Security - User management

In this lecture

- Data protection in Teradata.
- User management.

Transient Journal

- Teradata uses transient journal to protect data from transaction failures.
- Whenever any transactions are run, transient journal keeps a copy of the before images of the affected rows until the transaction is successful or rolled back successfully.
- Then, the before images are discarded.
- Transient journal is kept in each AMPs.

Fallback

- Fallback option can be used at table creation or after table creation.
- Fallback ensures that a second copy of the rows of the table is always stored in another AMP to protect the data from AMP failure.
- However, fallback occupies twice the storage and I/O for Insert/Delete/Update.

Down AMP Recovery Journal

- The Down AMP recovery journal is activated when the AMP fails and the table is fallback protected.
- This journal keeps track of all the changes to the data of the failed AMP. The journal is activated on the remaining AMPs in the cluster.
- Once the failed AMP is live then the data from the Down AMP recovery journal is synchronized with the AMP.
- Once this is done, the journal is discarded.

Cliques

- **Clique** is a mechanism used by Teradata to protect data from Node failures.
- A clique is nothing but a set of Teradata nodes that share a common set of Disk Arrays.
- When a node fails, then the vprocs from the failed node will migrate to other nodes in the clique and continue to access their disk arrays.

Hot Standby Node

- **Hot Standby Node** is a node that does not participate in the production environment.
- If a node fails then the vprocs from the failed nodes will migrate to the hot standby node.
- Once the failed node is recovered it becomes the hot standby node.
- Hot Standby nodes are used to maintain the performance in case of node failures.

RAID

- Redundant Array of Independent Disks (RAID) is a mechanism used to protect data from Disk Failures.
- Any changes to the data in primary disk is reflected in mirror copy also.
- If the primary disk fails, then the data from mirror disk can be accessed.

Secure transactions

- Secure transactions can be defined as *if one statement in the SQL request fails then the entire transaction must fails and the system must rolls back to original state.*
- There are 3 ways to achieve implicit transactions in Teradata.
  - MultiStatement
  - BEGIN / END Transaction (BT / ET)
  - MACRO

MultiStatement

- A MultiStatement request is a sequence of DMLs with semi-colon(;) at the start of next DML(instead at the end of current DML) and so on.
- There should not be any DDL statement within MultiStatement request.
- The advantage is that dirty reads are not possible as the processed data will be available to other sessions only after all DMLS are successfully executed in MultiStatement request.
- Performance of multiple DMLs executed as multistatement will be better than running DMLs individually.

BEGIN / END Transaction (BT / ET)

- Statements within BT / ET works as an implicit transaction but dirty reads are possible for other sessions.
- To start implicit transaction, `BT;` is executed first and then series of statements can be executed and finally `ET;` must be issued to commit the changes to database.
- Disadvantage is that dirty reads are possible for the participating table, this means that intermediate data will be available as the current state of data to the other sessions.

Macro

- DMLs defined within a macro work as an implicit transaction.
- Advantage is that dirty reads are not possible as the processed data will be available to other sessions only after all the statements are executed successfully in the macro.

User Management

Users

- A user is created using `CREATE USER` command.
- In Teradata, a user is also similar to a database.
- They both can be assigned space and contain database objects except that the user is assigned a password.

```
CREATE USER username
AS
[PERMANENT|PERM] = n BYTES
PASSWORD=password
TEMPORARY=n BYTES
SPOOL=n BYTES;
```

- For creating a user, the values for user name, Permanent space and Password are mandatory. Other fields are optional.

User Creation

```
CREATE USER tutorial_user AS
   PASSWORD=tutorial_user
   PERM = 100000000   -- 100 MB
   SPOOL= 10000000    -- 10 MB
   ;
```

Database Creation

```
CREATE DATABASE tutorial_db
    FROM DBC
    AS
    PERM = 1000000000   -- 1000 MB
;
```

Grant

- Grant on Database: Providing complete access(creation/dropping/modifying objects) on tutorial_db to tutorial_user.

```
GRANT ALL ON tutorial_db TO tutorial_user;
```

- Access to DBC Tables: Providing select access to tutorial_user on DBC data dictionary tables.

```
GRANT SELECT ON dbc TO tutorial_user;
```

Grant Privileges

- `GRANT` command is used to assign one or more privileges on the database objects to the user or database.
- `GRANT privileges ON objectname TO username;` Privileges can be `ALL`, `INSERT`, `SELECT`, `UPDATE`, `REFERENCES`.

Revoke Privileges

- `REVOKE` command removes the privileges from the users or databases.
- Syntax: `REVOKE [ALL|privileges] ON objectname FROM username;`
- **Example**

  ```
  REVOKE INSERT,SELECT ON Employee FROM TD01;
  ```

MODIFY

- Change Password:

  ```
  MODIFY USER tutorial_user AS PASSWORD= "NewPassword";
  ```

- Default Database:

  ```
  MODIFY USER tutorial_user AS DEFAULT
  DATABASE=tutorial_db;
  ```

- Access to specific object creation:
  - Procedure Creation Access:

    ```
    GRANT CREATE PROCEDURE ON tutorial_db TO
    tutorial_user;
    ```

  - Function Creation Access:

```
GRANT CREATE FUNCTION ON tutorial_db to tutorial_user;
GRANT DROP FUNCTION ON tutorial_db to tutorial_user;
GRANT EXECUTE FUNCTION ON tutorial_db to tutorial_user;
```

Account, Profile and Role

- One user can have multiple accounts, for different workload and priority purposes.
- One account can be applied to many users, so that they share the same accounting properties.
- Account IDs may begin with $L, $M, $H, or $R, meaning priorities low, medium, high and rush respectively.

Accounts

- While creating a new user, the user may be assigned to an account.
- `ACCOUNT` option in `CREATE USER` is used to assign the account.
- A user may be assigned to multiple accounts.

```
CREATE USER username
PERM=n BYTES
PASSWORD=password
ACCOUNT=accountid
```

Specify Account Id

- The user can specify the account id while logging into Teradata system or after being logged into the system using the `SET SESSION` command.

```
.LOGON username, passowrd,accountid
OR
SET SESSION ACCOUNT=accountid
```

Profile

A profile is a set of common user parameters that can be applied to a group of users. The parameters include:

- Account IDs.
- Default database.
- Spool space allocation.
- Temporary space allocation.
- Password attributes.

Profile - Example

```
/*Create a Profile*/
CREATE PROFILE Employee_P AS
ACCOUNT = 'L_load_&S&D&H',
DEFAULT DATABASE = EmployeeDB,
SPOOL = 1E9, TEMPORARY = 1E7,
PASSWORD ATTRIBUTES = (EXPIRE=90, MAXLOGONATTEMPTS=3);

/*Assign the Profile to a user*/
CREATE/MODIFY USER Emp01 AS PERM=0,
PASSWORD=123, PROFILE=Employee_P;

/*Remove a profile from a user*/
MODIFY USER Emp01 AS PROFILE=NULL;
```

Role

- A role is simply a collection of access rights.
- It can be viewed as a "pseudo-user" with priviledges on a number of database objects.
- Any user granted a role can take on the identity of the pseudo-user and access all of the objects it has rights to.

Role - Example

```
/*Create a new Role*/
CREATE ROLE Role_A;

/* Grant some privileges to this Role*/
GRANT SELECT, EXECUTE ON View1 TO Role_A;

/*Assign/revoke this role to a user*/
GRANT/REVOKE Role_A TO User1;

/*The user can change its role*/
SET ROLE Role_B;

/* Or void the current role*/
SET ROLE NONE;

/*Or have all valid roles (for that user) to be active*/
SET ROLE ALL;
```

Row-level security

- RLS allows you to restrict data access on a row-by-row basis in
- Row-level security policies can be used in addition to the standard GRANT privileges.
- RLS supports both hierarchical and non-hierarchical security schemes.
- These are implemented as user defined functions (written in C).

Row-level security (cont.)

- A **hierarchical security scheme** defines hierarchical security levels. Users with higher security levels automatically have access to rows protected by lower security levels.
    - **Example:** top secret, secret, classified, unclassified.
- A **non-hierarchical security scheme** defines distinct and unrelated security categories. Access granted to one type of protected row does not automatically allow access to rows protected with any other security category.
    - **Example:** Information from different countries.

Locks

Teradata Locks

Locking prevents multiple users who are trying to change the same data at the same time and in turn this helps in preventing data corruption.
There are four types of locks:

- **Exclusive:** prevents any other type of concurrent access.
- **Write:** prevents other reads, writes, exclusives access.
- **Read:** prevents writes and exclusives access.
- **Access:** prevents exclusive access only.

Implicit vs Explicit

Teradata Locking can be Implicit (automatically by Teradata) or Explicit (specified by users)

- Implicit lock based on the SQL command:
  - `SELECT` applies a Read lock
  - `UPDATE` applies a Write lock
  - `CREATE TABLE` applies an Exclusive lock

Implicit vs Explicit (cont.)

- Explicit lock using `LOCKING` modifier:

```
/* Access lock for Select */
LOCKING FOR ACCESS SELECT * FROM tutorial.employee;

/* Exclusive lock for Update */
LOCKING FOR EXCLUSIVE
UPDATE tutorial.employee SET job_title = 'CEO'
WHERE emp_no=1000245;

/*Write lock for Update*/
LOCKING FOR WRITE NOWAIT
UPDATE tutorial.employee SET job_title = 'CEO'
WHERE emp_no=1000245;
```

When `NOWAIT` is used in the query request and if the requested lock cannot be granted immediately then the query will be aborted by Teradata