ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System Theory Assignment #13**

**Submitted by:**

Divya Jyoti Pokharel

013BSCCSIT019

**Submitted to:**

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

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

**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.

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](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**

Encryption is the conversion of electronic [data](http://searchdatamanagement.techtarget.com/definition/data) into another form, called [ciphertext](http://searchcio-midmarket.techtarget.com/definition/ciphertext), which cannot be easily understood by anyone except authorized parties.

Encryption is a great way to keep valuable data safe—whether you’re transmitting it over the Internet, backing it up on a server, or just carrying it through airport security on your laptop. Encrypting your data makes it completely unreadable to anyone but you or its intended recipient.

**TRANSITIVITY, 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.

* **Reflexive rule** − If alpha is a set of attributes and beta is\_subset\_of alpha, then alpha holds beta.
* **Augmentation rule** − If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
* **Transitivity rule** − Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

**BCNF AND DECOMPOSITION INTO BCNF**

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that:

For any non-trivial functional dependency, X → A, X must be a super-key.

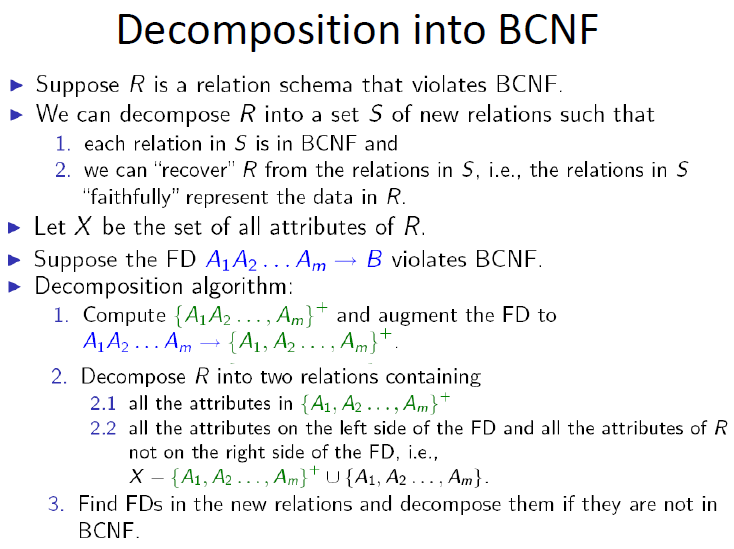
In the above image, Stu\_ID is the super-key in the relation Student\_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu\_ID → Stu\_Name, Zip

and

Zip → City

Which confirms that both the relations are in BCNF.



**CHARACTERIZING SCHEDULES BASED ON RECOVERABILITY**

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.

**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.

**TRANSACTIONS XUPPORTS ON 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 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. No repeatable read: A transaction may read a given value from a table.

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.