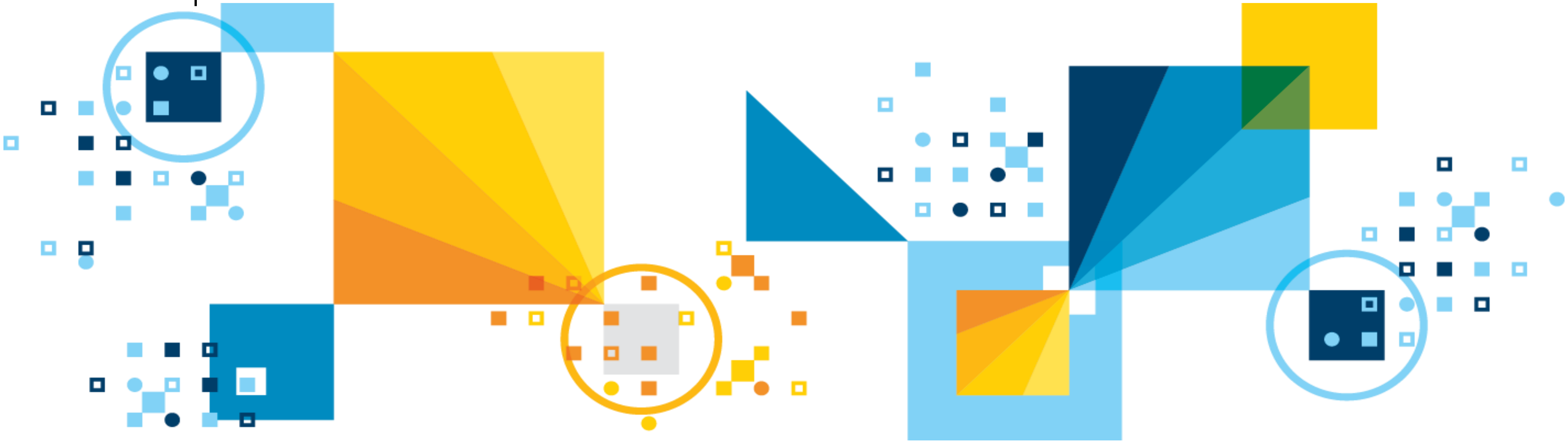
DB2 Security

Module ID	10111
Length	1 hour + 1.5 hours Hands on Lab



For questions about this presentation contact askdata@ca.ibm.com

January 30, 2015

Disclaimer

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

Module Information

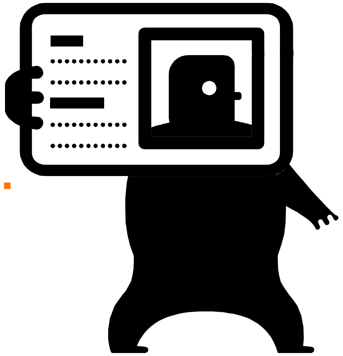
- You should have completed or acquired the necessary knowledge for the following modules in order to complete this module:
  - DB2 Fundamentals

- After completing this module, you should be able to:
  - Explain the concepts of:
    - Authentication
    - Authorization
    - Privileges
    - Roles
    - Trusted context
    - Audit
    - Data encryption
  - Be able to perform the following tasks:
    - Implement row permissions and column masks

Module Content

- Authentication
- Authorization
- Privileges
- Roles
- Trusted context
- Audit
- Data encryption
- Advanced security
  - Label-based Access Control (LBAC)
  - Row and Column Access Control (RCAC)
    - Working with Row Permissions and Column Masks
    - Using RCAC with Views, UDFs and Triggers
    - SQL and Built-in Scalar Functions for Defining RCAC rules

Authentication



- **Determines who the user is by identifying with a password**
  - **User authentication is completed by a security facility outside DB2.**
  - DB2 does not store nor maintain user/password information.
  - By default, the operating system's security mechanism is used.

- The following authentication types are available:
  - **CLIENT:**
    - Authentication performed on the client
  - **SERVER:**
    - Authentication occurs on the server
  - **SERVER_ENCRYPT:**
    - Same as SERVER but user/password are encrypted
  - **DATA_ENCRYPT:**
    - SERVER authentication and encryption of user data
  - **DATA_ENCRYPT_CMP:**
    - Same as DATA_ENCRYPT, but also allows use of SERVER_ENCRYPT for products that don't support DATA_ENCRYPT
  - **KERBEROS:**
    - Uses Kerberos security protocol
  - **KRB_SERVER_ENCRYPT:**
    - Server accepts KERBEROS or SERVER_ENCRYPT methods
  - **GSSPLUGIN:**
    - Uses a GSS-API plug-in to perform authentication
  - **GSS_SERVER_ENCRYPT**
    - Server accepts GSSPLUGIN or SERVER_ENCRYPT methods

Authorization

- **Determines what database operations users can perform and which data objects users can access**

- A user can acquire permissions in several ways
  - Granted directly to the **user ID**
  - Inherited from a **user group**
  - Inherited through **Roles** assigned to the user ID
  - Permissions acquired through a **trusted context**
  - Permissions assigned to **PUBLIC**.
    - *PUBLIC is a special group that consists of all users, including future users.*

- There are 3 types of permissions that can be granted
  - **Authority Levels**
  - **Privileges**
  - **LBAC Credentials** (*briefly covered in this presentation*)

Authorities

- An **Authority** is a group of privileges and permissions over database manager operations.
  - **Database-specific authorities**
  - **System (or instance level) authorities**

Level	Authorization	Description
System	SYSADM	System administrator
	SYSCTRL	Control over operations that affect system resources
	SYSMAINT	Ability to perform maintenance operations
	SYSMON	Ability to use database system monitor
Database	DBADM	Database administrator (only may be granted by SECADM)
	SECADM	Security administrator
	SQLADM	Authority to monitor and tune SQL within a database
	WLMADM	Ability to manage WLM assets
	EXPLAIN	Explain query plans without access to data
	ACCESSCTRL	Ability to issue limited grant and revoke statements
	DATAACCESS	Data access authority
	LOAD	Use of load utility

Privileges

- A **Privilege** is a single permission enabling the user to create/access a database resource.
  - Stored in the database catalog

Explicit

- User
- Group
- Role

Implicit

- When a database or database object is created

Indirect

- Inherited through execution of packaged code

- **Explicit**
  - GRANT

```
DB2 GRANT SELECT ON TABLE person TO USER employees
```

  - REVOKE

```
DB2 REVOKE SELECT ON TABLE person TO USER employees
```

- **Implicit**
  - Owner of the table is implicitly granted owner privileges

```
DB2 CREATE TABLE mytable
```

- **Indirect**
  - Execute permission on package that executes a select, indirectly granted select privileges

Roles

- Database object that groups together one or more privileges.
- Similar to User Groups but without the same restrictions
  - Hierarchical: you can assign roles to roles
  - Stored inside the database: changes to roles and its permissions are effective immediately to users
- Can be assigned to users, groups, PUBLIC or to other roles via a GRANT statement
- Step 1 – Create Role

```
CREATE ROLE developer
```

- Step 2 – Assign Privileges to a Role

```
GRANT SELECT ON TABLE server TO ROLE developer
```

- Step 3 – Grant Role to Users

```
GRANT ROLE developer TO USER Bob, USER Alice
```

- Step 4 – Revoke Role as Necessary

```
REVOKE ROLE developer FROM USER Bob
```

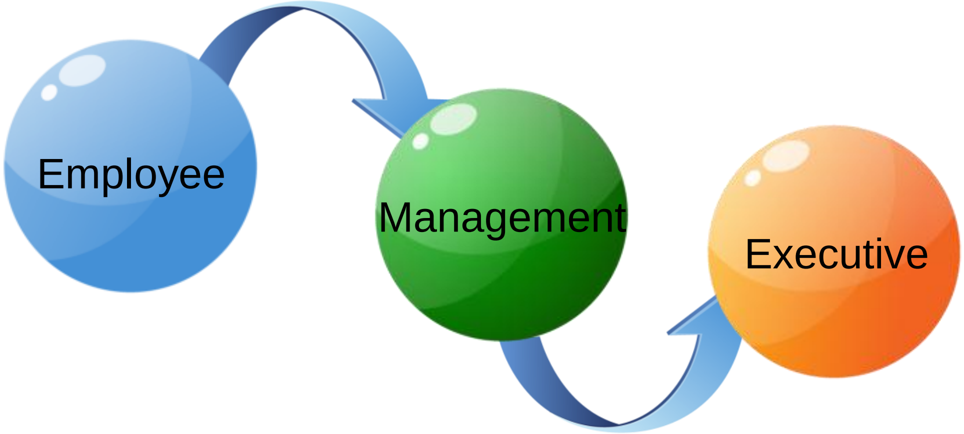
Roles – Admin Option and Hierarchies

- ADMIN OPTION
  - Allow for the user to grant or revoke the role when granting the role

```
GRANT ROLE developer TO USER Bob WITH ADMIN OPTION
```

- Hierarchies
  - A role may be granted membership into another role
  - Inherit privileges

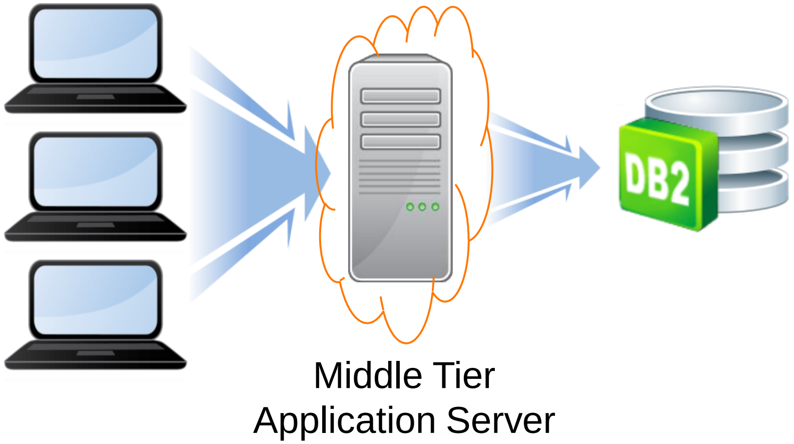
```
CREATE ROLE employee;
CREATE ROLE management;
CREATE ROLE executive;

GRANT ROLE employee TO ROLE management;
GRANT ROLE management TO ROLE executive;
```



Trusted Context

- A trusted relationship between the DB and the application that allows extra capabilities:
  - Switch current user ID
  - Acquire additional privileges via role inheritance
- Provide a means whereby the end-user identity in a three-tier environment can be easily and efficiently propagated to the database server
- Relationship identified by connection attributes
  - IP Address, Domain Name, Authorization ID, Data Encryption used

```
CREATE TRUSTED CONTEXT ctxt
BASED UPON CONNECTION USING SYSTEM AUTHID smith
ATTRIBUTES(address ‘192.168.2.27’) DEFAULT ROLE managerRole ENABLE
```



Creating and Applying Auditing

- Step 1: **Create an Audit Policy** to monitor and track activities. You can generate records for specific:
  - Activity Types (eg: All activities, Security Maintenance or all SQL Statements)
  - Activity Type Outcomes (Successes, Failures, Both, None)

```
CREATE AUDIT POLICY telleraction CATEGORIES execute
STATUS both ERROR TYPE audit
```

- Step 2: **Apply the Audit Policy** to any of the following objects or users
  - A whole database, tables, trusted contexts, authorization IDs (users, groups and roles), authorities
    - Example: Create an audit policy to monitor all SQL statements executed by the users in the teller role and all error messages

```
AUDIT ROLE teller USING POLICY telleraction
```

Collecting Audit Records

- Step 3: Allow regular database activities to continue, audit records will be created

- Step 4: Archive audit records
  - Use the provided SYSPROC.AUDIT_ARCHIVE stored procedure/table function to archive the active audit log

```
CALL SYSPROC.AUDIT_ARCHIVE(NULL, NULL)
```

- Step 5: Put Audit records into delimited file format
  - Use the provided SYSPROC.DELIM_EXTRACT stored procedure to extract data from the archive logs and load into a delimited file format

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT(NULL, '$HOME/AUDIT_DELIM_EXTRACT',
NULL, '%20070618%', 'CATEGORY EXECUTE')
```

Reading and Loading Audit Records

- Step 6: Create and load audit tables
  - Run the provided script db2audit.ddl to create the tables necessary to hold the audit data
  - The script creates 8 needed tables for audit files: AUDIT, CHECKING, OBJMAINT, SECMAINT, SYSADMIN, VALIDATE, CONTEXT and EXECUTE
  - Load data into the tables using the LOAD command

```
LOAD from execute.del of DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.EXECUTE
```

- Step 7: Begin querying the tables and creating reports!

```
SELECT category, event, appid, appname, userid, authid FROM
schema.EXECUTE
```

CATEGORY	EVENT	APPID	APPNAME	USERID	AUTHID
EXECUTE	STATEMENT	*LOCAL.prodrig.060410172044	myapp	smith	SMITH
EXECUTE	STATEMENT	*LOCAL.db2inst1.060410171009	db2bp	sam	SAM
EXECUTE	ROLLBACK	*LOCAL.db2inst1.060107111009	db2bp	sam	SAM

Data Encryption

- External Tools
  - **IBM Database Encryption Expert** to encrypt the underlying operating system data and backup files
  - **OS file-level encryption** (eg: AIX’s encrypted file system)
- Built-in function can encrypt data based on a provided password
  - ENCRYPT **(<data>, <pwd>, <pwd_hint>)**
    - Encrypts <*data*> based on password <*pwd*>
    - Result is a VARCHAR FOR BIT DATA value

```
INSERT INTO EMP(SSN)
VALUES ENCRYPT('289-46-8832','Pacific','Ocean')
```

- Built-in function to retrieve password hints
  - GETHINT **(<enc_data>)**
    - Returns the password hint if one was provided
  - ***Password management is responsibility of the user***

```
SELECT GETHINT(SSN) FROM EMP;
```

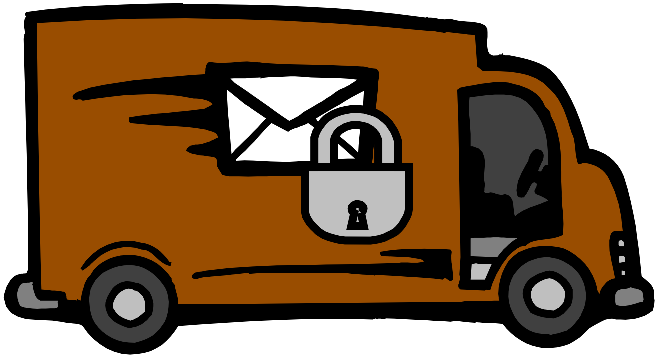
Data Decryption

- Built-in functions can decrypt data based on a provided password
  - DECRYPT_BIN **(<enc_data>, <pwd>)**
    - Decrypts <enc_data> and returns the value as VARCHAR FOR BIT DATA
  - DECRYPT_CHAR **(<enc_data>, <pwd>)**
    - Decrypts <enc_data> and returns the value as VARCHAR

```
SELECT DECRYPT_CHAR(SSN,'Pacific') FROM EMP
```

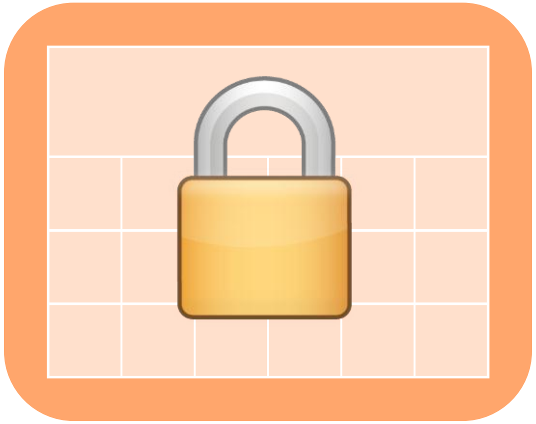
Encrypting Data in Transit

- **DATA_ENCRYPT authentication type**
  - During authentication, user and password are encrypted
  - The following data is encrypted:
    - **SQL** and **XQuery** statements
    - SQL program **variable data**
    - **Output data** from the server processing of an SQL or XQuery statement and including a description of the data
    - Some or all of the **answer set data** resulting from a query
    - Large object (**LOB**) data streaming
    - **SQLDA descriptors**
- **Secure Sockets Layer (SSL)**
  - CLI, CLP, and .Net Data Provider client applications and applications that use the IBM® Data Server Driver for JDBC and SQLJ (type 4 connections) support SSL
  - DB2 also supports SSL's successor, Transport Layer Security (TLS)



Fine Grained Access Control

- Regular SQL privileges can only protect tables as a whole.
- DB2 offers 2 solutions that complement the table privileges model:

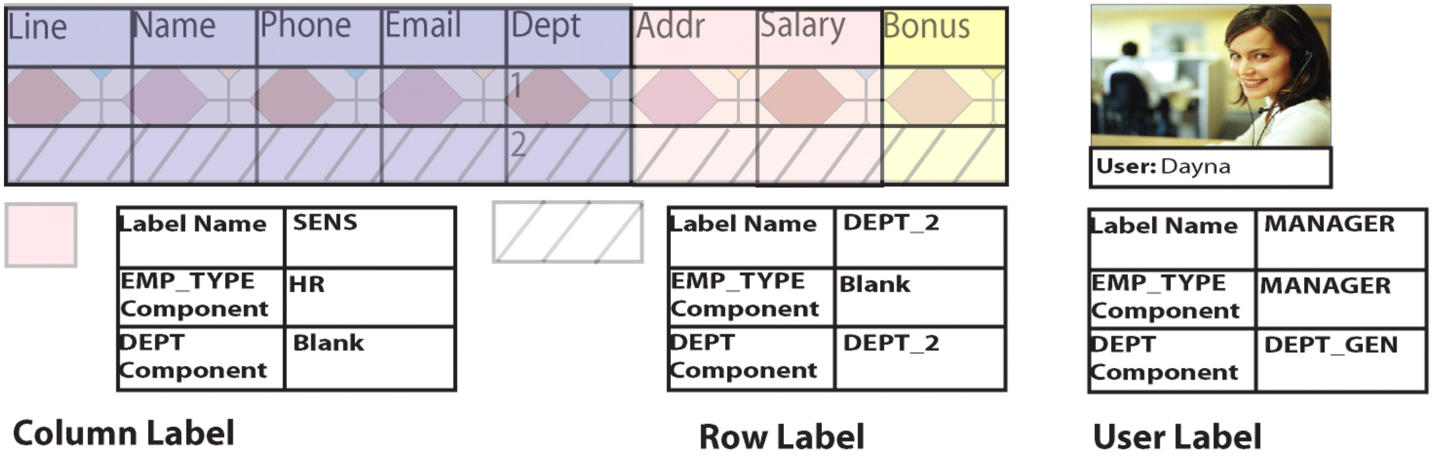


- **LBAC (Label-Based Access Control)** is a fixed label security model designed for environments with classified data
- Hierarchical access scenarios
  - Great for large companies with well defined data and user classifications
  - Suited for such applications as those intelligence and defense communities

- **RCAC (Row and Column Access Control)** is a general purpose security model
  - No data or user classification required
  - Best suited for commercial customers

Label-Based Access Control

- **Label Based Access Control**
  - Sets security labels at the row level, column level or both
- LBAC complements the traditional DB2 Discretionary Access Control (DAC)
- How LBAC works
  - Assign labels to database objects and users
  - DB2 compares the labels whenever an object is accessed
  - Access is granted based on the LBAC security policy defined



User: Dayna

Label Name	SENS
EMP_TYPE Component	HR
DEPT Component	Blank

Column Label

Label Name	DEPT_2
EMP_TYPE Component	Blank
DEPT Component	DEPT_2

Row Label

Label Name	MANAGER
EMP_TYPE Component	MANAGER
DEPT Component	DEPT_GEN

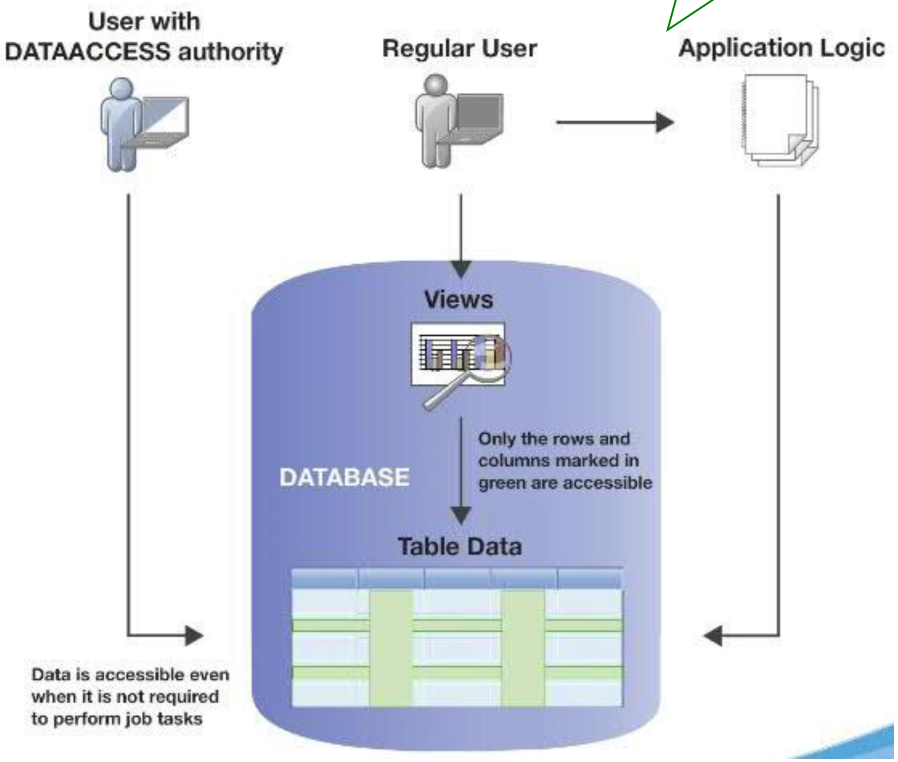
User Label

Why use Row and Column Access Control?

- Restricting portions of a table is not possible through regular SQL privileges
  - views or application logic are often used for this purpose
- Users with direct access to databases can bypass these layers
  - E.g.: Users with DATAACCESS authority can still view all data
- DB2 10 provides a **NEW** way to control data access at row / column level
  - Set up rich security policies
  - Prevents administrators with DATAACCESS authority from accessing all data in a database
  - No dependency on application logic
  - Facilitates table level multi-tenancy

How can we ensure that managers only see data for their own employees?

How can we ensure that a social security number column is masked out for unauthorized users?



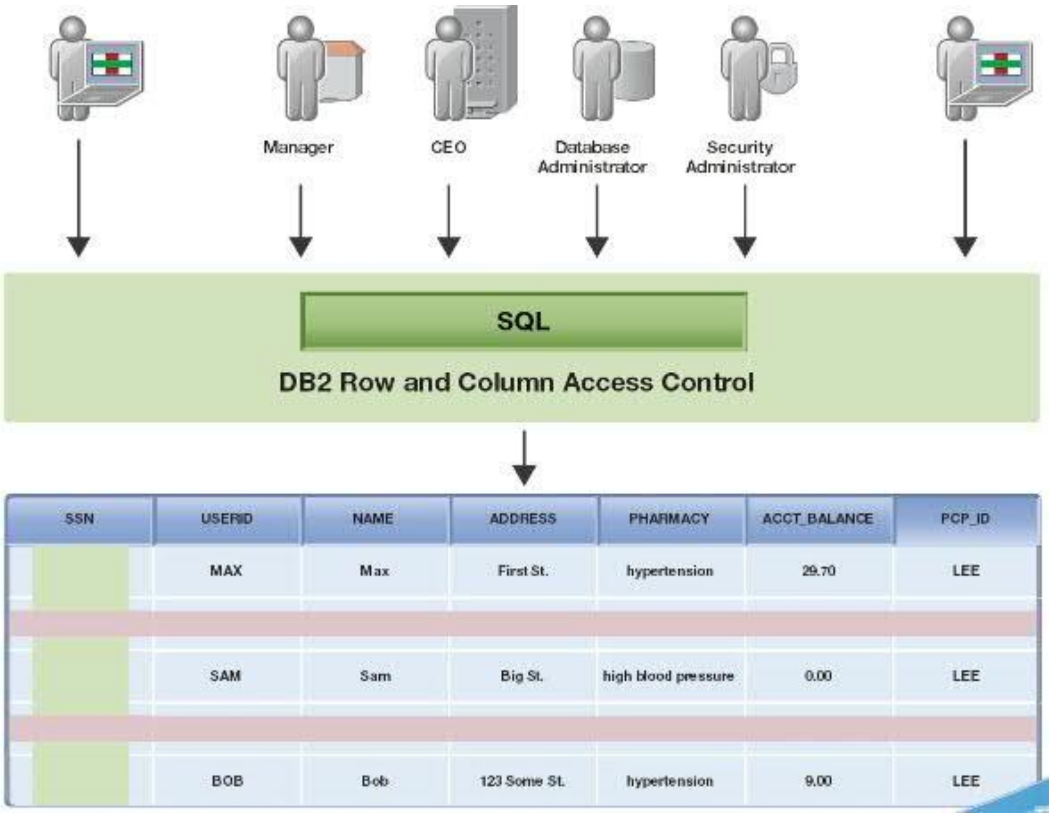
Only the rows and columns marked in green are accessible

Data is accessible even when it is not required to perform job tasks

What is Row and Column Access Control?

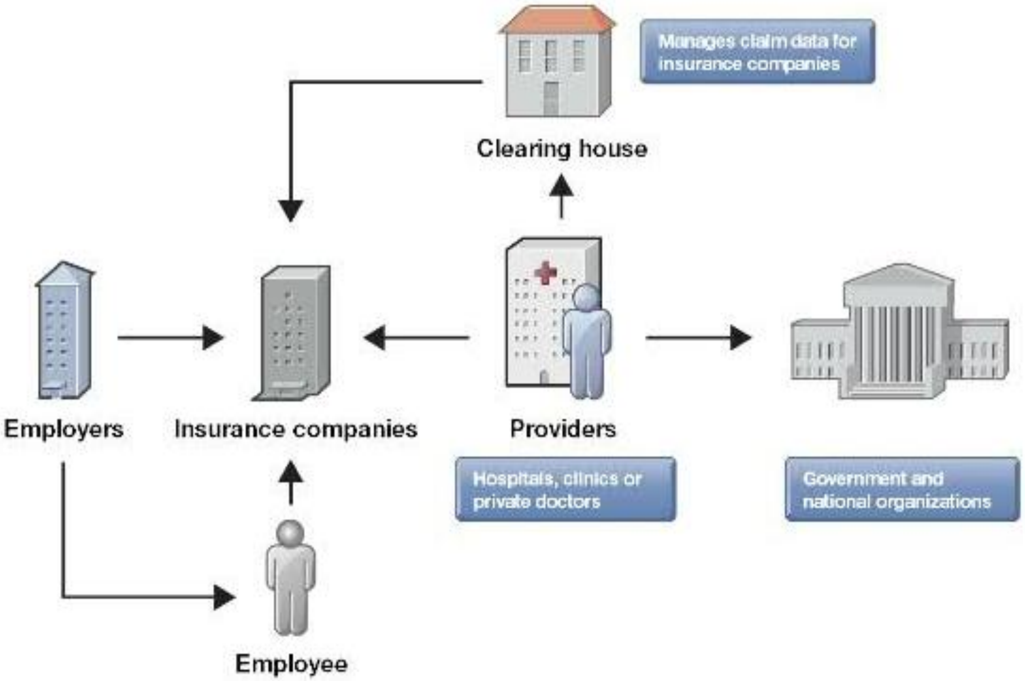
RCAC is:

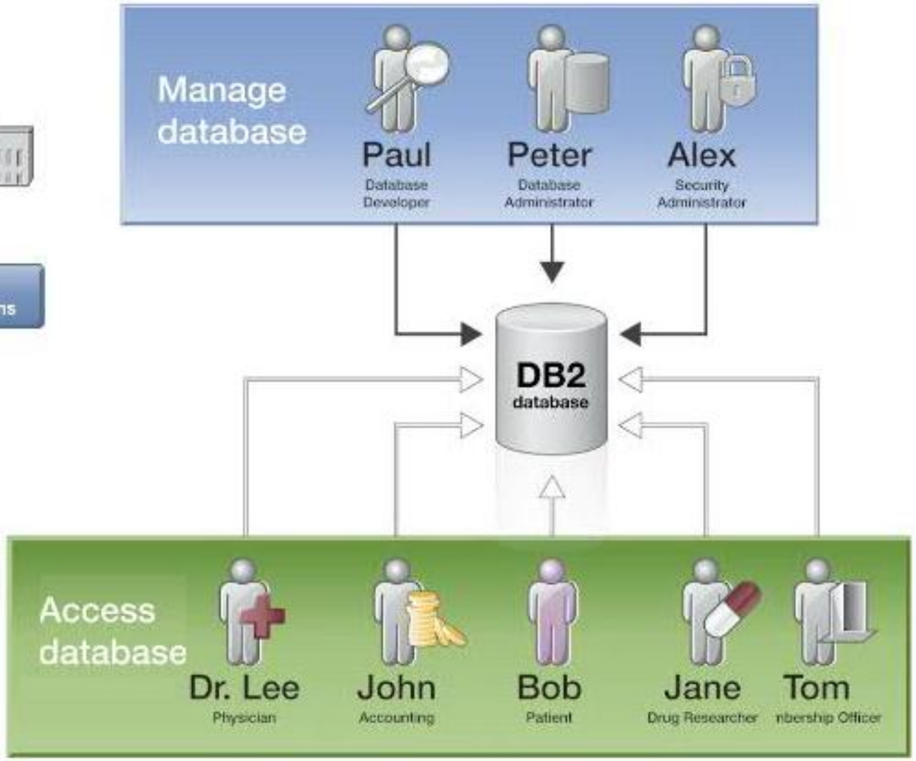
- Additional layer of data security introduced in **DB2 10**
- Complementary to table level authorization
- Controls access to a table at the row, column or both levels
  - Restrict access to a subset of the table's data
- Two sets of rules: permissions for rows and masks for columns

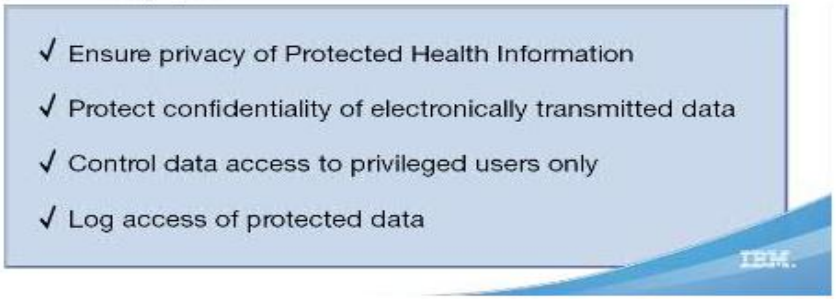


SSN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
	MAX	Max	First St.	hypertension	29.70	LEE
	SAM	Sam	Big St.	high blood pressure	0.00	LEE
	BOB	Bob	123 Some St.	hypertension	9.00	LEE

Sample Scenario: Health Care Insurance Industry


Manages claim data for insurance companies
Clearing house
Employers
Insurance companies
Providers
Hospitals, clinics or private doctors
Government and national organizations
Employee





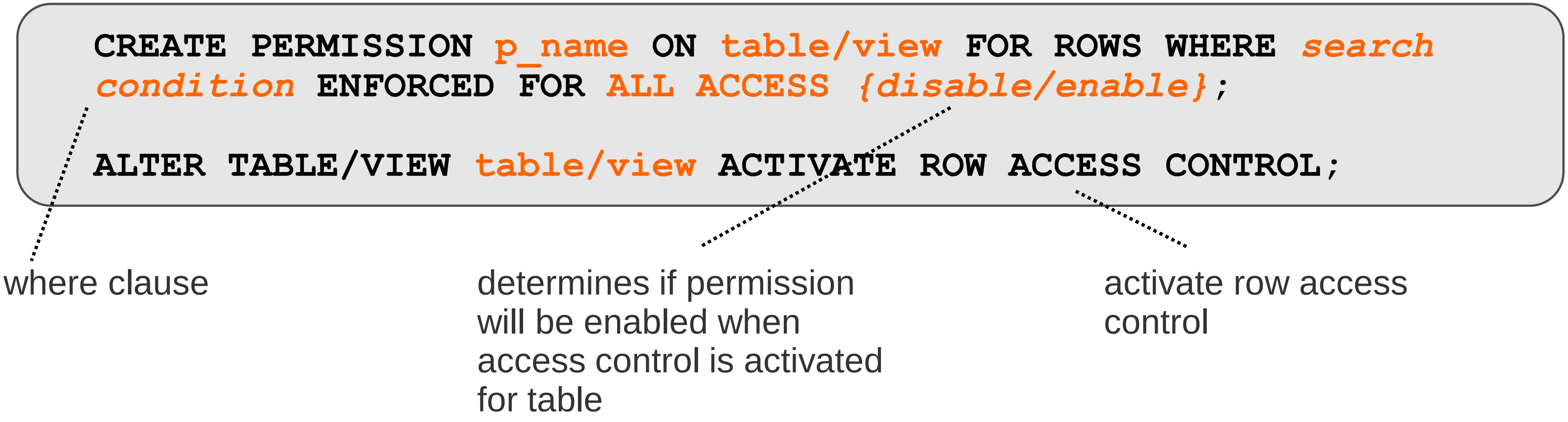
- ✓ Ensure privacy of Protected Health Information
- ✓ Protect confidentiality of electronically transmitted data
- ✓ Control data access to privileged users only
- ✓ Log access of protected data

Row Permission

- **A access control rule defined by a SQL statement that describes what set of rows a user has access to.**

To use RCAC row permissions:

- Create the permission with access rule defined by a search condition
- 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

determines if permission will be enabled when access control is activated for table

activate row access control

Sample Scenario: Create Row Permission

Our scenario has the following access control requirements:

1 **Patients**

- Can only access their own data

2 **Physicians**

- Can only access their own patients' data

3 **Membership officers**
Accounting
Drug researchers

- Can access all data

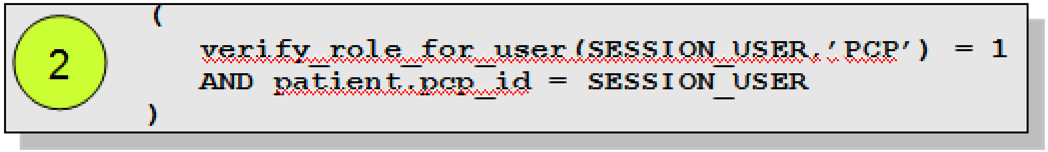
Nobody else sees any data

Sample Scenario: Create Row Permission (cont’d)

```
    CREATE PERMISSION row_access ON patient
    FOR ROWS WHERE
    (
1      verify_role_for_user(SESSION_USER,’PATIENT’) = 1
       AND patient.userid = SESSION_USER
    )
    OR
    (
2      verify_role_for_user(SESSION_USER,’PCP’) = 1
       AND patient.pcp_id = SESSION_USER
    )
    OR
    (
       verify_role_for_user(SESSION_USER,’MEMBERSHIP’) = 1 OR
3      verify_role_for_user(SESSION_USER,’ACCOUNTING’) = 1 OR
       verify_role_for_user(SESSION_USER,’DRUG_RESEARCH’) = 1
    )
    ENFORCED FOR ALL ACCESS
    ENABLE;


    ALTER TABLE patient ACTIVATE ROW ACCESS CONTROL;
```

Only users with SECADM authority can create RCAC rules

Sample Scenario: Update Table with Row Permissions Defined



2

```
(
    verify_role_for_user(SESSION_USER,’PCP’) = 1
    AND patient.pcp_id = SESSION_USER
)
```

Doctors can only see the data of their own patients

```
UPDATE patient SET pharmacy = ‘codeine’ WHERE name = ‘Sam’
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123 551 234	MAX	Max	First St.	hypertension	89.70	LEE
123 589 812	MIKE	Mike	Long St.	diabetics	8.30	JAMES
123 119 856	SAM	Sam	Big St.	codeine	12.50	LEE
123 191 454	DOUG	Doug	Good St.	influenza	7.68	JAMES
123 456 789	BOB	Bob	123 Some St.	hypertension	9.00	LEE

Successful update statement!

Sample Scenario: Update Table with Row Permissions Defined

2

```
(
    verify_role_for_user(SESSION_USER,'PCP') = 1
    AND patient.pcp_id = SESSION_USER
)
```

Doctors can only see the data of their own patients

```
UPDATE patient SET pharmacy = 'codeine' WHERE name = 'Doug'
```

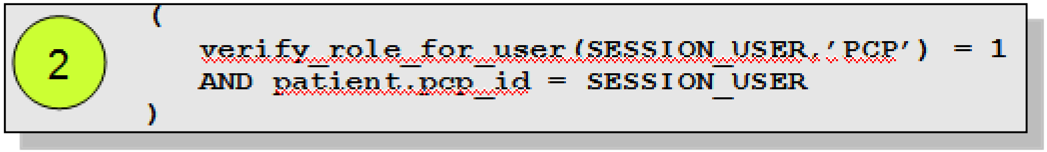
SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123 551 234	MAX	Max	First St.	hypertension	89.70	LEE
123 589 812	MIKE	Mike	Long St.	diabetics	8.30	JAMES
123 119 856	SAM	Sam	Big St.	codeine	12.50	LEE
123 191 454	DOUG	Doug	Good St.	influenza	7.68	JAMES
123 456 789	BOB	Bob	123 Some St.	hypertension	9.00	LEE

Unsuccessful update statement!

> ***No row was found for FETCH, UPDATE or DELETE; or the result of a query is an empty table.***

If you can't view a row, you can't update it either.

Sample Scenario: Select from Table with Row Permissions Defined



2

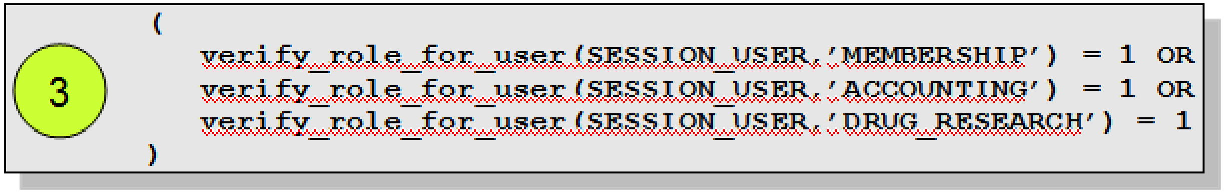
```
(
    verify_role_for_user(SESSION_USER,‘PCP’) = 1
    AND patient.pcp_id = SESSION_USER
)
```

Doctors can only see the data of their own patients

```
SELECT * FROM patient
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123 551 234	MAX	Max	First St.	hypertension	89.70	LEE
123 119 856	SAM	Sam	Big St.	codeine	12.50	LEE
123 456 789	BOB	Bob	123 Some St.	hypertension	9.00	LEE

Sample Scenario: Select from Table with Permission cont’d



3

```
(
   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
)
```

Accounting, drug researchers and membership officers can see all data

```
SELECT * FROM patient
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123 551 234	MAX	Max	First St.	hypertension	89.70	LEE
123 589 812	MIKE	Mike	Long St.	diabetics	8.30	JAMES
123 119 856	SAM	Sam	Big St.	codeine	12.50	LEE
123 191 454	DOUG	Doug	Good St.	influenza	7.68	JAMES
123 456 789	BOB	Bob	123 Some St.	hypertension	9.00	LEE

Sample Scenario: Select from Table with Permission cont’d

1

```
   (
      verify_role_for_user(SESSION_USER,’PATIENT’) = 1
      AND patient.userid = SESSION_USER
   )
```

Patients can only see their own data

```
SELECT * FROM patient
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123 456 789	BOB	Bob	123 Some St.	hypertension	9.00	LEE

Database Administrators can see no data.

```
SELECT * FROM patient
```

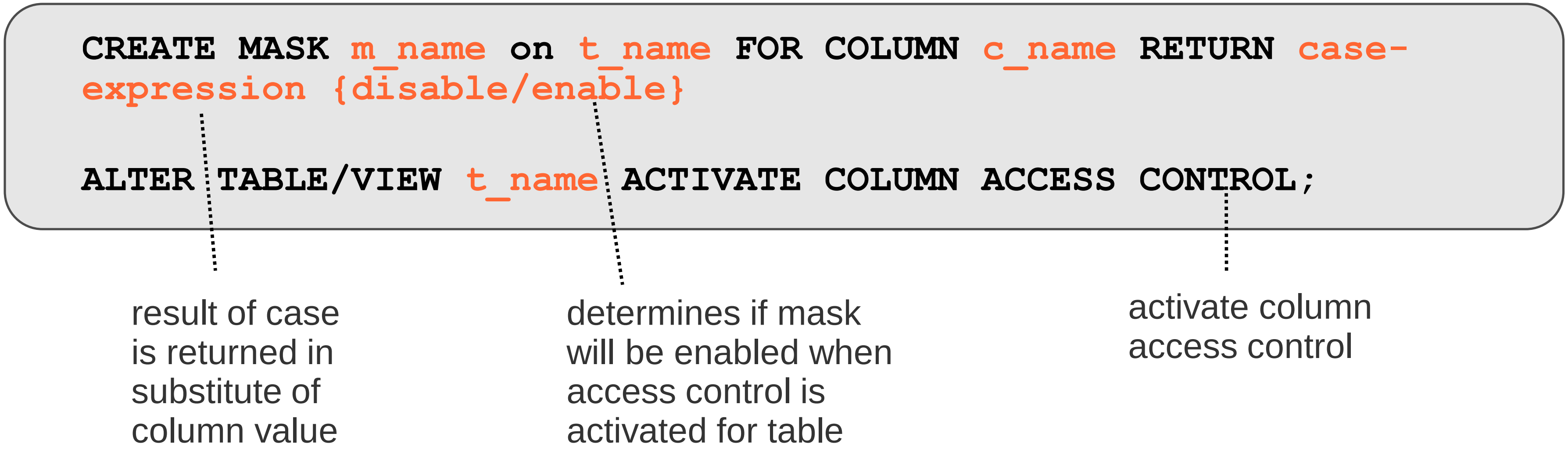
SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID

Column Mask

- **A access control rule defined by a SQL CASE expression that describes what column values a user is permitted to see and under what conditions.**

To protect a column with an RCAC column mask

- Define the column mask with using a SQL 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 table to activate column access control



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

ALTER TABLE/VIEW t_name ACTIVATE COLUMN ACCESS CONTROL;
```

result of case is returned in substitute of column value

determines if mask will be enabled when access control is activated for table

activate column access control

Sample Scenario: Create Column Mask

Scenario has the following permissions attached:

1

Account balance column

- Accounting can see the account balance
- Everyone else sees 0.00

2

SIN number column

- Patients can see full SIN number
- Everyone else sees ‘XXX XXX ‘ + last three digits of SIN

Sample Scenario: Create Column Mask   (cont’d)

1

```
CREATE MASK acct_balance_mask ON 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 sin_mask ON patient FOR
COLUMN sin RETURN
   CASE
      WHEN verify_role_for_user(SESSION_USER,
         ‘PATIENT’) = 1
         THEN sin
      ELSE
         ‘XXX XXX ‘ || SUBSTR(sin,8,3)
   END
ENABLE;

ALTER TABLE patient ACTIVATE COLUMN ACCESS CONTROL;
```

Sample Scenario: Select from Table with a Column Mask

Column Access Control → Accountants can see account balances.

Column Access Control → Accountants cannot see SIN numbers.

Row Access Control → Accountants can see all rows.

```
SELECT * FROM patient
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
XXX XXX 234	MAX	Max	First St.	hypertension	89.70	LEE
XXX XXX 812	MIKE	Mike	Long St.	diabetics	8.30	JAMES
XXX XXX 856	SAM	Sam	Big St.	codeine	12.50	LEE
XXX XXX 454	DOUG	Doug	Good St.	influenza	7.68	JAMES
XXX XXX 789	BOB	Bob	123 Some St.	hypertension	9.00	LEE

Sample Scenario: Select from Table with a Column Mask (cont’d)

Column Access Control → Drug researchers cannot see account balances.

Column Access Control → Drug researchers cannot see SIN numbers.

Row Access Control → Drug researchers can see all rows.

```
SELECT * FROM patient
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
XXX XXX 234	MAX	Max	First St.	hypertension	0.00	LEE
XXX XXX 812	MIKE	Mike	Long St.	diabetics	0.00	JAMES
XXX XXX 856	SAM	Sam	Big St.	codeine	0.00	LEE
XXX XXX 454	DOUG	Doug	Good St.	influenza	0.00	JAMES
XXX XXX 789	BOB	Bob	123 Some St.	hypertension	0.00	LEE

Sample Scenario: Select from Table with a Column Mask (cont’d)

Column Access Control → Doctors cannot see account balances.

Column Access Control → Doctors cannot see SIN numbers.

Row Access Control → Doctors can only see the rows of their own patients

```
SELECT * FROM patient
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
XXX XXX 234	MAX	Max	First St.	hypertension	0.00	LEE
XXX XXX 856	SAM	Sam	Big St.	codeine	0.00	LEE
XXX XXX 789	BOB	Bob	123 Some St.	hypertension	0.00	LEE

Sample Scenario: Select from Table with a Column Mask (cont'd)

Column Access Control → Patients cannot see account balances.

Column Access Control → Patients can see SIN numbers.

Row Access Control → Patients can only see their own data.

```
SELECT * FROM patient
```

SIN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123 456 789	BOB	Bob	123 Some St.	hypertension	0.00	LEE

Using Views with RCAC-protected Tables

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

```
CREATE VIEW patient_info_view AS
   SELECT p.sin, p.name, c.choice
   FROM patient p, patientchoice c
   WHERE p.sin = c.sin
         AND c.choice = ‘drug-research’
         AND c.value = ‘opt-in’;
```

```
  SELECT * FROM patient_info_view;
```

SIN	NAME	CHOICE
XXX XXX 856	Sam	drug-research
XXX XXX 789	Bob	drug-research

Using UDFs and Triggers with RCAC-protected Tables

- UDFs must be defined as SECURED when referenced from within row and column access control definitions

```
  ALTER FUNCTION ACCBALDISPLAY SECURED;
```

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

- UDFs invoked on columns protected with a column mask must be defined as SECURED. For instance, the SELECT statement below will fail unless CALC is defined as a secure UDF.

```
  SELECT CALC(ACC_BALANCE) FROM PATIENT;
```

- Triggers must be defined as SECURED if the subject table is protected with row or column access control.

```
  ALTER TRIGGER T_LOG_CHANGES SECURED;
```

SQL Statements for Managing RCAC rules

```
CREATE [OR REPLACE] PERMISSION p_name ON t_name FOR ROWS
WHERE search condition ENFORCED FOR ALL ACCESS
DISABLE/ENABLE
```

```
CREATE [OR REPLACE] MASK m_name ON t_name FOR COLUMN
c_name RETURN case expression DISABLE/ENABLE
```

```
ALTER MASK/PERMISSION c_name/p_name DISABLE/ENABLE
```

```
DROP MASK/PERMISSION c_name/p_name
```

```
ALTER TABLE t_name ACTIVATE/DEACTIVATE ROW/COLUMN ACCESS
CONTROL
```

New Built-in Scalar Functions

- **`verify_role_for_user`** ***(user, role1, role2, …)***
  - The result is 1 if any of the roles associated with the *user* are in list of *role1*, *role2*, etc.
  - Else 0.
- **`verify_group_for_user (user, group1, group2, …)`**
  - The result is 1 if any of the groups associated with the *user* are in list of *group1*, *group2*, etc.
  - Else 0.
- **`verify_trusted_context_role_for_user (user, role1, role2, …)`**
  - The result is 1 if when the role acquired through a trusted connection is in (or contains) any of the roles in the list of *role1*, *role2*, …
  - Else 0.

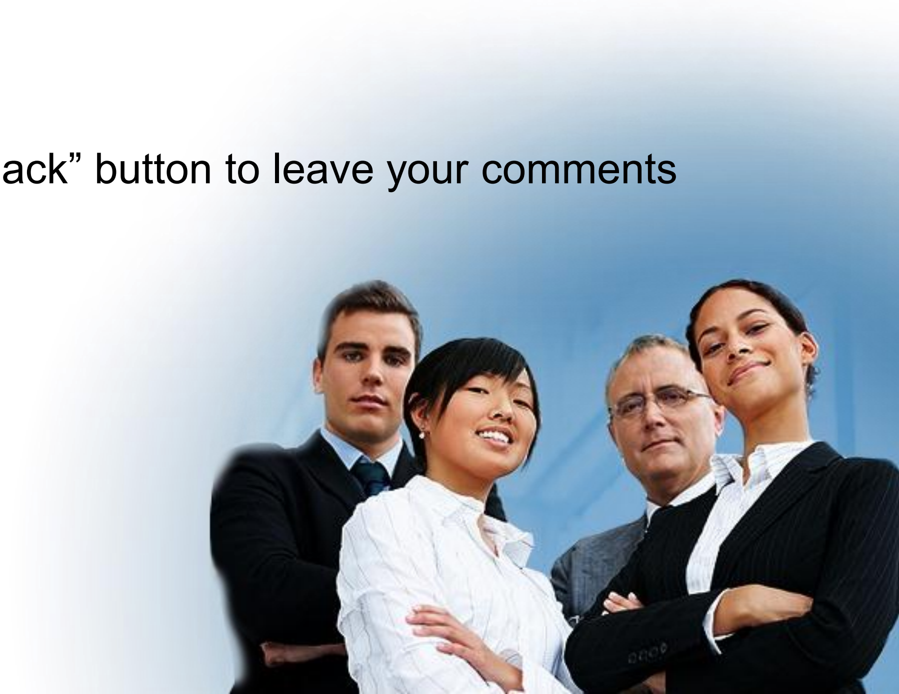
Summary

- Identifying the incoming connection is done by authentication, what the connection is able to do is classified by authorizations and privileges
- Roles allow for the grouping of many different privileges for a centralized point
- Trusted context allows for the identification of the incoming connection by defined attributes in a multi-tier environment
- Auditing allows for the monitoring of work
- Built-in functions allow for the encryption and decryption of data stored
- Advanced security mechanisms include LBAC and RCAC
    - **Label-Based Access Control** controls access by assignment of security labels to objects and users
    - **Row and Access Column Control** controls access by definition of row permissions and column masks

The next steps…

The Next Steps…

- Complete the Hands on Lab for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module
  - Download the workbook and the virtual machine image
  - Follow the instructions in the workbook to complete the lab
- Complete the online quiz for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module and select the quiz
- Provide feedback on the module
  - Log onto SKI, go to “My Learning” page
  - Find the module and select the “Leave Feedback” button to leave your comments



Questions?
askdata@ca.ibm.com