**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Lab Assignment #**

**Submitted by:**

Binod Marikhu  
013BSCCSIT013

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav**  Lecturer  St. Xavier’s College |  |

**Date of Submission:** November 5, 2015

**GRANT and REVOKE authorization**

Use of grant and revoke privileges to control access

The SQL GRANT statement lets you grant explicit privileges to authorization IDs. The REVOKE statement lets you take them away. Only a privilege that has been explicitly granted can be revoked.

Granting privileges is very flexible. For example, consider table privileges. You can grant all the privileges on a table to an ID. Alternatively, you can grant separate, specific privileges that allow that ID to retrieve data from the table, insert rows, delete rows, or update specific columns. By granting or not granting those privileges on views of the table, you can effectively determine exactly what action an ID can or cannot take on the table.

You can use the GRANT statement to assign privileges as follows:

* Grant privileges to a single ID or to several IDs in one statement.
* Grant a specific privilege on one object in a single statement, grant a list of privileges, or grant privileges over a list of objects.
* Grant ALL, for all the privileges of accessing a single table or for all privileges that are associated with a specific package.

To grant or revoke a privilege using one of the SQL GRANT or REVOKE statements, the user must have the following permissions for the GRANT/REVOKE statement to succeed:

* [Superuser](javascript:toggleBlock('14712')) or privilege WITH GRANT OPTION
* USAGE privilege on the schema
* Appropriate privileges on the object

The syntax for granting and revoking privileges is different for each database object, such as schema, database, table, view, sequence, procedure, function, resource pool, and so on.

Normally, a superuser first [creates a user](http://my.vertica.com/docs/6.1.x/HTML/index.htm#3046.htm) and then uses GRANT syntax to define the user's privileges or roles or both. For example, the following series of statements creates user Carol and grants Carol access to the apps database in the PUBLIC schema and also lets Carol grant SELECT privileges to other users on the applog table:

=> CREATE USER Carol;

=> GRANT USAGE ON SCHEMA PUBLIC to Carol;

=> GRANT ALL ON DATABASE apps TO Carol;

=> GRANT SELECT ON applog TO Carol WITH GRANT OPTION;

revoke removes a previously granted or denied permission.

**Syntax**

REVOKE [ GRANT OPTION FOR ]

{

[ ALL [ PRIVILEGES ] ]

|

permission [ ( column [ ,...n ] ) ] [ ,...n ]

}

[ ON [ class :: ] securable ]

{ TO | FROM } principal [ ,...n ]

[ CASCADE] [ AS principal ]

**char**

**DATA ENCRYPTION**

Encryption is the process of translating plain text data ([plaintext](https://msdn.microsoft.com/en-us/library/windows/desktop/ms721603(v=vs.85).aspx#_security_plaintext_gly)) into something that appears to be random and meaningless ([ciphertext](https://msdn.microsoft.com/en-us/library/windows/desktop/ms721572(v=vs.85).aspx" \l "_security_ciphertext_gly)). Decryption is the process of converting ciphertext back to plaintext.

To encrypt more than a small amount of data, [symmetric encryption](https://msdn.microsoft.com/en-us/library/windows/desktop/ms721625(v=vs.85).aspx#_security_symmetric_encryption_gly) is used. A [symmetric key](https://msdn.microsoft.com/en-us/library/windows/desktop/ms721625(v=vs.85).aspx#_security_symmetric_key_gly) is used during both the encryption and decryption processes. To decrypt a particular piece of ciphertext, the key that was used to encrypt the data must be used.

The goal of every encryption algorithm is to make it as difficult as possible to decrypt the generated ciphertext without using the key. If a really good encryption algorithm is used, there is no technique significantly better than methodically trying every possible key. For such an algorithm, the longer the key, the more difficult it is to decrypt a piece of ciphertext without possessing the key.

It is difficult to determine the quality of an encryption algorithm. Algorithms that look promising sometimes turn out to be very easy to break, given the proper attack. When selecting an encryption algorithm, it is a good idea to choose one that has been in use for several years and has successfully resisted all attacks.

**Transitive , Reflexivity and Augmentation properties of FDs**

If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

**Reflexivity rule**

If A is a set of attributes, and B is a set of attributes that are completely contained in A, the A implies B.

**Augmentation rule**

If A implies B, and C is a set of attributes, then if A implies B, then AC implies BC.

**Transitivity rule**

If A implies B and B implies C, then A implies C.

**bcnf and DECOMPOSITION INTO BCNF**

The previous normalization forms are considered elementary, and should be applied on tables during our design process. This normalization form however, and the following forms, are done in special tables.

A table is considered in BCNF (Boyce-Codd Normal Form) if it’s already in 3NF AND doesn’t contain any nontrivial functional dependencies. That is it doesn’t contain any field (other than the primary key) that can determine the value of another field. Let’s take the following table:

|  |  |  |
| --- | --- | --- |
| **Student** | **Subject** | **Teacher** |
| Smith | Math | Dr. White |
| Smith | English | Dr. Brown |
| Jones | Math | Dr. White |
| Jones | English | Dr. Brown |
| Doe | Math | Dr. Green |

By taking into consideration the following conditions:

* For each subject, every student is educated by one teacher.
* Every teacher teaches one subject only.
* Each subject can be teached by more than one teacher.

It’s clear we have the following functional dependency:  
Teacher ->  Subject

And the left side of this dependency is not the primary key.

So, to convert the table from 3NF to BCNF, we do these steps:

* Determine in the table, a key other than the primary key. That can be left side to the functional dependency.
* Delete the key in the right side of our functional dependency in the main table.
* Make a table for this dependency, with it’s key being the left side of the dependency, as the following:

|  |  |
| --- | --- |
| **Student** | **Teacher** |
| Smith | Dr. White |
| Smith | Dr. Brown |
| Jones | Dr. White |
| Jones | Dr. Brown |
| Doe | Dr. Green |

And

|  |  |
| --- | --- |
| **Teacher** | **Subject** |
| Dr. White | Math |
| Dr. Brown | English |
| Dr. Green | Math |

Given: relation R with FD’s F

Look among the given FD’s for a BCNF violation X → Y

If any FD following from F violates BCNF, then there will surely be an FD in F itself that violates BCNF

Compute X +

Not all attributes, or else X is a superkey

Decompose R Using X → Y

Replace R by relations with schemas: 1. R1 = X + 2. R2 = R – (X + – X )

**characterizing schedules based on recoverability**

**Transaction schedule or history:**

* When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms what is known as a transaction schedule (or history).
* A schedule (or history) S of n transactions T1, T2, …, Tn:
* It is an ordering of the operations of the transactions subject to the constraint that, for each transaction Ti that participates in S, the operations of T1 in S must appear in the same order in which they occur in T1.
* Note, however, that operations from other transactions Tj can be interleaved with the operations of Ti in S.

**Schedules classified on recoverability:**

**Recoverable schedule:**

* One where no transaction needs to be rolled back.
* A schedule S is recoverable if no transaction T in S commits until all transactions T’ that have written an item that T reads have committed.

**Cascadeless schedule:**

* One where every transaction reads only the items that are written by committed transactions.

**Schedules requiring cascaded rollback:**

* A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back.

**Strict Schedules:**

* A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

**characterizing schedules based on serializability**

**Serial schedule:**

* A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule.
* Otherwise, the schedule is called nonserial schedule.

**Serializable schedule:**

* A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.

**Result equivalent:**

* Two schedules are called result equivalent if they produce the same final state of the database.

**Conflict equivalent:**

* Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

**Conflict serializable:**

* A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S’.

**View equivalence:**

* A less restrictive definition of equivalence of schedules

**View serializability:**

* Definition of serializability based on view equivalence.
* A schedule is view serializable if it is view equivalent to a serial schedule.

**transaction supports in sql**

* A single SQL statement is always considered to be atomic.
* Either the statement completes execution without error or it fails and leaves the database unchanged.
* With SQL, there is no explicit Begin Transaction statement.
* Transaction initiation is done implicitly when particular SQL statements are encountered.
* Every transaction must have an explicit end statement, which is either a COMMIT or ROLLBACK.

Sample SQL transaction:

EXEC SQL whenever sqlerror go to UNDO;

EXEC SQL SET TRANSACTION

READ WRITE

DIAGNOSTICS SIZE 5

ISOLATION LEVEL SERIALIZABLE;

EXEC SQL INSERT

INTO EMPLOYEE (FNAME, LNAME, SSN, DNO, SALARY)

VALUES ('Robert','Smith','991004321',2,35000);

EXEC SQL UPDATE EMPLOYEE

SET SALARY = SALARY \* 1.1

WHERE DNO = 2;

EXEC SQL COMMIT;

GOTO THE\_END;

UNDO: EXEC SQL ROLLBACK;

THE\_END: