(ISC)²®

SECURITY CONGRESS | 2017

congress.isc2.org

#ISC2Congress

LEADERS OF TOMORROW

Row and Column Access Control in Db2 11

(Db2 on Linux, UNIX and Windows)

Philip K. Gunning, CISSP

Privacy and Data Protection Mandate

» Regulations and Standards stipulate that an individual is allowed access only to the subset of that information that is needed to perform their job function

Privacy and Data Protection Mandate

- US Health Insurance Portability and Accountability Act (HIPAA)
  - Doctor only authorized to view medical records of their own patients but not records of other patients
- Payment Card Industry Data Security Standard (PCI DSS)
  - Access to cardholder data such as card number must be restricted by business need-to-know
- European General Data Protection Regulation (GDPR)
  - Enforcement starts May 2018

Legacy Security

- Database views
  - Can get complicated for many users
  - Requires ongoing database administration and tuning
- Application-based security
  - Application filtering logic and maintenance required or application specific configuration logic and tables required

Label Based Access Control -- LBAC

- Security labels and security policy objects map security requirements into security tables
- Database “Security” Administrator creates security label objects for users to access protected tables
  - Grants security labels and exemptions to users
  - Alters tables to add security label column and associates a security policy with the table
- Rarely suitable for commercial customers as it requires data to be classified and has set of fixed security rules
- No Read Up and No Write Down Rule
- Generally suitable for defense and intelligence organizations requiring multilevel security
- Little commercial use

Row and Column Access Control -- RCAC

- » RCAC satisfies legal and regulatory requirements by controlling access at the row and column levels
- » Prior to RCAC, Views on tables and application logic were the primary means used to limit access to data
  - Views required significant Database Administration overhead such as design, tuning for performance, review of access requirements and creation for many users and groups
- » RCAC is just another tool in your arsenal to protect data!

Row and Column Access Control

- Additional layer of data security introduced in DB2 V10
- Complementary to table level authorization
- Allows access only to subset of data useful for job task
- Controls access to a table at the row, column or both
- Two sets of rules
  - Permissions for rows
  - Masks for columns

How Does RCAC Work?

- The table-privileges security model is applied first to determine whether the user is allowed access to the table
- If allowed, row permissions are applied to determine specific rows
- Column masks are then applied to determine whether the user sees the actual or masked value in the column

How Does RCAC Work?

- For example, when Dr. Jones queries the patients table, he sees only rows that represent patients under his care
- A column mask defined on the phone # column ensures that Dr. Smith sees only phone numbers for patients that have shared their phone # with him
- For other patients the phone # would be set to NULL or masked out according to the column mask definition

Benefits of RCAC

- Key advantage is that no database user is inherently exempted from RCAC rules
- Even users with DATAACCESS authority have no access
- RCAC by default can only be managed by the user designated as the Database Security Administrator (SECADM)

Row and Column Access Control (RCAC)

- RCAC was first introduced in Db2 in version 10 in 2012
- Transparent to applications and no application changes needed
- Enhanced in latest Db2 version 11.1.2.2
- Included in all editions of Db2
- Discretionary access control

SECADM

» SECADM is Db2 Security Administrator group created at Database Creation
  - Required in order to create RCAC
  - Separate from DBADM
  - Separates security functions from Database Administrators
  - Creates and assigns ROLES to GROUPS
  - GRANTS access to ROLES

Row and Column Access Control (RCAC)

- RCAC is used to control access to rows and columns based on rules defined on the rows and columns
- Supports ROLES and uses them to restrict the access based on RCAC

RCAC – How it Works

- Table-privilege security model is applied first to determine if user allowed to access table
- If allowed, row permissions are applied next to determine specific rows user has access to
- Column masks are then applied to determine whether the user sees the actual or masked value in the column

RCAC

- Examples that follow use the Db2 **SAMPLE** database and sample examples supplied as part of the **SAMPLES** folder with all Db2 installs

Scenario: Health Care

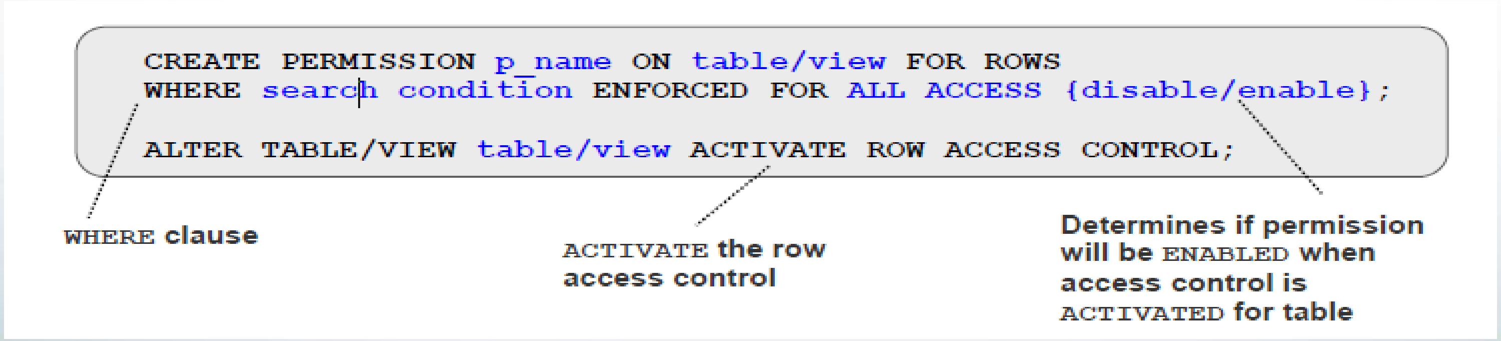
- In our scenario we will use patients, doctors, pharmacists, administrators and accountants to show how you can use RCAC to either allow or limit access to these ROLES
- Tables used are: Patients, Patientchoice, and Acct_history

Scenario: Create Permission

- **Patients**
  - Can only access their own data
- **Physicians**
  - Can only access their own patients’ data
- **Membership officers, Accounting, Drug Researchers**
  - Can access all data
- **Nobody** else sees any data

Create PERMISSION Syntax

» To create a permission governing access to rows
  - CREATE the permission with access rule defined by search condition
    - Choose to enforce for all DML or simply SELECT
  - ENABLE or DISABLE the permission
    - If enabled this access rule will be implemented when row access control is ACTIVATED for the affected table
  - ALTER the table to activate ROW access control



```
CREATE PERMISSION p_name ON table/view FOR ROWS
WHERE search condition ENFORCED FOR ALL ACCESS {disable/enable};

ALTER TABLE/VIEW table/view ACTIVATE ROW ACCESS CONTROL;
```

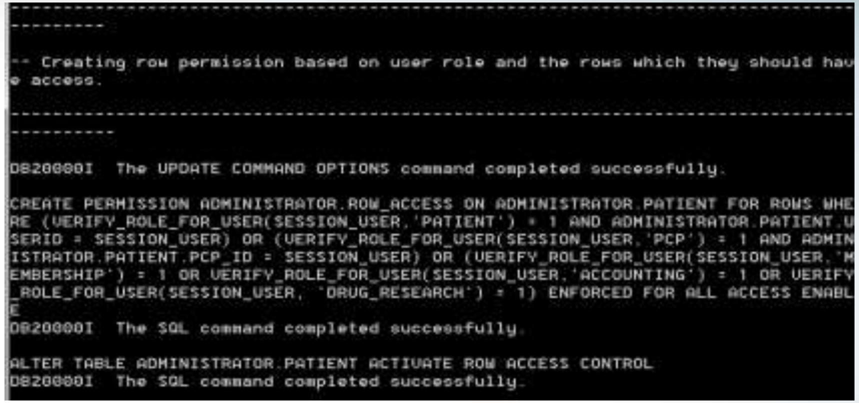
WHERE clause

ACTIVATE the row access control

Determines if permission will be ENABLED when access control is ACTIVATED for table

Create Permission

```
-- ROLE PATIENT is allowed to access his or her own row;
-- ROLE PCP is allowed to access his or her patients rows;
-- ROLE MEMBERSHIP, ACCOUNTING, and DRUG_RESEARCH are;
-- allowed to access all rows.;

CREATE PERMISSION ADMINISTRATOR.ROW_ACCESS ON ADMINISTRATOR.PATIENT
FOR ROWS WHERE (VERIFY_ROLE_FOR_USER(SESSION_USER,'PATIENT') = 1
AND
ADMINISTRATOR.PATIENT.USERID = SESSION_USER) OR
(VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1
AND
ADMINISTRATOR.PATIENT.PCP_ID = SESSION_USER) OR
        (VERIFY_ROLE_FOR_USER(SESSION_USER,'MEMBERSHIP') = 1 OR
        VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1 OR
        VERIFY_ROLE_FOR_USER(SESSION_USER, 'DRUG_RESEARCH') = 1)
ENFORCED FOR ALL ACCESS
ENABLE@
--Altering the table to activate the row access control feature.;

ALTER TABLE ADMINISTRATOR.PATIENT ACTIVATE ROW ACCESS CONTROL@
```

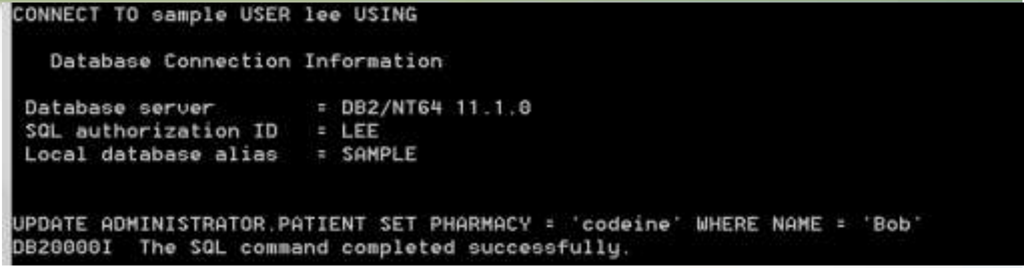


```
--------------------------------------------------------------------------------
---------

-- Creating row permission based on user role and the rows which they should hav
e access.

--------------------------------------------------------------------------------
----------

DB20000I  The UPDATE COMMAND OPTIONS command completed successfully.

CREATE PERMISSION ADMINISTRATOR.ROW_ACCESS ON ADMINISTRATOR.PATIENT FOR ROWS WHE
RE (VERIFY_ROLE_FOR_USER(SESSION_USER,'PATIENT') = 1 AND ADMINISTRATOR.PATIENT.U
SERID = SESSION_USER) OR (VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1 AND ADMIN
ISTRATOR.PATIENT.PCP_ID = SESSION_USER) OR (VERIFY_ROLE_FOR_USER(SESSION_USER,'M
EMBERSHIP') = 1 OR VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1 OR VERIFY
_ROLE_FOR_USER(SESSION_USER, 'DRUG_RESEARCH') = 1) ENFORCED FOR ALL ACCESS ENABL
E
DB20000I  The SQL command completed successfully.

ALTER TABLE ADMINISTRATOR.PATIENT ACTIVATE ROW ACCESS CONTROL
DB20000I  The SQL command completed successfully.
```

Scenario: Update Table with Permissions

As Dr. Lee -- UPDATE ADMINISTRATOR.PATIENT

SET PHARMACY = 'codeine' WHERE NAME = **'Bob'**



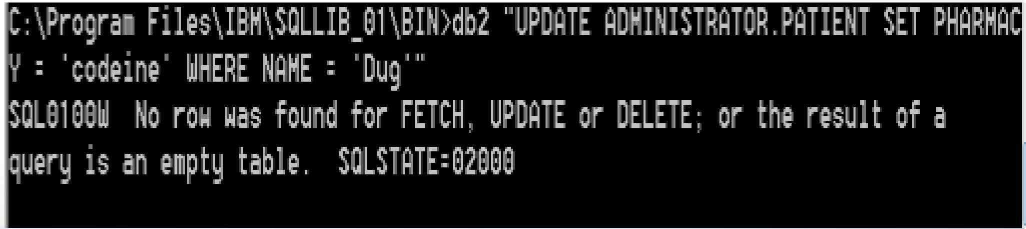
```
CONNECT TO sample USER lee USING

   Database Connection Information

 Database server        = DB2/NT64 11.1.0
 SQL authorization ID   = LEE
 Local database alias   = SAMPLE

UPDATE ADMINISTRATOR.PATIENT SET PHARMACY = 'codeine' WHERE NAME = 'Bob'
DB20000I  The SQL command completed successfully.
```

However, when Dr. Lee tries to update the pharmacy code for patient Dug, the update fails as Dug is not a patient of Dr. Lee.

UPDATE ADMINISTRATOR.PATIENT SET

PHARMACY = 'codeine'

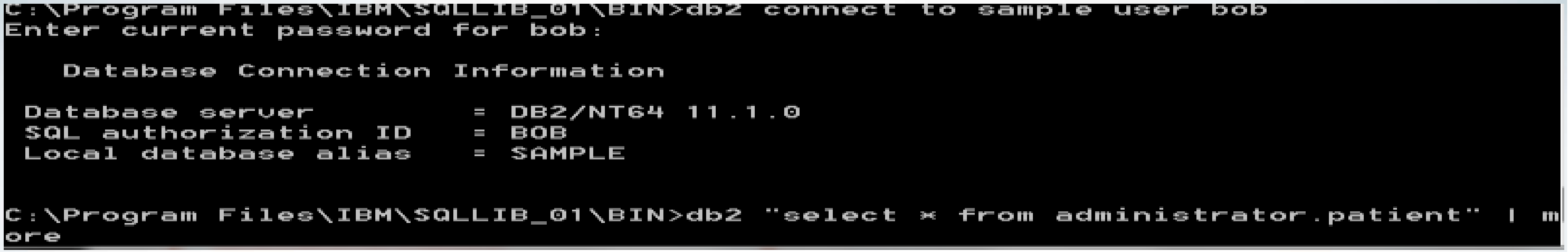
WHERE NAME = 'Dug'



```
C:\Program Files\IBM\SQLLIB_01\BIN>db2 "UPDATE ADMINISTRATOR.PATIENT SET PHARMAC
Y = 'codeine' WHERE NAME = 'Dug'"
SQL0100W  No row was found for FETCH, UPDATE or DELETE; or the result of a
query is an empty table.  SQLSTATE=02000
```

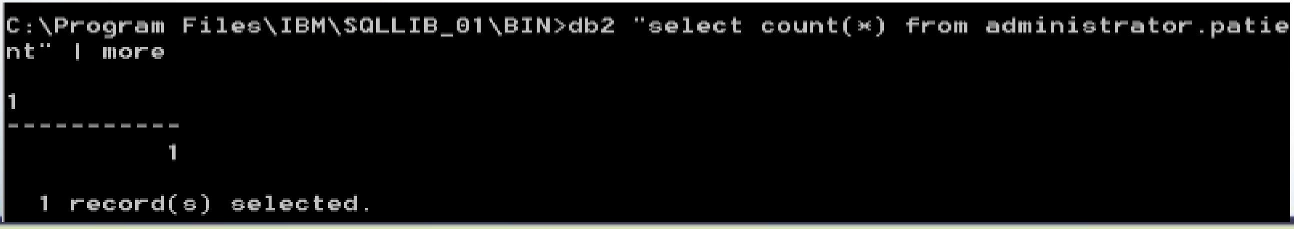
Scenario: Select from table as patient

» Only one row is returned for Bob, he cannot see data of other patients:



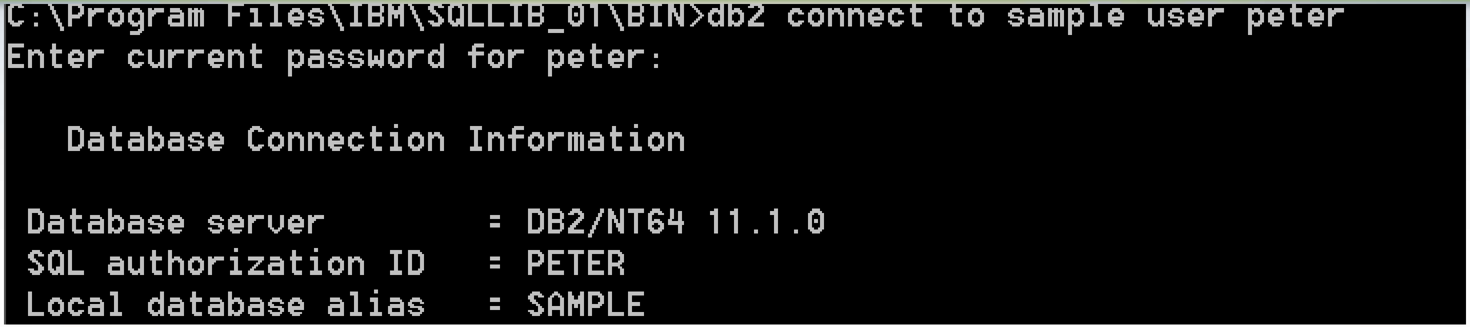
```
C:\Program Files\IBM\SQLLIB_01\BIN>db2 connect to sample user bob
Enter current password for bob:

   Database Connection Information

 Database server        = DB2/NT64 11.1.0
 SQL authorization ID   = BOB
 Local database alias   = SAMPLE


C:\Program Files\IBM\SQLLIB_01\BIN>db2 "select * from administrator.patient" | m
ore
```

Even a Select count(*) run by Bob only returns a count of 1 row



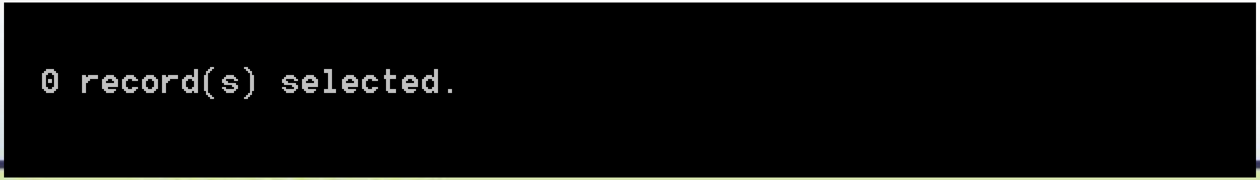
```
C:\Program Files\IBM\SQLLIB_01\BIN>db2 "select count(*) from administrator.patie
nt" | more

1
-----------
          1

  1 record(s) selected.
```

Scenario: Select from Table as Database Administrator – NO Rows Returned



```
C:\Program Files\IBM\SQLLIB_01\BIN>db2 connect to sample user peter
Enter current password for peter:

   Database Connection Information

 Database server        = DB2/NT64 11.1.0
 SQL authorization ID   = PETER
 Local database alias   = SAMPLE
```

» C:\Program Files\IBM\SQLLIB_01\BIN>db2 "select * from administrator.patient"



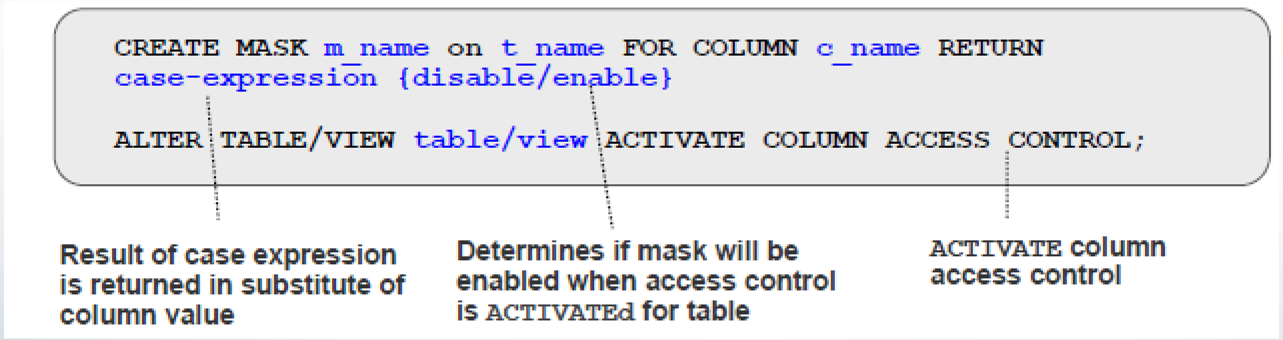
```

  0 record(s) selected.

```

Scenario: Create Column Mask

- This scenario has the following permissions attached:
- Account balance column
  - Accounting can see the balance
  - Everyone else sees 0.00
- SSN number column
  - Patients and Membership can see full SSN number
  - Everyone else sees ‘XXX XXX ‘ + last three digits of SSN

Create COLUMN MASK Syntax

» To create a mask for a column
  - CREATE the mask with visibility of column value determined by case expression
  - ENABLE or DISABLE the mask
    - If enabled this access rule will be implemented when column access control is ACTIVATED for the affected table
  - ALTER the table to activate column mask access control



```
CREATE MASK m_name on t_name FOR COLUMN c_name RETURN
case-expression {disable/enable}

ALTER TABLE/VIEW table/view ACTIVATE COLUMN ACCESS CONTROL;
```

Result of case expression is returned in substitute of column value

Determines if mask will be enabled when access control is ACTIVATEd for table

ACTIVATE column access control

Create Column Mask

» Three steps to create a mask for a column
  - Create the mask with visibility of column value determined by CASE expression
  - Enable or Disable the permission, determining if this access rule will be implemented when column access control is enabled for the affected table
  - Alter the table to Activate row access control

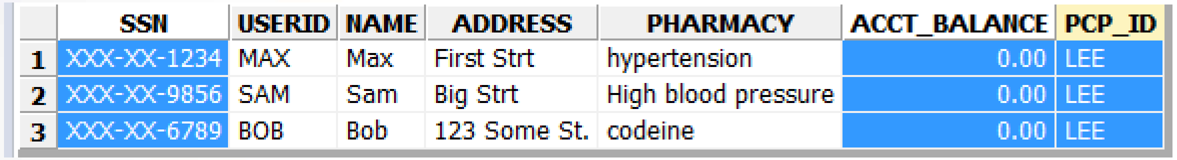
```
>>-CREATE--+------------+--MASK--mask-name--ON--table-name------>
           '-OR REPLACE-'

>--+-------------------------+--------------------------------->
   | .-AS-.                  |
   '-+----+--correlation-name-'

>--FOR COLUMN--column-name--RETURN--case-expression
```

Scenario: Create Column Mask

```
1.  CREATE MASK ADMINISTRATOR.ACCT_BALANCE_MASK ON ADMINISTRATOR.PATIENT FOR
COLUMN  ACCT_BALANCE RETURN
            CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1
                                    THEN ACCT_BALANCE
                        ELSE 0.00
            END
ENABLE@

2. CREATE MASK ADMINISTRATOR.SSN_MASK ON ADMINISTRATOR.PATIENT FOR
COLUMN SSN RETURN
            CASE WHEN
                        VERIFY_ROLE_FOR_USER(SESSION_USER,'PATIENT') = 1 OR
                        VERIFY_ROLE_FOR_USER(SESSION_USER,'MEMBERSHIP') = 1
                        THEN SSN
                        ELSE CHAR('XXX-XX-' || SUBSTR(SSN,8,4))
            END
ENABLE@
```

Scenario: Select from Table with Mask

- Query the table as Dr. Lee
  - Balance of 0.00 due to column mask

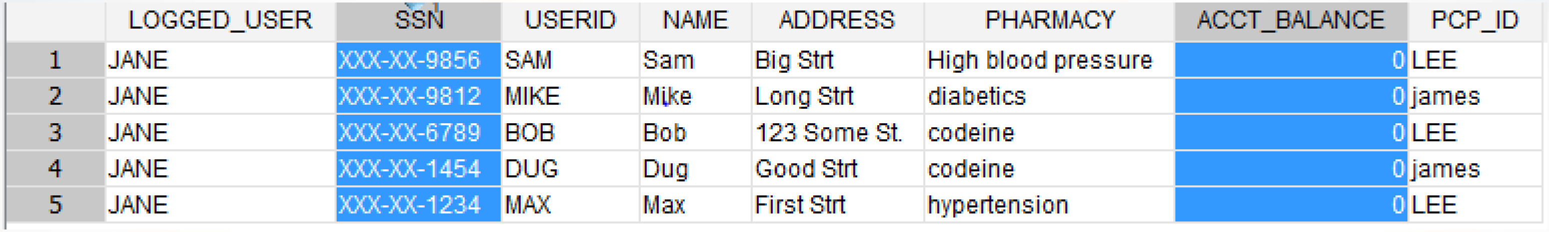


	SSN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
1	XXX-XX-1234	MAX	Max	First Strt	hypertension	0.00	LEE
2	XXX-XX-9856	SAM	Sam	Big Strt	High blood pressure	0.00	LEE
3	XXX-XX-6789	BOB	Bob	123 Some St.	codeine	0.00	LEE

- Column Access Control
  - Doctors cannot see account balances
  - Doctors cannot see SSN numbers
- Row Access control
  - Doctors can only see the rows of their own patients

Scenario: Select from Table with Mask

» A drug researcher, Jane queries the patient table:



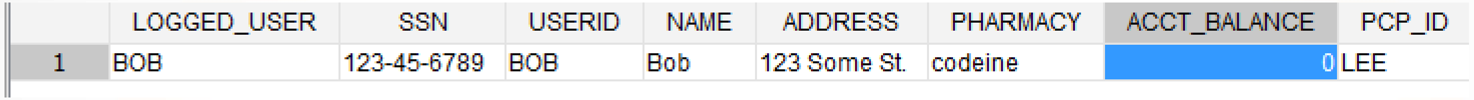
	LOGGED_USER	SSN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
1	JANE	XXX-XX-9856	SAM	Sam	Big Strt	High blood pressure	0	LEE
2	JANE	XXX-XX-9812	MIKE	Mike	Long Strt	diabetics	0	james
3	JANE	XXX-XX-6789	BOB	Bob	123 Some St.	codeine	0	LEE
4	JANE	XXX-XX-1454	DUG	Dug	Good Strt	codeine	0	james
5	JANE	XXX-XX-1234	MAX	Max	First Strt	hypertension	0	LEE

» Column Access Control
- Drug researchers cannot see account balances
- Drug researchers cannot see SSN numbers

» Row Access Control
- Drug researchers can see all rows

Scenario: Select from Table with Mask

- Patients can only see their own data

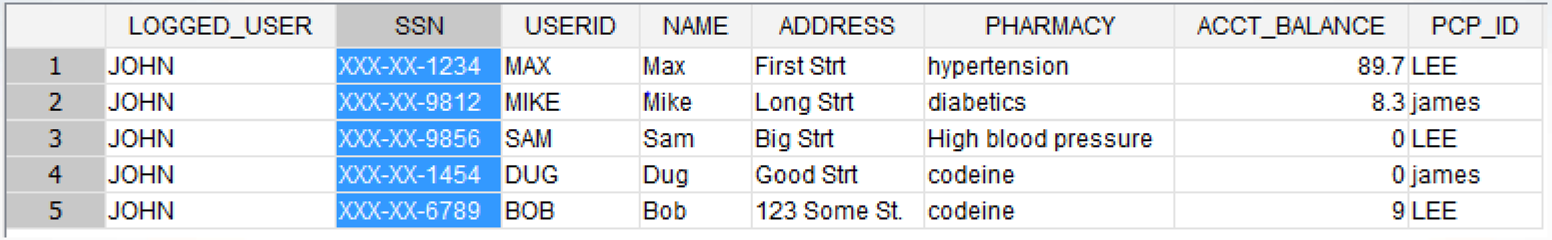


	LOGGED_USER	SSN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
1	BOB	123-45-6789	BOB	Bob	123 Some St.	codeine	0	LEE

- Column Access Control
  - Patients cannot see account balances
  - Patients can see SSN numbers
- Row Access Control
  - Patients can only see their own data

Scenario: Select from Table with Mask

» Accountants can see all rows



	LOGGED_USER	SSN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
1	JOHN	XXX-XX-1234	MAX	Max	First Strt	hypertension	89.7	LEE
2	JOHN	XXX-XX-9812	MIKE	Mike	Long Strt	diabetics	8.3	james
3	JOHN	XXX-XX-9856	SAM	Sam	Big Strt	High blood pressure	0	LEE
4	JOHN	XXX-XX-1454	DUG	Dug	Good Strt	codeine	0	james
5	JOHN	XXX-XX-6789	BOB	Bob	123 Some St.	codeine	9	LEE

»

- Accountants can see account balances
- Accountants cannot see SSN numbers

» Row Access Control

- Accountants can see all rows

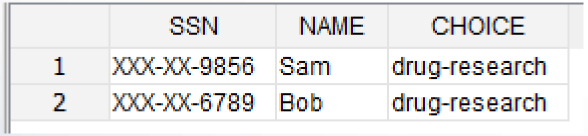
Using views with RCAC - Protected

- Views can be created on RCAC-protected tables
  - When querying the view, data is returned based on the **RCAC rules defined on the base table**

Using Views with RCAC-Protected Tables

```
CREATE VIEW ADMINISTRATOR.PATIENT_INFO_VIEW AS
SELECT P.SSN, P.NAME,C.CHOICE FROM ADMINISTRATOR.PATIENT P, ADMINISTRATOR.PATIENTCHOICE C
WHERE P.SSN = C.SSN AND
        C.CHOICE = 'drug-research' AND
        C.VALUE = 'opt-in'
```

As Dr. Lee, SELECT * FROM ADMINISTRATOR.PATIENT_INFO_VIEW;



	SSN	NAME	CHOICE
1	XXX-XX-9856	Sam	drug-research
2	XXX-XX-6789	Bob	drug-research

SQL Packages

- » Tables defined with RCAC have implications on any SQL package or cached dynamic SQL sections
- » RCAC incorporated by the SQL Compiler into SQL sections for query execution
- » These sections must remain in sync with RCAC at all times
- » To this end, changes against a table with RCAC defined will **INVALIDATE** an SQL package or cached dynamic SQL sections that depend on that table

SQL Statement Behavior

- SELECT statements against a table with RCAC activated will only return data with the RCAC applied to the results
  - The SQL optimizer uses the RCAC rules to create a filter to the table before any user specified operations such as predicates, grouping or ordering are processed
- When using INSERT, UPDATE and DELETE statements the rules specified for all RCAC permissions defined on the table determine whether the row can be INSERTED, UPDATED or DELETED
- To be INSERTED, UPDATED or DELETED the row must be conformant row
- You cannot INSERT, UPDATE or DELETE a row that you cannot SELECT
- The UPDATED row must conform to the enabled row permissions
- MERGE processing operates similarly to INSERT and UPDATE statements

Using UDFs and Triggers with RCAC-protected Tables

- User-defined functions (UDFs) could possibly cause data leakage of RCAC defined columns
- To prevent data leakage, UDFs must be defined as **SECURED** when referenced from within row and column access control definitions
- UDFs by default are evaluated last after any row permissions defined on that table are evaluated
- Since evaluating the UDF last can affect performance, the SECADM or a delegate can declare the function as SECURE if they trust the UDF

Using UDFs and Triggers with RCAC-protected Tables

- Triggers pose similar problems as UDFs where data leakage could occur to trigger transition variables and transition tables
- In order to create a Trigger that goes against a table with row or column access controls activated it must be defined by a SECADM as **SECURE**

Restrictions and Considerations

- You cannot create a mask on a column which
  - Is an XML object
  - Is a LOB column or a distinct type column that is based on a LOB
  - Is a column referenced in an expression that defines a generated column
- UDFs and TRIGGERs must be created or altered with the SECURE keyword
  - Compromise between security vs. integrity
- Automatic Data Movement
  - Row permissions are automatically activated on these target tables
    - MQT, History Tables for Temporal tables, and detached table partitions for range-partitioned tables
- **db2look** can extract row permission definitions in order to mimic elsewhere
- **EXPLAIN** facility shows access plans with RCAC in-place
  - Can override with NORCAC option

Performance Considerations

- » EXPLAIN facility will show optimized access path with RCAC included
  - NORCAC explain mode to hide RCAC explain information
- » Some implementations may benefit from RCAC others may suffer a performance impact
- » Each scenario has to be reviewed via EXPLAIN and Monitoring done to determine if there is an impact and what that impact is
  - MON_GET table function monitoring
  - Event Monitors (application trace)
- » Alternatively review impact of placing access control under application control and associated infrastructure to support it
  - REORG
  - BACKUP
  - Database Administration costs

Data Movement Considerations

» Automatic activation of row-level access control for a subject table happens in these situations:
  - Creation of an MQT that is based on one or more tables for which row-level or column-level access control is activated
  - Creation of a staging table for an MQT that is based on one or more tables for which row-level or column-level access control is activated
  - The activation of row-level or column-level access control on a base table that is used in the definition of an MQT
  - The creation of a history table for a temporal table for which row-level or column-level access control is activated
  - The activation of row-level or column-level access control on a temporal table for which a history table exists
  - The detaching of a partition from a partitioned table for which row-level or column-level access control is activated

Db2 Utility Considerations

- The db2look utility can be used to extract row permission and column mask definitions in order to mimic them in another database
- db2look can be used to extract the DDL for row permissions for subject tables and dependent objects
- Run REORG, REORGCHK and RUNSTAT utilities on tables with RCAC definitions just as you would for a non-RCAC table
- Row permissions and column masks are not applied for non-SQL utilities: LOAD, REORG, RUNSTATS
- Row permissions and column masks are applied for utilities that use SQL such as EXPORT

Summary

- SECADM is the sole manager for security policy
- Two sets of rules
  - Row access is restricted via permissions
  - Column access is restricted via masks
- Allows access to only subset of data necessary for job function
- Same output regardless of access method
  - Data Studio, views, applications, etc

Summary

- Data-centric and transparent to client application
- Ideal for commercial applications, function-specific access control, and use in compliance
- Use EXPLAIN for any performance implications!
- RCAC is another important tool in your arsenal to protect data!

References

» Data security best practices, IBM Developer Works, Walid Rjaibi, CISSP, IBM STSM. April 2012

» DB2 10.1, Row and Column Access Control, Georg Baklarz, PhD, IBM Toronto Lab, August 2013

» DB2 11.1 Row and Column Access Control, IBM Knowledge Center

» DB2 Database Security Guide, IBM Knowledge Center

Thank You!

Row and Column Access Control for Db2 11 LUW

Philip K. Gunning, CISSP
pgunning@gts1consulting.com