**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Lab Assignment #11**

**Submitted by:**

Pankaj Niroula  
013BSCCSIT025

**Submitted to:**

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

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

**GRANT AND REVOKE AUTHORIZATIONS**

**GRANT**

*grant privilege statement*

<grant privilege statement> ::= GRANT <privileges> TO <grantee> [ { <comma> <grantee> }... ] [ WITH GRANT OPTION ] [ GRANTED BY <grantor> ]

Assign privileges on schema objects to roles or users. Each <grantee> is a role or a user. If [ WITH GRANT OPTION ] is specified, then the <grantee> can assign the privileges to other <grantee> objects.

<privileges> ::= <object privileges> ON <object name>

<object name> ::= [ TABLE ] <table name> | DOMAIN <domain name> | COLLATION <collation name> | CHARACTER SET <character set name> | TRANSLATION <transliteration name> | TYPE <user-defined type name> | SEQUENCE <sequence generator name> | <specific routine designator> | ROUTINE <routine name> | FUNCTION <function name> | PROCEDURE <procedure name>

<object privileges> ::= ALL PRIVILEGES | <action> [ { <comma> <action> }... ]

<action> ::= SELECT | SELECT <left paren> <privilege column list> <right paren> | DELETE | INSERT [ <left paren> <privilege column list> <right paren> ] | UPDATE [ <left paren> <privilege column list> <right paren> ] | REFERENCES [ <left paren> <privilege column list> <right paren> ] | USAGE | TRIGGER | EXECUTE

<privilege column list> ::= <column name list>

<grantee> ::= PUBLIC | <authorization identifier>

The <object privileges> that can be used depend on the type of the <object name>. These are discussed in the previous section. For a table, if <privilege column list> is not specified, then the privilege is granted on the table, which includes all of its columns and any column that may be added to it in the future. For routines, the name of the routine can be specified in two ways, either as the generic name as the specific name. HyperSQL allows referencing all overloaded versions of a routine at the same time, using its name. This differs from the SQL Standard which requires the use of <specific routine designator> to grant privileges separately on each different signature of the routine.

Each <grantee> is the name of a role or a user. Examples of GRANT statement are given below:

GRANT ALL ON SEQUENCE aSequence TO roleOrUser

GRANT SELECT ON aTable TO roleOrUser

GRANT SELECT, UPDATE ON aTABLE TO roleOrUser1, roleOrUser2

GRANT SELECT(columnA, columnB), UPDATE(columnA, columnB) ON TABLE aTable TO roleOrUser

GRANT EXECUTE ON SPECIFIC ROUTINE aroutine\_1234 TO rolOrUser

As mentioned in the general discussion, it is better to define a role for the collection of all the privileges required by an application. This role is then granted to any user. If further changes are made to the privileges of this role, they are automatically reflected in all the users that have the role.

**GRANT**

*grant role statement*

<grant role statement> ::= GRANT <role name> [ { <comma> <role name> }... ] TO <grantee> [ { <comma> <grantee> }... ] [ WITH ADMIN OPTION ] [ GRANTED BY <grantor> ]

Assign roles to roles or users. One or more roles can be assigned to one or more <grantee> objects. A <grantee> is a user or a role. If the [ WITH ADMIN OPTION ] is specified, then each <grantee> can grant the newly assigned roles to other grantees. An example of user and role creation with grants is given below:

CREATE USER appuser

CREATE ROLE approle

GRANT approle TO appuser

GRANT SELECT, UPDATE ON TABLE atable TO approle

GRANT USAGE ON SEQUENCE asequence to approle

GRANT EXECUTE ON ROUTINE aroutine TO approle

**REVOKE privilege**

*revoke statement*

<revoke privilege statement> ::= REVOKE [ GRANT OPTION FOR ] <privileges> FROM <grantee> [ { <comma> <grantee> }... ] [ GRANTED BY <grantor> ] RESTRICT | CASCADE

Revoke privileges from a user or role.

**REVOKE role**

*revoke role statement*

<revoke role statement> ::= REVOKE [ ADMIN OPTION FOR ] <role revoked> [ { <comma> <role revoked> }... ] FROM <grantee> [ { <comma> <grantee> }... ] [ GRANTED BY <grantor> ] RESTRICT | CASCADE

<role revoked> ::= <role name>

Revoke a role from users or roles.

**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.

The basic idea behind encryption is to apply an encryption algorithm, which may' be accessible to the intruder, to the original data and a user-specified or DBA-specified encryption key, 'which is kept secret. The output of the algorithm is the encrypted version of the data. There is also a decryption algorithm, which takes the encrypted data and the decryption key as input and then returns the original data. Without the correct decryption key, the decryption algorithm produces gibberish. Encryption and decryption keys may be same or· different but there must be relation between the both which must me secret.

**Techniques used for Encryption**

There are following techniques used for encryption process:

• Substitution Ciphers • Transposition Ciphers

**Substitution Ciphers**

In a substitution cipher each letter or group of letters is replaced by another letter or group of letters to mask them For example: a is replaced with D, b with E, c with F and z with C. In this way attack becomes DWWDFN. The substitution ciphers are not much secure because intruder can easily guess the substitution characters.

**Transposition Ciphers**

Substitution ciphers preserve the order of the plaintext symbols but mask them- ;-The transposition cipher in contrast reorders the letters but do not mask them. For this process a key is used. For example: iliveinqadian may be coded as divienaniqnli. The transposition ciphers are more secure as compared to substitution ciphers.

**Algorithms for Encryption Process**

There are commonly used algorithms for encryption process. These are:

• Data Encryption Standard (DES)

• Public Key Encryption

**Data Encryption Standard (DES)**

It uses both a substitution of characters and a rearrangement of their order on the basis of an encryption key. The main weakness of this approach is that authorized users must be told the encryption key, and the mechanism for communicating this information is vulnerable to clever intruders.

**Public Key Encryption**

Another approach to encryption, called public-key encryption, has become increasingly popular in recent years. The encryption scheme proposed by Rivest, Shamir, and Adheman, called RSA, is a well-knm.vn example of public-key encryption. Each authorized user has a public encryption key, known to everyone and a private decryption key (used by the decryption algorithm), chosen by the user and known only to him or her. The encryption and decryption algorithms themselves are assumed to be publicly known.

Consider user called Suneet. Anyone can send Suneet a secret message by encrypting the message using Sunset's publicly known encryption key. Only Suneet can decrypt this secret message because the decryption algorithm required Suneet's decryption key, known only to Suneet. Since users choose their own decryption keys, the weakness of DES is avoided.

The main issue for public-key encryption is how encryption and decryption keys are chosen. Technically, public-key encryption algorithms rely on the existence of one-way functions, which are functions whose inverse is computationally very hard to determine.

The RSA algorithm, for example is based on the observation that although checking whether a given number of prime is easy, determining the prime factors of a nonprime number is extremely hard. (Determining the prime factors of a number with over 100 digits can take years of CPU-time on the fastest available computers today.)

We now sketch the intuition behind the RSA algorithm, assuming that the data to be encrypted is an integer 1. To choose an encryption key and a decryption key, our friend Suneet-- create a public key by computing the product of two large prime numbers: PI and P2. The private key consists of the pair (PI, P2) and decryption algorithms cannot be used if the product of PI and P2 is known. So we publish the product PI \*P2, but an unauthorized user would need to be able to factor PIP2 to steal data. By choosing PI and P2 to be sufficiently large (over 100 digits), we can make it very difficult (or nearly impossible) for an intruder to factorize it.

Although this technique is secure, but it is also computationally expensive. A hybrid scheme used for secure communication is to use DES keys exchanged via a public-key encryption scheme and DES encryption is used on the data transmitted subsequently.

**Disadvantages of encryption**

There are following problems of Encryption:

* Key management (i.e. keeping keys secret) is a problem. Even in public-key encryption the decryption key must be kept secret.
* Even in a system that supports encryption, data must often be processed in plaintext form. Thus sensitive data may still be accessible to transaction programs.
* Encrypting data gives rise to serious technical problems at the level of physical storage organization. For example indexing over data, which is stored in encrypted form, can be very difficult.

**Transitivity, Reflexivity and Augmentation Properties of FDs**

**Transitivity Property**

A transitive dependency is a functional dependency which holds by virtue of transitivity. A transitive dependency can occur only in a relation that has three or more attributes. Let A, B, and C designate three distinct attributes (or distinct collections of attributes) in the relation. Suppose all three of the following conditions hold:

A → B

It is not the case that B → A

B → C

Then the functional dependency A → C (which follows from 1 and 3 by the axiom of transitivity) is a transitive dependency.

**Reflexivity Property**

This axiom says, if Y is a subset of X, then X determines Y (see Figure 11.1).

Ch-11-Axion-Reflexivity

Figure . Equation for axiom of reflexivity.

For example, **PartNo —> NT123**  where X (PartNo) is composed of more than one piece of information; i.e., Y (NT) and partID (123).

**Augmentation Property**

The axiom of augmentation, also known as a partial dependency, says if X determines Y, then XZ determines YZ for any Z (see Figure 11.2 ).

Ch-11-Axiom-of-Augmentation-300x34

Figure: Equation for axiom of augmentation.

The axiom of augmentation says that every non-key attribute must be fully dependent on the PK. In the example shown below, StudentName, Address, City, Prov, and PC (postal code) are only dependent on the StudentNo, not on the StudentNo and Grade.

StudentNo, Course —> StudentName, Address, City, Prov, PC, Grade, DateCompleted

This situation is not desirable because every non-key attribute has to be fully dependent on the PK. In this situation, student information is only partially dependent on the PK (StudentNo).

To fix this problem, we need to break the original table down into two as follows:

* Table 1: StudentNo, Course,  Grade, DateCompleted
* Table 2: StudentNo, StudentName, Address, City, Prov, PC

**Boyce Codd Normal Form**

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).

**Decomposition into BCNF**

This is how the relations are decomposed into BCNF:

**1. Determine BCNF:**  
For relation R to be in BCNF, all the functional dependencies (FDs) that hold in R need to satisfy property that the determinants X are all superkeys of R. i.e. if X->Y holds in R, then X must be a superkey of R to be in BCNF.

In your case, it can be shown that the only candidate key (minimal superkey) is ACE. Thus both FDs: A->B and C->D are violating BCNF as both A and C are not superkeys or R.

**2. Decompose R into BCNF form:**  
If R is not in BCNF, we decompose R into a set of relations S that are in BCNF.  
This can be accomplished with a very simple algorithm:

Initialize S = {R}

While S has a relation R' that is not in BCNF do:

Pick a FD: X->Y that holds in R' and violates BCNF

Add the relation XY to S

Update R' = R'-Y

Return S

In your case the iterative steps are as follows:

S = {ABCDE} // Intialization S = {R}

S = {ACDE, AB} // Pick FD: A->B which violates BCNF

S = {ACE, AB, CD} // Pick FD: C->D which violates BCNF

// Return S as all relations are in BCNF

Thus R(A,B,C,D,E) is decomposed into a set of relations: R1(A,C,E), R2(A,B) and R3(C,D) that satisfies BCNF.

Note also that in this case, functional dependency is preserved but normalization to BCNF does not guarantee this.

**Characterizing Schedules based on Recoverability**

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.

**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.

**Transactions supports 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. If a transaction executes at a lower isolation level than SERIALIZABLE, then one or more of the following three violations may occur:

1. Dirty read: A transaction may read the update of a transaction , which has not yet committed.

If fails and is aborted, then would have read a value that does not exist and is incorrect.

2. No repeatable read: A transaction may read a given value from a table. If another transaction later updates that value and reads that value again, will see a different value.

3. Phantoms: A transaction may read a set of rows from a table, perhaps based on some condition specified in the SQL WHERE-clause. Now suppose that a transaction inserts a new row that also satisfies the WHERE-clause condition used in, into the table used by. If is repeated, then will see a phantom, a row that previously did not exist.

Table summarizes the possible violations for the different isolation levels. An entry of "yes" indicates that a violation is possible and an entry of "no" indicates that it is not possible.