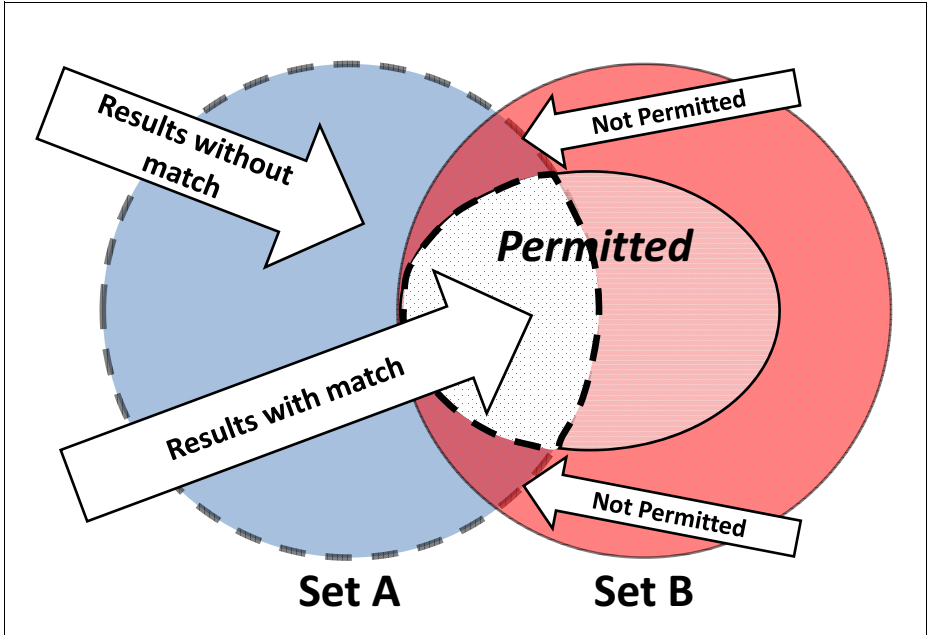


Figure 6-11 Outer join with RCAC permission

6.3.3 Exception joins

Exception joins preserve one side of two data sets. A row can be returned from the exception join query if it appears in the primary set (LEFT or RIGHT) and the row does not appear in the secondary set, as shown in Figure 6-12. Column values from the secondary set are returned as NULL by default.

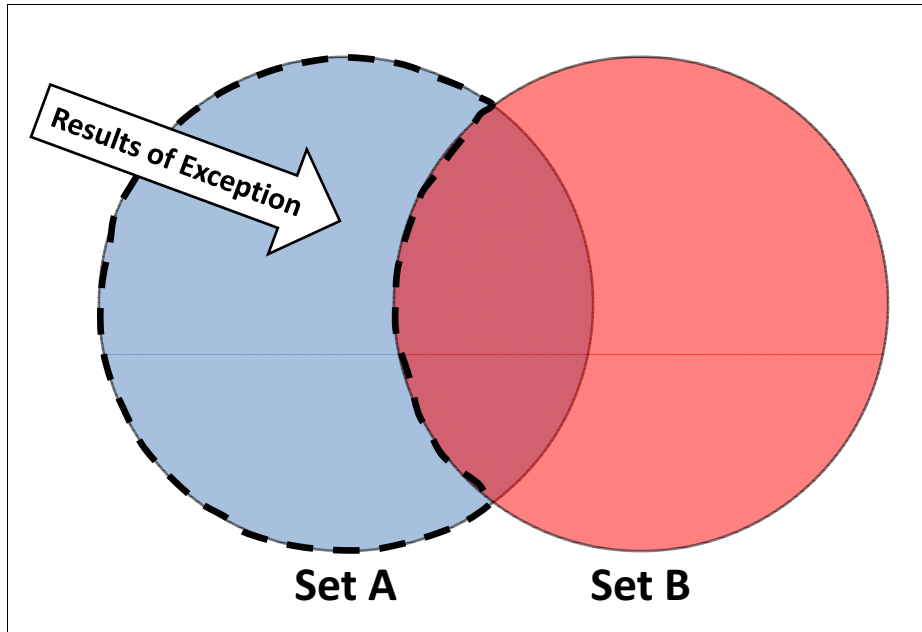


Figure 6-12 Exception join without RCAC permission

Given that row permission serves to eliminate logically rows from one or more sets, more rows can appear to be exceptions when RCAC is applied, as shown in Figure 6-13. Also, because column masks are applied after the query final results are determined, the masked value has no effect on the join processing and corresponding query result set.

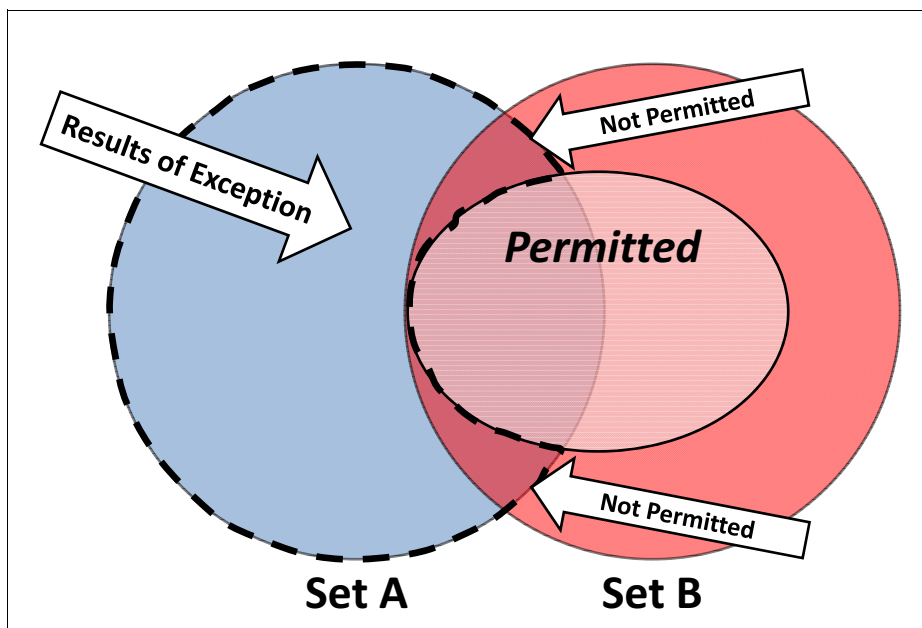


Figure 6-13 Exception join with RCAC permission

6.4 Monitoring, analyzing, and debugging with RCAC

It is assumed (and it is a critical success factor) that the database engineer or application developer has a thorough understanding of the DB2 for i Query Optimizer, Database Engine, and all the associated tools and techniques.

The monitoring, analyzing, and debugging process basically stays the same when RCAC row permissions or column masks are in place, with a few important differences:

- ▶ The underlying data access plan can be different and more complex based on the rule text.
- ▶ The database results can be reduced or modified based on the rule text and user profile.
- ▶ The run time of the request can be affected either positively or negatively based on the rule text.
- ▶ For high-level language record level access, query plans must be considered, and not just program code.

During analyzing and debugging, it is important to account for all of the RCAC definitions for each table or file to understand the logic and corresponding work that is associated with processing the row permissions and column masks. It is also important to realize that, depending on the user profile in effect at run time, the database actions and query results can be different.

RCAC is designed and implemented to be transparent to the user. It is possible for user “Mike” and user “Hernando” to run the exact same query, against the exact same data on the exact same system, and get different result sets. There is no error, no warning, and no indication that RCAC reduced or modified the respective answers that are returned. Furthermore, it is also likely that user “Mike” and user “Hernando” have different query run times even though it appears that everything is the same for both users. The actual query plan contains the RCAC logic, and this additional code path can alter the amount of work that is needed to produce results, based on the user running the query.

When monitoring, analyzing, and debugging a database process when RCAC is enabled, it is critical to keep as many of the “variables” the same as possible. Use a good scientific process. For example, when re-creating a problem situation running under the same user profile with the same data and under the same conditions, it is almost mandatory. Otherwise, the database behavior and query results can be different.

To successfully perform monitoring, analyzing, and debugging when RCAC is enabled likely involves changes in the security and data access policies of the organization, and require new responsibilities, authority, and oversight within the data-centric application development community. As such, establishing and staffing the position of “database engineer” becomes even more important.

6.4.1 Query monitoring and analysis tools

When monitoring and collecting metrics on database requests, DB2 for i provides additional information that indicates row permissions or column masks are being applied. This information is integrated and part of the standard tools, such as Visual Explain, SQL Plan Cache Snapshot, and SQL Performance Monitor.

Figure 6-14 shows how Visual Explain externalizes RCAC.

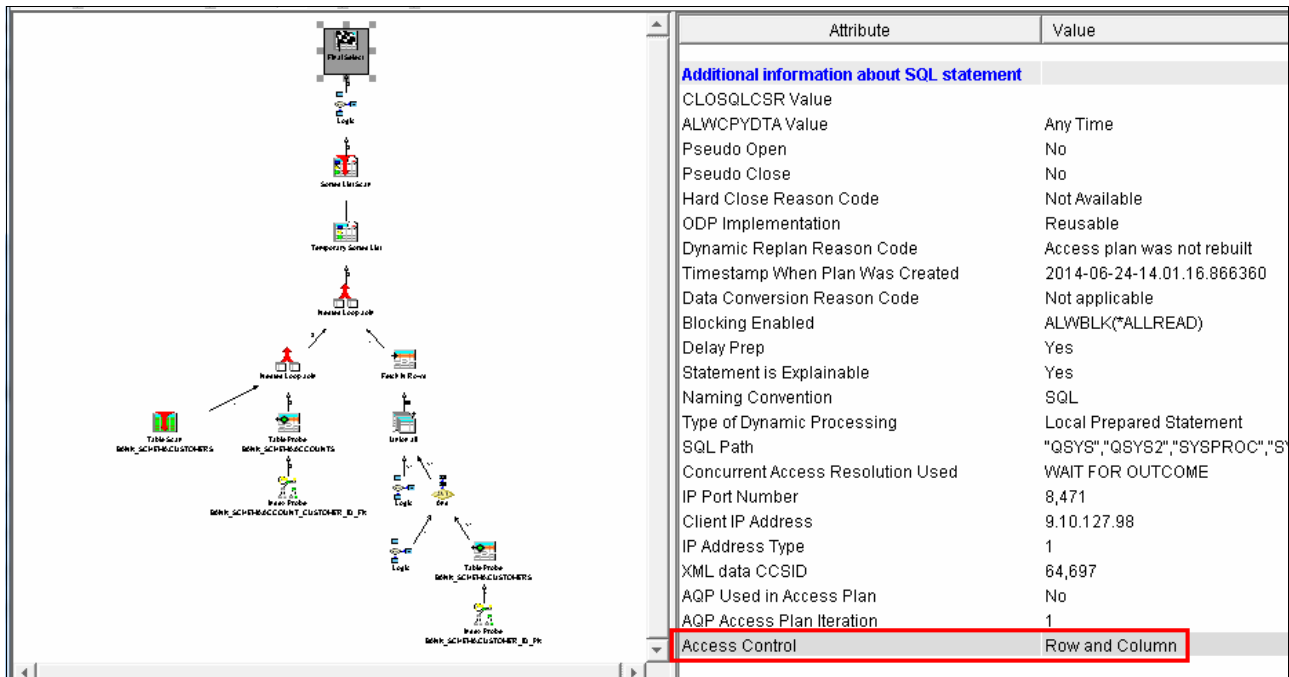


Figure 6-14 Visual Explain indicating that RCAC is applied

Figure 6-15 shows the main dashboard of an SQL Performance Monitor. Click **Summary**.

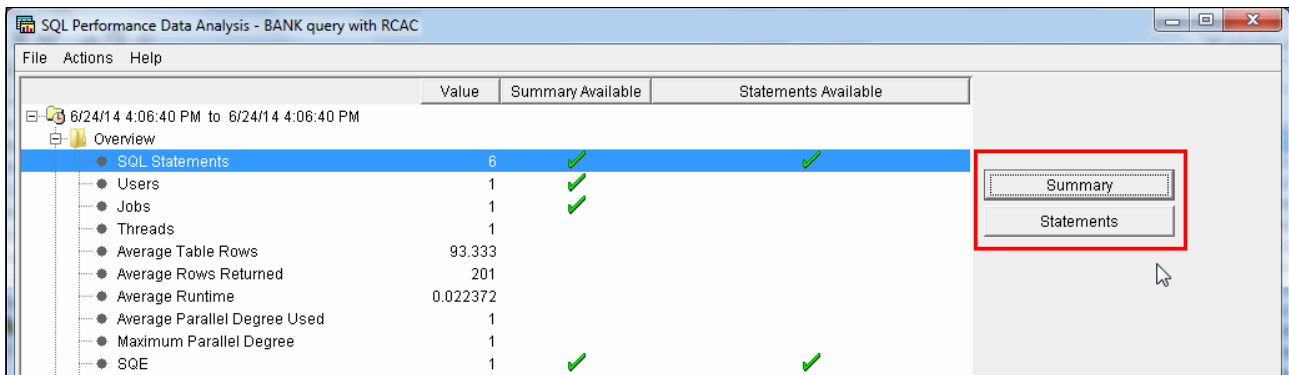


Figure 6-15 SQL Performance Monitor

Figure 6-16 shows the summary of an SQL Performance Monitor with an indication that RCAC is applied.

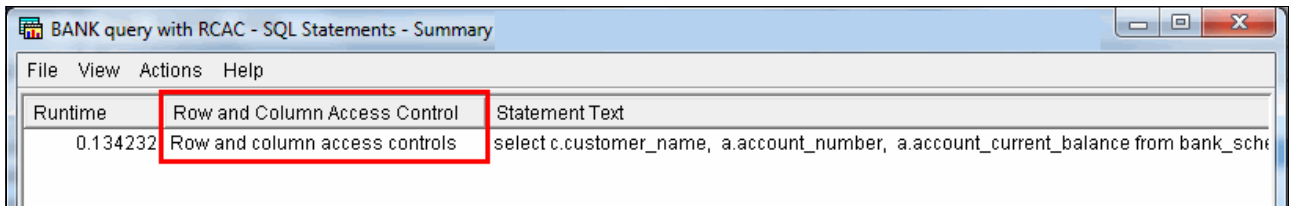


Figure 6-16 SQL Performance Monitor indicating that RCAC is applied

Figure 6-17 shows the statements of an SQL Performance Monitor and how RCAC is externalized.

Start Time	End Time	Runtime	Statement Outcome	SQLSTATE	SQLCODE	Operation	Row and Column Access Control	Stal
2014-06-24 16:06:40.473588	2014-06-24 16:06:40.542539	0.064792	Successful	00000	0	FETCH		
2014-06-24 16:06:40.473588	2014-06-24 16:06:40.537996	0.064408	Successful	00000	0	OPEN	Row and column access controls	sele
2014-06-24 16:06:40.538384	2014-06-24 16:06:40.542611	0.004224	Successful	00000	0	CLOSE	Row and column access controls	CLC
2014-06-24 16:06:40.469807	2014-06-24 16:06:40.470608	0.000808	Successful	00000	0	PREPARE...DESCRIBE		sele
2014-06-24 16:06:40.575708	2014-06-24 16:06:40.575939	0.000232	Successful	00000	0	CALL		CAL
2014-06-24 16:06:40.538388	2014-06-24 16:06:40.538388	0.000000	Successful	00000	0	CLOSE (Hard)		HAF

Figure 6-17 SQL Performance Monitor showing statements and RCAC

When implementing RCAC as part of a comprehensive and pervasive data access control initiative, consider that the database monitoring and analysis tools can collect literal values that are passed as part of SQL statements. These literal values can be viewed as part of the information collected. If any of the literals are based on or are used with masked columns, it is important to review the database engineer's policy for viewing these data elements. For example, supposed that column CUSTOMER_TAX_ID is deemed masked for the database engineer and the CUSTOMER_TAX_ID column is used in a predicate as follows:

```
WHERE CUSTOMER_TAX_ID = '123-45-7890'
```

The literal value of '123-45-7890' is visible to the analyst, effectively exposing sensitive information. If this is not acceptable, you must implement the SYSPROC.SET_COLUMN_ATTRIBUTE procedure.

The SET_COLUMN_ATTRIBUTE procedure sets the SECURE attribute for a column so that variable values that are used for the column cannot be seen in the SQL Performance Monitor, SQL Plan Cache Snapshot, or Visual Explain.

6.4.2 Index advisor

Because the RCAC rule text can be almost any valid SQL logic, including local selection predicates, join conditions, and subqueries, the standard query tuning techniques still apply. Without a doubt, a proper and adequate indexing strategy is a good starting point.

The index advisor is not specifically enhanced for RCAC, but because the rule text is a fully integrated part of the query plan, any opportunities for indexing is advised based on the current Query Optimizer functionality. If an index is advised because of the RCAC rule text logic, there is no RCAC reason code provided. Analyzing the query plan and the RCAC rule text provides the understanding as to why the index is being advised.

For example, the query that is shown in Figure 6-18 produces index advice for the user's predicate and the RCAC predicate.

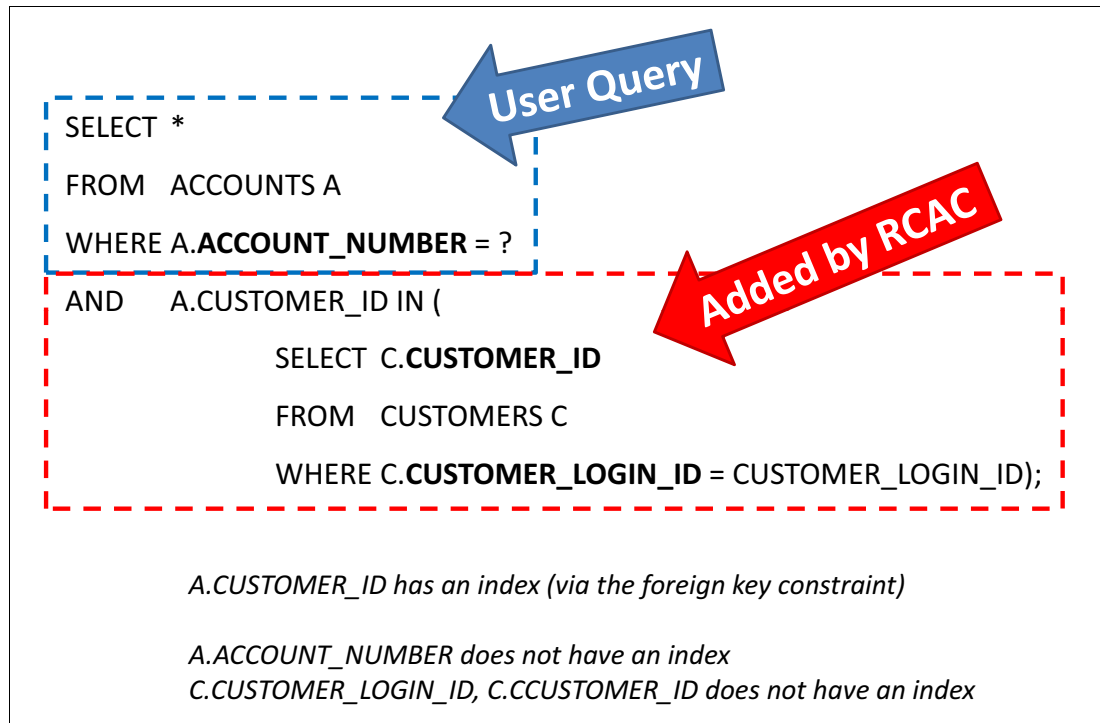


Figure 6-18 Index advice and RCAC

In Figure 6-19, index advisor is showing an index for the ACCOUNTS and CUSTOMERS tables based on the RCAC rule text.

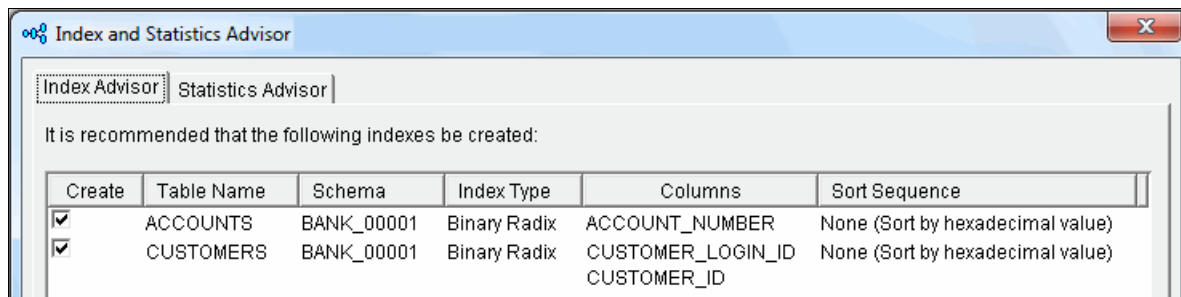


Figure 6-19 Index advisor based on the RCAC rule

For more information about creating and using indexes, see *IBM DB2 for i indexing methods and strategies*, found at:

http://www.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_wp_db2_i_indexing_methods_strategies

6.4.3 Metadata using catalogs

To make the discovery and identification of RCAC row permissions and column masks programmatically, query the QSYS2.SYSCONTROLS catalog view or the QSYS2.SYSCONTROLSDEP catalog view directly. Otherwise, the System i Navigator Database graphical interface can be used interactively.

Figure 6-20 shows the QSYS2.SYSCONROLS catalog view.

```
select *
from qsys2.syscontrols
where rcac_schema = 'BANK_SCHEMA';
```

RCAC_SCHEMA	RCAC_NAME	RCAC_OWNER	TABLE_SCHEMA	TABLE_NAME
BANK_SCHEMA	MASK_DRIVERS_LICENSE_ON_CUSTOMERS	MCAIN	BANK_SCHEMA	CUSTOMERS
BANK_SCHEMA	MASK_EMAIL_ON_CUSTOMERS	MCAIN	BANK_SCHEMA	CUSTOMERS
BANK_SCHEMA	MASK_LOGIN_ID_ON_CUSTOMERS	MCAIN	BANK_SCHEMA	CUSTOMERS
BANK_SCHEMA	MASK_SECURITY_QUESTION_ANSWER_ON_CUSTOMERS	MCAIN	BANK_SCHEMA	CUSTOMERS
BANK_SCHEMA	MASK_SECURITY_QUESTION_ON_ACCOUNTS	MCAIN	BANK_SCHEMA	ACCOUNTS
BANK_SCHEMA	MASK_SECURITY_QUESTION_ON_CUSTOMERS	MCAIN	BANK_SCHEMA	CUSTOMERS
BANK_SCHEMA	MASK_TAX_ID_ON_CUSTOMERS	MCAIN	BANK_SCHEMA	CUSTOMERS
BANK_SCHEMA	PERMISSION1_ON_ACCOUNTS	MCAIN	BANK_SCHEMA	ACCOUNTS
BANK_SCHEMA	PERMISSION1_ON_CUSTOMERS	MCAIN	BANK_SCHEMA	CUSTOMERS
BANK_SCHEMA	PERMISSION1_ON_TRANSACTIONS	MCAIN	BANK_SCHEMA	TRANSACTIONS

Figure 6-20 RCAC and catalogs

The SYSCONROLS catalog view contains the following columns:

- ▶ COLUMN_NAME
- ▶ CONTROL_TYPE
- ▶ CREATE_TIME
- ▶ ENABLE
- ▶ ENFORCED
- ▶ ASP_NUMBER
- ▶ IMPLICIT
- ▶ LABEL
- ▶ LAST_ALTERED
- ▶ LONG_COMMENT
- ▶ RCAC_NAME
- ▶ RCAC_OWNER
- ▶ RCAC_SCHEMA
- ▶ RULETEXT
- ▶ SYSTEM_COLUMN_NAME
- ▶ SYSTEM_TABLE_NAME
- ▶ SYSTEM_TABLE_SCHEMA
- ▶ TABLE_NAME
- ▶ TABLE_SCHEMA
- ▶ TBCORRELATION

The SYSCONROLSDEP catalog view contains the following columns:

- ▶ COLUMN_NAME
- ▶ CONTROL_TYPE
- ▶ IASP_NUMBER
- ▶ OBJECT_NAME
- ▶ OBJECT_SCHEMA
- ▶ OBJECT_TYPE
- ▶ PARM_SIGNATURE
- ▶ RCAC_NAME
- ▶ RCAC_SCHEMA
- ▶ SYSTEM_TABLE_NAME
- ▶ SYSTEM_TABLE_SCHEMA

For more information, see the *IBM i 7.2 DB2 for i SQL Reference Guide*, found at:

http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/db2/rbafzintro.htm?lang=en

6.5 Views, materialized query tables, and query rewrite with RCAC

This section covers the implications to views, materialized query tables (MQTs), and query rewrite when RCAC is activated on a table.

6.5.1 Views

Any access to an SQL view that is over one or more tables that have RCAC also have those row permissions and column masking rules applied. If an SQL view has predicates, those are logically ANDed with any search condition that is specified in the permissions that are defined on the underlying tables. The view does not have to project the columns that are referenced by the permissions. Figure 6-21 shows an example of a view definition and user query.

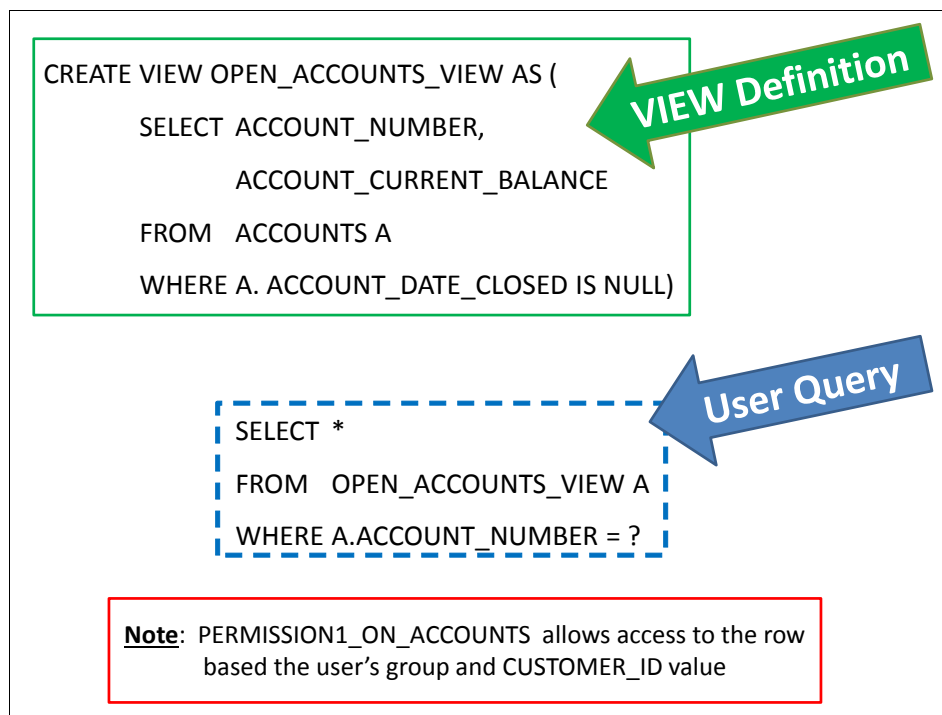


Figure 6-21 View definition and user query

What the query optimizer plans for and what the database engine runs is shown in the Figure 6-22.

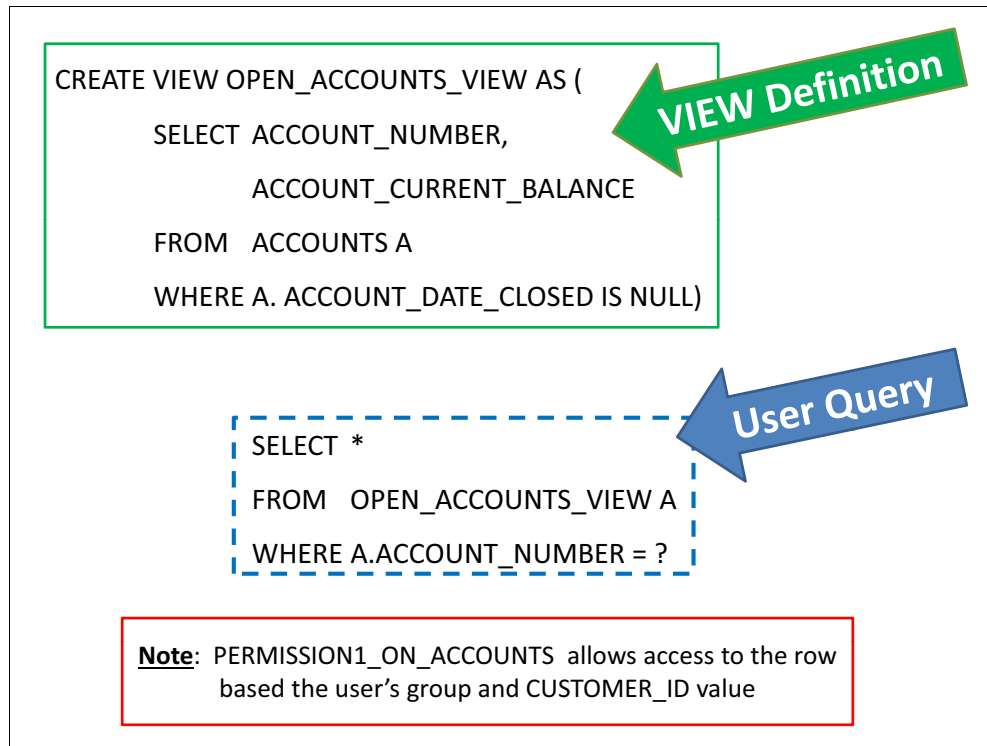


Figure 6-22 Query rewrite with RCAC

6.5.2 Materialized query tables

When the query to populate a materialized query table (MQT) is run by the system on either the create table or a refresh table, and one or more source tables have RCAC defined, the row permissions and column masks are ignored. This means that the MQT has all of the data.

Because the MQT is a copy of the base table data, when a permission is created on the base table, all the related MQTs are altered to have a default row permission. This default permission prevents any of the rows from being directly queried.

When a query implicitly uses an MQT, 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 row permissions and column masks.

The following example illustrates this scenario:

1. Create schema and tables:

```
CREATE SCHEMA Schema1;  
CREATE TABLE Schema1.employee(userID varchar(128), LocationID integer, Regionid  
integer);  
CREATE TABLE Schema1.Sales (INVOICE INTEGER NOT NULL, SALEAMT DECIMAL(5,2),  
TAXAMT DECIMAL(5,2), LOCATIONID INTEGER, REGIONID INTEGER);
```

2. Create a row permission that allows the employees to see only rows from the region they work in:

```
/* Create permission that only allows the employees to see rows from the region
they work in */
CREATE PERMISSION Schema1.Sales_PERM1
ON schema1.sales FOR ROWS
WHERE CURRENT_USER in (SELECT userId FROM schema1.employee E
WHERE e.regionid = regionid)
ENFORCED FOR ALL
ACCESS ENABLE;
```

3. Create an MQT to summarize sales by location:

```
-- Create MQT to summarize sales by location
-- This has all of the data. The schema1.sales_perm1 predicate was not applied
CREATE TABLE Schema1.Location_Sales_MQT as
AS (SELECT LocationID, SUM(Saleamt) as Total_Location_Sales
FROM SCHEMA1.SALES
GROUP BY LOCATIONID)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY USER;
```

4. Populate the MQT (permission is not applied):

```
/* Populate the MQT - Permission not applied here */
REFRESH TABLE Schema1.Location_Sales_MQT
```

The following query matches Location_Sales_MQT, but it cannot be used because it does not have column regionid, which is needed by the schema1.sales_PERM1 permission:

```
SELECT Locationid, sum(SALEAMT) FROM schema1.sales
GROUP BY locationid;
```

5. Create an MQT to summarize by region and location:

```
-- MQT to summarize by region and location
Create table schema1.Region_Location_Sales_MQT as
AS (SELECT REGIONID, LocationID, SUM(Saleamt) as Total_Location_Sales
FROM SCHEMA1.SALES
GROUP BY REGIONID, LOCATIONID)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY USER;
```

6. Populate the Region_location_Sales_MQT (permission not applied):

```
/* Populate the Region_location_Sales_MQT - Permission not applied here */
Refresh table schema1.Region_Location_Sales_MQT
```

The following query can use the Region_location_SALES_MQT because it has REGIONID, which is required for the schema1.sales_PERM1 permission:

```
SELECT Locationid, sum(SALEAMT) FROM schema1.sales
GROUP BY locationid;
```


This example has the following additional implications:

- ▶ Users must be prevented from explicitly querying the MQT or a view that is created over it. Those two cases bypass the row permission and column mask rules from the underlying tables.
- ▶ If the user writes code to update incrementally an MQT, that code must be run from a user that has permission to view all of the rows and all columns in their unmasked state. Otherwise, the MQT contents are not complete and queries that implicitly use the MQT might get wrong results.
- ▶ To prevent this, a check constraint can be created to cause an error if masked data was inserted into the MQT.

6.5.3 Query rewrite

Query rewrite is a technique that the optimizer can use to change the original request to improve performance.

For example, a query that references Table1 might be rewritten to access an MQT over Table1, or it might also be optimized to access only the fields in an index that is defined over Table1 and avoid touching Table1. With RCAC, defining these rewrites can still occur, but the MQT or index also must include all columns that are needed by the row permissions or column masks that are defined on Table1.

As part of adding RCAC, the impact to these potentially significant performance optimizations must be considered. Usage of MQTs or index-only access might be reduced or eliminated by enabling RCAC.

6.6 RCAC effects on performance and scalability

As with any discussion that is related to performance and scalability, nothing is certain or guaranteed. There are always many variables that are involved. First, a good foundation of knowledge and skill is required to appreciate fully what is occurring when a database request is handled within an RCAC enabled environment. Implementing the row permission or column masks involves the query optimizer and database engine. The process that identifies the rows that you have permission to access is considered a “query”, and as such a query plan must be formulated. In the case of SQL requests, the RCAC portion of the query is combined with the user's query, much like a query referencing a view.

For native record level access, this RCAC “query” is also built and used to test the permission. When a file is opened, the RCAC rule text logic is included, optimized, and run as part of the native read, write, update, or delete operation. The amount of work (and time) required to identify the record based on the user's permission is directly related to the complexity and depth of the logic that is needed to identify the records that can be returned.

A simple example to illustrate this concept is a random read using a keyed logical file (that is, an index). In its purest form, a random read uses two data access methods: index probe (find the key and RRN) and table probe (find the record using RRN). If the RCAC rule text specifies five nested subqueries to determine whether the user has access to the record, this logic must be added to the path. The subquery processing now becomes part of the original “random read” request. Instead of two simple I/Os to retrieve the record, there can be a minimum of 12 I/Os to retrieve the same record. These I/Os can be done with a result of “not found” if the user is not entitled to any of the records.

For programs that access records sequentially, in or out of key order, the added RCAC logic can have a profound effect on the performance and scalability. Reading the “next record” in order is no longer a simple matter of positioning to the next available key, as shown in Figure 6-23.

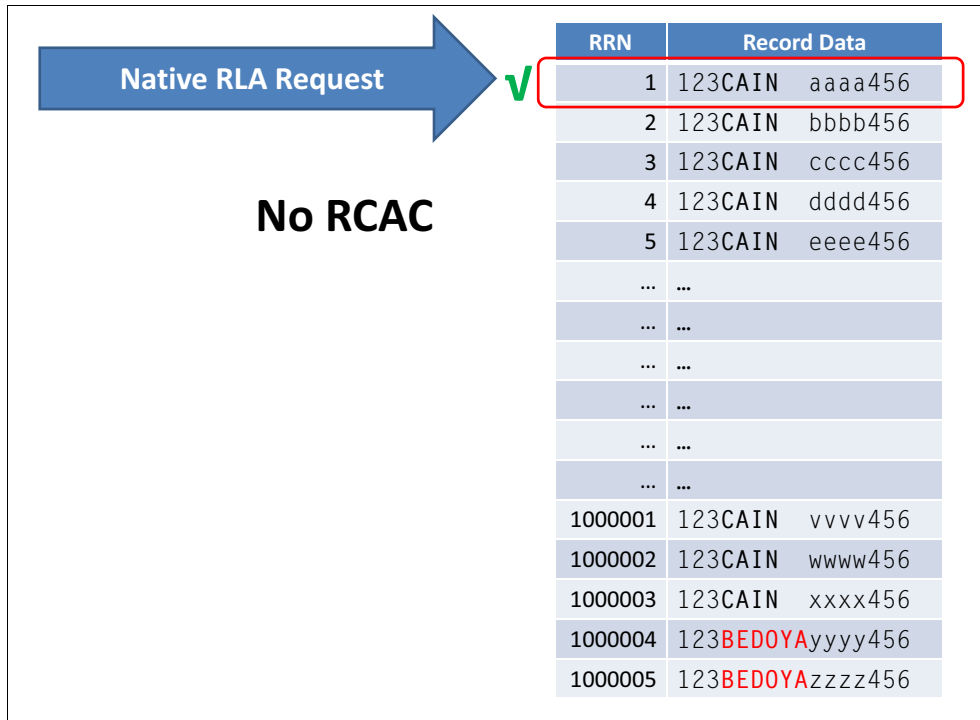


Figure 6-23 Native record access with no RCAC

Before the record, as identified by the key, is considered available, the RCAC logic must be run. If the record is rejected by RCAC, the next record in sequence that is permissible must be identified. This spinning through the records can take a long time and uses many resources, as shown in Figure 6-24.

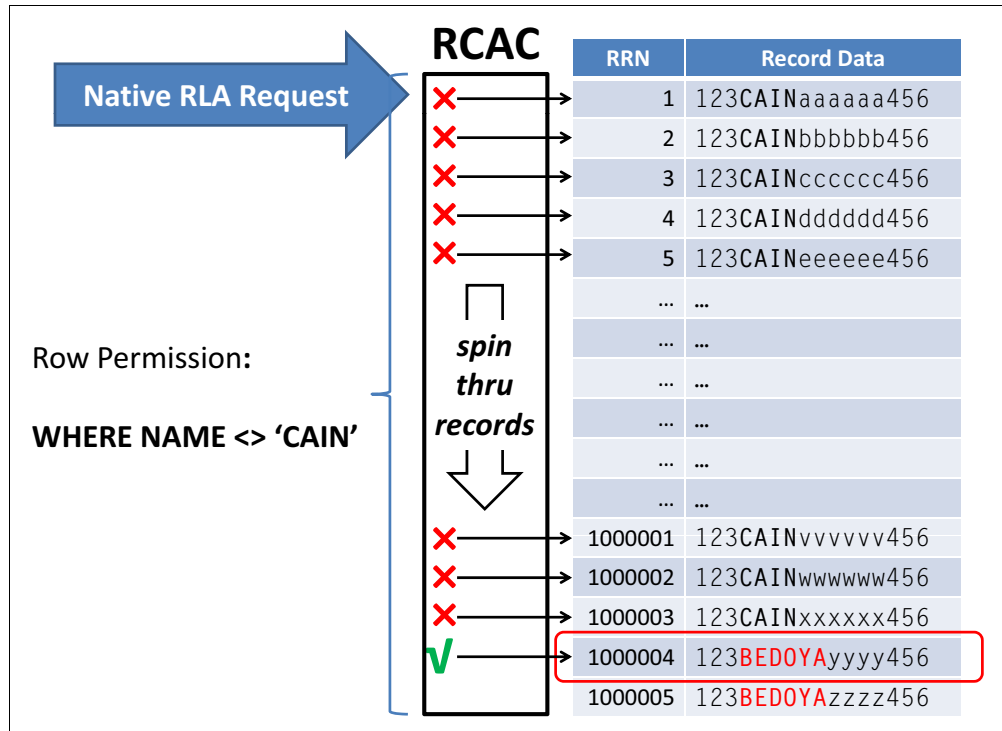


Figure 6-24 Native record level access with RCAC

After the row permissions and column masks are designed and implemented, adequate performance and scalability testing are recommended.

6.7 Exclusive lock to implement RCAC (availability issues)

When defining permissions or enabling RCAC, an exclusive lock on the base table is obtained. The impact to other applications depends on the order of create permission and the alter table to activate RCAC.

Consider the following scenarios:

- ▶ Scenario 1: Adding permissions and RCAC is not enabled on the table:
 - Job 1 reading data from the table (open for input) holds a *SHRRD on the member and a *SHRRD on the data.
 - Job 2 adding, updating, or deleting rows from table (open for output) holds a *SHRRD on the member and a *SHRUPD on the data.
 - Job 4 allocates the object and gets a *SHRRD on the file and a *EXCLRD on the data.
 - Job 3 attempts to add a permission to the table. Permission is added and the pseudo-closed cursors for Job1 and Job 2 are closed. Job 4 still holds the *SHRRD on the file and *EXCLRD on the data.

The net result from Scenario 1 is that you can add permissions without having to end the applications that are reading the base table.

- ▶ Scenario 2: Altering a table to activate RCAC requires that all applications using the table be ended. The alter table requires exclusive use of the table.
- ▶ Scenario 3: Altering the table to activate RCAC before the permissions are added. The alter table requires exclusive use of the table, as in scenario 2. All applications must be ended to perform this alter. After the alter is complete, any applications trying to read data do not get any results, and attempts to insert new rows returns the following message:
 SQ20471] INSERT or UPDATE does not satisfy row permissions.
 To create a permission in this case requires that you end all the applications, unlike scenario 1 where permissions can be added while the applications were active. In this case, the applications must be ended to run the create permission.

6.8 Avoiding propagation of masked data

Operations such as insert or update into a table with active column access control can fail if the input data is masked data. This can happen when data to be inserted or updated contains the masked value as a result of a **SELECT** from a table with active column access control.

For example, assume TABLE1 and TABLE2 have active column access control and for insert, selecting data from TABLE2 returns the masked data. The following INSERT returns an error:

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

The masked data that is returned from the **SELECT * FROM TABLE2** might not be valid input data for TABLE1 because of data type or column check constraint.

There are two ways to prevent this situation from happening: Define a check constraint or create a before trigger.

6.8.1 Check constraint solution

One way to prevent this problem is to define a check constraint.

As part of RCAC, new SQL syntax is provided to allow an action to be performed when a violation of the check constraints check condition occurs instead of giving that error. However, if the check condition is still not met after the action, a hard error is returned. A check constraint with the new on-violation-clause is allowed on both the **CREATE TABLE** and **ALTER TABLE** statements.

In the Example 6-4, 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.

Example 6-4 Check constraint to avoid masked data

```
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-') -- Allow any value other than the mask
    ON UPDATE VIOLATION PRESERVE COL2_ssn -- Don't update the mask portion of the existing value
    ON INSERT VIOLATION SET COL2_ssn = DEFAULT -- for insert set this to the default value.

```

6.8.2 Before trigger solution

The actions that are described in Example 6-4 on page 108 for ON UPDATE VIOLATION and ON INSERT VIOLATION also can be handled by a before trigger, as shown in Example 6-5.

Example 6-5 Before trigger to avoid masked data

```

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

```

6.9 Triggers and functions (SECURED)

There are some considerations that must be considered when there are triggers and functions on tables that have RCAC enabled. The purpose of SECURE for triggers and functions is so that a user who is allowed to create a trigger or function is not necessarily able to make it SECURE themselves. This prevents the trigger/function developer from adding code that skims off data that they are not allowed to see.

6.9.1 Triggers

Triggers have access to the data in rows outside of the row permission or column masking. An after trigger has access to the new row image after the permission has allowed the update or insert to occur. Therefore, the triggers can potentially change the insert or update image value so that it violates the permission.

Any triggers that are defined on a table must be created with an attribute that designates that it is SECURED when RCAC definitions are created or altered for that table, as shown in Example 6-6. The same applies to a view that has an instead of trigger. That trigger must be secure at the point RCAC is enabled for any of the underlying tables the view is over.

Example 6-6 Trigger SECURED

```
/* Trigger created with the SECURED attribute */
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
```

6.9.2 Functions

Within a **CREATE PERMISSION** or **CREATE MASK**, a function can be called. Because that UDF has access to the data before the RCAC rules are applied, the SECURE attribute is required on that function, as shown in Example 6-7.

Example 6-7 Specifying SECURED on a function

```
CREATE PERMISSION SCHEMA.PERM1 ON SCHEMA.TABLE1 FOR ROWS WHERE
MY_UDF(CURRENT_USER,COLUMN1) = 1
ENFORCED FOR ALL ACCESS ENABLE;

CREATE FUNCTION MY_UDF
  (INP1 CHAR(32),
   INP2 INTEGER)
Returns INTEGER
LANGUAGE SQL
CONTAINS SQL
SECURED
```

The SECURED attribute of MY_UDF signifies that the function is considered secure for RCAC. If a function is called from an SQL statement, and references a column in a table that has RCAC, it must be declared as secure. In that case, if the secure function calls other functions, they are not validated to confirm whether they are secure.

Consider the following examples:

- ▶ Table1 has RCAC defined and enabled. **SELECT MY_UDF2(Column2)** from schema.table1. MY_UDF2 must be created with the SECURED attribute. If MY_UDF2 invokes MY_UDF3, there is no checking to ensure that it is also created with SECURED.

NOT SECURED is the default on the create function unless SECURED is explicitly selected.

This same rule applies for any function that might be invoked with a masked column specified as an argument.

- ▶ Table2 column SSN has a column mask that is defined on it.
SELECT MY_UDF4(SSN) from table2. Because SSN has a column mask that is defined, MY_UDF4 must be created with the SECURED attribute.

6.10 RCAC is only one part of the solution

When designing and implementing RCAC row permissions, special attention should be given to the effectiveness and limitations of controlling data access. Data can be housed in objects other than tables or physical files. The role and responsibility of the database user, for example, the database engineer, must be reconciled with their respective authority and access privileges.

Figure 6-25 illustrates that object level security is the first check and that RCAC permissions provide control only on tables and physical files.

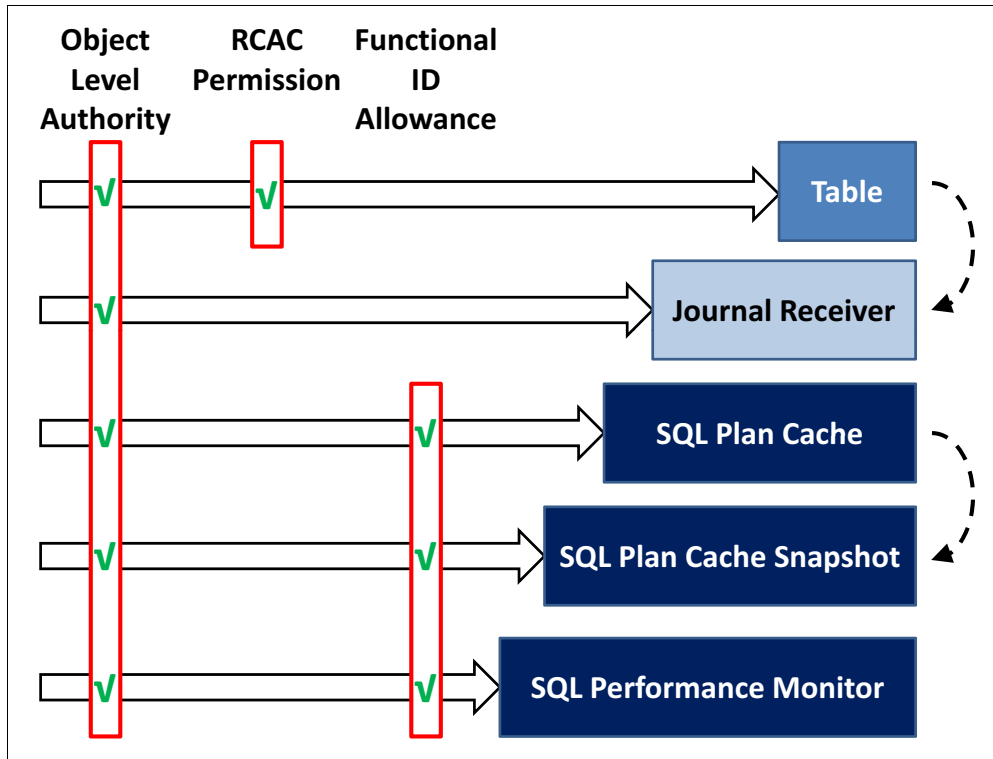


Figure 6-25 Object-level security and RCAC permissions

To get access to the table and the rows, the user must pass the object level authority test and the RCAC permission test.

The IBM i journal captures the transactional data and places an image of the row in the journal receiver. If the user has access to the journal receiver, the row image can be viewed if the user has authority to the journal receiver.

Although the SQL Plan Cache data, the SQL Plan Cache Snapshot data, and the SQL Performance Monitor data do not reveal the results of queries, they can show the literal values that are passed along with the SQL statements.

The ability to monitor, analyze, debug, and tune data-centric applications effectively and efficiently requires some understanding of the underlying data, or at least the attributes of the data. The organization must be willing to reconcile the conflicting requirements of “restricting access to data”, and “needing access to data”.



Row and Column Access Control management

After Row and Column Access Control (RCAC) definitions are defined and activated in a database, your management processes must be adjusted to accommodate these new security controls. This chapter highlights some of the changes that should be considered.

The following topics are covered in this chapter:

- ▶ Managing row permissions and column masks
- ▶ Managing tables with row permissions and column masks
- ▶ Monitoring and auditing function usage

7.1 Managing row permissions and column masks

This section focuses on the management of the RCAC row permissions and column masks.

7.1.1 Source management

The SQL statements that are used to define row permissions and column masks should be managed with a change management process. Ideally, you already are using a change management process for your database definitions, and that same process can be extended to cover your RCAC definitions.

If you are using SQL DDL to define your DB2 tables, then you have the option of adding the RCAC definitions to the same source file as the table definition. The benefit of this approach is that it keeps all DDL that is related to a table in a single source file. The downside is that if you must re-create only the RCAC definitions and leave the table unchanged, then you must identify and extract only the RCAC definitions from the source file. There are situations where the row permissions and column masks must be changed or re-created without changing the definition of the associated table.

7.1.2 Modifying definitions

After RCAC is activated for a table, the row permission and column mask definitions can be re-created to change the data access behavior for that table. Usage of the **OR REPLACE** clause on the **CREATE MASK** and **CREATE PERMISSION** SQL statements simplifies the re-creation process by folding in the deletion of the existing RCAC definition.

This capability makes it easy to change your RCAC definitions as you test the controls with your applications and identify tweaks that must be made to your RCAC implementation. However, re-creation of RCAC definitions does require an exclusive lock to be acquired on the table during the process.

7.1.3 Turning on and off

As described in 3.1.2, “Enabling and activating RCAC” on page 16, the SQL **ALTER** statement can turn on and off row permissions and column masks. The **ALTER MASK** and **ALTER PERMISSION** statements allow an individual row permission or column mask to be turned off with the **DISABLE** option and back on with the **ENABLE** option. The **ALTER TABLE** statement can deactivate enforcement of all the row permissions and column masks for a single table.

Important: Although these capabilities make it easy to temporarily turn off RCAC security so that you can make environment or application changes, these processes require an exclusive lock to be obtained on a table. Therefore, this activity must be planned carefully to avoid disruptions and outages.

7.1.4 Regenerating

DB2 also can regenerate an existing row permission or column mask. This regenerate option can be useful with more complex RCAC definitions that reference other DB2 objects.

For example, consider a row permission on an ACCOUNTS table (PERMISSION1_ON_ACCOUNTS). The ACCOUNTS table row permission references and compares columns in the CUSTOMERS table. When the definition of the CUSTOMERS table changes, DB2 does not check to determine whether the change to the CUSTOMERS table breaks the ACCOUNTS table row permission. If this table definition change does break the row permission, an error does not surface until an application tries to read rows from the ACCOUNTS table.

Instead of waiting for an application to detect this error, the **REGENERATE** option can be used on the ACCOUNTS row permission. The **REGENERATE** option returns an error if the change in the CUSTOMERS table definition causes the row permission to be invalid. In this way, the row permission can be proactively corrected before an application discovers the error.

7.2 Managing tables with row permissions and column masks

This section examines the object management considerations after RCAC is added to a DB2 table.

7.2.1 Save and restore

Row permissions and column masks are stored in the DB2 table object itself, so they are automatically saved and restored when the DB2 table object is saved and restored. Therefore, no adjustments must be made to your database backup process to accommodate RCAC.

Save and restore processing works fine with RCAC if the RCAC definition does not reference other DB2 objects other than the table over which they are defined. When the RCAC definition has dependencies on other DB2 objects, the restore process is much more challenging.

For example, assume that the BANKSCHEMA library (which is the system name or short name for the schema long name of BANK_SCHEMA) is saved and restored into a library named BANK_TEST. Recall from the example in 7.1.4, “Regenerating” on page 114 that the row permission on the ACCOUNTS table references the CUSTOMERS table (...SELECT C.CUSTOMER_ID FROM CUSTOMERS C...). After the restore operation, the ACCOUNTS row permission still references the CUSTOMERS table in BANK_SCHEMA because DB2 explicitly qualifies all object references when the row permission or column mask is created. The restore processing does not change the explicit qualification from BANK_SCHEMA to BANK_TEST. As a result, the restored ACCOUNTS row permission now depends on DB2 objects residing in a different schema, even though it was not created that way originally. For more details, see Figure 7-1.

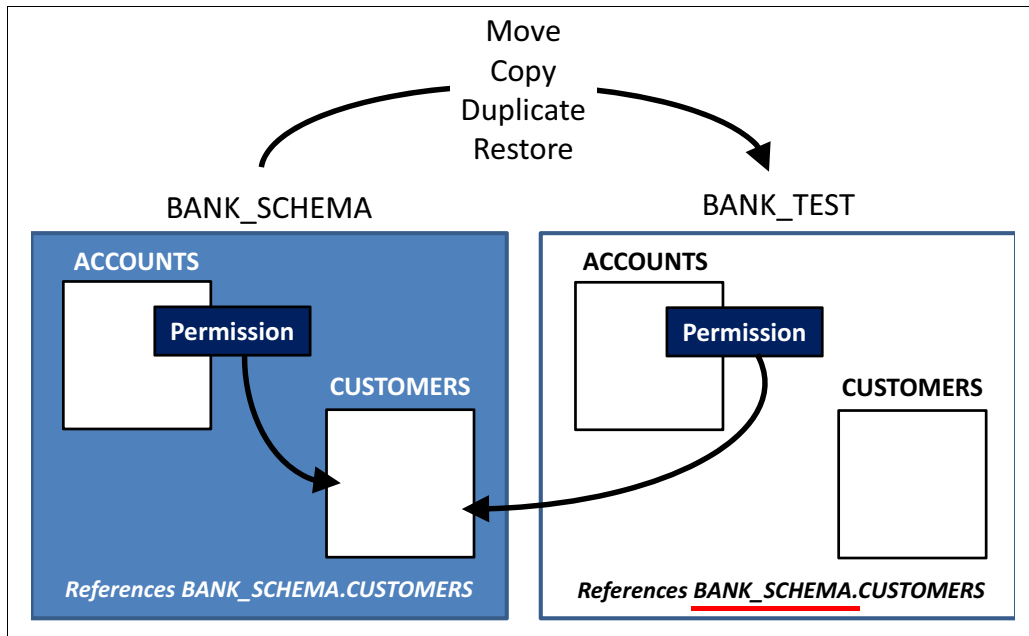


Figure 7-1 Restoring tables to different schemas

The only way to fix this issue is to re-create the row permission or column mask after the restore operation. Re-creation of the row permission or column mask is required only for definitions that reference other DB2 objects, but it is simpler to re-create all of the RCAC definitions instead of a subset. For example, generate the SQL using System i Navigator, clear the “Schema qualify names for objects” and select the “OR REPLACE clause”, and then run the generated script.

7.2.2 Table migration

There are several IBM i CL commands, such as Move Object (MOVOBJ), Create Duplicate Object (CRTDUPOBJ), and Copy Library (CPYLIB), which are used to migrate a table from one library to another one. Often, this migration is done to create different versions of the table that can be used for development or testing purposes.

The migration of a table with RCAC has the same challenges as restore processing. If the RCAC definition references other DB2 objects, then IBM i CL commands do not change the schema names that are explicitly qualified by the DB2 internal RCAC processing.

Again, re-creating the row permission or column mask is the only way to fix the issue of references to DB2 objects in other schemas.

7.3 Monitoring and auditing function usage

While establishing proper roles for users, separating duties using function usage IDs, and defining RCAC policies allows you to implement an effective and pervasive data access control scheme. How do you monitor and audit everyone who is involved in the implementation of that scheme? The answer is to use IBM i journaling. A special journal that is called QAUDJRN, also known as the *audit journal*, can provide a record and audit trail of many security relevant events that occur on the system, including RCAC-related events.

The tasks and operations of security administrators and database engineers who are collaborating can (and should) be effectively monitored and audited to ensure that the organization's data access control and governance policies are in place and enabled. For example, the Database Engineers can be involved in designing and developing functions and triggers that must be secured using the SECURE attribute. Otherwise, without properly securing functions and triggers, the RCAC controls can be bypassed.

A new journal entry type of "AX" for journal entry code "T" (audit trail) is now used for RCAC. More information about the journaling of RCAC operations can be found in the following documents:

- ▶ *IBM i Version 7.2 Journal Management Guide*, found at:

http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzaki/rzakiprintthis.htm?lang=en

- ▶ *IBM i Version 7.2 Security Reference Guide*, found at:

http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzar1/rzar1kickoff.htm?lang=en



Designing and planning for success

Although successfully implementing Row and Column Access Control (RCAC) is based on knowledge and skills, designing and planning are fundamental aspects. This chapter describes the need for a deep understanding of the technology, and good design, proper planning, and adequate testing.

The following topics are covered in this chapter:

- ▶ Implementing RCAC with good design and proper planning
- ▶ DB2 for i Center of Excellence

8.1 Implementing RCAC with good design and proper planning

By using RCAC, the row and column data that is returned to the requester can be controlled and governed by a set of data-centric policies that are defined with SQL and implemented within DB2 for i.

RCAC provides fine-grained access control and is complementary to IBM i object-level security. With the new RCAC feature of DB2 for i, the database engineer, in partnership with the data owner and security officer, can ensure that users have access to the data based on their level of authorization and responsibility.

This situation also can include separation of duties, such as allowing the application developers to design and implement the solutions, but restricting them from accessing the production data based on policy. Just because someone writes and owns the program, it does not mean that they have access to all the sensitive data that their program can potentially read.

This paper has described the following pervasive power and advantages of RCAC:

- ▶ Access can be controlled through simple or sophisticated logic.
- ▶ Virtually no application changes are required.
- ▶ The implementation of the access policy is part of the DB2 data access layer.
- ▶ Table data is protected regardless of the interface that is used.
- ▶ No user is inherently exempted from the access control policies.
- ▶ Groups of users can share policies and permissions.

A deep understanding of the technology, and proper planning, good design, adequate testing, and monitored deployment are critical for success. This includes the usage of quality assurance testing, and realistic performance and scalability exercises that serve to demonstrate that all of your requirements are being met. As part of the verification process, the usage of in-depth proofs of concepts and proofs of technology are recommended, if not essential. When RCAC is activated, the results of queries can change. Anticipating this change and realizing the effects of RCAC before going live are of the utmost importance.

With the ever-growing value of data, and the vast and varied database technology that is available today, it is crucial to have a person or persons on staff who specialize in data-centric design, development, and deployment. This role and responsibility falls on the database engineer. With the availability of DB2 RCAC, the importance of full-time database engineering has grown.

8.2 DB2 for i Center of Excellence

To further assist you with understanding and implementing RCAC, the DB2 for i Center of Excellence team offers an RCAC education and consulting workshop. In addition to knowledge transfer, a working session allows for a review of your data access control requirements, review of the current environment, solution ideation, and high-level solution design.

If you are interested in engaging with the DB2 for i Center of Excellence, contact Mike Cain at mcaain@us.ibm.com.



Database definitions for the RCAC banking example

This appendix provides the database definitions or DDLs to re-create the Row and Column Access Control (RCAC) scenario that is described in Chapter 4, “Implementing Row and Column Access Control: Banking example” on page 37. The script that is shown in Example A-1 is the DDL script that is used to implement this example.

Example A-1 DDL script to implement the RCAC banking example

```
/* Database Definitions for RCAC Bank Scenario */
/* Schema */
CREATE SCHEMA BANK_SCHEMA FOR SCHEMA BANKSCHEMA ;

/* Global Variable */
CREATE VARIABLE BANK_SCHEMA.CUSTOMER_LOGIN_ID
  VARCHAR( 30) ;

LABEL ON VARIABLE BANK_SCHEMA.CUSTOMER_LOGIN_ID IS 'Customer''s log in value passed by web application' ;

/* Tables */

CREATE TABLE BANK_SCHEMA.CUSTOMERS (
  CUSTOMER_ID FOR COLUMN CUST000001 INTEGER GENERATED ALWAYS AS IDENTITY (
    START WITH 1 INCREMENT BY 1
    NO MINVALUE NO MAXVALUE
    NO CYCLE NO ORDER
    CACHE 20 ),
  CUSTOMER_NAME FOR COLUMN CUST000002 VARCHAR(30) CCSID 37 NOT NULL ,
  CUSTOMER_ADDRESS FOR COLUMN CUST000003 VARCHAR(30) CCSID 37 NOT NULL ,
  CUSTOMER_CITY FOR COLUMN CUST000004 VARCHAR(30) CCSID 37 NOT NULL ,
  CUSTOMER_STATE FOR COLUMN CUST000005 CHAR(2) CCSID 37 NOT NULL ,
  CUSTOMER_PHONE FOR COLUMN CUST000006 CHAR(10) CCSID 37 NOT NULL ,
  CUSTOMER_EMAIL FOR COLUMN CUST000007 VARCHAR(30) CCSID 37 NOT NULL ,
  CUSTOMER_TAX_ID FOR COLUMN CUST000008 CHAR(11) CCSID 37 NOT NULL ,
  CUSTOMER_DRIVERS_LICENSE_NUMBER FOR COLUMN CUST000012 CHAR(13) CCSID 37 DEFAULT NULL ,
  CUSTOMER_LOGIN_ID FOR COLUMN CUST000009 VARCHAR(30) CCSID 37 DEFAULT NULL ,
  CUSTOMER_SECURITY_QUESTION FOR COLUMN CUST000010 VARCHAR(100) CCSID 37 DEFAULT NULL ,
```

```

CUSTOMER_SECURITY_QUESTION_ANSWER FOR COLUMN CUSTO00011 VARCHAR(100) CCSID 37 DEFAULT NULL ,
INSERT_TIMESTAMP FOR COLUMN INSER00001 TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP IMPLICITLY HIDDEN ,
UPDATE_TIMESTAMP FOR COLUMN UPDAT00001 TIMESTAMP GENERATED ALWAYS FOR EACH ROW ON UPDATE
    AS ROW CHANGE TIMESTAMP NOT NULL IMPLICITLY HIDDEN ,
CONSTRAINT BANK_SCHEMA.CUSTOMER_ID_PK PRIMARY KEY( CUSTOMER_ID ) ) ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
ADD CONSTRAINT BANK_SCHEMA.CUSTOMER_LOGIN_ID_UK
UNIQUE( CUSTOMER_LOGIN_ID ) ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
ADD CONSTRAINT BANK_SCHEMA.CUSTOMER_DRIVERS_LICENSE_CHECK
CHECK( CUSTOMER_DRIVERS_LICENSE_NUMBER <> '*****' )
ON UPDATE VIOLATION PRESERVE CUSTOMER_DRIVERS_LICENSE_NUMBER ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
ADD CONSTRAINT BANK_SCHEMA.CUSTOMER_EMAIL_CHECK
CHECK( CUSTOMER_EMAIL <> '****@****' )
ON UPDATE VIOLATION PRESERVE CUSTOMER_EMAIL ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
ADD CONSTRAINT BANK_SCHEMA.CUSTOMER_LOGIN_ID_CHECK
CHECK( CUSTOMER_LOGIN_ID <> '*****' )
ON INSERT VIOLATION SET CUSTOMER_LOGIN_ID = DEFAULT
ON UPDATE VIOLATION PRESERVE CUSTOMER_LOGIN_ID ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
ADD CONSTRAINT BANK_SCHEMA.CUSTOMER_SECURITY_QUESTION_CHECK
CHECK( CUSTOMER_SECURITY_QUESTION_ANSWER <> '*****' )
ON INSERT VIOLATION SET CUSTOMER_SECURITY_QUESTION_ANSWER = DEFAULT
ON UPDATE VIOLATION PRESERVE CUSTOMER_SECURITY_QUESTION_ANSWER ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
ADD CONSTRAINT BANK_SCHEMA.CUSTOMER_SECURITY_QUESTION_ANSWER
CHECK( CUSTOMER_SECURITY_QUESTION <> '*****' )
ON INSERT VIOLATION SET CUSTOMER_SECURITY_QUESTION = DEFAULT
ON UPDATE VIOLATION PRESERVE CUSTOMER_SECURITY_QUESTION ;

ALTER TABLE BANK_SCHEMA.CUSTOMERS
ADD CONSTRAINT BANK_SCHEMA.CUSTOMER_TAX_ID_CHECK
CHECK( CUSTOMER_TAX_ID <> 'XXX-XX-XXXX' AND SUBSTR ( CUSTOMER_TAX_ID , 1 , 7 ) <> 'XXX-XX-' )
ON UPDATE VIOLATION PRESERVE CUSTOMER_TAX_ID ;

CREATE TABLE BANK_SCHEMA.ACCOUNTS (
ACCOUNT_ID INTEGER GENERATED ALWAYS AS IDENTITY (
START WITH 1 INCREMENT BY 1
NO MINVALUE NO MAXVALUE
NO CYCLE NO ORDER
CACHE 20 ),
CUSTOMER_ID FOR COLUMN CUSTID INTEGER NOT NULL ,
ACCOUNT_NUMBER FOR COLUMN ACCOUNTNO VARCHAR(50) CCSID 37 NOT NULL ,
ACCOUNT_NAME FOR COLUMN ACCOUNTNAM CHAR(12) CCSID 37 NOT NULL ,
ACCOUNT_DATE_OPENED FOR COLUMN OPENDATE DATE DEFAULT CURRENT_DATE ,
ACCOUNT_DATE_CLOSED FOR COLUMN CLOSEDATE DATE DEFAULT NULL ,
ACCOUNT_CURRENT_BALANCE FOR COLUMN ACCTBAL DECIMAL(11, 2) NOT NULL DEFAULT 0 ,
INSERT_TIMESTAMP FOR COLUMN INSDATE TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP IMPLICITLY HIDDEN ,
UPDATE_TIMESTAMP FOR COLUMN UPDDATE TIMESTAMP GENERATED ALWAYS FOR EACH ROW ON UPDATE
    AS ROW CHANGE TIMESTAMP NOT NULL IMPLICITLY HIDDEN ,
CONSTRAINT BANK_SCHEMA.ACCOUNT_ID_PK PRIMARY KEY( ACCOUNT_ID ) );

```

```

ALTER TABLE BANK_SCHEMA.ACCOUNTS
  ADD CONSTRAINT BANK_SCHEMA.ACCOUNT_CUSTOMER_ID_FK
  FOREIGN KEY( CUSTOMER_ID )
  REFERENCES BANK_SCHEMA.CUSTOMERS ( CUSTO00001 )
  ON DELETE RESTRICT
  ON UPDATE RESTRICT ;

ALTER TABLE BANK_SCHEMA.ACCOUNTS
  ADD CONSTRAINT BANK_SCHEMA.ACCOUNT_NUMBER_CHECK
  CHECK( ACCOUNT_NUMBER <> '*****' )
  ON UPDATE VIOLATION PRESERVE ACCOUNT_NUMBER ;

CREATE TABLE BANK_SCHEMA.TRANSACTIONS FOR SYSTEM NAME TRANS (
  TRANSACTION_ID FOR COLUMN TRANS00001 INTEGER GENERATED ALWAYS AS IDENTITY (
  START WITH 1 INCREMENT BY 1
  NO MINVALUE NO MAXVALUE
  NO CYCLE NO ORDER
  CACHE 20 ),
  ACCOUNT_ID INTEGER NOT NULL ,
  TRANSACTION_TYPE FOR COLUMN TRANS00002 CHAR(1) CCSID 37 NOT NULL ,
  TRANSACTION_DATE FOR COLUMN TRANS00003 DATE NOT NULL DEFAULT CURRENT_DATE ,
  TRANSACTION_TIME FOR COLUMN TRANS00004 TIME NOT NULL DEFAULT CURRENT_TIME ,
  TRANSACTION_AMOUNT FOR COLUMN TRANS00005 DECIMAL(11, 2) NOT NULL ,
  INSERT_TIMESTAMP FOR COLUMN INSER00001 TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP IMPLICITLY HIDDEN ,
  UPDATE_TIMESTAMP FOR COLUMN UPDAT00001 TIMESTAMP GENERATED ALWAYS FOR EACH ROW ON UPDATE
  AS ROW CHANGE TIMESTAMP NOT NULL IMPLICITLY HIDDEN ,
  CONSTRAINT BANK_SCHEMA.TRANSACTION_ID_PK PRIMARY KEY( TRANSACTION_ID ) ) ;

ALTER TABLE BANK_SCHEMA.TRANSACTIONS
  ADD CONSTRAINT BANK_SCHEMA.TRANSACTIONS_ACCOUNT_ID_FK
  FOREIGN KEY( ACCOUNT_ID )
  REFERENCES BANK_SCHEMA.ACCOUNTS ( ACCOUNT_ID )
  ON DELETE RESTRICT
  ON UPDATE RESTRICT ;

/* Permissions and Masks */

CREATE PERMISSION BANK_SCHEMA.PERMISSION1_ON_CUSTOMERS ON BANK_SCHEMA.CUSTOMERS AS C
  FOR ROWS WHERE ( QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'DBE' , 'ADMIN' , 'TELLER' ) = 1 )
OR
( QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
AND ( C . CUSTOMER_LOGIN_ID = BANK_SCHEMA . CUSTOMER_LOGIN_ID ) )
  ENFORCED FOR ALL ACCESS
  ENABLE ;

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

CREATE MASK BANK_SCHEMA.MASK_TAX_ID_ON_CUSTOMERS ON BANK_SCHEMA.CUSTOMERS AS C
  FOR COLUMN CUSTOMER_TAX_ID
  RETURN CASE
  WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'ADMIN' ) = 1

```

```

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 ;

```

```

CREATE PERMISSION BANK_SCHEMA.PERMISSION1_ON_ACCOUNTS ON BANK_SCHEMA.ACCOUNTS AS A
  FOR ROWS WHERE ( QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'DBE' , 'ADMIN' , 'TELLER' ) = 1 )
OR
( QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
AND ( A . CUSTOMER_ID IN (
SELECT C . CUSTOMER_ID
FROM BANK_SCHEMA . CUSTOMERS C
WHERE C . CUSTOMER_LOGIN_ID = BANK_SCHEMA . CUSTOMER_LOGIN_ID

  ENFORCED FOR ALL ACCESS
  ENABLE ;

CREATE MASK BANK_SCHEMA.MASK_ACCOUNT_NUMBER_ON_ACCOUNTS ON BANK_SCHEMA.ACCOUNTS AS A
  FOR COLUMN ACCOUNT_NUMBER
  RETURN CASE
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'ADMIN' ) = 1
THEN A . ACCOUNT_NUMBER
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'TELLER' ) = 1
THEN A . ACCOUNT_NUMBER
WHEN QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
THEN A . ACCOUNT_NUMBER
ELSE '*****'
END
  ENABLE ;

ALTER TABLE BANK_SCHEMA.ACCOUNTS
  ACTIVATE ROW ACCESS CONTROL
  ACTIVATE COLUMN ACCESS CONTROL ;

CREATE PERMISSION BANK_SCHEMA.PERMISSION1_ON_TRANSACTIONS ON BANK_SCHEMA.TRANSACTIONS AS T
  FOR ROWS WHERE ( QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'DBE' , 'ADMIN' , 'TELLER' ) = 1 )
OR
( QSYS2 . VERIFY_GROUP_FOR_USER ( SESSION_USER , 'CUSTOMER' ) = 1
AND ( T . ACCOUNT_ID IN (
SELECT A . ACCOUNT_ID
FROM BANK_SCHEMA . ACCOUNTS A
WHERE A . CUSTOMER_ID IN (
SELECT C . CUSTOMER_ID
FROM BANK_SCHEMA . CUSTOMERS C
WHERE C . CUSTOMER_LOGIN_ID = BANK_SCHEMA . CUSTOMER_LOGIN_ID

  ENFORCED FOR ALL ACCESS
  ENABLE ;

ALTER TABLE BANK_SCHEMA.TRANSACTIONS
  ACTIVATE ROW ACCESS CONTROL ;

/* END */

```

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this paper.

Other publications

These publications are relevant as further information sources:

- ▶ *IBM DB2 for i indexing methods and strategies* white paper:
http://www.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_wp_db2_i_indexing_methods_strategies
- ▶ *IBM i Memo to Users Version 7.2:*
http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzahg/rzahgmtu.htm
- ▶ *IBM i Version 7.2 DB2 for i SQL Reference Guide:*
http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/db2/rbafzintro.htm?lang=en
- ▶ *IBM i Version 7.2 Journal Management Guide:*
http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzaki/rzakiprintthis.htm?lang=en
- ▶ *IBM i Version 7.2 Security Reference Guide:*
http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzar1/rzar1kickoff.htm?lang=en

Online resources

These websites are relevant as further information sources:

- ▶ Database programming topic of the IBM i 7.2 IBM Knowledge Center:
http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzahg/rzahgdbp.htm?lang=en
- ▶ Identity Theft Resource Center
<http://www.idtheftcenter.org>
- ▶ Ponemon Institute
<http://www.ponemon.org/>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



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