BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2102

DATABASE MANAGEMENT SYSTEM

Unit 7

SQL – 2

Table of Contents

1. INTRODUCTION

In the last Unit, SQL was introduced, and depicted that this language is available in a number of database management packages based on the relational model of data. It was also said that SQL is used for data definition,manipulation, and data control for a relational database and illustrated that all the three major facilities of SQL, namely, data manipulation, data definition, and data control are bound together in one integrated language framework.

In this unit, an introduction about the Views is given which are useful for hiding unneeded information and for collecting together information from more thanone relation into a single view. This Unit also introduces Embedded SQL which uses SQL commands within the host language program.

We discuss about the transaction processing which consists of a sequenceof operations and which must appear to be atomic.

1.1 Objectives:

After going through Unit 7, the learners should be able to:

- *Discuss about views and its definition*
- *Explain embedded SQL and Transactions*
- *Discuss the Transaction Processing*

2. VIEWS

A view is a virtual table that does not actually exist. It is made up of a query on other tables in the database. It could include only certain columns or rows from a table or from many tables. A view that restricts the user to certain rows is called a horizontal view, and a vertical view restricts the user to certain columns. You are not restricted to purely horizontal or vertical slices of data.

A view can be as complicated as you like. You can have grouped views, where the query contains a GROUP BY clause. This makes the view a summary of the data in a table or tables.

If the list of column names is omitted the columns in the view take the same name as in the underlying tables. You must specify column names if the query includes calculated columns or two columns with the same name. There are several advantages to views, including

Security: Users can be given access to only the rows/columns of tablesthat concern them. The table may contain data for the whole firm but they only see data for their department.

Data integrity: The WITH CHECK OPTION clause is used to enforce the query conditions on any updates to the view. If the view shows data for a particular office the user can only enter data for that office.

Simplicity: Even if a view is a multi-table query, querying the view still lookslike a single-table query.

Protection from change: If the structure of the database changes, the user's view of the data can remain the same.

There are two disadvantages to views:

Performance: A view may look like a single table but underneath the DBMSis usually still running multi-table queries. If the view is complex then even simple queries can take a long time.

Update restrictions: Updating the data through a view may or may not be possible. If the view is complex, the DBMS may decide it can't perform updates and make the view read-only.

The ISO standard specifies five conditions that a view must meet in order to allow updates:

- The view must not have a DISTINCT clause.
- The view must only name one table in the FROM clause.
- All the columns must be real columns – no expressions, calculated columns, or column functions
- The WHERE clause must not contain a sub-query
- There must be no GROUP BY or HAVING clause

You will find that most dialects of SQL are not quite so restrictive. The underlying principle is that updates are allowed if the rows and columns of the view are traceable back to actual rows and columns in tables.

The format of the view statement is as follows:

create view< view name > as query expression

A view is a relation (virtual other than base) and can be used in query expressions, that is, queries can be written using the view as a relation. Views generally are not stored, since the data in the base relations may change. The base relations on which a view is based are sometimes called the existing relations. The definition of a view in a create view statement is stored in the system catalog. Having been defined, it can be used as if the view really represented a real relation. However, such a virtual relation defined by a view is recomputed whenever a query refers to it.

Self-Assessment Questions – 1

1. A view is a_____table that is one that does not actually exist.
2. Users can be given access to only the rows/columns of_____that concern them.
3. If the structure of the database changes, the user's view of the data _____
4. The view must not have a_____clause.
5. The base relations on which a view is based are sometimes called the _____ relations.

3. EMBEDDED SQL

We have looked at a wide range of SQL query constructs, treating SQL as an independent language in its own right. A relational DBMS supports an *interactive SQL* interface, and users can directly enter SQL commands. Thissimple approach is fine as long as the task at hand can be accomplished entirely with SQL commands. In practice, we often encounter situations in which we need the greater flexibility of a general-purpose programming language, in addition to the data manipulation facilities provided by SQL. For example, we may want to integrate a database application with a nice graphical user interface, or we may want to ask a query that cannot be expressed in SQL.

To deal with such situations, the SQL standard defines how SQL commandscan be executed from within a program in a **host language** such as C or Java. The use of SQL commands within a host language program is called **embedded SQL**. Details of embedded SQL also depend on the host language. Although similar capabilities are supported for a variety of host languages, the syntax sometimes varies.

Conceptually, embedding SQL commands in a host language program is straightforward. SQL statements (i.e., not declarations) can be used wherever a statement in the host language is allowed (with a few restrictions). Of course, SQL statements must be clearly marked so that a preprocessor can deal with them before invoking the compiler for the host language. Also, any host language variables used to pass arguments into an SQL command must be declared in SQL. In particular, some special hostlanguage variables *must* be declared in SQL.

There are, however, two complications to bear in mind. First, the data types recognized by SQL may not be recognized by the host language, and vice versa. This mismatch is typically addressed by casting data values appropriately before passing them to or from SQL commands. (SQL, like C and other programming languages, provides an operator to cast values of one type into values of another type.) The second complication has to do with the fact that SQL is **set-oriented**; commands operate on and produce tables, which are sets (or multisets) of rows. Programming languages do not typically have a data type that corresponds to sets or multisets of rows. Thus, although SQL commands deal with tables, the interface to the host language is constrained to be one row at a time.

In our discussion of embedded SQL, we assume that the host language is Cfor concreteness because minor differences exist in how SQL statements are embedded in different host languages.

3.1 Declaring Variables and Exceptions

SQL statements can refer to variables defined in the host program. Such host-language variables must be prefixed by a colon (:) in SQL statements and must be declared between the commands EXEC SQL BEGIN DECLARE SECTION and EXEC SQL END DECLARE SECTION. The declarations are similar to how they would look in a C program and, as usual in C, are separated by semicolons. For example, we can declare variables *c_sname*, *c_sid*, *c_rating*, and *c_age* (with the initial *c* used as a naming convention to emphasize that these are host language variables) asfollows:

```
EXEC SQL BEGIN DECLARE SECTION
        char c_sname[20];
        long c_sid;
        short c_rating;
        float c_age;
EXEC SQL END DECLARE SECTION
```

The first question that arises is which SQL types correspond to the variousC types since we have just declared a collection of C variables whose values are intended to be read (and possibly set) in an SQL run-timeenvironment when an SQL statement that refers to them is executed. The SQL-92 standard defines such a correspondence between the host language types and SQL types for a number of host languages. In our example *c_sname* has the type CHARACTER(20) when referred to in an SQL statement, *c_sid* has the type INTEGER, *c_rating* has the type SMALLINT, and *c_age* has the type REAL.

An important point to consider is that SQL needs some way to report what went wrong if an error condition arises when executing an SQL statement. The SQL-92 standard recognizes two special variables for reporting errors, SQLCODE and SQLSTATE. SQLCODE is the older of the two and isdefined to return some negative value when an error condition arises, without specifying further just what error a particular negative integer denotes.

SQLSTATE, introduced in the SQL-92 standard for the first time, associatespredefined values with several common error conditions, thereby introducing some uniformity to how errors are reported. One of these two variables *must* be declared. The appropriate C type for SQLCODE is long and the appropriate C type for SQLSTATE is char[6], that is, a character string that is five characters long. (Recall the null-terminator in C strings!) In this unit, we will assume that SQLSTATE is declared.

3.2 Embedding SQL Statements

All SQL statements that are embedded within a host program must be clearly marked, with the details dependent on the host language; in C, SQL statements must be pre-fixed by EXEC SQL. An SQL statement can essentially appear in any place in the host language program, where a host language statement can appear.

As a simple example, the following embedded SQL statement inserts a row,whose column values are based on the values of the host language variables contained in it, into the Sailors relation:

EXEC SQL INSERT INTO Sailors VALUES (:*c_sname*, :*c_sid*, :*c_rating*, :*c_age*);

Observe that a semicolon terminates the command, as per the convention for terminating statements in C.

The SQLSTATE variable should be checked for errors and exceptions after each embedded SQL statement. SQL provides the WHENEVER command to simplify this tedious task:

EXEC SQL WHENEVER [SQLERROR / NOT FOUND] [CONTINUE | GOTO *stmt*]

The intent is that after each embedded SQL statement is executed, the value of SQLSTATE should be checked. If SQLERROR is specified and the value of SQLSTATE indicates an exception, control is transferred to *stmt*, which is presumably responsible for error/exception handling. Control is alsotransferred to *stmt* if NOT FOUND is specified and the value of SQLSTATE is 02000, which denotes NO DATA.

Self-Assessment Questions – 2

6. The use of SQL commands within a host language program is called _____ .
7. The_____variable should be checked for errors andexceptions after each embedded SQL statement.

4. TRANSACTION PROCESSING

A user writes data access/update programs in terms of the high-level query and update language supported by the DBMS. To understand how the DBMS handles such requests, with respect to concurrency control andrecovery, it is convenient to regard an execution of a user program, or **transaction**, as a series of **reads** and **writes** of database objects:
To read a database object, it is first brought into main memory (specifically, some frame in the buffer pool) from the disk, and then its value is copied into a program variable.

To write a database object, an in-memory copy of the object is first modified and then written to disk.

Database 'objects' are the units in which programs read or write information.The units could be pages, records, and so on, but this is dependent on the DBMS and is not central to the principles underlying concurrency control or recovery. In this unit, we will consider a database to be a fi*xed* collection of *independent* objects. When objects are added to or deleted from a database, or there are relationships between database objects that wewant to exploit for performance, some additional issues arise.

There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures:

1. Users should be able to regard the execution of each transaction as **atomic**: either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions (say, when a system crash occurs).
2. Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of the database. This property is called **consistency**, and the

DBMS assumes that it holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.

3. Users should be able to understand a transaction without consideringthe effect of other concurrently executing transactions, even if the DBMSinterleaves the actions of several transactions for performance reasons. This property is sometimes referred to as **isolation**: Transactions are isolated, or protected, from the effects of concurrently scheduling other transactions.
4. Once the DBMS informs the user that a transaction has been successfully completed, its effects should persist even if the system crashes before all its changes are reflected on the disk. This property is called **durability**.

The acronym ACID is sometimes used to refer to the four properties of transactions that we have presented here: atomicity, consistency, isolation, and durability. We now consider how each of these properties is ensured in a DBMS.

4.1 Consistency and Isolation

Users are responsible for ensuring transaction consistency. That is, the userwho submits a transaction must ensure that when run to completion by itselfagainst a 'consistent' database instance, the transaction will leave the database in a `consistent' state. For example, the user may (naturally!) havethe consistency criterion that fund transfers between bank accounts should not change the total amount of money in the accounts. To transfer money from one account to another, a transaction must debit one account, temporarily leaving the database inconsistent in a global sense, even though the new account balance may satisfy any integrity constraints with respect to the range of acceptable account balances. The user's notion of a consistent database is preserved when the second account is credited with the transferred amount. If a faulty transfer program always credits the second account with one dollar less than the amount debited from the first account, the DBMS cannot be expected to detect inconsistencies due to such errors in the user program's logic.

The isolation property is ensured by guaranteeing that even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions one after the other in some serial order. For example, if two transactions $T1$ and $T2$ are executed concurrently, the net effect is guaranteed to be equivalent to executing (all of) $T1$ followed

by executing *T*2 or executing *T*2 followed by executing *T*1. (The DBMS provides no guarantees about which of these orders is effectively chosen.) Ifeach transaction maps a consistent database instance to another consistent database instance, executing several transactions one after the other (on a consistent initial database instance) will also result in a consistent final database instance.

Database Consistency is the property in that every transaction sees a consistent database instance. Database consistency follows from transaction atomicity, isolation, and transaction consistency. Next, we discuss how atomicity and durability are guaranteed in a DBMS.

4.2 Atomicity and Durability

Transactions can be incomplete for three kinds of reasons. First, a transaction can be **aborted**, or terminated unsuccessfully, by the DBMS because some anomaly arises during the execution. If a transaction is aborted by the DBMS for some internal reason, it is automatically restarted and executed anew. Second, the system may crash (e.g., because the power supply is interrupted) while one or more transactions are in progress. Third, a transaction may encounter an unexpected situation (for example, read an unexpected data value or be unable to access some disk) and decide to abort (i.e., terminate itself).

Of course, since users think of transactions as being atomic, a transaction that is interrupted in the middle may leave the database in an inconsistent state. Thus a DBMS must find a way to remove the effects of partial transactions from the database, that is, it must ensure transaction atomicity: either all of a transaction's actions are carried out, or none are. A DBMS ensures transaction atomicity by *undoing* the actions of incomplete transactions. This means that users can ignore incomplete transactions in thinking about how the database is modified by transactions over time. Tobe able to do this, the DBMS maintains a record, called the *log*, of all, writes to the database.

The log is also used to ensure durability. If the system crashes before the changes made by a completed transaction are written to disk, the log isused to remember and restore these changes when the system restarts.

SELF ASSESSMENT QUESTIONS – 3

8. To write a database object, an in-memory copy of the object is first _____ and then written to disk.
9. Each transaction, run by itself with no concurrent execution of other transactions, must preserve the_____of the database.
10. The acronym ACID is sometimes used to refer to the_____of transactions.
11. _____are responsible for ensuring transaction consistency.
12. Transactions can be incomplete for_____kinds of reasons.
13. The_____is also used to ensure durability.

5. SUMMARY

In this unit, we dealt with the SQL data definition language used to create relations with specified schemas. We discussed Views, embedded SQL, and transaction processing properties. The SQL DDL supports a number of types including date and time types. SQL queries can be invoked from host languages, via embedded SQL.

6. TERMINAL QUESTIONS

1. What is View? With an example, use the format of view statement tocreate view.
2. What is an Embedded SQL statement? Describe briefly.
3. Explain any two important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures.

7. ANSWERS

Self-Assessment Questions

1. Virtual
2. Tables
3. can remain the same.
4. DISTINCT
5. Existing
6. embedded SQL
7. SQLSTATE
8. Modified
9. Consistency
10. four properties
11. Users
12. Three
13. Log

Terminal Questions

1. A view is a virtual table which does not actually exist. It is made up of a query on other tables in the database. It could include only certaincolumns or rows from a table or from many tables. (Refer section 7.2 for detail)
2. The use of SQL commands within a host language program is called embedded SQL. (Refer section 7.3 for detail)
3. There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures. They are (i) Atomicity (ii) Consistency (iii) Isolation (iv) Durability. (Refer section 7.4 for detail)