ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System Lab Assignment #11**

**SUBMITTED BY:**

**Victor Kumar Sapkota**

**013BSCCSIT047**

**SUBMITTED TO:**

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

**Submission date:** 5 November, 2015

**GRANT AND REVOKE AUTHORIZATIONS**

Authorization is the process of giving someone permission to do or have something. In multi-user computer systems, a system administrator defines for the system which users are allowed access to the system and what privileges of use (such as access to which file directories, hours of access, amount of allocated storage space, and so forth). 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:

* Super user 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 super user 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:

**The GRANT statements:**

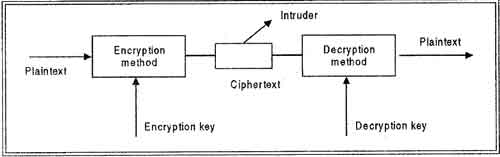
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GRANT (Database) | GRANT (Procedure) | GRANT (Resource Pool) | GRANT (Role) | GRANT (Schema) |
| GRANT (Sequence) | GRANT (Storage Location) | GRANT (Table) | GRANT (User Defined Extension) | GRANT (View) |

**The REVOKE ststements:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| The REVOKE statements | REVOKE (Database) | REVOKE (Procedure) | REVOKE (Resource Pool) | REVOKE (Role) |
| REVOKE (Schema) | REVOKE (Sequence) | REVOKE (Storage Location) | REVOKE (Table) | REVOKE (User Defined Extension) |

**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:** 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.

**TRANSIVITY, REFLEXIVITY AND AUGMENTATION PROPERTIES OF FDs**

As you know there are three Armstrong's Axioms for inferring all the functional dependencies on a relational database. (X, Y and Z are set of attributes)

1. **Reflexivity**: If X ⊆ Y, then Y → X
2. **Augmentation**: If X → Y, then XZ → YZ for any Z
3. **Transitivity**: if X → Y and Y → Z, then X → Z

I understand the augmentation and transitivity for example if we had such schema:

SOME\_SCHEMA(a, b, c, d)

with such functional dependencies:

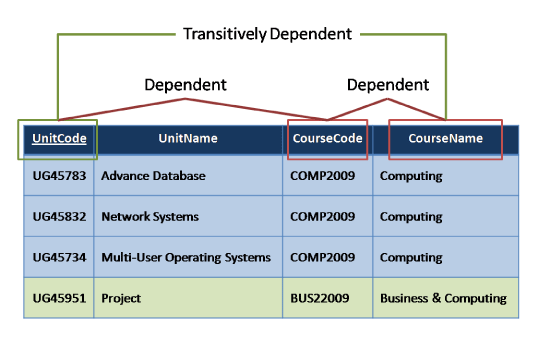
1. a → b
2. b → c

By using augmentation we could get ac → bc or by using transitivity we could get a → c

In Database Management System, a transitive dependency is a [functional dependency](https://en.wikipedia.org/wiki/Functional_dependency) which holds by virtue of [transitivity](https://en.wikipedia.org/wiki/Transitive_relation). A transitive dependency can occur only in a [relation](https://en.wikipedia.org/wiki/Relation_(mathematics)) 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:

1. A → B
2. It is not the case that B → A
3. B → C

Then the functional dependency A → C (which follows from 1 and 3 by the [axiom of transitivity](https://en.wikipedia.org/wiki/Armstrong%27s_axioms#Axioms)) is a transitive dependency.



**BCNF AND DECOMPOSITION INTO BCNF**

A relational schema R is considered to be in Boyce–Codd normal form (BCNF) if, for every one of its dependencies X → Y, one of the following conditions holds true:

* X → Y is a trivial functional dependency (i.e., Y is a subset of X)
* X is a superkey for schema R

Informally the Boyce-Codd normal form is expressed as “Each attribute must represent a fact about the key, the whole key, and nothing but the key.”

**Example:**

Let’s take a look at this table, with some typical data. The table is not in BCNF.

| **Author** | **Nationality** | **Book title** | **Genre** | **Number of pages** |
| --- | --- | --- | --- | --- |
| William Shakespeare | English | The Comedy of Errors | Comedy | 100 |
| Markus Winand | Austrian | SQL Performance Explained | Textbook | 200 |
| Jeffrey Ullman | American | A First Course in Database Systems | Textbook | 500 |
| Jennifer Widom | American | A First Course in Database Systems | Textbook | 500 |

The nontrivial functional dependencies in the table are:

**author → nationality**

**book title → genre, number of pages**

We can easily see that the only [key](http://www.vertabelo.com/blog/on-keys) is the set {author, book title}.

The same data can be stored in a BCNF schema. However, this time we would need three tables.

| **Author** | **Nationality** |
| --- | --- |
| William Shakespeare | English |
| Markus Winand | Austrian |
| Jeffrey Ullman | American |
| Jennifer Widom | American |

| **Book title** | **Genre** | **Number of pages** |
| --- | --- | --- |
| The Comedy of Errors | Comedy | 100 |
| SQL Performance Explained | Textbook | 200 |
| A First Course in Database Systems | Textbook | 500 |

| **Author** | **Book title** |
| --- | --- |
| William Shakespeare | The Comedy of Errors |
| Markus Winand | SQL Performance Explained |
| Jeffrey Ullman | A First Course in Database Systems |
| Jennifer Widom | A First Course in Database Systems |

The functional dependencies for this schema are the same as before:

**author → nationality**

**book title → genre, number of pages**

The key of the first table is {author}. The key of the second table is {book title}. The key of the third table is {author, book title}. There are no functional dependencies violating the BCNF rules, so the schema is in Boyce-Codd normal form.

**Decomposition Example:**

R(A, C, B, D, E) with functional dependencies: A -> B, C -> D

How do I go about decomposing it?

The steps I've taken are:  
A+ = AB  
C+ = CD  
R1 = A+ = **AB**  
R2 = ACDE (since elements of C+ still exist, continue decomposing)  
R3 = C+ = **CD**  
R4 = **ACE** (no FD closures reside in this relation)w

So now I know that ACE will compose the whole relation, but the answer for the decomposition is: AB, CD, ACE.

**CHARACTERIZING SCHEDULES BASED ON RECOVERABILITY AND SERIALIZABILITY**

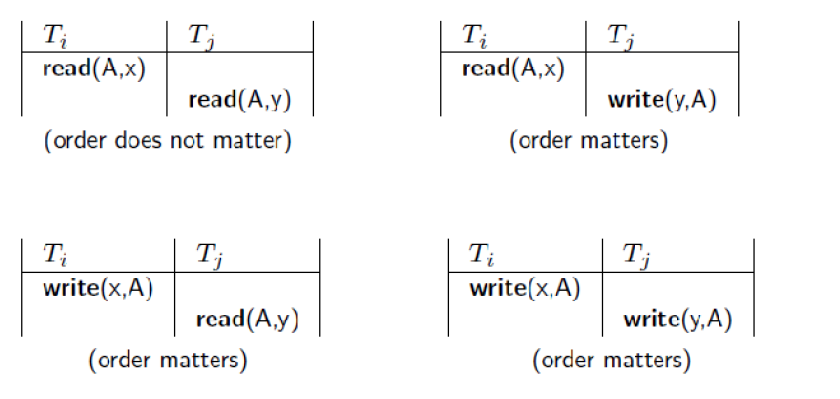
When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms what is known as a transaction schedule (orhistory). A schedule (or history) S of n transactions T1, T2, …, Tn:

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.

**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., 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 aserial 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.

**Atomicity of Transactions**

A transaction might commit after completing all its actions, or it could abort (or be aborted bythe DBMS) after executing some actions. A very important property guaranteed by the DBMS for all transactions is that they are atomic. That is, a user can think of a Xact as always executing all its actions in one step, or not executingany actions at all.\_ DBMS logs all actions so that it can undo the actions of aborted transactions.

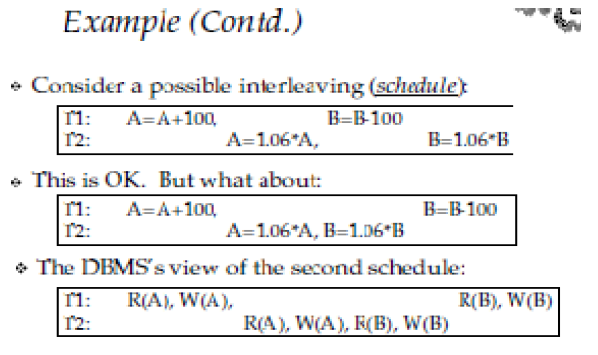
**Example**

Consider two transactions (Xacts):

T1: BEGIN A=A+100, B=B-100 END

T2: BEGIN A=1.06\*A, B=1.06\*B END

Intuitively, the first transaction is transferring $100 from B’s account to A’s account. Thesecond is crediting both accounts with a 6% interest payment.There is no guarantee that T1 will execute before T2 or vice-versa, if both are submittedtogether. However, the net effect must be equivalent to these two transactions running serially in some order.



**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. Nonrepeatable 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.