Chapter 4 SQL

SQL (Structured Query Language) is the most common language used to communicate with relational database management system. It was first originated in at the IBM Research Laboratory in San Jose, California. The project name was SYSTEM R whose purpose was to validate the feasibility of the implementation of relational model in database management system. The name of the language used in System R was 'sequel which was later replaced to SQL. After the popularity of system R project, a number of commercial RDBMS products were released, like SQL/DS of IBM, which was the first commercial relational database management system product. Other products included ORACLE from ORACLE Corp, INGRES from Relational Technology (1981), SYBASE from SYBASE Inc. (1986) and SQL Server from MICROSOFT.

To provide the directions for the development of RDBMS, American National Standard Institute (ANSI) and the International Standard Organization (ISO) approved a standard for the SQL. The official name of this language is International Standard Database Language SQL (1992). The latest version of SQL is referred to as SQL/92 or SQL2 further extensions are now being considered and are expected to result in SQL3.

Most of the RDBMS vendors support SQL92 version of SQL with a little bit of modification to extend the standard to increase their commercial market. In this chapter, we will follow SQL92 most of the time. However these features are implemented by using Oracle, as it is the most popular and most widely used RDBMS commercially.

As discussed earlier, SQL is used to communicate with any RDBMS package, its primary function is to support the definition, manipulation and management of data in relational database. It provides the syntax and semantics of data definition and manipulation of data. It also provides an interface for portability of database definition and application programs. The following are some advantages and disadvantages of SQL in brief.

(a) Non-procedural in nature
SQL is a non procedural or declarative language which means that in order to obtain a particular result, the programmer does not need to specify how to perform the operations or the steps for the operations that the computer needs to carry out. A programmer just has to specify WHAT is needed.

(b) Interactive and embedded SQL
SQL can be used **interactively** which allows the user to issue commands directly to the database management system. Result is received back as soon as the statement is produced. On the other hand, in embedded SQL, the SQL commands can be embedded inside a program written in high level languages like C, C++, Java etc.. The embedded SQL provides the additional programming features, which are not there in SQL language. Any SQL command that can be used interactively can also be used in embedded form with some minor changes.

(c) Common Language for All Relational Databases

Because all major relational database management systems support SQL, you can transfer all skills you have gained with SQL from one database to another. In addition, since all programs written in SQL are portable, they can often be moved from one database to another with very little modification.

(d) Easy to learn
SQL is termed as English like language and is very easy to learn and adopt.

(e) SQL operations work for a set of rows rather than record-by-record. It makes SQL more powerful than the other languages like COBOL.

The commands in SQL are divided into three sub languages, first is known as DDL (Data Definition Language) which includes the commands used to define or modify the database objects like tables, objects, index etc. The second sub language is DML (Data Manipulation Language), which includes statements that allow the processing or manipulation of database objects. Last is the Data Control Language which is having DBA related commands. These commands are very straightforward and easy to implement.

Following are the three categories or sub languages under SQL:

(a) Data Definition Language
(b) Data Manipulation Language
(c) Data Control Language

The brief description of all these sub languages is given in next section. Before going ahead, we will discuss the general structure of SQL statements. When writing SQL statement, it is useful to follow certain guidelines and rules to improve the readability of the statement and to facility their editing, if necessary. SQL statement are not case sensitive, but remember, data may be case sensitive depending upon the database. SQL statements end with a semicolon (;) and can be written in one or more line.

(a) Data Definition Language
Data definition language is used to define the database objects. These objects include tables, index and views etc. We can also moodily and drop these objects by using DDL commands.

Creating Table:- A table can be created in the Database by using CREATE TABLE command. The basic structure of this command is given:

```
CREATE  TABLE <tablename>
                    (Column1     datatype          [constraint],
                     Column2     datatype          [constraint],
                    :
                    :
```

```
Column n datatype [constraint],
[Constraint]);
```

Where

tablename is the name of the table, to be created. In most of the cases, table name can be from 1 to 30 characters long. It must begin with a alphabet, rest of the characters can be any combination of letters, digits or the underscore (_). It must not be a SQL reserved word. For example employees is a valid name which 3student is invalid as its first character is a digit.

Column1, Column2 …. Column n are the attributes of the table. Same rules as of table also apply for column names. The name of the columns should be chosen so that one item of data is stored for each record.

Datatype is the type, which a column will use to store the data. The main data types of ORACLE are listed in Table 1

DATATYPE	**DESCRIPTION**
CHAR	To store alphanumeric type of data of fixed length. Maximum value is 2000 characters. Default value is 1.
VARCHAR2	Similar to CHAR but can store variable number of characters. Maximum length is 4000 characters. No default value is allowed.
VARCHAR	Synonymous with VARCHAR2 however likely to change in future versions of Oracle. It is suggested to use VARCHAR2.
NUMBER	Stores fixed and floating point number with maximum precision and scale in 38 digits.
DATE	Stores point-in-time values (i.e. date and time) in a table. The default format is DD-MON-YY.
LONG	Can store up to 2 GB of characters. Only one long column per table is allowed.
RAW	Used to store binary data such as graphics, sound, animation etc. Maximum permissible data is of 2000 bytes.
LONG RAW	Contains raw binary data otherwise the same as LONG column. The values entered must be in hex notation. Maximum limit is 2 GB.
CLOB	Character Large Objects. Max data size is 4GB.
BLOB	Binary Large Objects. Max data size is 4GB.
NCLOB	National Language Support character Large Objects
BFILE	External Binary File. Maximum Limit is 4 GB.

Table 1. Data types available in ORACLE

Constraint are the conditions which can be attached to a column(s). It works like a flag. Whenever user attempts to load a value in that column, this value is checked against the data constraint. These constraints can either be placed at the column level or at the table level. Column level constraints can be applied to any one column at a time. They are local to that a specific column. The table level constraint are applied when the

constraint definition contains more than one column. These constraints are stored as a part of the global table definition. The general syntax for table level constraint is

CONSTRAINT <constraint_name> <type_constraint> (column)

Where

constraint_name is the name of the constraint.
type_constraint is any constraint listed below and column is the name of the attribute on which constraint is applied.

The major data constraints available in most of the RDBMS packages are given below.

(i) **NULL:** If a row lacks a data value for a particular column, that value is said to be NULL. Column of any data type may contain NULL unless the column was defined as NOT NULL, when table was created. Setting a NULL is appropriate when the actual value is unknown or the known value is meaningless. In Oracle all columns are by default NULL.

(ii) **NOT NULL:** If a column is defined as NOT NULL, then it becomes a mandatory column i.e. the user has to enter the data into it. It can take duplicate values if it is not a unique column but cannot take Nulls.

For example if CREATE TABLE command includes

…. Sal NUMBER (7) NOT NULL….

Then it means that Sal is a mandatory column and it needs a valid value corresponding to every record.

(iii) **UNIQUE:** It prevents duplicate values being entered in the column. A table may have more than one unique keys. An example of this key is

…., Phone NUMBER (7) UNIQUE,….

Here the column *Phone* is a number type of field with unique constraint, means that no two records in this table can have the same phone number. The syntax for having this constraint at table level will be

…, Phone NUMBER (7), CONSTRAINT uq_phone UNIQUE(Phone)…

where *uq_phone* is the name of the constraint which is of unique type and is applied on Phone column.

(iv) **CHECK:** This constraint is used to enforce some rules that can be evaluated based on a logical expression. For example we want that for a student database, the Roll_no column should contain the values starting from ‘a’ like

a101, a102 etc. or name column should be accepted only in upper case then these types of conditions can be imposed by using CHECK constraint. An example of check constraint is given here

… Roll_no VARCHAR2(4) CHECK (Roll_no LIKE 'a%')

where LIKE is a SQL operator which is used for pattern matching. The same constraint at table level will have the syntax:

… Roll_no VARCHAR2(4), CONSTRAINT chk_roll CHECK (Roll_no LIKE 'a%')

where chk_roll is the name of the constraint.

(v) **PRIMARY KEY:** A primary key is a one or more column used to uniquely identify each row in the table. These values should never be changed and should never be NULL. Through this key, rows in the table are individually identified. NULLs are not allowed in the Primary key column. Only one primary key per table is allowed however composite primary key can be created involving multiple attributes. An example of primary key syntax is given here

… Empno NUMBER (4) PRIMARY KEY,…

This constraint can also be applied at table level as follows.

…Empno NUMBER(4), CONSTRAINT pk_empno PRIMARY KEY (Empno)

where pk_empno is the name of the constraint which is of primary key type and is applicable on Empno column.

(vi) **FOREIGN KEY:** A foreign key is a column or combination of columns in one base table whose values are required to match some specific primary key value in some other base table. A foreign key must have a corresponding primary key value in the primary key table to have a meaning. A foreign key value can be left null. It rejects an INSERT or UPDATE of a value, if a corresponding value does not currently exist in the primary key table. It also rejects DELETE from the primary table if corresponding records exists in foreign key table. An example of creating foreign key constraint is given here

… Deptno NUMBER(2) REFERENCES DEPT (Deptno)….

In this example, we are adding a new column Deptno with datatype NUMBER (2) as a foreign key with respect to the column Deptno of DEPT table. Remember the Deptno column of dept table must be a primary key. The syntax at table level will be

… Deptno NUMBER(2), CONSTRAINT fk_deptno FOREIGN KEY (Deptno)

REFERENCES dept (Deptno)….

Where fk_deptno is the name of the foreign key constraint.

(vii) DEFAULT: We can assign a default value to the columns. Now when a row is inserted in the table then all the corresponding columns will take their values as specified. If some column is not having a value then in place of NULL it will store the default value.

An example of creating table is given below:

```
CREATE TABLE EMP
  (
        Empno NUMBER (4) PRIMARY KEY,
        Ename VARCHAR2(20),
        Job VARCHAR(20),
        Mgr NUMBER(4),
        Sal NUMBER(4),
        Comm NUMBER(4),
        Hiredate DATE,
        Deptno NUMBER(2),
    CONSTRAINT chk_ename CHECK (Ename=UPPER (Ename)),
    CONSTRAINT chk_sal CHECK (Sal>Comm),
    CONSTRAINT fk_deptno FOREIGN KEY (Deptno)
                                    REFERENCES DEPT (Deptno)
                        )
```

Modifying a Table:- Modifying a table means to change the definition of the table. This can be done by using ALTER TABLE command. This command may have one of the following clauses.

ADD| MODIFY| DROP| ENABLE| DISABLE

The general syntax for the ALTER TABLE is as follows

```
ALTER TABLE <table name>  [ADD| MODIFY| DROP| ENABLE| DISABLE] (Constraint| | column specification)
```

The brief description of all these clauses is given here.

ADD Clause: The ADD clause is used to add a column and/or constraints to an existing table. To add a column say Part_full_time to EMP table, the syntax will be as follows:

```
ALTER TABLE EMP ADD (Part_full_time CHAR(1));
```

This will add a column, Part_full_time to the EMP table, which will accept CHAR type of data and the maximum width for that data is 1 character. It means that this column will work as a flag which keeps the information about the status of the employee whether he is working as a full time employee or part time employee. If this table is having some existing data corresponding to other columns then it will take NULL for this column corresponding to those records. If we are adding a NOT NULL or Primary Key constraint along with the column description and the table is having some data then command will not be executed successfully and will give an error message. The following syntax will be executed only in that case if table is empty.

```
ALTER TABLE EMP ADD (Part_full_time CHAR (1) NOT NULL);
```

MODIFY Clause: This clause is used to modify the column specifications and the constraints. In case of constraints, only possibilities are to modify a NULL to NOT NULL and NOT NULL to NULL. Other constraints, which are to be modified, should be first deleted and then recreated with the modification.

Suppose we want to increase the width of a column, syntax is

```
ALTER TABLE EMP MODIFY (Sal NUMBER(5));
```

It will increase the width of commission column from NUMBER(4) to NUMBER(5). However, there is one restriction for decreasing the width of a column or changing the data type of the column and that is, the column must not contain any data. If the column is having some data corresponding to existing records then we cannot decrease the size or change the data type. But we can increase or decrease the number of decimal places in a NUMBER column at any time. For modification in the constraint, the syntax is

```
ALTER TABLE EMP MODIFY(Sal NUMBER(5) NOT NULL);
```

Now Sal column will become NOT NULL column (again it must not contain any NULL value).

DROP clause: We can remove a column from table directly by using the DROP clause. For example

```
ALTER TABLE EMP DROP COLUMN Part_full_time;
```

This will drop the Part_full_time column from the table along with the data.

We can also drop the constraints by using the DROP clause in the ALTER TABLE command. To drop a constraint, the different syntaxes are

```
ALTER TABLE EMP DROP CONSTRAINT chk_sal;
```

It will drop the constraint chk_sal (check(Sal>Comm) from the EMP table. Similarly to drop a primary key constraint we can use

```
ALTER TABLE EMP DROP pk_empno;
```

The above constraint can also be deleted by using the syntax

```
ALTER TABLE EMP DROP PRIMARY KEY;
```

In this case there is no need to specify the name of the primary key constraint as there can be only one primary key constraint in a single table. Now consider one more example of dropping constraint.

```
ALTER TABLE EMP DROP PRIMARY KEY CASCADE;
```

This command will drop all the dependencies of the Primary Key (i.e. the Foreign Key constraints in different table, which are based on this Primary Key) and then will drop this Primary key in a single step.

Dropping a table:- An existing table can be dropped from the database at any time by using the DROP TABLE Command. The syntax is

```
DROP TABLE EMP;
```

The above syntax on execution will drop the EMP table. To drop a table in ORACLE, one must either own the table or have been granted the DROP ANY TABLE permission dropping a table will also cause associated objects such as indexes and privileges etc. to be dropped.

Data Manipulation Language (DML)

Data Manipulation Language contains the commands, which are used to manipulate the data in the tables. These commands include INSERT, UPDATE, DELET, and SELECT. The brief discussion on all these commands is given here:

INSERT Command

This command is used to insert rows into table. On its simplest form, this command allows the user to add rows to a table one row at a table. The basic syntax of this command is as given:

```
INSERT INTO <tablename> (Column1, Column2,…. Column n )
                                VALUES (Value1, Value2…., Value n)
```

For example

```
INSERT  INTO EMP(Empno, Ename, Job, Mgr, Sal, Comm, Hiredate, Deptno)
                                VALUES
                        (1,'AKSHITA','Manager',9000,500,'27-Mar-2006',10);
```

This statement will add a new record in the EMP table and will be added at the end of table according the relational database property.

When data is not to be entered into every column in the table them either enter NULL corresponding to all those columns which do not require the value or specify only those columns which require a value for example the following two INSERT statement will insert the customize record into EMP table.

```
INSERT INTO EMP(Empno, Ename, Job, Mgr, Sal, Comm, Hiredate, Deptno )
                                        VALUES
                        ( 2,'Sparsh',NULL,NULL,NULL,NULL,NULL, NULL);
```

One more style for INSERT command is without specifying column names as given in the following:

```
INSERT INTO EMP VALUES     (            );
```

In this case, values must be in the same sequence as specified at the table creation time.

Changing Table Contents

To change the value of a column or a group of columns in a table corresponding to some search criteria UPDATE command is used. The general syntax of this command is as:

```
UPDATE <tablename>  SET <columnname> = <newvalue>
                                        [, column= newvalue] WHERE <search criteria>-
```

for example to update the salary of King , the following statement will be used:

```
UPDATE EMP SET Sal = 6000 WHERE Ename= 'King';
```

Multiple columns can be changed in a simple update statement for example.

```
UPDATE EMP SET Sal=6000, Deptno=10 where Ename = 'King'
```

In absence of WHERE clause, all the records will be updated with the same value.

Deleting records from the table

Records from the table can be deleted individually or in groups by using DELETE Command. The general syntax is

```
DELETE FROM  <tablename> WHERE <search condition>
```

For example:

```
DELETE FROM EMP WHERE Deptno=10;
```

The syntax will delete all the records from Deptno 10. In absence of WHERE Clause, this syntax will delete all the records from the table.

```
DELETE FROM EMP ;
```

The sub-queries can be also be used in DELETE Command. For example, if we have to delete all the records from Accounts Department then the syntax will be as follows:

```
DELETE FROM EMP where Deptno = (SELECT Deptno from DEPT
        WHERE Dname= 'Accounts')
```

Retrieving the information
SELECT statement allows to query the data contained in the tables.

The most basic SELECT statement is the statement which includes only the mandatory clauses i.e. SELECT and FROM clause. It retrieves the complete information of the table without having any restriction on rows for example, the syntax

```
SELECT * FROM EMP;
```

It will give the complete EMP table and refers to all fields of the table. We can restrict it to selected fields also. For example:-

```
SELECT Ename, Sal, Deptno FROM EMP;
```

It will give just three fields i. ENAME, SAL and DEPTNO.

Eliminating Duplicates:- The above written SELECT statement given all the records from the database. These statements do not eliminate the duplicates from the result (of any) DISTINCT clause is used to eliminate duplicates from the result. The statement having DISTINCT clauses as:

```
SELECT DISTINCT Deptno FROM EMP;
```

This statement gives all the distinct department numbers from EMP table as there may be many duplicates Deptno in the table.

Retrieving data of selected columns and selected rows
The syntax is as follows:

```
SELECT <column names> FROM <tablename> [WHERE condition];
```

For example:

```
SELECT Ename, Desig FROM EMP WHERE Deptno=10;
```

The above syntax will give the name and designation of all the employees from department number 10. The WHERE clause expects three elements- a column name, a comparison operator and content or the list of values to be compared against the column values from the table. One more example is given below, which is using the expressions.

```
SELECT Ename, Sal*12 ANNUAL_SAL FROM EMP;
```

It uses the concept of *alias.* ANNUAL_SAL is an alias for expression Sal*12 . In output the alias name will be displayed in place of expression Sal*12.

ORDER BY clause is used for sorting the data in either ascending or descending order depending on the condition specified in the select statement. In increasing order, lowest numeric values, earliest date and alphabetical characters will be displayed first.

```
SELECT * FROM <tablename> ORDER BY <columnname> [sort order]
```

For example

```
SELECT * FROM EMP ORDER BY Sal;
```

It will give the data in the increasing order of salary, all records having null values in the salary column will be displayed in the last. For getting the data in decreasing order of salary, we use DESC at the end of the above syntax.

```
SELECT * FROM EMP ORDER BY Sal DESC;
```

For getting data in descending order of Deptno and ascending order of Sal, we will use the following syntax:

```
SELECT * FROM EMP ORDER BY Deptno DESC, Sal ASC;
```

Preventing Duplicate Rows

To see only the unique rows, we have to use the DISTINCT clause along with the column name(s) for which the uniqueness is required. The example is given below

```
SELECT DISTINCT Deptno FROM EMP;
```

It will return all the department numbers without any repeat ion, though all the departments may have multiple occurrences in the table.

The concatenation Operator (||)

It allows columns to be linked to others columns, arithmetic expressions or constant values to create a character expression. Columns on either side of the operator are combined to make one single column. e.g.

```
SELECT Ename||‘ is  ’||Job||‘. ’ FROM EMP;
```

ENAME||'is'||JOB||'.'
AKSHITA is Manager.

Logical Operators: We can use AND, OR and NOT logical operators in SQL statements. For example:

```
SELECT * FROM EMP WHERE Deptno=10 AND Designation='Manager';
```

It will display all the information of Managers from department number 10.

SQL Operators

BETWEEN …AND…: To retrieve the data within a specified range of values, the BETWEEN operator is used. For example

```
SELECT * FROM EMP WHERE Sal BETWEEN 5000 AND 10000;
```

The above SELECT statement retrieves the information of those employees who are getting salary in between 5000 and 10000 (inclusive of both the values).

IN: If we have to compare a list of values against a column, then we have to use IN operator. For example

```
SELECT * FROM EMP WHERE Deptno IN (10,20)
```

This syntax will give the information of all those employees who are either in department number 10 or 20.

Like: This operator is applicable in case of character columns. It allows for a comparison of one string value with another string value which is not identical. We use two wildcards (% and _) for this purpose. The different forms of like are given as:

```
SELECT * FROM emp WHERE ename LIKE 'A%';
```

It gives the details of those employees whose name starts from character A.

```
SELECT * FROM EMP WHERE Ename LIKE '%h';
```

It returns the rows in which the Ename column ends with character 'h'

```
SELECT * FROM EMP WHERE Ename LIKE '%r%';
```

It will display the information about those employees who include 'r' in their names.

```
SELECT * FROM EMP WHERE Ename LIKE 'Ta__';
```

It will display the information of those employees where length of name is just four characters and first two letters are 'T' and 'a' respectively.
IS NULL: If we want to display all those employees for whom salary is not yet assigned. The probable query may be

```
SELECT Ename FROM EMP WHERE Sal=NULL;
```

But unfortunately, it does not return any row in spite of having NULL values in sal column. When we compare any salary to NULL like whether 5000=NULL or 3000=NULL or even NULL=NULL, it always return false because nothing can be equal to null not even null. Null is unknown value, thus returning no record. To overcome this problem we have SQL operator ***IS NULL***, which is used as follows:

```
SELECT Ename FROM EMP WHERE Sal IS NULL;
```

Functions
Functions make the basic query block more powerful, and are used to manipulate data values. Functions accept one or more arguments and return one value. An argument is a user supplied constant, variable or column reference, which can be passed to a function in the following format:

Function_name(arg1, arg2…)

Functions can be used to

- Perform calculations on data.
- Modify individual data item
- Manipulate output for group of rows
- Alter date formats for display
- Convert column data types.

Some functions operate on single row only; other manipulates group of rows. Single row functions act on each row returned in the query and return one result per row. It expects one or more user arguments and can be used anywhere in user variables, columns or expressions e.g. select, where, order by clause etc.

SINGLE ROW FUNCTIONS

- Act on each row returned in the query.
- Return one result per row
- Expect one or more user arguments
- May be nested.

Single row functions can be further grouped together by the data type of their arguments and return values. Single row functions can of following types:

Character Functions, Number Functions, Date Functions and Conversion function that accept any data type as input.

In the following section, we shall use a table called *dual* for any calculation or expression. *Dual* is a dummy table in Oracle owned by the user **system** and may be accessible to all users. It contains one column, *DUMMY* and one row with value 'X;. The *dual* table is useful when we want to retrieve the value of a constant, pseudo column or expression that is not derived from a table.

(a) ***Character Functions:*** Single row character functions accept character data as input and can return both character and number values.

LOWER (***col/value***): Converts a string into lower cases.
UPPER (***col/value***): Converts a string into upper cases.
INITCAP (***col/value***): Converts the string with first character of the word into upper case and rest in lower cases.

An example in which all three functions are being used is given below:

```
SELECT LOWER ('SPARSH') LOWER, UPPER ('sparSH') UPPER, INITCAP
('JOHN AbraHam') INITIAL_CAP FROM DUAL;
```

The output is as follows:

```
LOWER       UPPER          INITIAL_CAP
----------  -----------    ------------
sparsh      SPARSH         John Abraham
```

LPAD (***col/value,n,'string'***): Pads the column or literal values from left, to a total width of n character positions. The leading positions are filled with 'string'. If string is omitted, value is padded with spaces. An example is given below:

```
SELECT LPAD (Dname, 15, '*') dept1, LPAD (Dname,15) dname2 FROM DEPT;
```

RPAD (***col/value,n,'string'***) :- Pads the columns or literal values from right, to a total width of n character positions. The leading positions are filled with 'string'. If string is omitted, value is padded with spaces.

```
SELECT RPAD (Dname, 15, '*') dept1, RPAD (Dname,15)dname2 FROM DEPT;
```

SUBSTR (***col/value, position, n***):- Returns a string of n characters long from the column or literal value, starting at the position number *position*. If n is omitted, string is extracted from *position* to the end.

```
SELECT SUBSTR (Dname, 1, 5) DEPTT FROM DEPT;
```

INSTR (*col/value, 'string'*):- Finds the character position of first occurrence of string

INSTR (col/value,'string',pos, n): Finds the character position of nth occurrence of string in column or literal values starting at the position number *pos.*

```
SELECT dname, INSTR (Dname, 'y') POS1, INSTR (Dname, 's',1, 2) POS2 FROM DEPT;
```

Here INSTR (Dname, 'y') returns the position of first occurrence of y while INSTR (Dname,'s', 1,2) returns the position of 2nd occurrence of **s** starting from 1st character. As 'Chemistry' does not have 2nd 's' so it returns 0.

LTRIM(***col/value,'string'***):- Returns string after removing characters from its left side up to the first character not in the set.

RTRIM(***col/value,'string'***):- Returns string after removing characters from its right side up to the first character not in the set.

```
SELECT LTRIM ('xxxXxxJOHN','x') RES1, RTRIM ('JOHNxxXxxx','x') RES2 FROM DUAL;
```

REPLACE (***char,search string,replacement string***):- Returns char with every occurrence of search string replaced with replacement string. If replacement string is omitted or null, all occurrence of search string are removed. If search string is null, char is returned. It allows to substitute one string for another as well as to remove character strings.

```
SELECT REPLACE ('JACK AND JUE','J','BL') CHANGE  FROM DUAL;
```

ASCII(*character*) :- It gives the corresponding ASCII value for a given character.
CHR(*number*) :- It gives the character to the corresponding ASCII value.
UID :- Returns an integer that uniquely identifies the current user.
USER :- Returns the current user name.

An example of all the above written function is given below

```
SELECT ASCII ('B'), CHR(75), USER, UID FROM DUAL;
```

NUMBER FUNCTIONS:- These functions are used with numeric data. The argument to the function is number. The various number functions are discussed below.

ROUND (*col/val, n*):- This function rounds the value to n decimal place. If n is (-)ve, the no. to left of decimal are rounded. If n is omitted then it rounds the value and returns integer value.

```
SELECT ROUND (45.923,1), ROUND (45.923), ROUND (45.923, -1) FROM dual;
```

TRUNC(*col/val, n*):- Truncates the value to n decimal places.

```
SELECT TRUNC (45.923,1), TRUNC (45.923), TRUNC(45.923,-1) FROM dual;
```

CEIL(*col/val*):- Finds smallest integer >= to the column/ expression or value.

```
SELECT CEIL(11.9), CEIL(11.1), CEIL(11) FROM dual;
```

FLOOR (*col/val*):- Finds largest integer <= column or expression or value.

```
SELECT FLOOR (11.9), FLOOR (11.1), FLOOR (11) FROM dual;
```

POWER(*cal/val, n*):- Calculates the power n of col/value, where n is an integer.

SQRT(*col/val*) :- Calculates the square root of the col/value.

EXP(*n*) :- Calculates the value of e raised to the power of n.

SIGN(*col/val*) :- Finds the sign of the col/value according to the following rule.

-1 → (-)ve
0 → zero
+1 → (+)ve

i.e. If it returns –1 then it means that number is (-)ve . If 0 then number is zero and if it returns 1 then number is positive.

ABS(*col/val*) :- Finds the absolute value of the col/value.

MOD(*value1,value2*) :- Returns remainder after dividing value1 by value2.

The syntax for all these functions is given in the following example.

```
SELECT POWER(2,4), SQRT(36),EXP(2), SIGN(-44), ABS(-44), MOD(7,3) FROM DUAL;
```

DATE FUNCTIONS:- Date functions are used to manipulate and extract values from the date column of a table. Following functions are used with dates:

SYSDATE:- SYSDATE is a pseudo column that contains the current date and time. It requires no arguments when selected and returns the current date.

```
SELECT SYSDATE FROM DUAL;
```

ADD_MONTHS(***date, count***):- Adds count months to the date. If (-)ve value of count is used then it subtracts it from date.

```
SELECT SYSDATE, ADD_MONTHS (SYSDATE,2) ADDITION, ADD_MONTHS (SYSDATE,-2) SUBTRACTION FROM DUAL;
```

LAST_DAY(***date***) :- Returns the date of the last day of the month containing date.

```
SELECT SYSDATE, LAST_DAY(SYSDATE) FROM dual;
```

MONTHS_BETWEEN(***date2,date1***):- Calculates the number of months between two dates (date2-date1).

```
SELECT SYSDATE, MONTHS_BETWEEN(SYSDATE,'01-JAN-02') DIFFERENCE FROM DUAL;
```

NEXT_DAY(***date,'day'***) :- Returns the date of the next day after date having the *day*.

```
SELECT NEXT_DAY('22-AUG-97', 'SUNDAY') FROM DUAL;
```

NEW_TIMES('d','z1','z2') :- It gives the date and time in other zone z2 when date and time in this zone z1 are d.

```
SELECT NEW_TIME ('22-AUG-97','GMT','AST') FROM DUAL;
```

Where z1 and z2 can be anyone of AST (Atlantic standard time), BST (Bering standard time), CST (Central Standard Time), EST(Eastern Standard Time), GMT (Greenwich Mean Time), HST (Alaska Hawaii Standard time), PST (Pacific Standard time).

CONVERSION FUNCTION:- Converts one data type to another. These are three conversion functions, the brief description of which is given below:

TO_CHAR(*input, format*): Converts a date or number into character string.

```
SELECT TO_CHAR(SYSDATE,'ddth/mm/yyyy') FROM DUAL;

SELECT TO_CHAR(320) || 'New Mandi' FROM DUAL;
```

TO_DATE(***date, format***):- Converts any date format to the default format(dd-mon-yy).

```
SELECT TO_DATE('1997,25th/JUL','yyyy,ddth/mon') FROM DUAL;
```

TO_NUMBER(***col/value***):- Converts a string to a number.

```
SELECT TO_NUMBER(SUBSTR('320, New Mandi',1,3)) H_NO FROM DUAL;
```

It uses the nesting of functions, first of all SUBSTR function is applied to the date which extracts a string containing first three characters of the input data that is 320. Since it is a string data, so TO_NUMBER function converts it to the number.

GROUP FUNCTIONS:- Group functions operate on set of rows, result is based on group of rows rather than one result per row as returned by single row functions. By default all the rows in a table are treated as one group. The *group by* clause of the *select* statement is used to divide rows into small groups. The main group functions available in oracle are

(i) MAX(*expr*) :- Maximum value of *expr*
(ii) MIN(*expr*) :- Minimum value of *expr*
(iii)COUNT(*) :- Counts the number of records in the table.
COUNT(*expr*) :- Counts the number of values in *expr* except NULL.
(iv)SUM(*expr*) :- Sum values of *expr*, ignoring NULL values.
(v) AVG (*expr*) :- Average value of *expr*, ignoring NULL values.
(vi)STDDEV :- Standard deviation of *expr*, ignoring NULL values.
(vii) VARIANCE :- variance of *expr*, ignoring NULL values.

An example of group functions is given below.

```
SELECT MAX (Sal), MIN (Sal), COUNT (Sal), AVG (Sal), SUM (Sal), COUNT
  (*) FROM EMP;
```

MAX (SAL)	MIN (SAL)	COUNT (SAL)	AVG (SAL)	SUM (SAL)	COUNT (*)
8000	4000	6	5500	33000	7

All group functions except count(*) ignore null value.

Count (*) counts the number of records in a table, While count (sal) counts the number of known salaries (ignores NULL).

Group By and Having clauses:- These are parallel to where and order by clauses except that they act on groups rather than on individual rows.

If the objective is to find out the maximum salary of each department then the query will be

SELECT Deptno, MAX (Sal) FROM EMP GROUP BY Deptno;

DEPTNO	MAX(SAL)
10	8000
20	6000

Here in this example, we mixed the column name Deptno and group function MAX. This mix is only possible because Deptno column is referenced in *group by* clause, without which it will display error message. This error is avoided by using Deptno in *group by* clause, which forces the MAX to calculate maximum salary grouped for each department.

HAVING clause is very much similar to *WHERE* clause except that having clause is applicable only for grouped data. Here the rows are further restricted according to the having clause.

e.g. If we want to find out the maximum salary for all departments having more than three employees, then the query will be-

```
SELECT Deptno, MAX (Sal) FROM EMP
        GROUP BY Deptno HAVING COUNT(*) >3;
```

```
   DEPTNO          MAX(SAL)
----------      ------------
     10              8000
```

In this case, deptno 20 is having only 3 employees so it will not be included in the result set. We can also use the order by clause with *group by* & *HAVING* clauses to arrange the records in any specific order.

```
SELECT Deptno, MAX (Sal) FROM EMP
        GROUP BY Deptno HAVING COUNT (*) >2
                ORDER BY MAX(Sal);
```

```
    DEPTNO          MAX(SAL)
    ----------   ------------
       20             6000
       10             8000
```

RETRIEVING THE DATA FROM MORE THAN ONE TABLE

Suppose we want to retrieve the employee names along with their designations and corresponding department names. For this we have to take in consideration two tables, namely emp for ename and desig and dept for dname. For this type of query we have to use the joining of two tables based on a common column in both the tables. In this section we will study the Cartesian product and the various type of joins.

Cartesian Product:- The meaning of Cartesian product is explained through the following example: Suppose there are two sets A={a, b}, B={c, d}, the Cartesian product A X B={(a, c),(a, d),(b, c),(b, d)} where X denotes the Cartesian product.

From above example, in Cartesian product every element of set A is combined with every element of set B to form resultant elements of Cartesian product. We can perform this Cartesian product on tables also. For example the following command:

```
SELECT D.Dname, E.Ename, E.Designation FROM DEPT D, EMP E;
```

would give the Cartesian product. It will take a department from *dept* table and will associate it with all the employees from emp table, thus giving the result. If the table are

EMPNO	ENAME	DESIGNATION	SALARY	DEPTNO
1001	Axay	CEO	8000	10
1002	Ashish	Manager	5000	10
1003	Sparsh	Manager	6000	20
1004	Meenal	Sales Person	4000	20
1005	Tanu	Sales Person	4500	20
1006	Rahul	Manager		10
1007	Vivek	Admin Officer	5500	10

EMP

DEPTNO	DNAME	LOCATION
1	Physics	Delhi
2	Chemistry	Bombay

DEPT

Then the output of the Cartesian product will be as shown in the following table.

DNAME	ENAME	DESIG
Physics	Axay	CEO
Physics	Ashish	Manager
Physics	Sparsh	Manager
Physics	Meenal	Sales Person
Physics	Tanu	Sales Person
Physics	Rahul	Manager
Physics	Vivek	Admin Officer
Chemistry	Axay	CEO
Chemistry	Ashish	Manager
Chemistry	Sparsh	Manager
Chemistry	Meenal	Sales Person
Chemistry	Tanu	Sales Person
Chemistry	Rahul	Manager
Chemistry	Vivek	Admin Officer

Result of DEPT X EMP

Thus if there are m records in table1 and n records in table2 then after cartesian product we will get m*n records as an output. But not all the records are relevant. Our requirement is to display the department along with all the employees working in that

department. For this purpose we will have to use the joining of the tables. We have the following types of joining.

- Inner-join
- Outer Join
- Self Join

Inner Join:- Inner Join finds the exact match of the common column in both the tables. If match is found then that information will be displayed, otherwise not. The criteria for that is DEPT.Deptno=EMP.Deptno

An example of inner join is given as follows:

```
SELECT D.Dname, E.Ename, E.Desig FROM DEPT D, EMP E
                                    WHERE D.Deptno=E.Deptno;
```

The WHERE clause specifies the join condition based on the common attribute Deptno. If a department exists in both the tables then it displays the desired information (dname, ename and salary). For example if in a particular department, no employee exists then it will not even display that department name.

It is clear from the above examples that, in order to join two tables we need a join condition. Similarly, to join three tables we need two join conditions and so on. A simple rule is

The minimum number of join conditions= The number of tables used minus one

Outer Join: In inner join or equi join, a matching common column value is checked in both the tables for equality. In outer join, it picks all the records from one table and take only the matching records from another table, displaying NULL for non-matching values. There are several variations in outer join. In ***Left outer join***, the syntax is as follows:

```
SELECT D.Dname, E.Ename, E.Desig FROM DEPT D Natural Left Outer Join EMP E;
```

This syntax will give all rows from the dept table and the matching rows from the emp table i.e. if there are some departments where no employee exists in emp table even then it will display that department name, leaving other information NULL (ename and desig). In ***Right outer join***, it will display all the employees from emp table and the matching departments from dept table. For example, if there are some employees for whom the department is not assigned even then it will display the name and designation of those employees, leaving the corresponding dname column NULL.

In Full outer join, both the emp and dept rows (with match and without match) will be displayed.

Note: In Oracle, the syntax for outer join is as follows

```
SELECT D.Dname, E.Ename, E.Designation FROM Dept D, EMP E
                WHERE D.Deptno=E.Deptno(+);
```

It uses a (+) sign for outer join. In the above example, it will take all the departments from dept table and take only the matching employees from emp table.

Self Join:- Self join means the joining of a table to itself. Here, two rows from the same table are joined to form a resulting row. To join a table to itself, two copies of the same table have to be opened in the memory. Here we use concept of alias for both the copies of the same table, which will open two identical tables in different memory locations. An example of self join is given below.

Retrieve the employee names and their manager names from the following table.

EMPNO	ENAME	MGRNO
1001	Axay	1003
1002	Sparsh	1003
1003	Vivek	1004
1004	Abhay	-

The query for the above said problem will be

```
SELECT E1.Ename Ename, E2.Ename Mgrname FROM EMP E1, EMP E2
                WHERE E1.Mgrno=E2.Empno;
```

In this query, e1 and e2 are the two aliases of emp table. After opening e1 and e2 in memory locations, it will compare manager number of each employee of e1 to employee numbers of e2 and then will give the desired result as

ENAME	MGRNAME
Axay	Vivek
Sparsh	Vivek
Vivek	Abhay
Abhay	-

Sub Queries:- This is a very powerful and flexible feature of SQL. Through sub queries, we can embed a query within an update, delete or select statements. Sub queries are useful where the desired output is based on some query or when we do not have the predefined list of values. Some examples of sub queries are as given below.

Example1:- Delete all those employees who are in 'Physics' department.
If we are not using the sub query , we will have to follow the two steps. First we will query the deptno of 'Physics' department, as department name is not in emp table. In second step, we will perform the delete operation based on that department number. But if we are using the sub query, we can delete all those records in a single step. The syntax is as

```
DELETE FROM EMP WHERE
        Deptno = (SELECT Deptno FROM DEPT WHERE
Dname='Physics');
```

Firstly, it will execute the SELECT statement and then the output of this Statement will used as an input for the DELETE operation, thus doing both the things in one syntax.

Example2: Update the salary of 'Axay' to the highest salary the company.

```
UPDATE EMP SET Sal=(SELECT MAX(Sal) FROM Emp)
        WHERE Ename='Axay';
```

Example 3: Retrieve the second highest salary.

```
SELECT MAX (Sal) FROM EMP WHERE
        Sal <> (SELECT MAX(Sal) FROM EMP);
```

Example 4: Get the name of the employees who are getting the second highest salary.

```
SELECT Ename FROM EMP WHERE
        Sal=(SELECT MAX (Sal) FROM EMP WHERE
                Sal <> (SELECT MAX (Sal) FROM EMP));
```

Example 5: List all the employees from Physics department.

```
SELECT Ename FROM EMP WHERE
        Deptno=(SELECT Deptno FROM Dept WHERE
                Dname='Physics');
```

Example 6: Retrieve the name of all the employees who work in Mr. Axay's department and getting the same salary as Mr. Ashish is getting in Chemistry department.

```
SELECT Ename FROM EMP WHERE
        Deptno = (SELECT Deptno FROM EMP WHERE Ename='Axay')
                AND
        Sal=(SELECT Sal FROM EMP WHERE Ename='Ashish'
                AND
        Deptno=(SELECT Deptno FROM DEPT WHERE Dname='Chemistry'));
```

CORRELATED QUERY:- In normal sub-query, SELECT statements can be nested in the WHERE clause of the outer query. Correlated sub query is a nested sub query, which is executed once for each row considered by the main query and which on execution

uses a candidate row. This causes the correlated sub query to value from a column in the outer query be processed in a different way from the ordinary nested sub query. For example:

```
SELECT Ename, Sal, Designation, Deptno FROM EMP E WHERE
        Sal>(SELECT AVG (Sal) FROM EMP WHERE
                Deptno=E.Deptno) ORDER BY Deptno;
```

The above query gives the details of employees who earn a salary greater than the average salary for their department. But this query has an unusual thing and that is E.Deptno column of outer query used in the inner query. So why it is executing? The reason is that it is a correlated sub query. In correlated sub query, a sub query may refer to a column in a table used in its main query.

In the above example, it takes a row from emp table for outer query and compare the sal of this row against value returned by inner query. Inner query uses the deptno of the outer query to get the average salary of that department. If the candidate row (salary from outer query) meet the condition (Sal>AVG(Sal) of that department) then it displays the information, otherwise not.

The correlated sub query is one way of reading every row in the table, and comparing values in each row against related data. The inner SELECT statement is executed once for each candidate row.

EXISTS: It is used for testing of existence, whether a sub query returns any row or not. If it returns at least one row, it is TRUE, otherwise false. It is typically used only with a correlated sub query. An example of EXISTS clause is given below.

```
SELECT Ename FROM EMP E WHERE
        EXISTS (SELECT * FROM DEPT WHERE
                EMP.Mgrno=E.Empno);
```

It finds the employees who have at least one person reporting to them.

SET OPERATORS:- Suppose a company has two divisions, technical and accounts. In our data base, we have different tables for each division, described below. Now we want to combine the information from these two tables. For that, we have to use one of the set operators, which are described as follows.

EMPNO	ENAME	SAL
1001	Axay	5000
1002	Sparsh	7000
1003	Vivek	9000
1004	Abhay	9500

EMP_ACCOUNTS

EMPNO	ENAME	SAL
2001	Arun	3000
2002	Anuj	6000
2003	Manoj	2500
2004	Sparsh	4500

EMP_TECHNICAL

Union: If we want to display the combine list of employees from accounts as well as from technical divisions then we will have to use the UNION operator as follows:

```
SELECT  Ename FROM EMP_ACCOUNTS
        UNION
SELECT Ename FROM EMP_TECHNICAL;
```

Union gives all the records from both the queries after eliminating the duplicate records.

Intersect:- If we want to know all those employees who are working in both the departments then we will have to use the intersect operator. The syntax is as follows:

```
SELECT  Ename FROM EMP_ACCOUNTS
        INTERSECT
SELECT Ename FROM EMP_TECHNICAL;
```

Intersect gives a single set of records which are common in both the queries. For above example, it will display

```
ENAME
-----------
Sparsh
```

Minus:- If we need the employees who are working in accounts department but not in technical department, then minus operator will do that job. Syntax is given as follows:

```
SELECT  Ename FROM EMP_ACCOUNTS
        MINUS
SELECT Ename FROM EMP_TECHNICAL;
```

Minus gives only those records, which are in first query but not in second query. The output of this query will be

```
ENAME
--------
Axay
Vivek
Abhay
```

Restriction for all the set operators: The select statements must have the same number of columns and having the same data types for corresponding columns.

DATA CONTROL LANGUAGE

When an INSERT, UPDATE or DELETE operations are performed on the database, we can reverse or rollback the work what we have done. The process of saving the work done permanently or rolling back work is controlled by two commands, namely

COMMIT and ROLLBACK. COMMIT makes changes in the current transaction permanent and ends the transaction. The syntax for committing and rolling back the work given as:

```
COMMIT;
ROLLBACK;
```

INSERT, UPDATE and DELETE commands are not made final until we commit them. All the DDL commands have an implicit COMMIT command. If we enter a DDL statement after several DML commands, it automatically causes a COMMIT command to execute and saves all the DML operations on the database permanently. On the other hand ROLLBACK command is used to undo work. But it only undoes those operations, which are not committed to the server permanently. It searches the last commit executed and undo all the operations after that.

SECURITY

Oracle provides the excellent security features for the database. The security is provided by granting and revoking the privileges on the database to/from the user.

Granting and Revoking Privileges:- .

The syntax for granting a privilege to a user or to a role is given as:

```
GRANT {privilege | Role} [,{privilege | Role}….] on <object>
TO
{user | Role} [,{user | Role},…] WITH ADMIN OPTION;
```

The WITH ADMIN OPTION clause permits the user, who is getting this permission to grant it to anther user.

For revoking, the syntax will be as follows:

```
REVOKE {system privilege | Role} [,{privilege | Role}….]
FROM
{user | Role} [,{user | Role},…];
```

A user can grant privileges on any object he or she owns. Suppose user SCOTT wants to grant read only permission to user arun then syntax will be as follows:

```
GRANT SELECT ON EMP TO ARUN;
```

A user can grant SELECT, INSERT, UPDATE and DELETE permissions of the table, which he/she owns to any other user. He can also grant the EXECUTE permission for a

procedure or function to other users. Similarly SELECT and ALTER permissions for sequences can also be granted to other users.

```
GRANT INSERT, UPDATE ON EMP TO ARUN;
```

```
GRANT EXECUTE ON ADD_NUMBERS TO ARUN;
```

Creating a Role:- A role is a set of privileges or the type of access that each user needs, depending on his/her status and responsibilities. We can grant several privileges on different database objects to a role and that role can be granted to any user like a ordinary privilege. A role can be created and then used in granting permissions to different users by using the following syntaxes:

```
CREATE ROLE role1;
```

```
GRANT  SELECT, INSERT ON EMP TO role1;
GRANT DELETE ON DEPT TO ROLE1;
GRANT ROLE1 to arun;
```

The above syntaxes creates a role, ROLE1. Then some privileges on EMP and DEPT tables are granted to ROLE1 and finally this role is being granted to user ARUN. We can use the REVOKE command for the roles in a similar way as discussed earlier.

4.5 VIEWS

Due to the security reasons, we may need to prevent all users from accessing all columns or all data of a table. In other words, we want to give permission to access a specific portion of data to selected users only. Views give us the flexibility to rearrange the way to see the table, a portion of the table or a group of the tables without actually creating any copies of the underlying data. In simplified way, a view is like a window through which data on tables can be viewed or modified. A view is stored as a SELECT statement only. It does not have any data of its own; rather it manipulates the data in the underlying table.

A view is a virtual table where data is not stored physically but give the convenient method to retrieve and manipulate the information as needed. It also allows users to make simple queries to retrieve the results from complicated queries, which also gives the advantage to hide the logic on which the view is created.
There are two types of views, one is simple view and another is complex view. Simple views derive data from a single table and contain no function or grouped data. While complex views may be derived from many tables and contain functions and grouped data. The syntax for creating a view is as given below.

```
CREATE [OR REPLACE] [FORCE] VIEW <view name> AS
                        <SELECT statement> [WITH CHECK OPTION];
```

An ORDER BY clause cannot be used in a view definition. Though we can use ORDER BY clause while retrieving the data through view. An example to create a simple view is given as follows.

```
CREATE OR REPLACE VIEW EMP_10 AS
                    SELECT * FROM EMP WHERE Deptno=10;
```

This will create a view emp_10, which is based on EMP table and taking all the data of department number 10. Now we can use this view like a table, for data manipulation and retrieval purpose. For example

```
SELECT * FROM EMP_10 WHERE Sal>5000;
```

This will display the employees of department 10 who are getting more than 5000. Now there is no need to specify the condition for Deptno because that is implicit in the definition of EMP_10 view. Similarly we can use other data manipulation commands on this view.

```
INSERT INTO EMP_10 VALUES (1008, 'Rashmi', 5000, '10-JUL-11', 10);
```

```
INSERT INTO EMP_10 VALUES (1009, 'Suresh', 3000, '01-JUL-11', 20);
```

The above two syntaxes will add two employees to emp table through emp_10 view. It is to be noted here that INSERT is possible through view if it does not contain a NOT NULL or Primary Key column in its definition. The reason is that in absence of that column in view, it will insert NULL to that column automatically and will display error message that it cannot insert a NULL value to a NOT NULL column. However this restriction is not applicable in case of UPDATE and DELETE operations.

The WITH CHECK OPTION specifies that only those records which meet the criteria of view (specified in the WHERE clause of view creation syntax) can be manipulated. For example if this clause is used in the above example then through this view we can not add a record of Deptno 20. So the second INSERT command will give the error message.

FORCE option creates the view even if the base table on which the view is based does not exist or there are insufficient privileges.

Some examples of complex view are given as follows.

```
CREATE OR REPLACE VIEW EMP_DEPT AS
            SELECT Ename, Dname FROM EMP, DEPT
                        WHERE EMP.Deptno=DEPT.Deptno;
```

This syntax will create a view, which has the employee names along with their department names.

```
CREATE OR REPLACE VIEW EMP_MAX AS
        SELECT Deptno, MAX(Sal) max_sal FROM EMP GROUP BY Deptno;
```

This view will have all the department numbers along with the maximum salary of each department. It is mandatory to use an alias for a column which is derived from function or an expression, as is used for MAX(sal). In case of complex view, the following restrictions are enforced.

DELETE is prohibited through view, if it contains:

(i) JOIN Condition
(ii) GROUP Functions
(iii) GROUP BY Clause
(iv) DISTINCT Clause

UPDATE is prohibited if the view contains:

(i) Any of the above
(ii) Columns defined by expressions (e.g. Sal+Commission)

INSERT is prohibited if the view contains:

(i) Any of the above
(ii) Any NOT NULL columns are not selected by the view.

Dropping a view: To drop a view, DROP VIEW command is used. For example, to drop emp_10 view the syntax is

```
DROP VIEW EMP_10;
```

Summary

SQL is the primary interface language for any database system. It is categorized into DDL, DML and DCL. These consist of various commands used for the database management. These commands include CREATE, ALTER, DROP, INSERT, UPDATE, DELETE and SELECT. Several options like nested query, set options, joins can be used in queries for retrieving of complex information.

Solved Exercises

Problem 1. Database consists of the following tables.

STUDENT(S#, Sname)
COURSE(C#, Title, Teacher_name)
RESULT(S#, C#, Marks)

Write queries in SQL to get
i) List of students who appear in all courses taught by a teacher named Dr A Sharma and scored more than 60 marks.

ii) The subject titles (s) in which there are maximum failures. (passing marks in a subject is 40.)
iii) Name (s) of student (s) obtaining highest marks in course number C# = 101.

Solution:
i)To display the list of students who appear in all courses taught by a teacher named Dr A Sharma and scored more than 60 marks.

```
SELECT Sname FROM STUDENT
        WHERE S# IN(SELECT S# FROM RESULT
                WHERE Marks > 60 AND
                        C#  IN(SELECT C# FROM COURSE
                                WHERE Teacher_name ='Dr A Sharma'));
```

First inner most query will return all the course codes taught by teacher namedDr A Sharma. Then student's code of those students who have taught by Dr A Sharma and have got greater than 60 marks will be selected by second query which then be used to give the name of the students by the outer query.

ii) To display the subject titles (s) in which there are maximum failures.(passing marks in a subject is 40.)

```
SELECT Title FROM COURSE, RESULT
        WHERE RESULT.C# = COURSE.C# AND Marks<40
                GROUP BY COURSE.C# HAVING
                        COUNT(*) = (SELECT MAX(COUNT(*))
                                FROM RESULT WHERE Marks < 40
                                        GROUP BY C#);
```

iii) To display the name (s) of student (s) obtaining highest marks in course number (c#) 101.

```
SELECT Sname FROM STUDENT
        WHERE S# IN(SELECT S# FROM RESULT
                WHERE Marks = (SELECT MAX(Marks) FROM RESULT
                        WHERE C# = 101));
```

The inner most query will give the highest marks of course 101. Then next query will be used to give the student code of those students who have got those marks and finally the outer most query will return the name of the students.

Problem 2. Consider the following database.

EMPLOYEE (Ename, Street, City)
WORKS (Ename, Bank-name, Salary)
BANK (Bank-name, City)
MANAGER (Ename, Mgrname)

Give an expression in SQL for each of the following queries.

a) Find the names of all employees who work for State Bank of India.
b) Find all the employees in the database who live in the same cities as the bank for which they work.
c) Find the names, street and cities of residence of all employees who work for State Bank of India and earn more than Rs. 15,000. (Assume that all employees have distinct names.)

Solution

a) SELECT Ename FROM EMPLOYEE
WHERE Bank_name = 'State Bank of India';

b) SELECT Ename FROM employee, works, bank
WHER EEMPLOYEE.Ename = WORKS.Ename
AND WORKS.Bank_name= BANK.Bank_name
AND EMPLOYEE.City=BANK.City;

c) SELECT EMPLOYEE.Ename, Street, City FROM EMPLOYEE, WORKS
WHERE EMPLOYEE.Ename = WORKS.Ename
AND Bank_name='State Bank of India'
AND Salary >15000;

Problem 3

Consider the following tables.

EMP (Emp_no, Name, Salary, Supervisor_no, Sex_code, Dept_code)
DEPT(Dept_cd, Dept_name)

Write down queries in SQL for getting following information:
(i) Employees and their supervisor if they are of opposite sex.
(ii) Employees getting more salary than their supervisor.
(iii) Department name and total number of employees in each department who earn more than average salary for their department.
(iv) Department(s) having maximum employees earning more than 25000.
(v) Name of employee(s) who earn maximum salary in their organization.

Solution

(i) Name of the employees and their supervisors if they are of opposite sex.
This is case of self join, which will have two aliases of the same table EMP as

SELECT E1.Name, E2.Name Supervisor
FROM EMP AS E1, EMP AS E2
WHERE E1.Supervisoe_no = E2.Emp_no
AND E1.Sex_code <> E2.Sex_code;

(ii) Employees getting more salary than their supervisor.
Again it will be solved by using self join concept.

SELECT E1.Name, E2.Name Supervisor

```
    FROM EMP AS E1, EMP AS E2
        WHERE E1.Supervisoe_no = E2.Emp_no
            AND E1.Salary > E2.Salary;
```

(iii) Department name and total number of employees in each department who earn more than average salary for their department.

This will use the correlated query concept.

```
SELECT Dept_name, COUNT(*)
    FROM EMP E, DEPT
        WHERE E.Dept_code =DEPT.Dept_cd
            AND Salary > (SELECT AVG(Salary) FROM EMP
                WHERE EMP.Dept_code = E.Dept_code)
                    GROUP BY Dept_name ;
```

(iv) Departments having maximum employees earning more than 25000.

```
SELECT Dept_name, COUNT (*)
    FROM EMP E, DEPT
        WHERE E.Dept_code =DEPT.Dept_cd AND Salary>25000
            GROUP BY Dept_name
                HAVING COUNT(*) >= ALL (SELECT COUNT(*)
                    FROM EMP GROUP BY Dept_code);
```

(v) Name of employee(s) who earn maximum salary in the organization.

```
SELECT Name FROM EMP
    WHERE Salary = (SELECT MAX (Salary) FROM EMP);
```