
Secure your database

Skill Level: Intermediate

Paul Read (paul_read@uk.ibm.com)
Product Introduction Manager
IBM

20 Dec 2007

IBM® DB2® 9.5 provides new options for tighter security, and allows for more granularity and flexibility in administration of the database. This tutorial is the first of two tutorials that cover roles and trusted contexts. Follow the exercises in this tutorial, and learn how to take advantage of the new DB2 feature roles in combination with other essential e-business technologies such as Web services, Web application server, and DB2 database server.

Section 1. Before you start

About this tutorial

This tutorial will take you through a series of exercises to familiarize yourself with roles, a new feature in DB2 9.5. This tutorial is intended for DB2 technical specialists, database administrators, and programmers. You should have a good understanding of DB2 on Linux, UNIX, and Windows (hereafter called DB2 LUW). You should also be familiar with the DB2 command window and running DB2 scripts. Roles may also be created and managed using the new IBM Data Studio. However, they are not supported in the DB2 Control Center.
Objectives

This tutorial will help familiarize you with concepts and features of roles in DB2 9.5. In these exercises, you will learn:

- The basic concepts for roles
- How to create and manage roles
- How to use various SQL queries and tools to analyze the usage of roles

System requirements

To run the examples in this tutorial, you need:

- **DB2 9.5 Express-C**
- Microsoft Windows 2003, XP or **Linux** (Validated Environment)
- Java® Runtime Environment 1.4.2 or later

Also ensure that your hardware meets the requirements for DB2 9.5. (Refer to the DB2 9.5 system requirements page.)

DB2 9.5 Express C is available from the above link. DB2 9.5 is a full installation, not a fixpack upgrade. By default, DB2 will automatically start after installation unless you request it not to automatically start.

Use the sample scripts and data provided in the accompanying zip file (see Download section) to demonstrate the concepts in this tutorial. Extract the contents into a subdirectory called DB2Roles (C:\DB2Roles or home/userid/DB2Roles). This directory will be referred to simply as DB2Roles throughout the tutorial. This tutorial assumes that you have used the default directories for the DB2 installation. This tutorial requires the creation of a number of userids, and all the exercises will use the ids created.

Definition of a role

A role is a database object to which one or more DB2 privileges, authorities, or other roles can be granted or revoked. A role does not have an owner and it can only be created or dropped by the security administrator (SECADM).

By associating a role with a user, the user inherits all the privileges held by the role, in addition to privileges already held by the user.
The key advantage of database roles is that they simplify the administration and management of privileges in a database. For instance:

- Security administrators can control access to their databases at a level of abstraction that is close to the structure of their organizations. For example, if the company has 12 branches and everyone within each branch has a set of identical privileges, then the SECADM would set 12 roles and then grant membership to users based on their location.

- Users are granted membership in the roles based on their job responsibilities. As the user's job responsibilities change, which may be frequent in a large organization, user membership in roles can be easily granted and revoked. For example, if a user moves from the New York branch to the Boston branch, then the SECADM simply revokes his access to the role for New York and grants access to the role for the Boston branch.

- The assignment of privileges is simplified. Instead of granting the same set of privileges to each individual user in a particular job function, the administrator can grant this set of privileges to a role representing that job function and then grant that role to the users in that job function. For example, individual jobs can often require many different privileges for a user. However, if the privileges are granted to a role, then it is simple to grant or revoke the privileges without having to maintain large scripts for each job. If the SECADM needs to alter the privileges for a role, he can alter it in one place without having to replicate to process for all users.

- Roles can be updated without updating the privileges for every user on an individual basis. For example, if the SECADM needs to alter the privileges for the branch in New York, she can alter the definition for the role without having to replicate the process for all users.

All DB2 privileges and authorities that can be granted within a database, with the exception of SECADM, can be granted to a role. By granting privileges and authorities to roles only, and making users members in roles, the administration and management of privileges in the database is greatly simplified.

Section 2. Building the environment

This section introduces some sample code, available for download with this tutorial, to describe the behavior of roles. (See Download section.) The samples are based on database objects that can be added to the SAMPLE database. The data needed
to fill the tables is in a delimited ASCII file in the subdirectory DB2Roles with the name data.del. All scripts are in the subdirectory Section2.

To run the scripts in a DB2 command window, use:

```sql
db2 -tvf scriptname.sql
```

Batch scripts (suffixed .bat) can be run by typing the names.

**Instructions**

First, log on to your system using the administrator/root userid, then create the following userids:

- DB2SEC for SECADM
- MARK, ALLY, and SAUL will be the management team
- ADAM, DEBS, PETE, YANG, MARY, and ANNE
- ROSE, STAN, ALAN, LORI, EVAN, and KLEM

These userids do not need any special authorities as they will only be used with the DB2 database; all privileges and authorities required will be issued using SQL. You also need a group called Pension_gp. Place ROSE into this group. No other group authorities should be set. If using Windows, please ensure that users are not in the Administrators group.

The following diagram shows the hierarchy with the company and the roles that will be used in the exercises:

**Figure 1. Roles structure**
Please note that some names appear in two boxes because some individuals cover work for two departments and therefore they need to be in more than one role. KLEM is a new recruit and therefore not yet assigned to a role.

In the exercises that follow, DB2inst1 is used as the DB2 database administrator and the SAMPLE database as the database. All scripts use the DB2inst1 userid and a password of "password". You must edit the scripts to match your system if you use a different database or database administration id (DBA). Please verify that the "Sample" database or another usable database already exists. If one does not, please create the SAMPLE database using the DB2SAMPL command, as it is used throughout the rest of the tutorial.

Now, add the new users and tables to the database for this set of exercises:

1. Add the users to the database.

2. Connect to the database as the DBA and grant the following privileges:

   Listing 1. Grant privileges to database users

   ```sql
   GRANT CONNECT ON DATABASE TO USER MARK;
   GRANT CONNECT ON DATABASE TO USER ALLY;
   GRANT CONNECT ON DATABASE TO USER ADAM;
   GRANT CONNECT ON DATABASE TO USER DEBS;
   GRANT CONNECT ON DATABASE TO USER PETE;
   ```
GRANT CONNECT ON DATABASE TO USER YANG;
GRANT CONNECT ON DATABASE TO USER SAUL;
GRANT CONNECT ON DATABASE TO USER MARY;
GRANT CONNECT ON DATABASE TO USER ANNE;
GRANT CONNECT ON DATABASE TO USER ROSE;
GRANT CONNECT ON DATABASE TO USER STAN;
GRANT CONNECT ON DATABASE TO USER ALAN;
GRANT CONNECT ON DATABASE TO USER LORI;
GRANT CONNECT ON DATABASE TO USER EVAN;
GRANT SECADM ON DATABASE TO USER DB2SEC;

or use the AddUsers.sql script to add the new users.

3. Create the new tables for the SAMPLE database with the luwebase.sql script. It assumes that you are putting the tables into USERSPACE1. The following adds the base tables:

Listing 2. Add the base tables

LE_DEPARTMENT
LE_EMPLOYEE
LE_EMP_PHOTO
LE_EMP_RESUME
LE_ORG
LE_SALES
LE_PENSIONS

4. Load data into the new tables with the luweload.sql script. You must edit the script for Linux or if you have changed the locations of the load files.

5. To complete the setup, run the luweplus.sql script to create additional views, indices, and other database objects.

Please check for errors and warnings when you run the SQL scripts. You may also want to check the objects using the Control Center or another tool to verify that you have the tables and data.

Section 3. Set up and configure roles
In this section, you will learn how to:

- Create new roles
- Assign privileges to a role
- Grant membership to roles

The authority to manage membership in a role is vested in SECADM. The SECADM uses standard DML commands to manage the roles.

All DML commands for use with roles can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package.

The **CREATE ROLE** statement defines a role at the current server. The role must have a *unique* identifying name:

```
CREATE ROLE *role-name*
```

The **DROP ROLE** statement deletes a role at the current server. The role must already exist at the current server, and the role cannot be deleted if it is currently in use as a connection attribute (SESSION_USER) or as a trusted context. **Only** the SECADM can drop a role.

```
DROP ROLE *role-name*
```

The **GRANT** and **REVOKE ROLE** statements are used to grant and revoke roles to users, groups, or to other roles. They can also be used to delegate authority by using the **WITH ADMIN OPTION**. The **WITH ADMIN OPTION** allows the specified authorization ID the authority to grant or revoke the role-name to or from others. For example:

**Listing 3. Grant or revoke the role-name to or from others**

```
GRANT ROLE *role-name* TO *authorization-name*

GRANT ROLE *role-name* TO *authorization-name* WITH ADMIN OPTION

REVOKE ROLE *role-name* FROM *authorization-name*

REVOKE ADMIN OPTION FOR ROLE *role-name* FROM *authorization-name*
```
The COMMENT ON statement can be used to provide a description of the ROLE in the system catalogs. For example:

```
COMMENT ON ROLE *role-name* IS *'comment'*
```

The scripts needed for this section are in the subdirectory Section3 of the available download.

All the system catalog views that have details about users or groups have been updated to include the new roles. There are two new views in DB2 9.5 that show the details of roles:

1. **SYSCAT.ROLES**: This view has one row for each role defined, and contains identifiers for the role, plus the creation date and associated audit policy information.

2. **SYSCAT.ROLEAUTH**: This view has one row for each user, group, or role that has been granted authority over a role and whether they have grant authority on the role.

To simplify administration of roles, there is a system function, called `AUTH_LIST_ROLES_FOR_AUTHID`, provided to enable the administrator the ability to verify which roles have been granted to a user, group, or role. (This function will be described and demonstrated in the next section, "Maintain and manage roles ".)

The role is simply a database object that can be used to define authorities over other database objects like tables, views, triggers, and trusted contexts. Users can be assigned to a role to enable them to access data within the database. In this tutorial, learn to create a hierarchy of roles that identify the structure of the small company, as illustrated in Figure 1.

You will create the following basic roles:

- Sales: For the sales department
- Mktg: For the marketing department
- Admin: For the administration department
- Payroll: For the payroll department
- Pension: For Yang, who is the pensions administrator
- SalesMgr: To include the roles Sales and Mktg
- AdminMgr: To include the roles Admin and Payroll
- CEO: To include the roles AdminMgr and SalesMgr

Instructions

Log on to your system using the security administrator userid and password that you created in the previous section. Change the current directory to Section3.

1. Create the basic roles using the following code:

```
Listing 4. Create the basic roles

CREATE ROLE Sales;
CREATE ROLE Mktg;
CREATE ROLE Admin;
CREATE ROLE Payroll;
CREATE ROLE Pension;
CREATE ROLE SalesMgr;
CREATE ROLE AdminMgr;
CREATE ROLE CEO;
```

or use the CreateRoles.sql script to create the roles.

The Privileges table, below, shows the privileges to be granted to the roles:

```
Table 1. Privileges

<table>
<thead>
<tr>
<th>Role</th>
<th>Privileges</th>
<th>Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>UPDATE, INSERT</td>
<td>LE_SALES</td>
</tr>
<tr>
<td>Mktg</td>
<td>SELECT</td>
<td>LE_ORG, LE_SALES</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>LE_DEPARTMENT, LE_EMPLOYEE, LE_ORG, LE_SALES</td>
</tr>
<tr>
<td>Admin</td>
<td>UPDATE</td>
<td>LE_EMPLOYEE, LE_DEPARTMENT</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>LE_DEPARTMENT, LE_EMPLOYEE, LE_EMP(Photo, LE_EMP_RESUME, LE_PENSIONS</td>
</tr>
</tbody>
</table>
```
2. Grant privileges to the roles using the following commands:

**Listing 5. Grant privileges to the roles**

```sql
GRANT SELECT ON TABLE db2inst1.le_org TO ROLE Sales;
GRANT SELECT, UPDATE, INSERT ON TABLE db2inst1.le_sales TO ROLE Sales;
GRANT SELECT ON TABLE db2inst1.le_org TO ROLE Mktg;
GRANT SELECT ON TABLE db2inst1.le_sales TO ROLE Mktg;
GRANT SELECT ON TABLE db2inst1.le_department TO ROLE Admin;
GRANT SELECT ON TABLE db2inst1.le_employee TO ROLE Admin;
GRANT SELECT ON TABLE db2inst1.le_employee TO ROLE Payroll;
GRANT UPDATE ON TABLE db2inst1.le_department TO ROLE Admin;
GRANT UPDATE ON TABLE db2inst1.le_employee TO ROLE Admin;
GRANT SELECT ON TABLE db2inst1.le_department TO ROLE Payroll;
GRANT SELECT ON TABLE db2inst1.le_employee TO ROLE Payroll;
GRANT SELECT ON TABLE db2inst1.le_emp_photo TO ROLE Payroll;
GRANT SELECT ON TABLE db2inst1.le_emp_resume TO ROLE Payroll;
GRANT UPDATE, INSERT, DELETE ON TABLE db2inst1.le_employee TO ROLE Payroll;
GRANT SELECT, UPDATE, INSERT, DELETE, ALTER ON TABLE db2inst1.le_pensions TO ROLE Payroll;
```

or use the `AssignRoles.sql` script to create the roles.

Now you need to put the individuals into their respective roles.

3. Grant membership of the roles using the following commands:

**Listing 6. Grant membership of the roles**

```sql
GRANT ROLE SalesMgr, AdminMgr to ROLE CEO;
GRANT ROLE CEO TO USER Mark;
GRANT ROLE Admin, Payroll to ROLE AdminMgr;
GRANT ROLE AdminMgr TO USER Ally;
GRANT ROLE Sales, Mktg to ROLE SalesMgr;
GRANT ROLE SalesMgr TO USER Saul;
GRANT ROLE Admin TO USER Adam, Debs, Mary;
GRANT ROLE Mktg TO USER Mary, Anne, Rose;
GRANT ROLE Payroll TO USER Adam, Pete, Yang;
GRANT ROLE Pension TO USER Yang;
GRANT ROLE Sales TO USER Stan, Alan, Lori, Evan;
```

or use the `AssignUsers.sql` script to grant membership of the roles.

Now that you have the basic structure, you can run some queries to
validate the privileges and authorities that belong to the users.

4. Connect to the SAMPLE database as Anne, and run two queries:

CONNECT TO SAMPLE USER Anne USING password
SELECT * FROM db2inst1.le_org
SELECT * FROM db2inst1.le_employee

or use the AnneQueries.sql script to run the queries. The SELECT from the ORG table will work, but Anne cannot access the EMPLOYEE table because Anne is only in the role MKTG. The output should be similar to the following:

**Listing 7. Query output**

CONNECT TO SAMPLE USER Anne USING

Database Connection Information

Database server = DB2/NT 9.5.0
SQL authorization ID = ANNE
Local database alias = SAMPLE

SELECT * FROM db2inst1.le_org

DEPTNUMB  DEPTNAME  MANAGER  DIVISION  LOCATION
---------  ----------  -------  --------  --------
C21       Head Office C01  Corporate  London
C21       Eastern     C01  Finance    Vienna
C21       Western     C01  Finance    Paris

9 record(s) selected.

SELECT * FROM db2inst1.le_employee
SQL0551N "ANNE" does not have the privilege to perform operation "SELECT" on object "DB2INST1.LE_EMPLOYEE". SQLSTATE=42501

5. Change the user to Mary, and repeat the queries:

CONNECT TO SAMPLE USER Mary USING password
SELECT * FROM db2inst1.le_org
SELECT * FROM db2inst1.le_employee

Both queries work for Mary because she is in both the MKTG and ADMIN roles. Therefore, you will be able to see the content of le_employee table (see Listing 8).

6. Connect to the SAMPLE database as Debs, and run two queries:
or use the DebsQueries.sql script to run the queries. The SELECT from the EMPLOYEE table will work, but Debs cannot access the PENSIONS table because Debs is only in the role Admin (see Listing 9).

7. Change the user to Adam, and repeat the queries:

```
CONNECT TO SAMPLE USER Adam USING password
SELECT * FROM db2inst1.le_employee
SELECT * FROM db2inst1.le_pensions
```

Both queries work for Adam because he is in the PAYROLL and the ADMIN roles. You can, therefore, see the le_pensions table (see Listing 10).

8. Optionally, change the user to Mark (CEO role). He is be able to run all the queries.

---

### Section 4. Maintain and manage roles

In this tutorial alone, it is not easy to see the benefits of roles in administering a database. However, if you look at a large company with thousands of employees in hundreds of roles, the benefits become clearer. For example, if the company has an average of 20 sales representatives per branch, 10 branches per region or state, and 50 regions within the sales structure, suddenly you have 10,000 sales representatives and probably another 1,000 associated staff. In previous releases of DB2, you would have had to grant privileges to each person or compromise security access to the data. Assume that you have different roles for each branch, region, and support roles; you still have over 500 roles. However, when a person joins the company, leaves the company, or moves position, it would be a simple grant or revoke of a role. If you are to assume an attrition rate of 5%, that is still over 500 changes in a year. However, if you delegate the administration to region or branch levels, the administration tasks at these levels becomes easily manageable.

In this section, you will look at:

- The WITH ADMIN OPTION
- The effects of revoking role authorities
• Querying membership authorities

Instructions

Log on to your system using the security administrator userid and password that you created in the "Building the environment" section of this tutorial. Change the current directory to Section4.

First, let's explore some of effects of revoking role membership:

1. Connect to the SAMPLE database as Mary. Run a join between the SALES and EMPLOYEE tables, and then create a view over the same two tables:
   
   **Listing 11. Testing Mary’s authorities**

   ```sql
   CONNECT TO SAMPLE USER Mary USING password
   SELECT e.firstnme, e.lastname, SUM(s.sales) 
   FROM db2inst1.le_employee e, db2inst1.le_sales s 
   WHERE e.firstnme=s.sales_person 
   GROUP BY e.firstnme, e.lastname 
   CREATE VIEW total_sales (fname, lname, sales) 
   AS SELECT e.firstnme, e.lastname, SUM(s.sales) 
   FROM db2inst1.le_employee e, db2inst1.le_sales s 
   WHERE e.firstnme=s.sales_person 
   GROUP BY e.firstnme, e.lastname
   ``

   or use the Mary01.sql script to run the SQL. The SELECT should produce a list of sales by salesperson, and the create view duplicates this data as a view. These commands work because Mary is a member of the Admin and Sales roles.
   
   **Listing 12. Results of Mary’s tests**

   ```sql
   CONNECT TO SAMPLE USER Mary USING 
   
   Database Connection Information 
   
   Database server = DB2/NT 9.5.0 
   SQL authorization ID = MARY 
   Local database alias = SAMPLE 
   
   SELECT e.firstnme, e.lastname, 
   SUM(s.sales) as Total_Sales 
   FROM db2inst1.le_employee e, db2inst1.le_sales s WHERE e.firstnme=s.sales_person 
   GROUP BY e.firstnme, e.lastname 
   
   ----------------- --------------- ----------- 
   FIRSTNME LASTNAME TOTAL_SALES 
   ------- --------------- -------- 
   ALAN OMAHONY 8 
   EVAN WELSH 34 
   ```
LORI       VERLOAD       57
STAN       DELIVER       64

4 record(s) selected.

CREATE VIEW total_sales (fname, lname, sales) AS
SELECT e.firstname, e.lastname, SUM(s.sales)
FROM db2inst1.le_employee e, db2inst1.le_sales s
WHERE e.firstname=s.sales_person
GROUP BY e.firstname, e.lastname

DB20000I The SQL command completed successfully.

Authorities or privileges inherited through role membership are treated
exactly the same as if they had been granted directly.

2. Now, let's remove Mary from the Admin role and try to use the SELECT
statement to use the view created by Mary:

Listing 13. Revoke ADMIN role from Mary

CONNECT TO SAMPLE USER db2sec USING password
REVOKE ROLE admin FROM Mary
CONNECT TO SAMPLE USER Mary USING password
SELECT e.firstname, e.lastname, SUM(s.sales)
FROM db2inst1.le_employee e, db2inst1.le_sales s
WHERE e.firstname=s.sales_person
GROUP BY e.firstname, e.lastname

SELECT * FROM total_sales

or use the Mary02.sql script to run the SQL. The SELECT fails because
Mary is not in the Admin role and, therefore, no longer has select
authority on LE_EMPLOYEE. The second SELECT also fails because the
view is marked as invalid.

Listing 14. Mary's no longer able to access ADMIN tables

CONNECT TO SAMPLE USER DB2SEC USING
Database Connection Information
Database server = DB2/NT 9.5.0
SQL authorization ID = DB2SEC
Local database alias = SAMPLE

REVOKE ROLE admin FROM Mary
DB20000I The SQL command completed successfully.

CONNECT TO SAMPLE USER Mary USING
Database Connection Information
Database server = DB2/NT 9.5.0
SQL authorization ID = MARY
Local database alias = SAMPLE
SELECT e.firstnme, e.lastname, SUM(s.sales) as Total_Sales
FROM db2inst1.le_employee e, db2inst1.le_sales s
WHERE e.firstnme=s.sales_person GROUP BY e.firstnme, e.lastname
SQL0551N "MARY" does not have the privilege to perform operation "SELECT" on object "DB2INST1.LE_EMPLOYEE". SQLSTATE=42501

SELECT * FROM total_sales
SQL0575N View or materialized query table "MARY.TOTAL_SALES" cannot be used because it has been marked inoperative. SQLSTATE=51024

To delegate the GRANT/REVOKE authority, use the WITH ADMIN OPTION.

3. The SECADM grants to Stan the ADMIN OPTION for the role SALES so that he can handle all new Salespeople. Stan grants access to the new salesman Klem. Test Klem's access.

Listing 15. Delegating ADMIN OPTION

CONNECT TO SAMPLE USER db2sec USING password
GRANT ROLE sales TO Stan WITH ADMIN OPTION
CONNECT TO SAMPLE USER Stan USING password
GRANT ROLE sales TO Klem WITH ADMIN OPTION

or use the Stan01.sql script to run the SQL. The first grant succeeds, but Stan is not able to pass the authority to another user. Therefore, the second grant fails. So just grant normal access to Klem.

Listing 16. Only SECADM can delegate ADMIN OPTION

CONNECT TO SAMPLE USER DB2SEC USING Database Connection Information
  Database server = DB2/NT 9.5.0
  SQL authorization ID = DB2SEC
  Local database alias = SAMPLE

GRANT ROLE sales TO Stan WITH ADMIN OPTION
DB20000I The SQL command completed successfully.

CONNECT TO SAMPLE USER Stan USING Database Connection Information
  Database server = DB2/NT 9.5.0
  SQL authorization ID = STAN
  Local database alias = SAMPLE

GRANT ROLE sales TO Klem WITH ADMIN OPTION
DB21034E The command was processed as an SQL statement because it was not a valid Command Line Processor command. During SQL processing it returned:
SQL0551N "STAN" does not have the privilege to perform operation "GRANT" on object "SALES". SQLSTATE=42501
4. Grant normal access to Klem:

**Listing 17. Testing delegated powers**

CONNECT TO SAMPLE USER Stan USING password
GRANT ROLE sales TO Klem
CONNECT TO SAMPLE USER Klem USING password
SELECT * FROM db2inst1.le_sales

or use the Stan02.sql script to run the SQL. Klem is now a member of the role Sales and he has all the necessary power for his new role as a salesman.

**Listing 18. Successful delegation of authority**

CONNECT TO SAMPLE USER Stan USING

Database Connection Information

Database server = DB2/NT 9.5.0
SQL authorization ID = STAN
Local database alias = SAMPLE

GRANT ROLE sales TO Klem
DB20000I The SQL command completed successfully.

CONNECT TO SAMPLE USER Klem USING

Database Connection Information

Database server = DB2/NT 9.5.0
SQL authorization ID = KLEM
Local database alias = SAMPLE

SELECT * FROM db2inst1.le_sales

<table>
<thead>
<tr>
<th>SALES_DATE</th>
<th>SALES_PERSON</th>
<th>REGION</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>31/12/2005</td>
<td>STAN</td>
<td>London</td>
<td>7</td>
</tr>
<tr>
<td>29/03/2006</td>
<td>STAN</td>
<td>London</td>
<td>3</td>
</tr>
<tr>
<td>30/03/2006</td>
<td>STAN</td>
<td>London</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If Stan was to retire and his authority, what would happen to Klem? Let's remove Stan from the database SAMPLE and test Klem's access:

**Listing 19. Revoking Stan's access**
CONNECT TO SAMPLE USER db2sec USING password
REVOKE ROLE sales FROM Stan
CONNECT TO SAMPLE USER Klem USING password
SELECT * FROM db2inst1.le_sales

or use the Stan03.sql script to run the SQL. Stan no longer has membership of the role Sales, and Klem retains all the necessary authorities.

Listing 20. Klem still has full access

CONNECT TO SAMPLE USER DB2SEC USING

Database Connection Information
Database server = DB2/NT 9.5.0
SQL authorization ID = DB2SEC
Local database alias = SAMPLE

REVOKE ROLE sales FROM Stan
DB20000I The SQL command completed successfully.
CONNECT TO SAMPLE USER Klem USING

Database Connection Information
Database server = DB2/NT 9.5.0
SQL authorization ID = KLEM
Local database alias = SAMPLE

SELECT * FROM db2inst1.le_sales
SALES_DATE   SALES_PERSON  REGION  SALES
------------- --------------- --------------- -----------
31/12/2005   STAN           London  7
29/03/2006   STAN           London  3
30/03/2006   STAN           London  8
.
.

5. If the administrator needs to query the roles, he can query the views in the catalogs SYSCAT.ROLES and SYSCAT.ROLEAUTH. However, DB2 9.5 provides a function, AUTH_LIST_ROLES_FOR_AUTHID, to query to usage of roles. The function has two parameters, AUTHID and TYPE. This exercise uses the AUTH_LIST_ROLES_FOR_AUTHID function to find the roles assigned to user(TYPE) Ally(AUTHID) and then for the role(TYPE) CEO(AUTHID):

Listing 21. Querying ROLES

CONNECT TO SAMPLE USER db2sec USING password
or use the Admin01.sql script to run the SQL. There is a third TYPE 'G', which represents groups.

Listing 22. Authorities for user Ally and role CEO

<table>
<thead>
<tr>
<th>Roles For Ally</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2SEC U ADMINMGR R</td>
<td>ADMMG</td>
<td>2007-11-29-17.14.54.765001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2SEC U ADMINMGR R</td>
<td>PAYROLL</td>
<td>2007-11-29-17.14.54.781002</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2SEC U ALLY U</td>
<td>ADMINMGR2007-11-29-17.14.54.828002</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles For CEO</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2SEC U SALESMGR R</td>
<td>SALES</td>
<td>2007-11-29-17.14.54.718000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2SEC U SALESMGR R</td>
<td>MKTG</td>
<td>2007-11-29-17.14.54.734002</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2SEC U ADMINMGR R</td>
<td>PAYROLL</td>
<td>2007-11-29-17.14.54.781002</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2SEC U CEO R</td>
<td>SALESMGR2007-11-29-17.14.54.812002</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2SEC U CEO R</td>
<td>ADMINMGR2007-11-29-17.14.54.828002</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Section 5. (Optional) Comparing roles and groups

In this section, explore the advantage of roles over groups. The key advantage of using roles instead of groups in DB2 is that roles are managed by the database. DB2, therefore, knows when membership in a role changes and can consider roles for all operations. This is not the case with groups because they are managed outside the database and are controlled by the operating system or an LDAP server (including Microsoft Active Directory). Because of this, DB2 imposes some restrictions on when privileges and authorities assigned to groups are taken into account when performing an authorization check. These restrictions include not taking groups into account when creating the following objects:

1. Views
2. Triggers
3. Materialized Query Tables (MQTs)
4. Packages with static SQL

5. SQL routines

Let's take a look at this with a view and a trigger.

The scripts needed to fill the tables are in a delimited ASCII file in the subdirectory DB2Roles with the name data.del.

Instructions

Log on to your system using the security administrator userid and password that you created in the “Building the environment” section of this tutorial. Change the current directory to Section5.

1. Create the table LE_PEN_DELETES using the following command:

```sql
CREATE TABLE db2inst1.le_pen_deletes
  (X int,
   D_DATE date)
```

or use the Yang01.sql script to run the SQL.

Listing 23. Created le_pen_deletes table

```sql
CONNECT TO SAMPLE USER DB2INST1 USING Database Connection Information
  Database server = DB2/NT 9.5.0
  SQL authorization ID = DB2INST1
  Local database alias = SAMPLE

CREATE TABLE db2inst1.le_pen_deletes (User VARCHAR(128), D_DATE DATE)
DB20000I The SQL command completed successfully.
```

2. Grant INSERT privilege to the role PENSION using the SECADM user DB2SEC:

```sql
GRANT INSERT ON db2inst1.le_pen_deletes TO ROLE Pension
```

or use the Yang02.sql script to run the SQL.

Listing 24. Grant INSERT on table
3. As user Yang, create the trigger PENS_TRG to insert a row into LE_PEN_DELETES every time a row is deleted in the LE_PENSIONS:

**Listing 25. Create trigger using ROLE authority**

```sql
CREATE TRIGGER PENS_TRG
AFTER DELETE on db2inst1.le_pensions
FOR EACH STATEMENT
MODE DB2SQL
INSERT INTO db2inst1.le_pen_deletes
VALUES (1, Current Date)
```

or use the Yang03.sql script to run the SQL.

The trigger should have completed successfully.

**Listing 26. Successful trigger creation**

```sql
CONNECT TO SAMPLE USER Yang USING
Database Connection Information
Database server = DB2/NT 9.5.0
SQL authorization ID = YANG
Local database alias = SAMPLE

CREATE TRIGGER pens_trg
AFTER DELETE ON db2inst1.le_pensions
FOR EACH STATEMENT
MODE DB2SQL
INSERT INTO db2inst1.le_pen_deletes
VALUES (Current User, Current Date)
```

4. Test it using the following code:

```sql
DELETE FROM db2inst1.le_pensions
WHERE EMPNO="012213"
SELECT * FROM db2inst1.le_pen_deletes
```
or use the Yang04.sql script to run the SQL.

Listing 27. Test the trigger

```sql
CONNECT TO SAMPLE USER Yang USING

   Database Connection Information

   Database server       = DB2/NT 9.5.0
   SQL authorization ID   = YANG
   Local database alias   = SAMPLE

DELETE FROM db2inst1.le_pensions WHERE EMPNO='012213'
DB20000I   The SQL command completed successfully.

CONNECT TO SAMPLE USER db2inst1 USING

   Database Connection Information

   Database server       = DB2/NT 9.5.0
   SQL authorization ID   = DB2INST1
   Local database alias   = SAMPLE

SELECT * FROM db2inst1.le_pen_deletes

USER D_DATE
------- ----------
YANG 30/11/2007
```

Let's try to replicate using groups. In the section "Building the environment," you should have created a group called PENSION_GP, and ROSE should be a member of the group. Rose is normally in the role MKTG and, therefore, has no access to any Pension role-related database objects.

5. Grant the authorities to the group PENSION_GP to enable the group to manipulate the pension tables:

Listing 28. Grant group authorities

```sql
GRANT CONNECT ON DATABASE TO GROUP Pension_gp
GRANT INSERT, UPDATE, ALTER, DELETE, SELECT ON db2inst1.le_pen_deletes
TO GROUP Pension-gp
GRANT DELETE ON db2inst1.le_pensions TO GROUP Pension_gp
```

or use the Rose01.sql script to run the SQL.

Listing 29. Group authorities granted

```sql
CONNECT TO SAMPLE USER db2inst1 USING

   Database Connection Information
```
As user Rose, create the trigger PENS_TRG_F to insert a row into LE_PEN_DELETES every time a row is deleted in the LE_PEN_DELETES:

Listing 30. Create trigger using GROUP authority

```sql
CREATE TRIGGER PENS_TRG_F
AFTER DELETE ON db2inst1.le_pensions
FOR EACH STATEMENT
MODE DB2SQL
INSERT INTO db2inst1.le_pen_deletes
VALUES (1, Current Date)
```

or use the Rose02.sql script to run the SQL.

This statement should generate the following error messages:

Listing 31. Trigger creation failed

```
CONNECT TO SAMPLE USER Rose USING
Database Connection Information
Database server = DB2/NT 9.5.0
SQL authorization ID = ROSE
Local database alias = SAMPLE

CREATE TRIGGER pens_trg_f
AFTER DELETE ON db2inst1.le_pensions
FOR EACH STATEMENT
MODE DB2SQL
INSERT INTO db2inst1.le_pen_deletes
VALUES ( Current User, Current Date)
```

DB21034E The command was processed as an SQL statement because it was not a valid Command Line Processor command. During SQL processing it returned: SQL0551N "ROSE" does not have the privilege to perform operation "ALTER TABLE" on object "DB2INST1.LE_PEN_DELETES".  LINE NUMBER=2.  SQLSTATE=42501

Although the group Pension_gp has the privilege "ALTER" and Rose is in the group, she is not able to create the trigger because group authorities
are not considered.

7. Now, as user Yang, create the view PENS_VIEW to view selected information from the table LE_PENSIONS, and query the data using the view:

**Listing 32. Create view using ROLE authority**

```sql
CREATE VIEW PENS_VIEW AS
(SELECT DISTINCT(job), SUM(base), SUM(avc)
FROM db2inst1.le_pensions
GROUP BY JOB)
SELECT * FROM PENS_VIEW
```

or use the Yang05.sql script to run the SQL. The view should complete successfully, and you can select data from the new view.

**Listing 33. Successful view creation**

```sql
CONNECT TO SAMPLE USER Yang USING
Database Connection Information
Database server = DB2/NT 9.5.0
SQL authorization ID = YANG
Local database alias = SAMPLE

CREATE VIEW pens_view (career, tot_pen, tot_avc) AS SELECT job, SUM(base), SUM(AVC)
FROM db2inst1.le_pensions GROUP BY job
DB20000I The SQL command completed successfully.

SELECT * FROM pens_view
```

<table>
<thead>
<tr>
<th>CAREER</th>
<th>TOT_PEN</th>
<th>TOT_AVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYST</td>
<td>265750.00</td>
<td>146060.00</td>
</tr>
<tr>
<td>CEO</td>
<td>1527500.00</td>
<td>4220.00</td>
</tr>
<tr>
<td>CLERK</td>
<td>266400.00</td>
<td>28130.00</td>
</tr>
<tr>
<td>MANAGER</td>
<td>811624.00</td>
<td>16360.00</td>
</tr>
<tr>
<td>P LEADER</td>
<td>821895.00</td>
<td>186391.00</td>
</tr>
<tr>
<td>SALESREP</td>
<td>220670.00</td>
<td>23054.00</td>
</tr>
</tbody>
</table>

8. Now, as user Rose, try to create the view PENS_VIEW_F to view selected information from the table LE_PENSIONS, and also try to select the data directly:

**Listing 34. Create view using GROUP authority**

```sql
CREATE VIEW PENS_VIEW_F AS
```
(SELECT DISTINCT(job), SUM(base), SUM(avc)
FROM db2inst1.le_pensions
GROUP BY JOB)

or use Rose03.sql script to run the SQL.
The create view statement should generate error messages. However, the direct select works because of the limitations on groups.

Listing 35. Failed view creation

CONNECT TO SAMPLE USER Rose USING

Database Connection Information

Database server = DB2/NT 9.5.0
SQL authorization ID = ROSE
Local database alias = SAMPLE

CREATE VIEW pens_view_f (career, tot_pen, tot_avc) AS SELECT job, SUM(base), SUM(AVC)
FROM db2inst1.le_pensions GROUP BY job
DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL0551N "ROSE" does not have the privilege to perform operation "SELECT" on
object "DB2INST1.LE_PENSIONS". SQLSTATE=42501

SELECT job, SUM(base), SUM(AVC) FROM db2inst1.le_pensions GROUP BY job

<table>
<thead>
<tr>
<th>JOB</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYST</td>
<td>265750.00</td>
<td>146060.00</td>
</tr>
<tr>
<td>CEO</td>
<td>1527500.00</td>
<td>4220.00</td>
</tr>
<tr>
<td>CLERK</td>
<td>266400.00</td>
<td>28130.00</td>
</tr>
<tr>
<td>MANAGER</td>
<td>811624.00</td>
<td>16350.00</td>
</tr>
<tr>
<td>P LEADER</td>
<td>821895.00</td>
<td>186391.00</td>
</tr>
<tr>
<td>SALESREP</td>
<td>220670.00</td>
<td>23054.00</td>
</tr>
</tbody>
</table>

Section 6. Conclusion

This tutorial has introduced the concept of roles using exercises and samples. It has demonstrated how to use and take advantage of this new DB2 feature, roles. This tutorial has also covered the basic administration of roles.
### Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample DB2 scripts and data for this tutorial</td>
<td>Roles_Scripts_Data.zip</td>
<td>520 KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Information about download methods
Resources

Learn

- **DB2 Express-C**: Learn about the free version of DB2 Express Edition for the community.


- **DB2 9 articles on developerWorks**: Check out this collection of articles to learn all you want to know about DB2 9.

- **developerWorks Information Management zone**: Learn more about Information Management. Find technical documentation, how-to articles, education, downloads, product information, and more.

- **developerWorks DB2 for Linux, UNIX, and Windows resource page**: Find articles and tutorials, and link to documentation, support, learning resources, and more.

- Stay current with **developerWorks technical events and webcasts**.

- **Technology bookstore**: Browse for books on these and other technical topics.

Get products and technologies

- **DB2 Express-C**: Download DB2 Express-C to try out the features described in this article.

- Build your next development project with **IBM trial software**, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.

- Check out **developerWorks blogs** and get involved in the **developerWorks community**.

About the author

Paul Read

Paul Read is a product introduction manager in the Global BetaWorks team, based in the IBM Lab in Hursley, Hampshire, United Kingdom. Paul is the lead professional for Beta and Early Support Programs for Information Management products on Linux, Unix, and Windows. He has also provided technical consultancy and skills transfer for the Information Management software products across all platforms.