**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Theory Assignment #11**

**Submitted by:**

Dikita Tuladhar  
013BSCCSIT018

**Submitted to:**

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

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

1. **GRANT and REVOKE Authorization**

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.

The following examples show specific table privileges that you can grant to users.

* GRANT SELECT ON DEPT TO PUBLIC;

This statement grants SELECT privileges on the DEPT table. Granting the select privilege to PUBLIC gives the privilege to all users at the current server.

* GRANT UPDATE (EMPNO,DEPT) ON TABLE EMP TO NATZ;

This statement grants UPDATE privileges on columns EMPNO and DEPT in the EMP table to user NATZ.

* GRANT ALL ON TABLE EMP TO KWAN,ALONZO WITH GRANT OPTION;

This statement grants all privileges on the EMP table to users KWAN and ALONZO. The WITH GRANT OPTION clause allows these two users to grant the table privileges to others.

A user with SYSADM or SYSCTRL authority can issue the following statements:

* REVOKE CREATETAB ON DATABASE DB1 FROM PGMR01 BY ALL;

In this statement, the CREATETAB privilege that user PGMR01 holds is revoked regardless of who or how many people explicitly granted this privilege to this user.

* REVOKE CREATETAB, CREATETS ON DATABASE DB1 FROM PGMR01 BY DBUTIL1;

This statement revokes privileges that are granted by DBUTIL1 and leaves intact the same privileges if they were granted by any other ID.

1. **Data Encryption**

Encryption is the process of encoding messages or information in such a way that only authorized parties can read it. Encryption does not of itself prevent interception, but denies the message content to the interceptor. In an encryption scheme, the intended communication information or message, referred to as plaintext, is encrypted using an encryption algorithm, generating cipher text that can only be read if decrypted. For technical reasons, an encryption scheme usually uses a pseudo-random encryption key generated by an algorithm. It is in principle possible to decrypt the message without possessing the key, but, for a well-designed encryption scheme, large computational resources and skill are required. An authorized recipient can easily decrypt the message with the key provided by the originator to recipients, but not to unauthorized interceptors.

The Data Encryption Standard (DES, /ˌdiːˌiːˈɛs/ or /ˈdɛz/) was once a predominant symmetric-key algorithm for the encryption of electronic data. It was highly influential in the advancement of modern cryptography in the academic world. Developed in the early 1970s at IBM and based on an earlier design by Horst Feistel, the algorithm was submitted to the National Bureau of Standards (NBS) following the agency's invitation to propose a candidate for the protection of sensitive, unclassified electronic government data. In 1976, after consultation with the National Security Agency (NSA), the NBS eventually selected a slightly modified version (strengthened against differential cryptanalysis, but weakened against brute force attacks), which was published as an official Federal Information Processing Standard (FIPS) for the United States in 1977. The publication of an NSA-approved encryption standard simultaneously resulted in its quick international adoption and widespread academic scrutiny. Controversies arose out of classified design elements, a relatively short key length of the symmetric-key block cipher design, and the involvement of the NSA, nourishing suspicions about a backdoor. The intense academic scrutiny the algorithm received over time led to the modern understanding of block ciphers and their cryptanalysis.

DES is now considered to be insecure for many applications. This is mainly due to the 56-bit key size being too small; in January 1999, distributed.net and the Electronic Frontier Foundation collaborated to publicly break a DES key in 22 hours and 15 minutes (see chronology). There are also some analytical results which demonstrate theoretical weaknesses in the cipher, although they are infeasible to mount in practice. The algorithm is believed to be practically secure in the form of Triple DES, although there are theoretical attacks. In recent years, the cipher has been superseded by the Advanced Encryption Standard (AES). Furthermore, DES has been withdrawn as a standard by the National Institute of Standards and Technology (formerly the National Bureau of Standards).

Some documentation makes a distinction between DES as a standard and DES as an algorithm, referring to the algorithm as the DEA (Data Encryption Algorithm).

1. **Transitivity, 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*

If A is a set of attributes, and B is a set of attributes that are completely contained in A, the A implies B.

1. **Augmentation**: *If X → Y, then XZ → YZ for any Z*

If A implies B, and C is a set of attributes, then if A implies B, then AC implies BC.

1. **Transitivity**: *if X → Y and Y → Z, then X → Z*

If A implies B and B imply C, then A implies 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 designates three distinct attributes (or distinct collections of attributes) in the relation. Suppose all three of the following conditions hold:

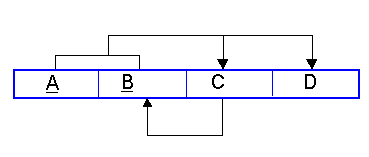
1. A → B
2. It is not the case that B → A
3. B → C

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.

In [database normalization](https://en.wikipedia.org/wiki/Database_normalization), one of the important features of [third normal form](https://en.wikipedia.org/wiki/Third_normal_form) is that it excludes certain types of transitive dependencies. [E.F. Codd](https://en.wikipedia.org/wiki/E.F._Codd), the inventor of the [relational model](https://en.wikipedia.org/wiki/Relational_model), introduced the concepts of transitive dependence and third normal form in 1971.

1. **BCNF and Decomposition into BCNF**

Usually tables that are in Third Normal Form are already in Boyce Codd Normal Form. Boyce Codd Normal Form (BCNF) is considered a special condition of third Normal form. A table is in BCNF if every determinant is a candidate key. A table can be in 3NF but not in BCNF. This occurs when a non key attribute is a determinant of a key attribute. The dependency diagram may look like the one below:



The table is in 3NF. A and B are the keys and C and D depend on both A and B. There are no transitive dependencies existing between the non key attributes, C and D.

The table is not in BCNF because a dependency exists between C and B. In other words if we know the value of C we can determine the value of B.

We can also show the dependencies as:

A B -> C D

C -> B

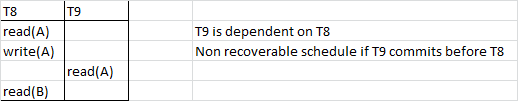
**Converting to BCNF**

The situation is resolved by following the steps below:

|  |  |
| --- | --- |
| 1 | The determinant, Offering#, becomes part of the key and the dependant attribute T\_Code, becomes a non key attribute. So the Dependency diagram is now S\_Num, Offering# -> T\_Code,  Review Date |
| 2 | There are problems with this structure as T\_Code is now dependant on only part of the key. This violates the rules for 2NF, so the table needs to be divided with the partial dependency becoming a new table. The dependencies would then be S\_Num, Offering# ® T\_Code,  Review Date Offering#  -> T\_Code |
| 3 | The original table is divided into two new tables. Each is in 3NF and in BCNF.  StudentReview   |  |  |  | | --- | --- | --- | | S\_Num | Offering# | Review Date | | 123599 | 01764 | 2nd March | | 123599 | 01765 | 12th April | | 123599 | 01789 | 2nd May | | 346700 | 01764 | 3rd March | | 346700 | 01765 | 7th May |   OfferingTeacher   |  |  | | --- | --- | | Offering# | T\_Code | | 01764 | FIT104 | | 01765 | PIT305 | | 01789 | PIT107 | |  |  | |

1. **Characteristics Schedules based on Recoverability**

A recoverable schedule is one where, for each pair of Transaction Ti and Tj such that Tj reads data item previously written by Ti the commit operation of Ti appears before the commit operation Tj.

[](https://sites.google.com/site/projectcodebank/computer-engineering-notes/recoverable-and-cascadeless/Untitled1.png?attredirects=0)

Suppose that the system allows T9 to commit immediately after execution of read (A) instruction. Thus T9 commit before T8 does.

Now suppose that T8 fails before it commits. Since T9 has read the value of data item A written by T8 we must abort T9 to ensure transaction Atomicity.

However, T9 has already committed and cannot be aborted. Thus we have a situation where it is impossible to recover correctly from the failure of T8.

1. **Characteristics Schedules based on Serializability**

In concurrency control of databases, transaction processing (transaction management), and various transactional applications (e.g., transactional memory and software transactional memory), both centralized and distributed, a transaction schedule is serializable if its outcome (e.g., the resulting database state) is equal to the outcome of its transactions executed serially, i.e., sequentially without overlapping in time. Transactions are normally executed concurrently (they overlap), since this is the most efficient way. Serializability is the major correctness criterion for concurrent transactions' executions. It is considered the highest level of isolation between transactions, and plays an essential role in concurrency control. As such it is supported in all general purpose database systems. Strong strict two-phase locking (SS2PL) is a popular serializability mechanism utilized in most of the database systems (in various variants) since their early days in the 1970s.

If executing interleaved transaction results in same outcome as serial schedule(running transaction in some sequnence) then they are considered serializable. this schedule is type of nonserial schedule.

Serializabality might be compromised in some cases but recoverability compromise would mean violating database integrity. Isolation levels decide tradeoff between correctness and concurrency.

1. **Transaction 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. **Non-repeatable read:** A transaction may read a given value from a table. If another transaction later updates then the value is read and 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Table  Possible Violations Based on Isolation Levels as Defined in SQL | | | |
| Type of Violation | | | |
| Isolation level | Dirty read | Nonrepeatable read | Phantom |
| READ UNCOMMITTTED | yes | yes | yes |
| READ COMMITTED | no | yes | yes |
| REPEATABLE READ | no | no | yes |
| SERIALIZABLE | no | no | no |

A sample SQL transaction might look like the following:

EXEC SQL WHENEVER SQLERROR GOTO 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: ...;

The above transaction consists of first inserting a new row in the EMPLOYEE table and then updating the salary of all employees who work in department 2. If an error occurs on any of the SQL statements, the entire transaction is rolled back. This implies that any updated salary (by this transaction) would be restored to its previous value and that the newly inserted row would be removed.

As we have seen, SQL provides a number of transaction-oriented features. The DBA or database programmers can take advantage of these options to try improving transaction performance by relaxing serializability if that is acceptable for their applications.