ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

Database Management System

Assignment # 11

**SUBMITTED BY:**

Arun Sanjel

013BSCCSIT010

**SUBMITTED TO:**

|  |  |
| --- | --- |
| Mr. Sanjay Kr. Yadav  **Lecturer** |  |
| **Department of Computer Science** | |

Submission Date: November 5, 2015

# GRANT and REVOKE authorizations

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;

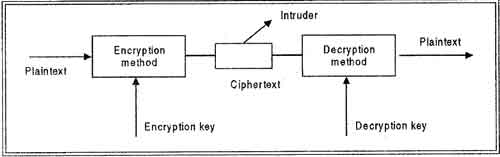
=> GRANT USAGE ON SCHEMA PUBLIC to Carol;

=> GRANT ALL ON DATABASE apps TO Carol;

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

# DATA ENCRYPTION

A DBMS can use encryption to protect information in certain situations where the normal security mechanisms of the DBMS are not adequate. For example, an intruder may steal tapes containing some data or tap a communication line. By storing and transmitting data in an encrypted form, the DBMS ensures that such stolen data is not intelligible to the intruder. Thus, encryption is a technique to provide privacy of data.



In encryption, the message to be encrypted is known as plaintext. The plaintext is transformed by a function that is parameterized by a key. The output of the encryption process is known as the cipher text. Ciphertext is then transmitted over the network. The process of converting the plaintext to ciphertext is called as Encryption and process of converting the ciphertext to plaintext is called as Decryption. Encryption is performed at the transmitting end and decryption is performed at the receiving end. For encryption process we need the encryption key and for decryption process we need decryption key as shown in figure. Without the knowledge of decryption key intruder cannot break the ciphertext to plaintext. This process is also called as Cryptography.

# 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

When a relation has more than one candidate key, anomalies may result even though the relation is in 3NF.3NF does not deal satisfactorily with the case of a relation with overlapping candidate keys i.e. composite candidate keys with at least one attribute in common.

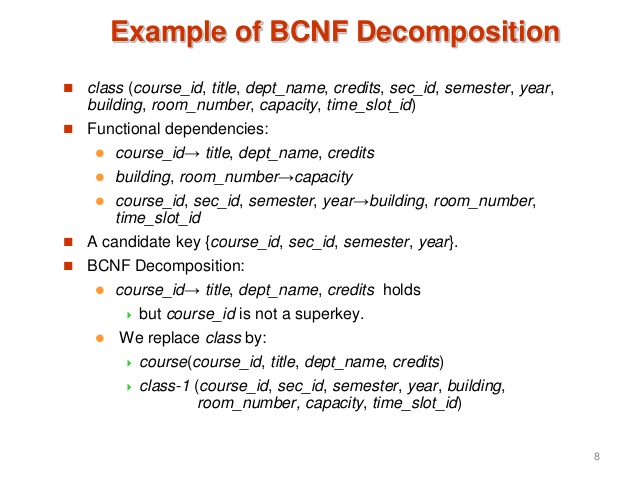
BCNF is based on the concept of a determinant. A determinant is any attribute (simple or composite) on which some other attribute is fully functionally dependent. A relation is in BCNF is, and only if, every determinant is a candidate key. Consider the following relation and determinants.

R(a,b,c,d)

a,c -> b,d

a,d -> b

Here, the first determinant suggests that the primary key of R could be changed from a,b to a,c. If this change was done all of the non-key attributes present in R could still be determined, and therefore this change is legal. However, the second determinant indicates that a,d determines b, but a,d could not be the key of R as a,d does not determine all of the non key attributes of R (it does not determine c). We would say that the first determinate is a candidate key, but the second determinant is not a candidate key, and thus this relation is not in BCNF (but is in 3rd normal form).



# 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:

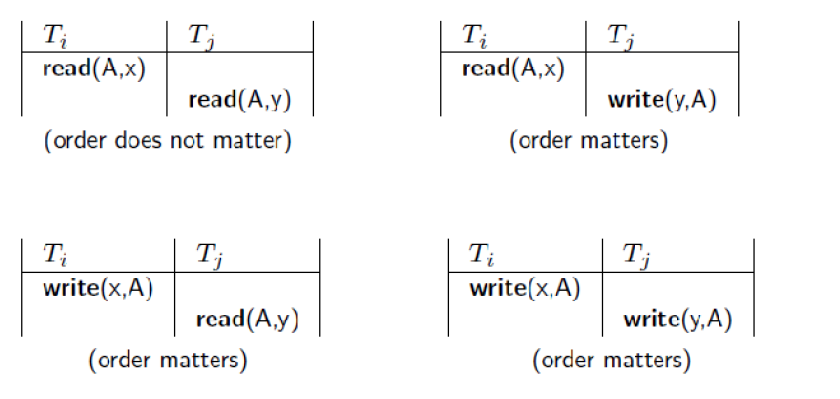
# A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

# Characterizing Schedules based on Serializability

DBMS must control concurrent execution of transactions to ensure read consistency, i.e., to avoid dirty reads etc. A (possibly concurrent) schedule S is serializable if it is equivalent to a serial schedule S0, i.e., S has the same result database state as S0.

**How to ensure serializability of concurrent transactions?**

Conflicts between operations of two transactions:



 A schedule S is serializable with regard to the above conflicts iff S can be transformed into a  
serial schedule S' by a series of swaps of non-conflicting operations.  
Checks for serializability are based on precedence graph that describes dependencies among  
concurrent transactions; if the graph has no cycle, and then the transactions are serializable.  
- they can be executed concurrently without affecting each other’s transaction result.

# Transactions Support in SQL

The definition of an SQL-transaction is that it is a logical unit of work and is guaranteed to be atomic. A single SQL statement is always considered to be atomic—either it 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. However, every transaction must have an explicit end statement, which is either a COMMIT or a ROLLBACK. Every transaction has certain characteristics attributed to it. These characteristics are specified by a SET TRANSACTION statement in SQL2. The characteristics are the *access mode,*the *diagnostic area size,*and the *isolation level.*

The **access mode**can be specified as READ ONLY or READ WRITE. The default is READ WRITE, unless the isolation level of READ UNCOMMITTED is specified, in which case READ ONLY is assumed. A mode of READ WRITE allows update, insert, delete and create commands to be executed. A mode of READ ONLY, as the name implies, is simply for data retrieval.

The **diagnostic area size**option, DIAGNOSTIC SIZE *n,*specifies an integer value *n,*indicating the number of conditions that can be held simultaneously in the diagnostic area. These conditions supply feedback information (errors or exceptions) to the user on the most recently executed SQL statement.

The **isolation level**option is specified using the statement ISOLATION LEVEL <isolation>, where the value for <isolation> can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE. The default isolation level is SERIALIZABLE, although some systems use as READ COMMITTED their default. The use of the term SERIALIZABLE here is based on not allowing violations that cause dirty read, unrepeatable read, and phantoms, and it is thus not identical to the way serializability.