**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Lab Assignment #13**

**Submitted by:**

Sazjan Neupane

013BSCCSIT036

**Submitted to:**

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

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

**GRANT and REVOKE authorization**

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

**Grant and Revoke 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.

|  |
| --- |
| **Note:**  In general, you grant system privileges only to administrative personnel and application developers. End users normally do not require the associated capabilities. |

Use either of the following to grant or revoke system privileges to users and roles:

* Oracle Enterprise Manager Console
* The SQL statements GRANT and REVOKE

Only users who have been granted a specific system privilege with the ADMIN OPTION or users with the system privileges GRANT ANY PRIVILEGE or GRANT ANY OBJECT PRIVILEGE can grant or revoke system privileges to other users.

**Syntax**

REVOKE [ GRANT OPTION FOR ]

{

[ ALL [ PRIVILEGES ] ]

|

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

}

[ ON [ class :: ] securable ]

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

[ CASCADE] [ AS principal ]

**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.

Encryption is the process of obfuscating data by the use of a key or password. This can make the data useless without the corresponding decryption key or password. Encryption does not solve access control problems. However, it enhances security by limiting data loss even if access controls are bypassed. For example, if the database host computer is misconfigured and a hacker obtains sensitive data, that stolen information might be useless if it is encrypted.

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. 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: 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: