ST. XAVIER’S COLLEGE

**Maitighar,Kathmandu**

**(Affiliated to Tribhuvan University)**



**Database Management System**

**Theory Assignment #13**

**Submitted By**

Aashish Raj Shrestha

B.Sc. CSIT

Year II/IV Semester

013BSCIT002

**Submitted To:**

Er. Sanjay Kumar Yadav

Lecturer,

Department of Computer Science

St. Xavier’s College

Maitighar, Kathmandu

**Submitted On:**

5th November, 2015

1. **GRANT and REVOKE Authorizations**

**Authorization** or authorization is the function of specifying access rights to resources related to information security and computer security in general and to access control in particular. More formally, "to **authorize**" is to define an access policy.

Authorization includes primarily two processes:

* Permitting only certain users to access, process, or alter data
* Applying varying limitations on user access or actions. The limitations placed on (or removed from) users can apply to objects, such as schemas, tables, or rows; or to resources, such as time (CPU, connect, or idle times).

A privilege is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include the right to:

* Connect to the database (create a session)
* Create a table
* Select rows from another user's table
* Execute another user's stored procedure

You grant privileges to users so these users can accomplish tasks required for their jobs. You should grant a privilege only to a user who requires that privilege to accomplish the necessary work. Excessive granting of unnecessary privileges can compromise security.

A user can receive a privilege in two different ways:

* You can grant privileges to users explicitly. For example, you can explicitly grant to user SCOTT the privilege to insert records into the employees table.
* You can also grant privileges to a role (a named group of privileges), and then grant the role to one or more users. For example, you can grant the privileges to select, insert, update, and delete records from the employees table to the role named clerk, which in turn you can grant to users scott and brian.

**System Privileges**

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.

There are over 100 distinct system privileges to manage as described in the following subsections:

* Granting and Revoking System Privileges
* Who Can Grant or Revoke System Privileges?

**Granting and Revoking System Privileges**

You can grant or revoke system privileges to users and roles. If you grant system privileges to roles, then you can use the roles to manage system privileges. For example, roles permit privileges to be made selectively available.

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 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 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;

**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.

**GRANT Role**

|  |
| --- |
| GRANT role\_name [, role\_name] ...  TO principal\_specification [, principal\_specification] ...  [ WITH ADMIN OPTION ];   principal\_specification    : USER user    | ROLE role |

Grant one or more roles to other roles or users.

If “WITH ADMIN OPTION” is specified, then the user gets privileges to grant the role to other users/roles.

If the grant statement ends up creating a cycling relationship between roles, the command will fail with an error.

**REVOKE Role**

|  |
| --- |
| REVOKE [ADMIN OPTION FOR] role\_name [, role\_name] ...  FROM principal\_specification [, principal\_specification] ... ;   principal\_specification    : USER user    | ROLE role |

Revokes the membership of the roles from the user/roles in the FROM clause.

1. **Data Encryption**

Database encryption is the process of converting data, within a database, in plain text format into a meaningless cipher text by means of a suitable algorithm.  
Database decryption is converting the meaningless cipher text into the original information using keys generated by the encryption algorithms.  
Database encryption can be provided at the file or column level.

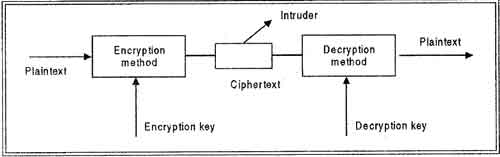
Encryption of a database is costly and requires more storage space than the original data.

The steps in encrypting a database are:

1. Determine the criticality of the need for encryption
2. Determine what data needs to be encrypted
3. Determine which algorithms best suit the encryption standard
4. Determine how the keys will be managed

Numerous algorithms are used for encryption. These algorithms generate keys related to the encrypted data. These keys set a link between the encryption and decryption procedures. The encrypted data can be decrypted only by using these keys.  
Different databases, such as SQL, Oracle, Access and DB2, have unique encryption and decryption methods.

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**

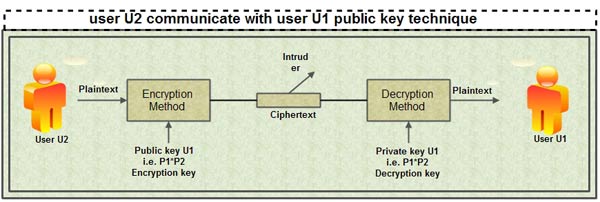
  Another approach to encryption, called public-key encryption, has become increasingly popular in recent years. The encryption scheme proposed by Rivest, Shamir, and Adheman, called RSA, is a well-knm.vnexample of public-key encryption. Each authorized user has a public encryption key, known to everyone 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 0f 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.

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

A functional dependency is defined as a constraint between two sets of attributes in a relation from a database.

Given a relation *R*, a set of attributes *X* in *R* is said to **functionally determine** another attribute *Y*, also in *R*, (written *X* → *Y*) if and only if each *X* value is associated with at most one *Y* value.

In other words, *X* is the *determinant set* and *Y* is the *dependent attribute*. Thus, given a tuple and the values of the attributes in *X*, one can determine the corresponding value of the *Y* attribute.

EXAMPLE:

Employee:

|  |  |  |  |
| --- | --- | --- | --- |
| SSN | Name | JobType | DeptName |
| 557-78-6587 | Lance Smith | Accountant | Salary |
| 214-45-2398 | Lance Smith | Engineer | Product |

Note: Name is functionally dependent on SSN because an employee’s name can be uniquely determined from their SSN. Name does not determine SSN, because more than one employee can have the same name.

Whereas a key is a set of attributes that uniquely identifies an entire tuple, a functional dependency allows us to express constraints that uniquely identify the values of certain attributes.

However, a candidate key is always a determinant, but a determinant doesn’t need to be a key.

Let a relation *R* have some functional dependencies *F* specified. The *closure of F* (usually written as ***F+***) is the set of all functional dependencies that may be logically derived from *F*. Often *F* is the set of most obvious and important functional dependencies and ***F+***, the closure, is the set of all the functional dependencies including *F* and those that can be deduced from *F*. The closure is important and may, for example, be needed in finding one or more candidate keys of the relation.

**Example:**

Student:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SNo | SName | CNo | CName | Addr | Instr. | Office |
| 5425 | Susan Ross | 102 | Calc I | …San Jose, CA | P. Smith | B42 Room 12 |
| 7845 | Dave Turco | 541 | Bio 10 | ...San Diego, CA | L. Talip | B24 Room 210 |

**Axioms:**

*1. Reflexivity Rule* --- If *X* is a set of attributes and *Y* is a subset of *X*, then ***X → Y*** holds. Each subset of *X* is functionally dependent on *X*.

*2.* *Augmentation Rule* --- If ***X → Y*** holds and *W* is a set of attributes, then ***WX → WY*** holds.

*3. Transitivity Rule* --- If ***X → Y*** and ***Y → Z*** hold, then ***X → Z*** holds.

**Derived Theorems:**

4. *Union Rule* --- If ***X → Y*** and ***X → Z*** holds, then ***X → YZ*** holds.

5. *Decomposition Rule* --- If ***X → YZ*** holds, then so do ***X → Y*** and ***X → Z***.

6. *Pseudotransitivity Rule* --- If ***X → Y*** and ***WY → Z*** hold then so does ***WX → Z***.

EXAMPLE:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SNo | SName | CNo | CName | Addr | Instr. | Office |

Based on the rules provided, the following dependencies can be derived.

***(SNo, CNo) → SNo*** (Rule 1) -- subset

***(SNo, CNo) → CNo*** (Rule 1)   
***(SNo, CNo) → (SName, CName)*** (Rule 2) -- augmentation

***CNo → office*** (Rule 3) -- transitivity

***SNo → (SName, address)*** (Union Rule)   
etc.

Using the first rule alone, from our example we have 2^7 = 128 subsets. This will further lead to many more functional dependencies. This defeats the purpose of normalizing relations.

**So what now?**

One way is to deal with one attribute or a set of attributes at a time and find its closure (i.e. all functional dependencies relating to them). The aim of this exercise is to find what attributes depend on a given set of attributes and therefore ought to be together.

**Approach:**

Step 1 Let ***X^c <- X***

Step 2 Let the next dependency be ***A -> B***. If *A* is

in ***X^c*** and *B* is not, ***X^c <- X^c + B***.

Step 3 Continue step 2 until no new attributes can

be added to ***X^c***.

Then The Given Examplewill have,

Consider the following relation: student(SNo, SName, CNo, CName).

We wish to determine the closure of (SNo, CNo). We have the following functional dependencies.

***SNo -> SName***

***CNo -> CName***

* *Step 1* --- ***X^c <- X***, that is, ***X^c <- (SNo, CNo)***   
  *Step 2* --- Consider ***SNo -> SName***, since *SNo* is in ***X^c*** and *SName* is not, we have: ***X^c <- (SNo, CNo) + SName***   
  *Step 3* --- Consider ***CNo -> CName***, since *CNo* is in ***X^c*** and *CName* is not, we have: ***X^c <- (SNo, CNo, SName) + CName***   
  *Step 4* --- Again, consider ***SNo -> SName*** but this does not change ***X^c***.  
  *Step 5* --- Again, consider ***CNo -> CName*** but this does not change ***X^c***.
* Therefore ***X+ = X^c = (SNo, CNo, SName, CName)***.
* This shows that all the attributes in the relation *student (SNo, CNo, SName, CName)* are dependent on *(SNo, CNo)* and therefore *(SNo, CNo)* is a candidate key of the present relation. In this case, it is the only candidate key.

**Normal Form:**

Initially Codd (1972) presented three normal forms (1NF, 2NF and 3NF) all based on functional dependencies among the attributes of a relation. Later Boyce and Codd proposed another normal form called the Boyce-Codd normal form (BCNF). The fourth and fifth normal forms are based on multi-value and join dependencies and were proposed later. The primary objective of normalization is to avoid anomalies.

1. **BCNF and De-composition into BCNF**

**Boyce-Codd Normal Form (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).

Decomposition Algorithm

Algorithm BCNF(R: relation, F: FD set)

Begin

1. Compute F+

2. Result 🡪 {R}

3. While some Ri in Result not in BCNF Do

a. Chose (X🡪Y) in F+ s.t.

(X🡪Y) covered by Ri

X -/-> Ri  ( X not a superkey for Ri )

b. Decompose Ri on (X🡪Y)

Ri1  🡨 X U Y

Ri2 🡨 Ri - Y

c. Result 🡨 Result - {Ri} U { Ri1, Ri2}

4. return Result

End

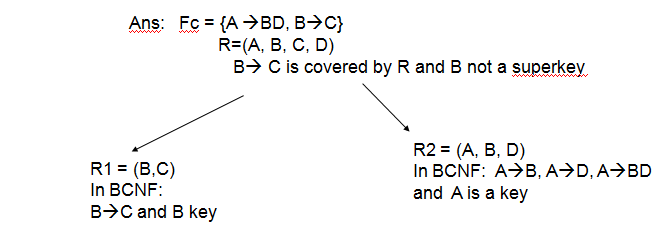
**BCNF Decomposition**

Example:

R= (A, B, C, D)

F = (A🡪B, AB🡪D, B🡪C)

Decompose R into BCNF



1. **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**:

* 1. One where no transaction needs to be rolled back.
  2. 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.

1. **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.

* + 1. 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’.

Being serializable is not the same as being serial

Being serializable implies that the schedule is a correct schedule.

* + It will leave the database in a consistent state.
  + The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.

1. **Transactions supports in SQL.**

The SQL Server drivers support two-phase commit and can fully participate in a distributed transaction when the transaction environment parameter convertAllToDistributed is set to true. You can use SQL Server with its two-phase commit capability both under MTS, and directly through an XA connection. In both cases, Microsoft DTC must be running on the server.  
  
If you are working under MTS, start an OLE transaction. The SQL Server data source is automatically included in the distributed transaction.  
If the connection to the data is through an XA connection, the connection is made automatically. The daemon server mode must be configured to Single-client mode.  
  
To use distributed transactions from an ODBC-based application, ensure that AUTOCOMMIT is set to 0.