This tutorial introduces authentication, authorization, privileges, and roles as they relate to IBM® DB2® 10.1. It also introduces granular access control and trusted contexts. This is the second in a series of six tutorials designed to help you prepare for the DB2 10.1 Fundamentals certification exam (610). It is assumed that you already have basic knowledge of database concepts and operating system security.

View more content in this series

Before you start

About this series

If you want to become an IBM Certified Database Associate, you're in the right place. This series of six DB2 certification tutorials provides the basics on topics you’ll need to understand to take exam 610 (DB2 Fundamentals 10.1). Although its main purpose is to help you to take the test, this series can also be used to discover some of the new and exciting features of DB2 10.1 and help you learn about many of the features in DB2 10 for z/OS® and DB2 10.1 for Linux®, UNIX® and Windows®.

You can review the DB2 9 tutorials in the DB2 9 Fundamentals certification 730 prep series.

About this tutorial

This tutorial introduces the concepts of authentication, authorization, privileges, and roles as they relate to DB2 10.1. It also introduces granular access control and trusted contexts. It is the second in a series of seven tutorials designed to help you prepare for the DB2 10.1 Fundamentals certification exam (610).

Objectives

At the end of this tutorial, you will have knowledge of the following security objectives of the DB2 10.1 Fundamentals certification exam (610):

- Restricting data access
Different privileges and authorities
Identifying results (grant/revoke/connect statements) when given a DCL SQL statement
Row and Column Access Control (RCAC)
Roles and trusted contexts

Prerequisites
To get the most benefit from the tutorial, it's recommended that you have some background in database administration and usage. Having access to a DB2 server would also help you test many of the concepts and let you practice the tasks presented throughout the series. Product installation is not covered here, but you can find information in Related topics.

System requirements
As mentioned, it is strongly recommended that you install DB2 to take better advantage of this tutorial and the others in the series. DB2 has a FREE, fully functional edition that can be downloaded from DB2 Express-C. You can use it on your own computer without concerns about licensing, and it should meet your learning needs.

Acknowledgements
Although the material in this tutorial covers much of what is on the exam, it was created based on the exam objectives and not on specific questions. When taking the exam you may find questions that are not explicitly covered here. The best way to get prepared for the exam is to have experience with the topics listed in the exam objectives. This tutorial contains some content from its previous edition, "DB2 9 Fundamentals exam 730 prep, Part 2: Security" written by Graham G. Milne.

Security fundamentals
This section introduces authentication and authorization.

DB2 security overview
Security, especially data security, is of utmost importance today. DB2 database products provide security features you can use to protect your sensitive data. Access to the DB2 database system, which is managed by facilities that reside outside the DB2 database system, is called authentication. Access within the DB2 database system is managed by the database manager and is called authorization.

Authentication
Authentication is the process by which a system verifies a user's identity. DB2 uses a security plugin that communicates with the security facility to authenticate users. By default, DB2 integrates with the operating system (OS) security. It also ships with plug-ins for Kerberos and LDAP. You can even create your own custom security plugin to meet your business requirements.

The authentication process produces a DB2 authorization ID. Group membership information for the user is also acquired during authentication. Default acquisition of group information relies on an OS-based group membership plugin module that is included when you install the DB2 database
system. If you prefer, you can acquire group membership information by using a specific group membership plugin module such as LDAP.

**Authorization**

After users have been authenticated, authorization determines which resources or objects they can access and what type of access or operations they can perform. A user can get access from four sources:

- Primary permissions: Granted directly to the user.
- Secondary permissions: Inherited from membership of a group or a role.
- Public permissions: Granted to the `PUBLIC`.
- Context-sensitive permissions: Inherited from a role granted using a trusted context.

Users may acquire different levels of authorization in DB2 systems. The highest level is system-level access, followed by database-level authority. Inside the database comes next and this is called object-level authorities. Finally, for that specific object, do you have access to all data or just a subset of the data? All columns or just a few columns? This is called content-based authorities or fine-grained security. It can be implemented using either Label Based Access Control (LBAC) or Row and Column Access Control (RCAC). The following sections provide more detail.

**Authentication methods for DB2 server**

Access to an instance or a database first requires that the user be authenticated. The authentication type for each instance determines how and where a user will be verified.

The authentication type for a server is a database manager configuration parameter. You have one configuration per instance and it affects all the contained databases. After you request a connection for any database, the same authentication mechanism is used. Even if you are switching a user (that requires authentication) in a trusted context, the same mechanism applies. For users that don't require authentication, you may define your own security plugin that takes a user ID only and returns a valid DB2 authorization ID for that user.

**Authentication types**

Table 1 shows the available authentication types.

**Table 1. Authentication types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER</td>
<td>Authentication takes place on the server.</td>
</tr>
<tr>
<td>SERVER_ENCRYPT</td>
<td>Authentication takes place on the server. Passwords are encrypted at the client machine before being sent to the server. With the <code>alternate_auth_enc</code> instance parameter you can tell DB2 to use Advanced Encryption Standard AES 256-bit algorithm for stronger encryption.</td>
</tr>
<tr>
<td>CLIENT</td>
<td>Authentication takes place on the client machine, with no further checks on the server. In cases where the client's OS doesn't have appropriate native security features, they are marked untrusted.</td>
</tr>
</tbody>
</table>
instance parameters TRUST_ALLCLNTS and TRUST_CLNTAUTH are specifically to control what should happen in such cases.

<table>
<thead>
<tr>
<th>Kerberos</th>
<th>Authentication is performed by the Kerberos security software.</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRB_SERVER_ENCRYPT</td>
<td>Authentication is performed by Kerberos security software if the clientsetting is KERBEROS. Otherwise, SERVER_ENCRYPT is used.</td>
</tr>
<tr>
<td>DATA_ENCRYPT</td>
<td>Authentication works the same as that shown with SERVER_ENCRYPT. In addition, user data are encrypted.</td>
</tr>
<tr>
<td>DATA_ENCRYPT_CMP</td>
<td>Authentication is the same as for DATA_ENCRYPT except that this scheme allows older clients that don't support the DATA_ENCRYPT scheme to connect using SERVER_ENCRYPT authentication. The data in this case will not be encrypted. If the client connecting supports DATA_ENCRYPT, it is forced to encrypt the data and cannot downgrade to SERVER_ENCRYPT authentication. This authentication type is only valid in the server's database manager configuration file and is not valid when used on the CATALOG DATABASE command on a client or gateway instance.</td>
</tr>
<tr>
<td>GSSPLUGIN</td>
<td>Authentication is controlled by an external GSS-API plugin.</td>
</tr>
<tr>
<td>GSS_SERVER_ENCRYPT</td>
<td>Authentication is controlled by an external GSS-API plugin. If the client doesn't support one of the server's GSS-API plug-ins, SERVER_ENCRYPT authentication is used.</td>
</tr>
</tbody>
</table>

### Handling untrusted clients

In the CLIENT authentication type there could be trusted and untrusted clients. The instance parameters TRUST_ALLCLNTS and TRUST_CLNTAUTH are specifically to control what should happen in such cases. TRUST_ALLCLNTS can be YES to trust all the untrusted clients. However, when it's set to NO, untrusted clients are authenticated on the server and must provide a user ID and password.

DB2 provides more flexibility for strict security, even for trusted clients. TRUST_CLNTAUTH tells DB2 where to authenticate the user. It could be CLIENT or SERVER. In the case of SERVER, authentication takes place on the server if user and password are provided. If they are not provided, authentication takes place at the client machine. Table 2 lists all the combinations for these two parameters and shows where authentication would take place with each configuration.

### Table 2. Authentication modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH combinations

<table>
<thead>
<tr>
<th>trust_allclnts</th>
<th>trust_clntauth</th>
<th>Untrusted non-DRDA client authentication (no user ID &amp; password)</th>
<th>Untrusted non-DRDA client authentication (with user ID &amp; password)</th>
<th>Trusted non-DRDA client authentication (no user ID &amp; password)</th>
<th>Trusted non-DRDA client authentication (with user ID &amp; password)</th>
<th>DRDA client authentication (no user ID &amp; password)</th>
<th>DRDA client authentication (with user ID &amp; password)</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
</tr>
<tr>
<td>YES</td>
<td>SERVER</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>SERVER</td>
</tr>
<tr>
<td>NO</td>
<td>CLIENT</td>
<td>SERVER</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
<td>CLIENT</td>
</tr>
<tr>
<td>NO</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>CLIENT</td>
<td>CLIENT</td>
</tr>
<tr>
<td>DRDAONLY</td>
<td>CLIENT</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>CLIENT</td>
<td>CLIENT</td>
</tr>
<tr>
<td>DRDAONLY</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>SERVER</td>
<td>CLIENT</td>
<td>SERVER</td>
</tr>
</tbody>
</table>
DB2 authorities

DB2 authorities control the following aspects of a database security plan:

- Authority level that a user is granted.
- Commands that a user is allowed to run.
- Data that a user is allowed to read or alter.
- Database objects a user is allowed to create, alter, or drop.

Authorities are made up of groups of privileges and higher-level database manager (instance-level) maintenance and utility operations. Of the four authorities available in DB2, SYSADM, SYSCTRL, SYSSMAINT, and SYSMON are system-level authorities. System-level authorities allow you to perform a variety of instance-level operations such as creating databases, monitoring activities, managing tablespace, and so on. No instance-level authority provides access to data in database tables.

Instance-level authorities can be granted to users through instance (dbm) configuration parameters. You can list users having instance-level authorities as follows:

```
> db2 get dbm cfg | grep -i SYS
....
SYSADM group name                        (SYSADM_GROUP) = DB2IADM1
SYSCTRL group name                      (SYSCTRL_GROUP) =
SYSSMAINT group name                    (SYSSMAINT_GROUP) =
SYSMON group name                        (SYSMON_GROUP) =
....
```

The other type of authority is database-level. Each database authority allows users holding it to perform some particular type of action on the database as a whole. Database authorities are different from privileges, which allow a certain action to be taken on a particular database object, such as a table or an index. The DBADM, LOAD, and SECADM authorities are assigned to a user, group, or role for a particular database. This can be done explicitly using the `GRANT` command.

The following sections describe how each authority is assigned and which commands users with that authority are allowed to perform. Any reference to group membership implies that the user and group names have already been defined at the OS level.

Users can determine what authorities and database-level privileges they have by issuing the following command:

```
db2 get authorizations
```

Instance-level authorities

This section describes instance-level authorities, including examples of commands users can use with different authorities.

Obtaining SYSADM authority

SYSADM authority is the highest level of administrative authority at the instance level. Users with SYSADM authority can run some utilities and issue some database and database manager
commands within the instance. SYSADM authority assigned to the group specified by the sysadm_group configuration parameter. Membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSADM authority can perform the following functions:

- Upgrade a database.
- Restore a database.
- Change the database manager configuration file (including specifying the groups having SYSADM, SYSCTRL, SYSMANT, or SYSMON authority).
- Grant and revoke table space privileges and also use any table space.

Note: When a user with SYSADM authority creates a database, that user is automatically granted ACCESSCTRL, DATAACCESS, DBADM and SECADM authority on the database. If you want to prevent that user from accessing that database as a database administrator or a security administrator, you must explicitly revoke these database authorities from the user.

SYSADM authority is controlled in the DBM CFG file via the SYSADM_GROUP parameter. When the instance is created, this parameter is set to Administrator on Windows (although it appears blank if you issue the `db2 get dbm cfg` command). On UNIX, it is set to the primary group of the user who created the instance.

Because SYSADM users are the only users allowed to update the DBM CFG, they are also the only ones allowed to grant any of the SYS* authorities to other groups. The following example illustrates how to grant SYSADM authority to the group `db2grp1`:

```
db2 update dbm cfg using SYSADM_GROUP db2grp1
```

Remember, this change will not take effect until the instance is stopped and then restarted. And, if you are not currently logged in as a member of `db2grp1`, you may not have authority to restart the instance! You would have to log out and log back in with an ID in the correct group or add your current ID to `db2grp1`.

**Obtaining SYSCTRL authority**

Users with SYSCTRL authority can perform all administrative and maintenance commands within the instance. However, unlike SYSADM users, they cannot access any data within the databases unless they are granted the privileges required to do so. Examples of commands a SYSCTRL user can perform against any database in the instance are:

- `db2start/db2stop`
- `db2 create/drop database`
- `db2 create/drop tablespace`
- `db2 backup/restore/rollforward database`
- `db2 runstats` (against any table)
- `db2 update db cfg for database dbname`

A user with SYSADM authority can assign SYSCTRL to a group using the following command:
db2 update dbm cfg using SYSCTRL_GROUP group name

Obtaining SYSMAINT authority

The commands that a user with SYSMAINT authority can issue are a subset of those allowed to users with SYSCTRL authority. SYSMAINT users can only perform tasks related to maintenance, such as:

- `db2start/db2stop`
- `db2 backup/restore/rollforward database`
- `db2 runstats` (against any table)
- `db2 update db cfg for database dbname`

Notice that users with SYSMAINT cannot create or drop databases or tablespaces. They also cannot access any data within the databases unless they are granted the explicit privileges required to do so.

If you have SYSADM authority, you can assign SYSMAINT authority to a group using the following command:

`db2 update dbm cfg using SYSMAINT_GROUP group name`

Obtaining SYSMON authority

SYSMON authority provides the ability to take database system monitor snapshots of a database manager instance or its databases. SYSMON authority is assigned to the group specified by the `sysmon_group` configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

SYSMON authority enables the user to run the following commands:

- `GET DATABASE MANAGER MONITOR SWITCHES`
- `GET MONITOR SWITCHES`
- `GET SNAPSHOT`
- `LIST ACTIVE DATABASES`
- `LIST APPLICATIONS`
- `LIST DATABASE PARTITION GROUPS`
- `LIST DCS APPLICATIONS`
- `LIST PACKAGES`
- `LIST TABLES`
- `LIST TABLESPACE CONTAINERS`
- `LIST TABLESPACES`
- `LIST UTILITIES`
- `RESET MONITOR`
- `UPDATE MONITOR SWITCHES`

With SYSMON authority, users can use the following APIs:
• db2GetSnapshot - Get snapshot
• db2GetSnapshotSize - Estimate size required for db2GetSnapshot() output buffer
• db2MonitorSwitches - Get/update monitor switches
• db2ResetMonitor - Reset monitor
• db2mtrk - Memory tracker

SYSMON authority enables the user to use all snapshot SQL table functions without previously running SYSPROC.SNAP_WRITE_FILE. SYSPROC.SNAP_WRITE_FILE takes a snapshot and saves its content into a file. If any snapshot table functions are called with null input parameters, the file content is returned instead of a real-time system snapshot.

Users with the SYSADM, SYSCTRL, or SYSMAINT authority level also possess SYSMON authority.

A user with SYSADM authority can assign SYSMON to a group using the following command:

db2 update dbm cfg using SYSMON_GROUP group name

Database-level authorities

This section covers the essentials about obtaining the database-level authorities.

Obtaining DBADM authority

DBADM authority is a database-level and not an instance-level authority. DBADM users have complete control over a database -- almost. DBADM users cannot perform such maintenance or administrative tasks as:

• drop database
• drop/create tablespace
• backup/restore database
• update db cfg for database db name

However, DBADM users can perform the following tasks:

• db2 create/drop table
• db2 grant/revoke (any privilege)
• db2 runstats (any table)

DBADM users are also automatically granted all privileges to non-security database objects. DBADM should be granted the DATAACCESS authority by the security administrator to be able to access the data. Since DBADM authority is a database-level authority, it can be assigned to users, roles, and groups. The following commands show different ways that you can give DBADM authority:

• db2 create database test
  This command gives implicit DBADM authority on the database named test to the user who issued the command.
• `db2 connect to sample
db2 grant dbadm on database to user tst1`
This command can only be issued by SYSADM users; it issues DBADM authority to the user `tst1` on the sample database. Note that the issuing user must be connected to the sample database before granting DBADM authority.

• `db2 grant dbadm on database to group db2grp1`
This command grants DBADM authority to everyone in the group `db2grp1`. Again, only SYSADM users can issue this command.

### Obtaining LOAD authority

LOAD authority is also considered a database-level authority and can therefore be granted to users, roles, and groups. As the name implies, LOAD authority allows users to issue the `LOAD` command against a table. The `LOAD` command is typically used as a faster alternative to insert or import commands when populating a table with large amounts of data. Depending on the type of `LOAD` you wish to perform, having LOAD authority alone may not be sufficient. Specific privileges on the table may also be required.

The following commands can be run by users with LOAD authority:

- `db2 quiesce tablespaces for table`
- `db2 list tablespaces`
- `db2 runstats (any table)`
- `db2 load insert (must have insert privilege on table)`
- `db2 load restart/terminate after load insert (must have insert privilege on table)`
- `db2 load replace (must have insert and delete privilege on table)`
- `db2 load restart/terminate after load replace (must have insert and delete privilege on table)`

Only users with either SYSADM or DBADM authority are permitted to grant or revoke LOAD authority to users or groups. The following examples show how LOAD authority can allow a user to LOAD data into a table called `sales`. Assume that the command `db2 connect to sample` has already been issued.

• `db2 grant load on database to user tst1
db2 grant insert on table sales to user tst1`
With LOAD authority and insert privilege, `tst1` could issue a `LOAD INSERT` or a `LOAD RESTART`, or `TERMINATE` after a `LOAD INSERT` against the sales table.

• `db2 grant load on database to group grp1
db2 grant delete on table sales to group grp1
db2 grant insert on table sales to group grp1`
With LOAD authority as well as delete and insert privileges, any member of `grp1` could issue a `LOAD REPLACE`, a `LOAD RESTART`, or `TERMINATE` after a `LOAD REPLACE` against the sales table.

### Obtaining SECADM authority

SECADM authority is considered a database-level authority but it can only be granted to a specific user, role, or group by a SYSADM user. A user with SECADM can perform the following:
• Create and drop security label components.
• Create and drop security policies.
• Create and drop security labels.
• Grant and revoke security labels.
• Grant and revoke LBAC rule exemptions.
• Grant and revoke setsessionuser privileges.
• Grant and revoke database privileges and authorities.
• Execute the SQL statement `TRANSFER OWNERSHIP` on objects that you do not own.
• Execute the following audit routines:
  • SYSPROC.AUDIT_ARCHIVE used to archive audit logs
  • SYSPROC.AUDIT_LIST_LOGS used to locate audit files present in a specific directory.
  • SYSPROC.AUDIT_DELIM_EXTRACT used to extract audit data to delimited files format.

No other user can perform these functions, not even the SYSADM, unless SECADM was explicitly granted to that SYSADM user. This is important because these security abilities are very powerful and should only be granted to a user who is defined as a security administrator.

SQLADM (SQL administrator)

The SQLADM authority level provides administrative authority to monitor and tune SQL statements within a single database. It can be granted by a user with ACCESSCTRL or SECADM authority.

Obtaining WLMADM (workload management administrator)

The WLMADM authority provides administrative authority to manage workload management objects such as service classes, work action sets, work class sets, and workloads. It can be granted by a user with ACCESSCTRL or SECADM authority.

Obtaining EXPLAIN (explain authority)

The EXPLAIN authority level provides administrative authority to explain query plans without gaining access to data. It can only be granted by a user with ACCESSCTRL or SECADM authority.

Obtaining ACCESSCTRL (access control authority)

The ACCESSCTRL authority level provides administrative authority to issue the following `GRANT` (and `REVOKE`) statements.

• GRANT (Database Authorities) 
  `ACCESSCTRL` authority does not give the holder the ability to grant `ACCESSCTRL`, `DATAACCESS`, `DBADM`, or `SECADM` authority. Only a user who has `SECADM` authority can grant these authorities.
• GRANT (Global Variable Privileges)
• GRANT (Index Privileges)
• GRANT (Module Privileges)
• GRANT (Package Privileges)
• GRANT (Routine Privileges)
• GRANT (Schema Privileges)
• GRANT (Sequence Privileges)
• GRANT (Server Privileges)
• GRANT (Table, View, or Nickname Privileges)
• GRANT (Table Space Privileges)
• GRANT (Workload Privileges)
• GRANT (XSR Object Privileges)

ACCESSCTRL authority can only be granted by a user with SECADM authority. The ACCESSCTRL authority cannot be granted to PUBLIC.

Obtaining DATAACCESS (data access authority)

The DATAACCESS authority level provides the following privileges and authorities:

• LOAD authority
• SELECT, INSERT, UPDATE, DELETE privilege on tables, views, nicknames, and materialized query tables
• EXECUTE privilege on packages
• EXECUTE privilege on modules
• EXECUTE privilege on routines
  Except on the audit routines: AUDIT_ARCHIVE, AUDIT_LIST_LOGS, AUDIT_DELIM_EXTRACT.
• READ privilege on all global variables and WRITE privilege on all global variables except variables that are read-only
• USAGE privilege on all XSR objects
• USAGE privilege on all sequences

It can be granted only by a user who holds SECADM authority. The DATAACCESS authority cannot be granted to PUBLIC.

DB2 privileges

Privileges were briefly mentioned in the preceding section. This section explores database and object privileges. Privileges are generally in two main categories: database-level privileges, which span all objects within the database, and object-level privileges, which are associated with a specific object.

The database-level privileges that a user might be given are:

• CREATETAB: To create tables within the database.
• BINDADD: To create packages in the database using the BIND command.
• CONNECT: To connect to the database.
• CREATE_NOT_FENCED: To create unfenced user-defined functions (UDFs).
• IMPLICIT_SCHEMA: To implicitly create schemas within the database without using the CREATE SCHEMA command.
• LOAD: To load data into a table.
• QUIESCE_CONNECT: To access a database while it is in a quiesced state.
• CREATEEXTERNAL_ROUTINE: To create a procedure for use by applications and other users of the database.
Database objects include tables, views, indexes, schemas, and packages. Fortunately, most of the object-level privileges are self explanatory. Table 3 summarizes these privileges.

### Table 3. Summary of privileges

<table>
<thead>
<tr>
<th>Privilege name</th>
<th>Relevant objects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL</td>
<td>Table, View, Index, Package, Alias, Distinct Type, User Defined function, Sequence</td>
<td>Provides full authority on the object. Users with this privilege can also grant or revoke privileges on the object to other users.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Table, View</td>
<td>Allows users to delete records from the object.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Table, View</td>
<td>Allows users to insert records into the object via the INSERT or IMPORT commands.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Table, View</td>
<td>Provides the ability to view the contents of the object using the select statement.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Table, View</td>
<td>Allows users to modify records within the object using the update statement.</td>
</tr>
<tr>
<td>ALTER</td>
<td>Table</td>
<td>Allows users to alter the object definition using the alter statement.</td>
</tr>
<tr>
<td>INDEX</td>
<td>Table</td>
<td>Allows users to create indexes on the object using the create index statement.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Table</td>
<td>Provides the ability to create or drop foreign key constraints on the object.</td>
</tr>
<tr>
<td>BIND</td>
<td>Package</td>
<td>Allows users to rebind existing packages.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Package, Procedure, Function, Method</td>
<td>Allows users to execute packages and routines.</td>
</tr>
<tr>
<td>ALTERIN</td>
<td>Schema</td>
<td>Allows users to modify definitions of objects within the schema.</td>
</tr>
<tr>
<td>CREATEIN</td>
<td>Schema</td>
<td>Allows users to create objects within the schema.</td>
</tr>
<tr>
<td>DROPIN</td>
<td>Schema</td>
<td>Allows users to drop objects within the schema.</td>
</tr>
</tbody>
</table>

Information on object-level privileges is stored in the system catalog views. The view names are: syscat.tabauth, syscat.colauth, syscat.indexauth, syscat.schemaauth, syscat.routineauth, and syscat.packageauth.

### Explicit privileges

Privileges can be explicitly granted and revoked to users, roles, or groups using the GRANT and REVOKE commands. Let's walk through an example of how you can use these commands on various objects.

1. While logged in as a user with Administrator authority on Windows, bring up two DB2 command windows. Make sure that the `db2instance` variable is set to `db2` in both windows!
2. From Window 1, issue the following command:
   ```sql
   db2 connect to sample
   ```
3. From Window 2, issue the following command:
   ```sql
   db2 connect to sample user test1 using password
   ```
Remember, the commands in Window 1 are being issued by a user with SYSADM authority. The commands in Window 2 are being issued by `tst1`, a user with no specific authority or privileges on the sample database. Note that the schema name associated with the tables in your sample database will be the name of the user that issued the `db2sampl` command. In these examples, that user is `GMILNE`.

4. From Window 2, issue the following command:
   ```sql
db2 select * from gmilne.org
   ```
   You should see this response:
   ```
   SQL0551N  "TEST1" does not have the privilege to perform operation "SELECT" on object "GMILNE.ORG".
   ```

5. To correct the situation, issue the following command from Window 1:
   ```sql
db2 grant select on table gmilne.org to user test1
   ```
   Now, the earlier command will succeed.

6. Issue a more ambitious command from Window 2, as follows:
   ```sql
db2 insert into gmilne.org values (100, 'Tutorial', 1, 'Eastern', 'Toronto')
   ```
   Again, you'll see an error message:
   ```
   SQL0551N  "TEST1" does not have the privilege to perform operation "INSERT" on object "GMILNE.ORG"
   ```

7. Enter the following command from Window 1:
   ```sql
db2 grant insert on table gmilne.org to group db2grp1
   ```
   The earlier failed INSERT command should now complete successfully because `test1` is a member of group `db2grp1`.

8. Enter the following command in Window 2:
   ```sql
db2 drop table gmilne.emp_photo
   ```
   Again, you'll see an error message:
   ```
   SQL0551N  "TEST1" does not have the privilege to perform operation "DROP TABLE" on object "GMILNE.EMP_PHOTO"
   ```

9. You'll have to grant that privilege, so enter the following from Window 1:
   ```sql
db2 grant dropin on schema gmilne to all
   ```
   The `DROP TABLE` command should now complete successfully.

We're finished with the example, so let's revoke all the privileges you just granted by issuing the following commands from Window 1:

```sql
db2 revoke select on table gmilne.org from user test1
db2 revoke insert on table gmilne.org from group db2grp1
db2 revoke dropin on schema gmilne from all
```

Revoking privileges from a group does not necessarily revoke it from all members of that group. For example, the following command could have been used to revoke all privileges (except CONTROL) from `db2grp1` on the table `gmilne.org`:

```sql
db2 revoke all on table gmilne.org from group db2grp1
```
db2 revoke all on table gmilne.org from group db2grp1

However, the user test1 (who is a member of db2grp1) would have kept the select privileges on that table because they were granted that privilege directly.

**Implicit privileges**

As you've seen, DB2 may grant privileges automatically when certain commands are issued without the need for an explicit GRANT statement being issued. Table 4 summarizes some commands that result in privileges being implicitly granted by the database manager. These privileges are implicitly revoked when the object created is dropped. They are not, however, revoked when higher-level privileges are explicitly revoked.

**Table 4. Resulting commands from privileges being implicitly granted by the database manager**

<table>
<thead>
<tr>
<th>Command issued</th>
<th>Privilege granted</th>
<th>To whom privilege is granted</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE mytable</td>
<td>CONTROL on mytable</td>
<td>User issuing command</td>
</tr>
<tr>
<td>CREATE SCHEMA myschema</td>
<td>CREATEIN, ALTERIN, DROPIN on myschema, plus the ability to grant these to others</td>
<td>User issuing command</td>
</tr>
<tr>
<td>CREATE VIEW myview</td>
<td>CONTROL on myview only if CONTROL is held on all tables and views referenced in the definition of myview</td>
<td>User issuing command</td>
</tr>
<tr>
<td>CREATE DATABASE mydb</td>
<td>SELECT on mydb’s system catalog tables, IMPLICIT_SCHEMA on mydb</td>
<td>PUBLIC**</td>
</tr>
</tbody>
</table>

*When a user creates a database, that user is implicitly granted DBADM authority on that database. With DBADM authority comes implicit CONNECT, CREATETAB, BINDADD, IMPLICIT_SCHEMA, and CREATE_NOT_FENCED privileges. These privileges will remain with the user even if the DBADM authority is revoked.

PUBLIC is a special DB2 group that includes all users of a particular database. Unlike the other groups we've discussed thus far, PUBLIC does not have to be defined at the OS level. There are some privileges granted to PUBLIC by default. For example, this group receives CONNECT privilege on the database and SELECT privilege on the catalog tables automatically. GRANT and REVOKE commands can be issued against the PUBLIC group, like so:

db2 grant select on table sysibm.systables to public
db2 revoke select on table sysibm.systables from public

**Indirect privileges**

Privileges can be obtained indirectly when packages are executed by the database manager. A package contains one or more SQL statements that have been converted into a format that DB2 uses internally to execute them. In other words, a package contains multiple SQL statements in an executable format. If all the statements in the package are static, a user would only require EXECUTE privilege on the package to successfully execute the statements in the package.
For example, assume `db2package1` executes the following static SQL statements:

```
db2 select * from org
```
```
db2 insert into test values (1, 2, 3)
```

In this case, a user with EXECUTE privilege on `db2package1` would indirectly be granted SELECT privilege on the table `org` and INSERT privilege on the table `test`.

## Roles

A role is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement. It can also be assigned to a trusted context by using a CREATE TRUSTED CONTEXT or ALTER TRUSTED CONTEXT statement. A role can be specified for the SESSION_USER ROLE connection attribute in a workload definition.

Roles provide several advantages that make it easier to manage privileges in a database system:

- **Roles can be used to mirror the hierarchy of an organization.**
- **Roles could represent job responsibilities where users are granted the roles representing their job responsibilities. If the users' responsibilities change, their role membership can change to reflect the new responsibilities.**
- **Simplified administration.** In the above example, you don't need to grant the same set of privileges to many users (under the same responsibility). You only need to grant them the role that combines the set of privileges. In addition, if the job responsibilities change, the roles themselves can change (granting new permissions or revoking existing permissions) to affect all users granted that role.
- **Users and groups are managed by third-party (such as OS or LDAP), but roles are managed by the database that makes it always available when you create views, triggers, materialized query tables (MQTs), static SQL and SQL routines. Privileges and authorities granted to groups (directly or indirectly) are not used. Roles have an advantage over groups.**
- **All the roles assigned to a user are enabled when that user establishes a connection, so all privileges and authorities granted to roles are taken into account when a user connects. Roles cannot be explicitly enabled or disabled.**
- **The security administrator can delegate management of a role to others. The security administrator can use the WITH ADMIN OPTION clause of the GRANT statement to delegate management of the role to another user so that user can control the role membership.**

**Note:** A role cannot own database objects.

All DB2 database privileges and authorities within a database can be granted to a role. For example, a role can be granted any of the following authorities and privileges:

- `DBADM`, `SECADM`, `DATAACCESS`, `ACCESSCTRL`, `SQLADM`, `WLMADM`, `LOAD`, and `IMPLICIT_SCHEMA` database authorities.
- `CONNECT`, `CREATETAB`, `CREATE_NOT_FENCED`, `BINDADD`, `CREATE_EXTERNAL_ROUTINE`, or `QUIESCE_CONNECT` database authorities.
• Any database object privilege (including CONTROL).

As previously mentioned, all the roles assigned to a user are enabled when that user establishes a connection. When combined with trusted contexts, a user can be assigned different roles based on their connection attributes. The following are examples of how to work with roles.

Creating roles

```sql
CREATE ROLE MARKETING;
CREATE ROLE ACCOUNTANT;
CREATE ROLE SALES;
```

Granting privileges to roles

```sql
--Grant privileges
GRANT SELECT, UPDATE ON TABLE CUSTOMER TO ROLE ACCOUNTANT;
GRANT SELECT, INSERT, UPDATE ON TABLE CUSTOMER TO ROLE SALES;
GRANT SELECT ON TABLE CUSTOMER TO ROLE MARKETING;
```

Granting roles to users

```sql
GRANT ROLE SALES TO USER USER1, USER USER2;
GRANT ROLE MARKETING TO USER USER3;
GRANT ROLE ACCOUNTANT TO USER USER4;
```

Revoking roles from users

```sql
REVOKE ROLE SALES FROM USER USER2;
```

Row and Column Access Control

DB2 Version 10.1 introduces Row and Column Access Control (RCAC) as an additional layer of data security. RCAC is sometimes referred to as fine-grained access control. RCAC controls access to a table at the row level, column level, or both. RCAC can be used to complement the table privileges model.

You can use RCAC to ensure that your users have access to only the data that is required for their work. For example, within a hospital system: doctors can only access data for their patients, only users from the accounting department can access employee salary information, and credit card information can be masked except for the card owner.

Regular SQL privileges cannot restrict access to portions of a table. This was usually done through views or application logic. However, users with direct access to the database can bypass these layers.

With RCAC, even higher level authorities such as users with DATAACCESS authority are not exempt from RCAC rules. Only users with security administrator (SECADM) authority can manage row and column access controls within a database. Therefore, you can use RCAC to prevent users with DATAACCESS authority from freely accessing all data in a database.
RCAC rules

RCAC is comprised of SQL rules that place access control at the table level around the data itself. RCAC permits all users to access the same table, as opposed to alternative views of a table. RCAC does, however, restrict access to the table based upon individual user permissions or roles as specified by a policy associated with the table.

There are two sets of RCAC rules:

Row permissions
Row permission is a database object that expresses a row access control rule for a specific table. A row access control rule is an SQL search condition that describes what set of rows a user has access to.

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

RCAC SQL syntax

Some examples of RCAC SQL syntax are shown below.

Creating row permission

```
CREATE PERMISSION <permission name> ON <table name>
FOR ROWS WHERE
.......

ENFORCED FOR ALL ACCESS
ENABLE;
```

Creating column mask

```
CREATE MASK <column mask name> ON <table name> FOR
COLUMN <column name> RETURN
CASE
  WHEN <condition>
    THEN <return value>
  ELSE <return value>
  ....

END
ENABLE;
```

Activating row access control on a table

```
ALTER TABLE <table name> ACTIVATE ROW ACCESS CONTROL;
```

Activating column access control on a table

```
ALTER TABLE <table name> ACTIVATE COLUMN ACCESS CONTROL;
```

RCAC 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 the list of role1, role2, and so on. Else 0.

`verify_group_for_user (user, group1, group2, ...)`

The result is 1 if any of the groups associated with the user are in the list of group1, group2, and so forth. 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, and so on. Else 0.

**Scenario: Implementing RCAC to STOREDB database**

The following scenario involves the table CUSTOMER in the bookstore database STOREDB. The CUSTOMER table stores individual customer information. Table 5 shows the columns within the CUSTOMER table.

**Table 5. CUSTOMER table columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Customer ID number</td>
</tr>
<tr>
<td>NAME</td>
<td>Customer name</td>
</tr>
<tr>
<td>CARD_NUMBER</td>
<td>Customer’s credit card number</td>
</tr>
<tr>
<td>SALES_REP</td>
<td>Sales Representative (Rep) responsible for the customer</td>
</tr>
<tr>
<td>CREDIT</td>
<td>Holds the credit balance of a customer</td>
</tr>
<tr>
<td>MARKETING</td>
<td>Indicates whether the customer opts to receive marketing bulletins</td>
</tr>
</tbody>
</table>

The bookstore has security challenges for the data in table CUSTOMER, as outlined in Table 6.

**Table 6. CUSTOMER table security challenges**

<table>
<thead>
<tr>
<th>Security challenge</th>
<th>RCAC feature that addresses the security challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limiting column access to only privileged users.</td>
<td>Column masks can be used to filter or hide sensitive data. Limiting row access to only privileged users.</td>
</tr>
<tr>
<td>• Only accountants can see customer credit.</td>
<td></td>
</tr>
<tr>
<td>• Only accountants can see credit card number details.</td>
<td></td>
</tr>
<tr>
<td>• Other users will have credit card information masked.</td>
<td></td>
</tr>
<tr>
<td>Limiting row access to only privileged users.</td>
<td>Row permissions can be implemented to control which row can be seen by Sales Reps and the Marketing department.</td>
</tr>
<tr>
<td>• Sales Reps can only see a list of their customers.</td>
<td></td>
</tr>
<tr>
<td>• Marketing can only see a list of customers that opt to receive marketing bulletins.</td>
<td></td>
</tr>
<tr>
<td>Restricting data on a need-to-know basis.</td>
<td>Row permissions can restrict table level data at the user level.</td>
</tr>
</tbody>
</table>

Table 7 shows the roles defined in the STOREDB database and their desired access rights to the CUSTOMER table.
Table 7. STOREDB database defined ROLES

<table>
<thead>
<tr>
<th>Role name</th>
<th>Role description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARKETING</td>
<td>For marketing operators. Can only read information about customers opting to receive marketing bulletins. Users having this role cannot read the credit balance and credit card information.</td>
</tr>
<tr>
<td>ACCOUNTING</td>
<td>For accountants. Full access.</td>
</tr>
<tr>
<td>SALES</td>
<td>For Sales Reps. Can read information about their own customers. Users having this role cannot read the credit balance or credit card information.</td>
</tr>
</tbody>
</table>

The users of the STOREDB database and their roles are shown in Table 8.

Table 8. STOREDB users

<table>
<thead>
<tr>
<th>Name</th>
<th>Duties</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECADM1</td>
<td>Chief Security Administrator, holds the SECADM authority.</td>
</tr>
<tr>
<td>USER1</td>
<td>Sales representative having the SALES role.</td>
</tr>
<tr>
<td>USER2</td>
<td>Sales representative having the SALES role.</td>
</tr>
<tr>
<td>USER3</td>
<td>Marketing officer having the MARKETING role.</td>
</tr>
<tr>
<td>USER4</td>
<td>Accountant having the ACCOUNTING role.</td>
</tr>
</tbody>
</table>

Implementing RCAC rules

Use the following steps to implement RCAC rules.

1. Creating the STOREDB database and CUSTOMER table:

   ```sql
   db2 CREATE DATABASE STOREDB USING CODESET UTF-8 TERRITORY US;
   ##Connect to database
   db2 CONNECT TO STOREDB;
   ##Create CUSTOMER table
   ##CUSTOMER table stores information regarding STOREDB.CUSTOMER
   db2 "CREATE TABLE STOREDB.CUSTOMER (ID VARCHAR(11), NAME VARCHAR(10), CARD_NUMBER VARCHAR(19), SALES_REP VARCHAR(10), CREDIT FLOAT, MARKETING INTEGER )";
   ```

2. Populating the STOREDB.CUSTOMER with arbitrary data:

   ```sql
   ##Insert CUSTOMER data
   INSERT INTO CUSTOMER VALUES('123-11-1111','Customer1','1234-5678-9012-3456','USER1',100,1)
   INSERT INTO CUSTOMER VALUES('123-22-2222','Customer2','3459-5344-6452-1234','USER2',200,0)
   INSERT INTO CUSTOMER VALUES('123-33-3333','Customer3','4335-5633-5673-9634','USER1',300,0)
   INSERT INTO CUSTOMER VALUES('123-44-4444','Customer4','7647-9246-2345-3452','USER2',400,1)
   ```

3. Creating the roles and granting them the adequate privileges:
--Creating roles and granting privileges
CREATE ROLE MARKETING;
CREATE ROLE ACCOUNTANT;
CREATE ROLE SALES;
GRANT ROLE SALES TO USER USER1;
GRANT ROLE SALES TO USER USER2;
GRANT ROLE MARKETING TO USER USER3;
GRANT ROLE ACCOUNTANT TO USER USER4;

--Grant privileges
GRANT SELECT, UPDATE ON TABLE STOREDB.CUSTOMER TO ROLE ACCOUNTANT;
GRANT SELECT, INSERT, UPDATE ON TABLE STOREDB.CUSTOMER TO ROLE SALES;
GRANT SELECT ON TABLE STOREDB.CUSTOMER TO ROLE MARKETING;
GRANT CONNECT ON DATABASE TO USER secadm1;

4. Using the SECADM1 account, create the following row permissions:
   a. ROW_ACCESS_SALES to restrict sales representative access to only their own customers. This is done by verifying the session user role to be SALES and the name to match the STOREDB.CUSTOMER.SALES_REP column.
      CREATE PERMISSION ROW_ACCESS_SALES ON STOREDB.CUSTOMER FOR ROWS WHERE
      (VERIFY_ROLE_FOR_USER(SESSION_USER, 'SALES') = 1 AND
      STOREDB.CUSTOMER.SALES_REP = SESSION_USER)
      ENFORCED FOR ALL ACCESS ENABLE;
   b. ROW_ACCESS_ACCOUNTANT to give the ACCOUNTANT role full access to the STOREDB.CUSTOMER data.
      CREATE PERMISSION ROW_ACCESS_ACCOUNTANT ON STOREDB.CUSTOMER FOR ROWS WHERE
      (VERIFY_ROLE_FOR_USER(SESSION_USER, 'ACCOUNTANT') = 1)
      ENFORCED FOR ALL ACCESS ENABLE;
   c. ROW_ACCESS_MARKETING to restrict access to only rows with STOREDB.CUSTOMER.MARKETING column value equal to 1 (customers opted to receive marketing bulletins).
      CREATE PERMISSION ROW_ACCESS_MARKETING ON STOREDB.CUSTOMER FOR ROWS WHERE
      (VERIFY_ROLE_FOR_USER(SESSION_USER, 'MARKETING') = 1 AND
      STOREDB.CUSTOMER.MARKETING = 1)
      ENFORCED FOR ALL ACCESS ENABLE;

5. Using the SECADM1 account, create the following column masks:
   a. CREDIT_MASK to allow only users having the ACCOUNTANT role to view content of the CREDIT column. Other users will get the value 0.0 for this column.
      CREATE MASK STOREDB.CREDIT_MASK ON STOREDB.CUSTOMER FOR COLUMN CREDIT RETURN
      CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER, 'ACCOUNTANT') = 1 THEN CREDIT
      ELSE 0.00 END
      ENABLE;
   b. CARD_NUMBER_MASK to allow only users with the ACCOUNTANT role to view the CARD_NUMBER column contents. Other users will get the first 12 left digits masked by "'XXXX-XXXX-XXXX-".
CREATE MASK STOREDB.CARD_NUMBER_MASK ON STOREDB.CUSTOMER FOR COLUMN CARD_NUMBER RETURN CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER, 'ACCOUNTANT') = 1 THEN CARD_NUMBER ELSE CHAR('XXXX-XXXX-XXXX-' || SUBSTR(CARD_NUMBER, 16, 4)) END ENABLE;

6. Activate RCAC on the table STOREDB.CUSTOMER:
ALTER TABLE STOREDB.CUSTOMER ACTIVATE COLUMN ACCESS CONTROL;
ALTER TABLE STOREDB.CUSTOMER ACTIVATE ROW ACCESS CONTROL;

**Querying the CUSTOMER table with different users**

Use the following steps to query the CUSTOMER table with different users.

1. Querying the table using USER1 (SALES role):
   > db2 connect to storedb user user1 using password
   
   Database Connection Information
   Database server         = DB2/LINUXX8664 10.5.0
   SQL authorization ID   = USER1
   Local database alias   = STOREDB
   > db2 "select * from storedb.customer"

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>CARD_NUMBER</th>
<th>SALES_REP</th>
<th>CREDIT</th>
<th>MARKETING</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-11-1111 Customer1</td>
<td>XXXX-XXXX-XXXX-3456</td>
<td>USER1</td>
<td>+0.00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>123-33-3333 Customer3</td>
<td>XXXX-XXXX-XXXX-9634</td>
<td>USER1</td>
<td>+0.00</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
   
   2 record(s) selected.

   • Only Customer1 and Customer3 data were retrieved in satisfaction of the row permission ROW_ACCESS_SALES (customers whose sales rep is USER1).
   • CREDIT column values is masked in satisfaction of column mask CREDIT_MASK (only users with ACCOUNTANT role can see the CREDIT column content).
   • The first left 12 digits of the CARD_NUMBER column is masked with “XXXX-XXXX-XXXX” in satisfaction of column mask CARD_NUMBER_MASK (only users with ACCOUNTANT role can see the CARD_NUMBER column content).

2. Querying the table using USER2 (SALES role):
   > db2 connect to storedb user user2 using password
   
   Database Connection Information
   Database server         = DB2/LINUXX8664 10.5.0
   SQL authorization ID   = USER2
   Local database alias   = STOREDB
   > db2 "select * from storedb.customer"

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>CARD_NUMBER</th>
<th>SALES_REP</th>
<th>CREDIT</th>
<th>MARKETING</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-2222 Customer2</td>
<td>XXXX-XXXX-XXXX-1234</td>
<td>USER2</td>
<td>+0.00</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>123-44-4444 Customer4</td>
<td>XXXX-XXXX-XXXX-3452</td>
<td>USER2</td>
<td>+0.00</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
   
   2 record(s) selected.
• Only Customer2 and Customer4 data were retrieved in satisfaction of the row permission ROW_ACCESS_SALES (customers whose sales rep is USER2).
• CREDIT column values is masked in satisfaction of column mask CREDIT_MASK (only users with ACCOUNTANT role can see the CREDIT column content).
• The first left 12 digits of the CARD_NUMBER column is masked with “XXXX-XXXX-XXXX” in satisfaction of column mask CARD_NUMBER_MASK (only users with ACCOUNTANT role can see the CARD_NUMBER column content).

3. Querying the table using USER3 (MARKETING role):

> db2 connect to storedb user user3 using password

Database Connection Information
Database server = DB2/LINUXX8664 10.5.0
SQL authorization ID = USER3
Local database alias = STOREDB

> db2 "select * from storedb.customer"

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>CARD_NUMBER</th>
<th>SALES_REP</th>
<th>CREDIT</th>
<th>MARKETING</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-11-1111</td>
<td>Customer1</td>
<td>XXXX-XXXX-XXXX-3456</td>
<td>USER1</td>
<td>+0.00</td>
<td>1</td>
</tr>
<tr>
<td>123-44-4444</td>
<td>Customer4</td>
<td>XXXX-XXXX-XXXX-3452</td>
<td>USER2</td>
<td>+0.00</td>
<td>1</td>
</tr>
</tbody>
</table>

2 record(s) selected.

• Only Customer1 and Customer4 data were retrieved in satisfaction of the row permission ROW_ACCESS_MARKETING (customers opt to receive marketing bulletins with MARKETING column has a value 1).
• CREDIT column values is masked in satisfaction of column mask CREDIT_MASK (only users with ACCOUNTANT role can see the CREDIT column content).
• The first left 12 digits of the CARD_NUMBER column is masked with “XXXX-XXXX-XXXX” in satisfaction of column mask CARD_NUMBER_MASK (only users with ACCOUNTANT role can see the CARD_NUMBER column content).

4. Querying the table using USER4 (ACCOUNTANT role):

> db2 connect to storedb user user4 using password

Database Connection Information
Database server = DB2/LINUXX8664 10.5.0
SQL authorization ID = USER4
Local database alias = STOREDB

> db2 "select * from storedb.customer"

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>CARD_NUMBER</th>
<th>SALES_REP</th>
<th>CREDIT</th>
<th>MARKETING</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-11-1111</td>
<td>Customer1</td>
<td>1234-5678-9012-3456</td>
<td>USER1</td>
<td>+100.0</td>
<td>1</td>
</tr>
<tr>
<td>123-22-2222</td>
<td>Customer2</td>
<td>3459-5344-6452-1234</td>
<td>USER2</td>
<td>+200.0</td>
<td>0</td>
</tr>
<tr>
<td>123-33-3333</td>
<td>Customer3</td>
<td>4335-5633-5673-9634</td>
<td>USER1</td>
<td>+300.0</td>
<td>0</td>
</tr>
<tr>
<td>123-44-4444</td>
<td>Customer4</td>
<td>7647-9246-2345-3452</td>
<td>USER2</td>
<td>+400.0</td>
<td>1</td>
</tr>
</tbody>
</table>

• USER4 can see all rows in satisfaction of row permission ROW_ACCESS_ACCOUNTANT.
• USER4 can see the CREDIT column values in satisfaction of column mask CREDIT_MASK (only users with ACCOUNTANT role can see the CREDIT column content).
• USER4 can see the CARD_NUMBER column values in satisfaction of column mask CARD_NUMBER_MASK (only users with ACCOUNTANT role can see the CARD_NUMBER column content).

Trusted contexts

Why do we need trusted contexts? Usually, when you have a database server you have applications that connect to that database server. Like many applications, your application likely has a lot of users but it accesses the database using only one user ID. That one user ID could have all superuser capabilities or at least must have all the capabilities required by all users and features of your applications. In this situation, you would have several security concerns as summarized below.

Loss of user identity
Some enterprises prefer to know the identity of the actual user accessing the database for access control purposes.

Diminished user accountability
Accountability through auditing is a basic principle in database security. Not knowing the user's identity makes it difficult to distinguish which transactions are performed by whom.

Over-granting privileges to that user
The authorization ID must have all the privileges necessary to execute all the requests from all the users. This presents the security issue of enabling users who do not need access to certain information to obtain access anyway.

Weakened security
The current approach requires that the authorization ID used by the middle tier to connect must be granted privileges on all resources that might be accessed by user requests. If that authorization ID is ever compromised, then all those resources will be exposed.

"Spill over" between users of the same connection
Changes by a previous user can affect the current user. Clearly, there's a need for a mechanism whereby the actual user's identity and database privileges are used for database requests performed by the middle tier on behalf of that user. You might consider establishing a new connection for each user using the user's ID and password. Although this seems simple, it can be inapplicable, hard to maintain, and probably slow. You may not have the authorization credentials all the time to establish a new connection. Even if you do, it's clear that connections are not pooled (which adds performance overhead). Even if you don't have centralized security setup, there's clearly maintenance overhead to maintain users in two locations.

Is there a way to overcome all of these concerns and issues? Yes—with trusted contexts.

What is a trusted context?
A trusted context is a database object that defines a trust relationship for a connection between the database and an external entity such as an application server. The trust relationship is based upon the following set of attributes:
System authorization ID
   Represents the user that establishes a database connection.

IP address (or domain name)
   Represents the host from which a database connection is established.

Data stream encryption
   Represents the encryption setting (if any) for the data communication between the database server and the database client.

When a user establishes a database connection, the DB2 database system checks whether the connection matches the definition of a trusted context object in the database. When a match occurs, the database connection is said to be trusted.

A trusted connection allows the initiator of this trusted connection to acquire additional capabilities that may not be available outside the scope of the trusted connection. The additional capabilities vary depending on whether the trusted connection is explicit or implicit. The initiator of an explicit trusted connection has the ability to:

- Switch the current user ID on the connection to a different user ID with or without authentication.
- Acquire additional privileges via the role inheritance feature of trusted contexts.

Trusted contexts and trusted connections

The security administrator is responsible for creating a trusted context object that matches certain connection attributes. After the trusted context object is created and enabled, application developers can request trusted connection to the database. That connection may be an implicit or explicit trusted connection.

Implicit trusted connections

An implicit trusted connection is the result of a normal connection request (that matches a definition in DB2) rather than an explicit trusted connection request. No application code changes are needed to obtain an implicit connection. The initiator of an implicit trusted connection can only acquire additional privileges via the role inheritance feature of trusted contexts; they cannot switch the user ID. Let's look at an example of an implicit trusted connection.

Assume you have a user db2inst3 that has very limited privileges. However, you want this user to acquire all the privileges associated with the role managerRole once he initiates a connection from the IP address 192.168.222.130. The sample code below shows how to create a trusted context to handle the situation.

```sql
CREATE TRUSTED CONTEXT CTX_1
   BASED UPON CONNECTION USING SYSTEM AUTHID db2inst3
   ATTRIBUTES (ADDRESS '192.168.222.130')
   DEFAULT ROLE managerRole
   ENABLE
```

If db2inst3 requests a connection from 192.168.222.130, the request is honored with a trusted connection because the connection attributes are satisfied by the trusted context CTX_1. Now that user db2inst3 has established a trusted connection, he or she can now acquire all the privileges...
and authorities associated with the trusted context role managerRole. These privileges and authorities may not be available to user user2 outside the scope of this trusted connection. If that same user (db2inst3) requested a connection from another server, they will get a non-trusted connection.

You might wonder, "What if that other server is defined in another trusted context?" That's a legitimate question but this can never happen. You can have only one trusted connection per authorization ID (db2inst3 in our example).

If user db2inst2, who doesn't have any associated trusted context definitions, requests a trusted connection from the same server (for example, 192.168.222.130), he or she will get a warning that a trusted connection could not be established and instead a non-trusted connection is returned. The connection request is not refused but the connection returned is a non-trusted connection.

Explicit trusted connections

An explicit trusted connection is the result of requesting a trusted connection explicitly. You might wonder, "How do I request it explicitly? How do I switch the user?" All you need are a few simple steps.

1. Create a trusted context object in the database.
2. Depending on your application type or, more specifically, database connectivity type, use the appropriate API to get a trusted connection or pooled connection.
3. Using that trusted connection or connection pool, request a trusted connection for a certain user or switch to another user. Again, use the appropriate API available to your application.

Let's look at an example tailored for a Java JDBC application using the following steps.

1. Create a trusted context object in the database.
   Typically you'd ask your security administrator to create a trusted context object for you in the subject database. In this example, we limit the trusted context object to the localhost (locally on the database server machine). Sample trusted context with multiple users:

   ```
   create trusted context CTX_2 BASED UPON CONNECTION
   USING SYSTEM AUTHID db2inst1
   ATTRIBUTES (ADDRESS '127.0.0.1')
   DEFAULT ROLE managerRole ENABLE
   WITH USE FOR
   db2inst2 WITH AUTHENTICATION,
   db2inst3 ROLE dummy_role WITHOUT AUTHENTICATION
   ```

   Note: An explicit trusted connection cannot be established if the CLIENT type of authentication is in effect.

2. Get pooled connection.
   Construct a new object of type `com.ibm.db2.jcc.DB2ConnectionPoolDataSource` and set the connection attributes (server name, port number, database name, and connection type). After you have the attributes set correctly, call the `getDB2TrustedPooledConnection` method and pass it to the authorization ID and password to get a `com.ibm.db2.jcc.DB2PooledConnection` as the first element of the returned array of objects. The second object is a cookie that you need to keep in a variable. You will need it while switching to other users.
3. Get connection for certain user or switch the users.
Using that DB2PooledConnection you got in the last step, you can get a new connection for every user defined in the trusted context object (db2inst2 and db2inst3 in our example). All you need to do is call your DB2PooledConnection object's method getDB2Connection(cookie, user, pass, null, null, null, properties) and pass it to the cookie (discussed in previous step), the user name, and password (if WITH AUTHENTICATION is used). The remaining options are available for Kerberos and DB2 for z/OS (see the DB2 Information Center).

Depending on your connectivity option to the database, you may have other APIs for step 2 and step 3. Available options for step 2 are summarized below.

<table>
<thead>
<tr>
<th>Connection type</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI/ODBC</td>
<td>SQLConnect, SQLSetConnectAttr</td>
</tr>
<tr>
<td>XA CLI/ODBC</td>
<td>Xa_open</td>
</tr>
<tr>
<td>JAVA</td>
<td>getDB2TrustedPooledConnection, getDB2TrustedXAConnection</td>
</tr>
</tbody>
</table>

Available options for step 3 are summarized below.

<table>
<thead>
<tr>
<th>Connection type</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI/ODBC</td>
<td>SQLSetConnectAttr</td>
</tr>
<tr>
<td>XA CLI/ODBC</td>
<td>SQLSetConnectAttr</td>
</tr>
<tr>
<td>JAVA</td>
<td>getDB2Connection, reuseDB2Connection</td>
</tr>
<tr>
<td>.NET</td>
<td>DB2Connection.ConnectionString keywords: TrustedContextSystemUserID and TrustedContextSystemPassword</td>
</tr>
</tbody>
</table>

**Summary**

Now that you've completed this tutorial, you should have a fundamental understanding of the following topics:

- Elements of a DB2 security plan. You should understand the structure of the entire DB2 environment, which includes client, servers, gateways, and hosts. You should also understand authentication, authorization, and privileges.
- DB2 authentication types. We explored how to set authentication types using the db2 update dbm cfg using authentication type command on the server and using the db2 catalog database command on the gateway and client.
- DB2 authorities. You should understand the basics of the SYSADM, SYSCTRL, SYMAINT, and SYMON authorities, which are set in the DBM CFG file. You should also understand the basics of the DBADM, LOAD, and SECADM authorities, which are set using the GRANT command and revoked using the REVOKE command. We also covered what command each authority is allowed to run.
- DB2 privileges. You should have an understanding of the different types of privileges and what they allow a user to do. Examples are CONTROL, INSERT, DELETE, CREATEIN, DROPIN, REFERENCES, and SELECT. You should know how a privilege is obtained and
revoked explicitly (GRANT/REVOKE commands), implicitly, or (for packages only) indirectly. You should also have a basic understanding of label-based access control and how to define different types of policies based on this new security concept.

- DB2 roles and trusted contexts.
- Row and Column Access Control (RCAC). You should have an understanding of the fine-grained access control, RCAC, introduced in DB2 version 10.1. You should be able to use RCAC to control access to a table at the row level, column level, or both through the new SQL statements CREATE PERMISSION and CREATE MASK. You should also be able to activate and deactivate RCAC on the table level.
Related topics

- Read the Preparation Guide for DB2 10.1 Fundamentals Exam 610 to learn in-depth information about each of the concepts presented in this tutorial. This guide is a compilation of topics from the DB2 10.1 Information Center documentation.
- IBM DB2 10.1 for LUW Information Center has the official documentation for DB2 Linux, UNIX and Windows 10.1. Use it to improve your experience with DB2 and get help whenever needed.
- "DB2 Express-C 10: Even more value" (developerWorks, 2012): Discover the new features brought to you on DB2 Express-C 10.