Chapter 8 - SQL

Background/History

- SQL ("Structured Query Language") is the concrete manifestation of the concepts in relational algebra. The original version, known as SEQUEL (for "Structured English QUEry Language") was developed at IBM Research.
- In 1986, ANSI and ISO jointly defined the SQL-86, or SQL1, standard. A revised and expanded standard emerged in 1992, called SQL2 (or SQL-92). SQL3 is in preparation. Among other things, it will incorporate object-oriented concepts.
- In the realm of relational databases, SQL is the predominant standard. The existence of such a standard tends to make it easier to migrate from one DBMS to another: as long as both support the standard, migration should not be terribly difficult.
- SQL is a comprehensive database language that includes statements for data definition, query, and update (making it both a DDL and DML).
- In addition, SQL has facilities for

    - defining views on databases
    - specifying security and authorization
    - defining integrity constraints
    - specifying transaction controls

Creating Tables

SQL CREATE TABLE Statement Syntax

```
CREATE TABLE <table_name> (
<column_name1> <datatype1> <constraint1>
<column_name2> <datatype2> <constraint2>
<constraint-list>
)
```

Ex:-

```
CREATE TABLE Emp_tab (
   Empno      NUMBER(5) PRIMARY KEY,
   Ename      VARCHAR2(15) NOT NULL,
   Job        VARCHAR2(10),
   Mgr        NUMBER(5),
   Hiredate   DATE DEFAULT (sysdate),
   Sal        NUMBER(7,2),
   Comm       NUMBER (7,2),
   Deptno     NUMBER(3) NOT NULL,
              CONSTRAINT dept_afkey REFERENCES Dept_tab(Deptno));
```

Altering Tables

What is the SQL ALTER TABLE Statement?

The SQL ALTER TABLE statement is the SQL command that makes changes to the definition of SQL table.

Why Use the SQL ALTER TABLE Statement?

Alter a table in an Oracle database for any of the following reasons:

- To add one or more new columns to the table
- To add one or more integrity constraints to a table
- To modify an existing column's definition (datatype, length, default value, and NOT NULL integrity constraint)
- To enable or disable integrity constraints associated with the table
- To drop integrity constraints associated with the table
- To drop a column from the table

When altering the column definitions of a table, you can only increase the length of an existing column, unless the table has no records. You can also decrease the length of a column in an empty table. For columns of datatype CHAR, increasing the length of a column might be a time consuming operation that requires substantial additional storage, especially if the table contains many rows. This is because the CHAR value in each row must be blank-padded to satisfy the new column length.

If you change the datatype (for example, from VARCHAR2 to CHAR), then the data in the column does not change. However, the length of new CHAR columns might change, due to blank-padding requirements.

How To Use the SQL ALTER TABLE Statement

SQL ALTER TABLE Statement Syntax

```
ALTER TABLE <table_name>
ADD <column_name1> <datatype1> <constraint1>

ALTER TABLE <table_name>
ADD  CONSTRAINT  <constraint_name1>  <constraint1>

 ALTER TABLE <table_name>
MODIFY <column_name1> <datatype1> <constraint1>

 ALTER TABLE <table_name>
DROP COLUMN <column_name1>
```

```
ALTER TABLE <table_name>
DROP  CONSTRAINT  <constraint_name1>
```

Dropping Tables

How To Use the SQL DROP TABLE Statement

SQL DROP TABLE Statement Syntax

```
DROP  TABLE <table_name>
```

If the table that you are dropping contains any primary or unique keys referenced by foreign keys of other tables, and if you intend to drop the FOREIGN KEY constraints of the child tables, then include the CASCADE option in the DROP TABLE command. For example:

DROP TABLE Emp_tab CASCADE CONSTRAINTS;

Dropping a table has the following effects:

- The table definition is removed from the data dictionary. All rows of the table are then inaccessible.
- All indexes and triggers associated with the table are dropped.
- All views and PL/SQL program units that depend on a dropped table remain, but become invalid (not usable).
- All synonyms for a dropped table remain, but return an error when used.
- All extents allocated for a non-clustered table that is dropped are returned to the free space of the table space and can be used by any other object requiring new extents.
- All rows corresponding to a clustered table are deleted from the blocks of the cluster.
- If the table is a master table for snapshots, then Oracle does not drop the snapshots, but does drop the snapshot log. The snapshots can still be used, but they cannot be refreshed unless the table is re-created.

Inserting Data into a Table

The syntax for inserting data into a table one row at a time is as follows:

```
INSERT INTO  <table_name>(column1 name, column2 name , ...)
VALUES (value1 , value2 , ...)
```

Assuming that we have a table that has the following structure,

Table ***Store_Information***

Column Name	Data Type
store_name	char(50)
Sales	float
Date	datetime

And now we wish to insert one additional row into the table representing the sales data for Los Angeles on January 10, 1999. On that day, this store had $900 in sales. We will hence use the following SQL script:

```
INSERT INTO Store_Information (store_name, Sales, Date)
VALUES ('Los Angeles', 900, 'Jan-10-1999')
```

Deleting Data from a Table

Sometimes we may wish to get rid of records from a table. To do so, we can use the **DELETE FROM** command. The syntax for this is

```
DELETE FROM  <table_name> WHERE  {condition}
```

It is easiest to use an example. Say we currently have a table as below:

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999

Boston	$700	Jan-08-1999

and we decide not to keep any information on Los Angeles in this table. To accomplish this, we type the following SQL statement:

```
DELETE FROM Store_Information
WHERE store_name = 'Los Angeles'
```

Now the content of table would look like,

Table ***Store_Information***

store_name	Sales	Date
San Diego	$250	Jan-07-1999
Boston	$700	Jan-08-1999

Modifying Data in a Table

Once there's data in the table, we might find that there is a need to modify the data. To do so, we can use the **UPDATE** command. The syntax for this is

```
UPDATE <table_name> SET "column_1" = [new value]
WHERE {condition}
```

For example, say we currently have a table as below:

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

and we notice that the sales for Los Angeles on 01/08/1999 is actually $500 instead of $300, and that particular entry needs to be updated. To do so, we use the following SQL:

```
UPDATE Store_Information
SET Sales = 500
WHERE store_name = 'Los Angeles'
AND Date = 'Jan-08-1999'
```

The resulting table would look like

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$500	Jan-08-1999
Boston	$700	Jan-08-1999

In this case, there is only one row that satisfies the condition in the **WHERE** clause. If there are multiple rows that satisfy the condition, all of them will be modified.

It is also possible to **UPDATE** multiple columns at the same time. The syntax in this case would look like the following:

```
UPDATE "table_name"
SET column_1 = [value1], column_2 = [value2]
WHERE {condition}
```

Retrieving Data from a Table

What do we use SQL commands for? A common use is to select data from the tables located in a database. Immediately, we see two keywords: we need to **SELECT** information **FROM** a table.

Hence we have the most basic SQL structure:

```
SELECT <attribute> FROM  <table_name>
```

To illustrate the above example, assume that we have the following table:

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

We shall use this table as an example throughout the tutorial (this table will appear in all sections). To select all the stores in this table, we key in,

SELECT store_name **FROM** Store_Information

Result:

store_name
Los Angeles
San Diego
Los Angeles
Boston

Multiple column names can be selected, as well as multiple table names.

A query in SQL can consist of up to six clauses, but only the first two- SELECT and FROM- are mandatory. The clauses are specified in the following order:

```
SELECT < attribute and function list >
FROM  < table list >
[WHERE < condition > ]
[GROUP BY < grouping attribute(s) > ]
[HAVING < group condition >]
[ORDER BY < attribute list >];
```

Refer Retrieval Queries of SQL in text book.

Views (Virtual Tables) in SQL

A **view** is a single table that is derived from other tables. These other tables could be based tables or previously defined views. A view is considered a **virtual table**.
E.g. For queries with the employee name and project name instead of specifying every time the join of EMPLOYEE, WORKS_ON and PROJECT tables, we define a view which is a result of these joins. The above tables are called **defining tables** of a view.

To create a View

```
V1:   CREATE VIEW    WORKS_ON1
      AS   SELECT    FNAME, LNAME, PNAME, HOURS
           FROM      EMPLOYEE, PROJECT, WORKS_ON
           WHERE     SSN=ESSN AND PNO=PNUMBER;
```

A query on a view:

```
      QV1: SELECT    FNAME, LNAME
      FROM           WORKS_ON1
      WHERE          PNAME='ProjectX';
```

WORKS_ON1

FNAME	LNAME	PNAME	HOURS

```
V2:   CREATE VIEW    DEPT_INFO(DEPT_NAME, NO_OF_EMPS,
TOTAL_SAL)
      AS   SELECT    DNAME, COUNT(*), SUM(SALARY)
           FROM      DEPARTMENT, EMPLOYEE
           WHERE     DNUMBER=DNO
           GROUP BY  DNAME;
```

DEPT_INFO

DEPT_NAME	NO_OF_EMPS	TOTAL_SAL

To dispose of a view

```
      V1A: DROP VIEW  WORKS_ON1;
```

A view is not realized at the time it is defined but rather at the time we specify a query on the view. Therefore the view automatically reflects all the changes in the base tables. A view is always *up to date.*

View Implementation

There are two approaches to **implement** a view for querying:

Query modification modifies the view query into a query on the underlying tables. It is inefficient for complex and multiple queries.

View materialization physically creates a view table when the view is first queried, and keeps the table for the following queries. Must have an automatic update of the view table when the underlying tables are modified (mostly *incremental update*). If the view is not queried for a certain period of time, the system removes the table, and recomputes it from scratch when queried again.

View Update

Update is complicated and can be ambiguous.

Update on a view defined on a single table (without aggregate functions) can be mapped on the underlying base table.

For a view involving joins, the mapping can be done in multiple ways, most likely causing ambiguity.

E.g. In the WORKS_ON view update the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

UV1: **UPDATE** WORKS_ON1
SET PNAME='ProductY'
WHERE LNAME='Smith' **AND** FNAME='John' **AND** PNAME='ProductX';

Consider two possible updates:

(a): **UPDATE** WORKS_ON **SET** PNO =
(**SELECT** PNUMBER **FROM** PROJECT **WHERE** PNAME='ProductY')
WHERE
ESSN **IN**
(**SELECT** SSN **FROM** EMPLOYEE **WHERE** LNAME='Smith' **AND** FNAME='John')
AND
PNO **IN** (**SELECT** PNUMBER **FROM** PROJECT **WHERE** PNAME='ProductX');

(b): **UPDATE** PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';

The (b) interpretation accomplishes as well the desired *view update*; however it doesn't reflect the user's intention.

Some view updates do not make sense, for example:

UV2: **UPDATE** DEPT_INFO
SET TOTAL_SAL=100000
WHERE DNAME='Research';

A large number of different updates on the underlying base tables can satisfy this view update.

In summary:

- A view with a single defining table is updateable if the view attributes contain the primary key of the basis relation because this maps each virtual view tuple to a single base tuple
- Views defined on multiple tables using joins are generally no updateable
- Views defined using grouping and aggregate functions are not updateable

Additional features of SQL

- Specifying general constrains by declarative assertions

The CREATE ASSERTION statement:

CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
(SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
WHERE E.SALARY>M.SALARY **AND** E.DNO=D.DNUMBER **AND**
D.MGRSSN=M.SSN));

The DBMS is responsible for ensuring that the condition is not violated.

- Granting and revoking of privileges to users

Each table is assigned an owner, and either the owner or the DBA can grant to use statements such as SELECT, INSERT, DELETE, or UPDATE.
In addition, the DBA can grant the privileges to create schema, tables, or views to user.
The commands are GRANT and REVOKE.

- Embedding SQL statements in programming languages

These are language bindings to various languages such as C, C++, COBOL, or PASCAL.SQL.

- Transaction control commands

These are functionalities for concurrency control and recovery control. Such as shared (read)and exclusive (write) locks, dealing with a deadlock problem by timeouts and deadlock detection for concurrency. UNDO and REDO recovery operations, checkpointing, caching, in-place updating and shadowing controls.

- Storage Definition Language commands

These are set of commands for specifying physical database design parameters, file structures for tables, and access paths such as indexes.