**ST.XAVIER,S COLLEGE**

**Maitighar, Kathmandu**

**(Affiliated to Tribhuwan University)**



**DATABASE MANAGEMENT SYSTEM**

**THEORY ASSIGNMENT**

**Submitted by:**

Abhishek Tamrakar

013BScCSIT003

2nd year/ 4th semester

**Submitted to:**

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

**GRANT AND REVOKE AUTHORIZATION:**

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](javascript:toggleBlock('14712')) 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](http://my.vertica.com/docs/6.1.x/HTML/index.htm#3046.htm) 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;

**DATA ENCRYPTION:**

**Database Encryption for Safeguarding Sensitive Data**

Encrypting sensitive data in databases has clearly gone beyond optional, and is now a firm requirement. Whether an organization is looking to secure intellectual property, comply with privacy or regulatory mandates, or simply guard the organization’s brand against the damage associated with data breaches, database encryption represents a vital imperative.

By providing database encryption for sensitive data in databases, organizations can establish a strong line of defense that can help secure sensitive assets against a range of threats. However, while the reasons to adopt database encryption are clear, that doesn’t mean the effort is simple. In fact, for many organizations, database encryption has presented a range of obstacles, including degraded database performance, laborious revisions to application code, and complex and time consuming key management efforts.

With Vormetric solutions, you can gain the capabilities you need to encrypt and secure sensitive assets in databases, while avoiding the challenges traditionally associated with database encryption.

**TRANSIVITY, REFLEXITY AND AUGMENTATION PROPERTIES OF FDs:**

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

Reflexivity: If X ⊆ Y, then Y → X

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

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:

a → b

b → c

**BCNF AND DECOMPOSITION INTO BCNF:**

It is a slightly stronger version of the third normal form (3NF). A table is in Boyce-Codd normal form if and only if for every one of its non-trivial [dependencies] X → Y, X is a super key—that is, X is either a candidate key or a superset thereof. Note the above set of tables “Tournament Winners” and “Player Dates of Birth” shown as in 3NF are also in BCNF. Only in rare cases does a 3NF table not meet the requirements of BCNF. A 3NF table which does not have multiple overlapping candidate keys is guaranteed to be in BCNF. An example of a 3NF table that does not meet BCNF is

Today's Court Bookings

Court Start Time End Time Rate Type

|  |
| --- |
| 1 09:30 10:30 SAVER |
|  |
| 1 11:00 12:00 SAVER |
|  |
| 1 14:00 15:30 STANDARD |
|  |
| 2 10:00 11:30 PREMIUM-B |
|  |
| 2 11:30 13:30 PREMIUM-B |
|  |
| 2 15:00 16:30 PREMIUM-A |

There are two courts available and there are four distinct rate types:

* SAVER, for Court 1 bookings made by members
* STANDARD, for Court 1 bookings made by non-members
* PREMIUM-A, for Court 2 bookings made by members
* PREMIUM-B, for Court 2 bookings made by non-members

So, Rate Type → Court is only non-trivial functional dependency that holds.

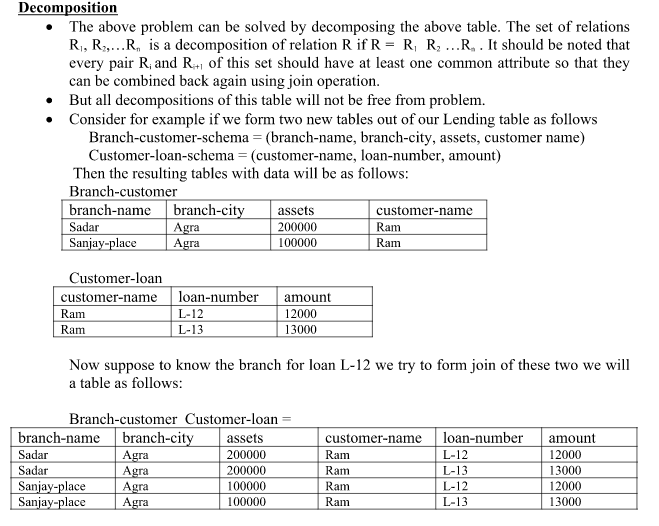
* We can observe that the table's candidate keys are:
* In the Today's Court Bookings table, there are no non-prime attributes: that is, all
*  {Court, Start Time}
*  {Court, End Time}
*  {Rate Type, Start Time}
*  {Rate Type, End Time}

attributes belong to candidate keys. Therefore the table adheres to both 2NF and 3NF Court, the determining attribute (Rate Type) is not a super key.

The table does not adhere to BCNF because in the dependency Rate Type → the design can be amended so that it meets BCNF as follows:

|  |
| --- |
| Rate Types Today’s Bookings |
|  |
| Rate Type Court Member Flag |
|  |
| SAVER 1 Yes |
|  |
| STANDARD 1 No |
|  |
| PREMIUM-A 2 Yes |
|  |
| PREMIUM-B 2 No |
|  |
| Rate Type Start Time End Time |
|  |
| SAVER 09:30 10:30 |
|  |
| SAVER 11:00 12:00 |
|  |
| STANDARD 14:00 15:30 |
|  |
| PREMIUM-B 10:00 11:30 |
|  |
| PREMIUM-B 11:30 13:30 |
|  |
| PREMIUM-A 15:00 16:30 |

The candidate keys for the Rate Types table are {Rate Type} and {Court, Member Flag};the candidate keys for the Today's Bookings table are {Rate Type, Start Time} and {Rate Type, End Time}. Both tables are in BCNF.



**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 inwhich they occur in T1.

Note, however, that operations from other transactions Tj can be interleaved with the operations of Ti in S.

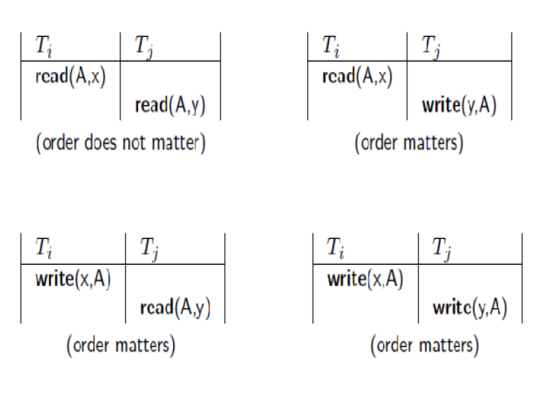
**CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY:**

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.

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 a serial 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.

**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.

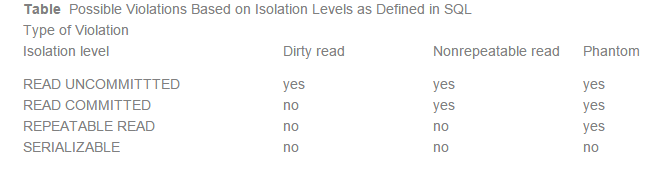
The isolation level option is specified using the statement ISOLATION LEVEL <isolation>, where the value for <isolation> can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE. The default isolation level is SERIALIZABLE, although some systems use as READ COMMITTED their default. The 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. 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.



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.