UNIT 3 STRUCTURED QUERY LANGUAGE (SQL)

Structure

1. Introduction

2. Objectives

3. **Lesson-1**

    What is SQL?

    Data Definition Language

4. Lesson-2

    Data Manipulation Language

    Functions in SQL

5. Lesson-3

    Data Control Language

6. Lesson-4

    Database Objects:

    Views

    Sequences

    Indexes

7. Lesson-5

    The Join operation

    Nested Queries

DMS,KU

8. Summary

9. Suggested Readings

10. Suggested Reading

INTRODUCTION

Database is an organized and related collection of information about an entity having controlled redundancy and serves multiple applications. DBMS (database management system) is application software that is developed to create and manipulate the data in database. A query language can easily access a data in a database. SQL (Structured Query Language) is language used by most relational database systems. IBM developed the SQL language in mid-1979. All communication with the clients and the RDBMS or between RDBMS is via SQL. Whether the client is a basic SQL engine or a disguised engine such as a GUI, report writer or one RDBMS talking to another, SQL statements pass from the client to the server. The server responds by processing the SQL and returning the results. The advantage of this approach is that the only network traffic is the initial query and the resulting response. The processing power of the client is reserved for running the application. SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL). It is a fourth generation language. SQL has many more features and advantages. Let us discuss the SQL in more detail in this unit. It should be noted that many commercial DBMSs may or may not implement all the details given here in this unit. For example, MS-ACCESS does not support some of these features. Even some of the constructs may not be portable, please consult the relevant DBMS manuals for any such issues.

OBJECTIVES

After going through this unit, you should be able to:

- Create, modify and delete database schema objects;
- Update database using SQL commands;
- Retrieve data from the database through queries and sub-queries;
- Handle join operations;
- Control database access;
- Deal with database objects like Tables, Views, Indexes, Sequences, and Synonyms using SQL.

Lesson-10

WHAT IS SQL?

Structured Query Language (SQL) is a standard fourth generation query language. It is commonly used with all relational databases for data definition, manipulation and control purposes. All the relational systems support SQL, thus allowing migration of database from one DBMS to another. In fact, this is one of the reasons of the major success of Relational DBMS. A user may be able to write a program using SQL for an application that involves data being stored in more than one DBMSs, provided these DBMSs support standard SQL. This feature is commonly referred to as portability. However, not all the features of SQL implemented in one RDBMS are available in others because of customization of SQL.

SQL provides an interface where the user can specify "What" are the expected results. The query execution plan and optimization is performed by the DBMS. The query plan and optimization determines how a query needs to be executed.

SQL is called a non-procedural language as it just specifies what is to be done rather than how it is to be done. Also, since SQL is a higher-level query language, it is closer to a language like English. Therefore, it is very user friendly. The American National

Standard Institute (ANSI) has designed standard versions of SQL. The first standard in this series was created in 1986. It was called SQL-86 or SQL1. This standard was revised and enhanced later and SQL-92 or SQL-2 was released. A newer standard of SQL is SQL3 which is also called SQL- 99. In this unit we will try to cover features from latest standards. However, some features may be found to be very specific to certain DBMSs.

Following are some of the important features of SQL:

- It is a non procedural language.
- It is an English-like language (High Level/ 4^{th} Generation).
- It can process a single record as well as sets of records at a time.
- It is different from a third generation language (C & C++). All SQL statements define what is to be done rather than how it is to be done.
- SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL).
- It insulates the user from the underlying structure and algorithm.
- SQL has facilities for defining database views, security, integrity constraints, transaction controls, etc.

There are many variants of SQL, but the standard SQL is the same for any DBMS environment. The following table shows the differences between SQL and one of its superset SQL*Plus which is used in Oracle. This will give you an idea of how various vendors have enhanced SQL to an environment. The non-standard features of SQL are not portable across databases, therefore, should not be used preferably while writing SQL queries.

DATA DEFINITION LANGUAGE (DDL)

As discussed in unit 2, the basic storage unit of a relational database management system is a table. It organizes the data in the form of rows and columns. But what does the data field column of table store? How do you define it using SQL? The Data definition language (DDL) defines a set of commands used in the creation and modification of schema objects such as tables, indexes, views etc. These commands provide the ability to create, alter and delete these objects. These commands are related to the management and administration of the databases. Before and after each DDL statement, the current transactions are implicitly committed, that is changes made by these commands are permanently stored in the databases. Let us discuss these commands in more detail:

CREATE TABLE Command: This command is used to create a table structure or what we call, schema**.** The general syntax for create table command is:

CREATE TABLE <table name>

(

Column_name1 data type (column width) [constraints],

Column_name2 data type (column width) [constraints],

Column_name3 data type (column width) [constraints],

……..);

Where <table name> is the name of the table, column name defines the name of the column or field, data type specifies the data type for the field and column width specifies the allocated size to the field.

General guidelines for creation of a table:

- Table name should start with an alphabet.
- Any combination of alphabets, numbers and underscore can be used
- Special symbols can't be used in table names

- In table name, blank spaces and single quotes are not allowed.
- Reserved words cannot be used as table name.
- Proper data types and size should be specified.
- Unique column name should be specified.

Column Constraints: The following constraints can be implemented on any column:

i. NOT NULL
ii. UNIQUE
iii. PRIMARY KEY
iv. CHECK
v. DEFAULT
vi. REFERENCES
vii. ON DELETE CASCADE/SET NULL

Most of these constraints have been discussed in the previous unit and the remaining will be discussed as and when found necessary.

Example 1:

CREATE TABLE EMPLOYEE (

ENO number (3) PRIMARY KEY,

ENAME char (20) NOT NULL,

AGE number (2) DEFAULT (18),

GENDER char (1) NOT NULL,

ADDRESS char (30),

SALARY number (10, 2) NOT NULL

DMS,KU

DNO number (2) [FOREIGN KEY] REFERENCES DEPARTMENT (DNO) ON DELETE CASCADE);

Note: [….] indicates that the keywords are optional.

The command above creates a table named EMPLOYEE (shown below) with ENO, ENAME, AGE, GENDER, ADDRESS, SALARY and DNO as its attributes with related data types and constraints. Please note the use of data type char. In many implementations of SQL on commercial DBMS like SQL server and oracle, a data type called varchar and varchar2 are used. Varchar or varchar2 basically is variable length character type subject to a maximum as specified in the declarations. The PRIMARY KEY is used to uniquely identify each individual tuple of the relation. NOT NULL specifies that the column value can't be NULL, it should contain some value. The References keyword in the query is used to implement Referential integrity constraint as discussed in the previous unit.

EMPLOYEE						
ENO	**ENAME**	**AGE**	**GENDER**	**ADDRESS**	**SALARY**	**DNO**
101	Tariq	35	M	Pattan	40000	10
102	Asif	32	M	Soura	35000	50
103	Rafi	39	M	Dalgate	45000	10
104	Ayub	41	M	Baramulla	43000	20
105	Bashir	48	M	Hawal	81000	10
106	Ajaz	31	M	Khanyar	25000	20

Example 2:

CREATE TABLE DEPARTMENT

(

DNO number (3) PRIMARY KEY,

DNAME char (25) NOT NULL,

DHEAD char (25) CHECK (DHEAD like "Prof. %")

);

The CHECK constraint is used to put a check on the values that can be inserted into the table. Various operators are used with CHECK constraint which will be discussed later in this unit.

ALTER TABLE Command: This command is used for modification of existing structure of the table in the following situation:

- When a new column is to be added to the table structure.
- When the existing column definition has to be changed, i.e., changing the width of the data type or the data type itself.
- When an existing column is to be deleted from the table.
- When integrity constraints have to be included or dropped.
- When a constraint has to be enabled or disabled.

The general syntax of the ALTER TABLE command is given below:

ALTER TABLE <table name> ADD (<column name> <data type>…);

ALTER TABLE <table name> MODIFY (<column name> <data type>…);

ALTER TABLE <table name> DELETE COLUMN <column name>;

ALTER TABLE <table name> ADD CONSTRAINT <constraint name>

< constraint type> (field name);

ALTER TABLE <table name> DROP<constraint name>;

ALTER TABLE <table name> ENABLE/DISABLE <constraint name>;

Example 3:

(i) Using ALTER command to add a new column to a table:

ALTER TABLE employee ADD [column] (PHONNO NUMBER (11));

(ii) Using ALTER command to modify a column:

ALTER TABLE employee MODIFY (GENDER char (6));

(iii) Using ALTER command to delete a column from a table:

ALTER TABLE employee DELETE [column] PHONNO;

The other Alter table command options are applied on named constraints. If a user knows the constraint names, (s)he can add, delete, enable or disable such constraints.

DROP TABLE Command: When an existing object is not required for further use, it is always better to remove it from the database. To delete the existing tables from the database the DROP TABLE command is used.

The general syntax of DROP TABLE command is:

DROP TABLE <table name>;

Example 4:

DROP TABLE employee;

Lesson-11

DATA MANIPULATION LANGUAGE

Data manipulation language (DML) defines a set of commands that are used to query and modify (i.e. Manipulate) data within existing database objects. In this case commit is not implicit that is changes are not permanent till explicitly committed. DML consist of SELECT, INSERT, UPDATE and DELETE statements.

These statements are discussed below with examples.

SELECT Statement: This statement is used for retrieving or displaying information from the databases. It can be coupled with many clauses. Let us discuss these clauses in more detail:

I. **Using Arithmetic operator:** Basic arithmetic operations like add, subtract, multiply, divide, modulus etc can be performed using various arithmetic operators shown below:

Operation	Operator
Add	**+**
Subtract	**-**
Multiply	*****

Divide	**/**
Modulus	**%**

Example 5:

(i) SELECT ENAME, SAL, SAL+300 FROM EMPLOYEE;

(ii) SELECT ENAME, SAL, SAL*12 FROM EMPLOYEE;

(iii) SELECT ENAME, SAL, SAL*12-300 FROM EMPLOYEE;

II. **Using Column aliases:** Alias names can be used instead of the exact column names while displaying information from tables. The alias name is given after the specified field within inverted commas as shown in the following example.

Example 6:

To print column names as EMPLOYEE NAME and SALARY*12 as ANNUAL SALARY for, the following query can be designed.

SELECT ENAME "EMPLOYEE NAME",

SALARY*12 "ANNUAL SALARY"

FROM EMPLOYEE;

III. **Concatenation operator:** If we want to print two or more columns as one string, we can use the concatenation (||) operator as shown below.

Example 7:

Printing name and job as one string as column name employees:

SELECT ENAME||JOB "EMPLOYEES" FROM EMP;

IV. **Using Literal Character String:** We can use character strings between column names while retrieving data from tables to make the output more understandable to

the users. The data will be displayed in meaningful English sentences as shown in the following example.

Example 8:

To print <name> IS A <job> as one string with column name employee

SELECT ENAME || ' IS A ' || JOB AS "EMPLOYEE" FROM EMPLOYEE;

V. **To eliminate duplicate rows:** Sometimes there may be similar data or records in the database. To eliminate duplicate rows, we make use of distinct operator. Its usage is shown below.

Example 9:

SELECT DISTINCT DEPTNO FROM EMPLOYEE;

SELECT DISTINCT ENAME FROM EMPLOYEE;

VI. **Special comparison operators used in where Clause:** In SQL, we use some special operators apart from the normal operators. These special operators are discussed here with examples.

a) **Between. …and…:** This operator is used to gives range between two values. Both the lower bound and upper bound values are included in the range as shown below.

Example 10:

SELECT ENAME, SALARY FROM EMPLOYEE

WHERE SALARY BETWEEN 20000 AND 40000;

SELECT ENO, ENAME, AGE FROM EMPLOYEE

WHERE AGE BETWEEN 20 AND 40;

Please note that the value before AND operator should be smaller than the value after it, otherwise the query will generate an error.

b) In (list): The IN operator is used to match any of a list of values with the values in the column to retrieve selective data from the tables.

Example 11:

SELECT ENO, ENAME, SALARY FROM EMPLOYEE

WHERE DNO IN (10, 20, 30);

SELECT ENO, ENAME, SALARY FROM EMPLOYEE

WHERE ENO IN (101, 105, 109);

The first query will display the records of only those employees who belong to department number 10, 20 or 30 only. Similarly the second query will display those records only where employee numbers are 101, 105 or 109. Note that the records will be displayed only in case a match is found otherwise no records will be displayed.

c) Like: This operator is used to match a character pattern. The character string or pattern to be matched is put with single quotes. Like operator is used only with char and Varchar2 data types to match a pattern. Following two special characters (Wild cards or Metacharacters) are used with LIKE operator to make it more robust.

1. % Denotes zero or many characters
2. _ (Underscore) Denotes one character

Note: A combination of % and _ can also be used.

Example 12:

(I) List the names of employees whose name starts with '**T**'

SELECT ENAME FROM EMPLOYEE

WHERE ENAME LIKE 'T%';

(II) List the employee names ending with '**r**'

SELECT ENAME FROM EMPLOYEE

WHERE ENAME LIKE '%r';

(III) List employee names starting with **T** and ending with **q.**

SELECT ENAME FROM EMPLOYEE

WHERE ENAME LIKE 'T%q';

(IV) List employee names having **V** as their fifth character**.**

SELECT ENAME FROM EMPLOYEE

WHERE ENAME LIKE '____I%';

d) IS NULL operator: This operator is used to check whether the value of some attribute is NULL. Unlike other operators "=" is not used with NULL. Null as discussed in unit 2 is treated specially by the SQL engine. It is something that has a value but is not equal to anything. Some properties of NULL are:

1. NULL is not equal to zero.
2. One NULL is not equal to another NULL.
3. NULL is not treated as a space.
4. Any operation performed on NULL results in a NULL value. ?

The use of IS NULL operator is shown below.

Example 13:

Find out the particulars of those employees whose DNO (department number) is not specified i.e. null

SELECT * FROM EMPLOYEE WHERE DNO IS NULL;

SELECT * FROM EMPLOYEE WHERE DNO=NULL; (will generate error)

e) **Logical Operators:** These operators are used to separate two or more conditions in the WHERE clause of an SQL statement. There are three logical operators viz.
   a. **AND**
   b. **OR**
   c. **NOT**

AND: means that the expressions on both sides must be true to return TRUE. If either of the expressions is false, AND returns FALSE.

Example 14:

To print those records of employees whose salary is more than 30000 and who reside in Srinagar.

SELECT * FROM EMPLOYEE

WHERE SALARY>30000 AND ADDRESS='Srinagar';

OR: We can use OR operator to sum up a series of conditions. If any of the comparisons is true, OR returns TRUE. In all other cases it returns FALSE.

Example 15:

Suppose you want to give a bonus of Rs- 500 to those employees who earn more than 50000 a month or who are older than 50 years. To print their salary we use the following query.

```
SELECT ENAME, SALARY+500 FROM EMPLOYEE
 WHERE salary > 50000   OR   age>= 50;
```

NOT: NOT negates everything that follows it. If the condition it applies to evaluates to TRUE, NOT make it FALSE. If the condition after the NOT is FALSE, it becomes TRUE.

Example 16:

```
SELECT * FROM EMPLOYEE WHERE ENAME NOT LIKE 'T%';
```

NOT can also be used with the operators IN, BETWEEN, and IS when applied to NULL. The examples of each are given below.

Example 17:

```
SELECT *   FROM EMPLOYEE
  WHERE ENAME IS NOT NULL;

SELECT *   FROM EMPLOYEE
  WHERE AGE NOT BETWEEN 18 AND 27;

SELECT *   FROM EMPLOYEE
  WHERE DNO NOT IN (10, 20, 30);
```

VII. **Operator Precedence:** All the operators used in SQL are evaluated as per some precedence rules. All the operators do not have equal priority. The order of evaluation of operators in a statement is called operator precedence. Operators of the same priority are evaluated from left to right. Parentheses are used to force prioritized evaluation. Multiplication and division operators have higher priority over addition and subtraction operators. Following table gives operator precedence of operators used in SQL.

Priority	Operator
1	Comparison operators
2	NOT
3	AND
4	OR
5	*, /
6	+, –

Example 18:

SELECT ENAME, SALARY, SALARY*12+100 FROM EMPLOYEE;

SELECT ENAME, SAL, (SALARY+100)*12 FROM EMPLOYEE;

In the first query SALARY field will be multiplied by 12 and then 100 will be added to the result whereas in second query 100 will be added to SALARY and the result will be multiplied by 12. So the results will be different.

All the operators used in SQL are evaluated as per some precedence rules. All the operators do not have equal priority. The order of evaluation of operators in a statement is called operator precedence. Following table gives operator precedence of operators used in SQL.

VIII. Order by clause: It is used in the last portion of SELECT statement to sort the data in the tables. Tuples can be sorted in ascending or descending order. By default it takes ascending order. Keywords ASC and DESC are used for this purpose. Sorting by column which is not in select list is possible by using column Alias.

Example 19: Simple sorting

SELECT ENO, ENAME, ADDRESS FROM EMPLOYEE

ORDER BY ENO DESC;

SELECT ENO, ENAME, SAL*12 "ANNUAL_SALARY"

FROM EMPLOYEE ORDER BY ANNUAL_SALARY;

Example 20:

Sorting by multiple columns; ascending order of department number and descending order of salary in each department.

SELECT ENAME, DNO, SALARY FROM EMPLOYEE

ORDER BY DNO, SALARY DESC;

IX. **Group By clause:** This clause is used to group database tuples on the basis of certain common attribute value such as job type of employees, employees of a particular department. If needed WHERE clause can still be used.

Example 21:

Find department number and Number of Employees working in that department.

SELECT DNO, COUNT (ENO) FROM EMPLOYEE

GROUP BY DNO;

Please note that while using group by and aggregate functions (discussed later) the only attributes that can be put in select clause are the aggregated functions and the attributes that have been used for grouping the information. For example, in the example 20, we cannot put ENAME attribute in the SELECT clause as it will not have a distinct value for the group. Please note the group here is created on DNO.

X. **Having clause:** With normal SELECT command we use WHERE clause to get selective tuples or attributes from the tables. But GROUP BY clause WHERE does not work. So to make a selective retrieval of data, we use HAVING clause.

Example 22:

Find department number and maximum salary of those departments where maximum salary is more than Rs 20000/-.

SELECT DNO, MAX (SALARY) FROM EMPLOYEE

GROUP BY DNO

HAVING MAX (SAL) > 20000;

Aggregate functions: These are the functions used to perform operations on data in multiple tuples. In other words, these are not single row functions but these act concurrently on data in many rows. Some of these functions are count, min, max, avg. These functions help in getting consolidated information from a group of tuples.

Example 23: Find the total number of employees.

SELECT COUNT (*) FROM EMPLOYEE; [Note: * stands for all rows, all columns]

Example 24:

Find the minimum, maximum and average salaries of employees of department number 10.

SELECT MIN (SALARY), MAX (SALARY), AVG (SALARY)

FROM EMPLOYEE WHERE DNO = '10';

INSERT INTO command: Once table structure or schema is created, the database is in an empty state. To populate the database with data, SQL provides us with the INSERT INTO command. This command can be used to insert Values in all columns or for some selected columns only. In place of values parameter substitution can also be used with insert. If data is not available for all the columns, then the column list must be included following the table name. The general syntax of INSERT INTO command is:

INSERT INTO <table name> VALUES (value1, value2, ……, value n);

INSERT INTO <table name> (attribute name, attribute name, …) VALUES (value1, value2, ……);

INSERT INTO <table name> VALUES (&1,&2, ……, &n);

Example 25:

Insert the employee records into EMPLOYEE table.

INSERT INTO EMPLOYEE

VALUES (101, 'Tariq', 35, 'M', 'Pattan', 35000, 10);

Example 26:

Insert values in a table using parameter substitution (& operator is used for it 1, 2 are the field numbers).

INSERT INTO EMPLOYEE

VALUES (&1,'&2','&3', &4, &5, NULL, &7);

Once this query is run, the user will be asked to supply values to the attributes one by one pressing Enter key after every value is given. The same query can be run again and again by pressing / (forward slash) every time we wish to run this query.

Note that instead of field numbers, field names or some alias names can also be used with the "&" operator.

Example 27: Insert values in a table specifying attribute names

INSERT INTO EMPLOYEE (ENAME, ADDRESS, AGE, ENO, SALARY, GENDER, DNO)

VALUES ('Majid', 'Lalbazar', 29, 109, 30000, M, 20);

When we don't know the order of attributes in tables, we specify the names of attributes followed by the values in the same order as shown in the query.

INERTING DATA FROM AN EXISTING TABLE: We can insert data into a new table from an existing table using INSERT INTO command. The syntax is :

INSERT INTO (<new or destination table name> [column name1, column name2, …])

VALUES AS SELECT <column name1, column name2, ….>

FROM <existing or source table name>;

Note that column names in source table are optional. If you want to put data in all columns, then column names need not be mentioned. Also if data from all columns in destination table has to be copied then instead of mentioning column names, * (meaning all columns) can be used. Please see example 27.

Example 28:

Assume table TEMP with fields EMPNO, EMPNAME, and SALARY. Insert data into TEMP from EMPLOYEE table.

INSERT INTO TEMP (EMPNO, EMPNAME, SALARY**)**

VALUES AS SELECT (ENO, ENAME, SALARY**)**

FROM EMPLOYEE;

Example 29:

Assume table TEMP1 with fields EMPNO, EMPNAME, AGE, SEX, ADDRESS, SAL, DEPTNO. Insert data into TEMP from EMPLOYEE table.

INSERT INTO TEMP1

VALUES AS SELECT *

FROM EMPLOYEE;

UPDATE Command: The purpose of UPDATE statement is to change or modify the values of existing records in the database. The update command is always followed by the keyword SET. The general syntax of UPDATE command is as:

UPDATE <table name>

SET <column name> = <value>

WHERE <condition>;

Example 30: using update command

UPDATE EMPLOYEE SET AGE = 32

WHERE ENAME='Tariq';

Note that the character, Date and Alphanumeric data has to be put in single quotes. We can also make use of sub-queries with update command.

DELETE Command: In addition to adding data to a database, we will also need to delete data from a database. The general syntax for the DELETE statement is

DELETE FROM <table name>

WHERE <condition>;

The first thing you will probably notice about the DELETE command is that it doesn't have a prompt. Users are accustomed to being prompted for assurance when, for instance, a directory or file is deleted at the operating system level. Are you sure? (Y/N) is a common question asked before the operation is performed. Using SQL, when you instruct the DBMS to delete a group of records from a table, it obeys your command without asking. That is, when you tell SQL to delete a group of records, it will really do it!

Depending on the use of the DELETE statement's WHERE clause, SQL can do the following:

- Delete single rows
- Delete multiple rows
- Delete all rows
- Delete no rows

Here are several points to remember when using the DELETE statement:

- The DELETE statement cannot delete an individual field's values (use UPDATE instead). The DELETE statement deletes entire records from a single table.
- Like INSERT and UPDATE, deleting records from one table can cause referential integrity problems within other tables. Keep this potential problem area in mind when modifying data within a database.
- Using the DELETE statement deletes only records, not the table itself. Use the DROP TABLE statement to remove an entire table.

Example 31:

This example shows you how to delete all the records from EMPLOYEE.

```
DELETE FROM EMPLOYEE;
   OR
DELETE * FROM EMPLOYEE;
```

Example 32:

Delete the records of employees who have got no increment.

```
DELETE FROM EMPLOYEE
    WHERE EMPNO NOT IN (SELECT EMPNO FROM INCR);
```

Functions:

Functions in SQL enable you to perform feats such as determining the sum of a column or converting all the characters of a string to uppercase. By the end of the day, you will understand and be able to use all the following:

- Aggregate functions
- Date and time functions
- Arithmetic functions
- Character functions
- Conversion functions
- Miscellaneous functions

These functions greatly increase your ability to manipulate the information you retrieve using the basic functions of SQL. The first five aggregate functions, COUNT, SUM, AVG, MAX, and MIN, are defined in the ANSI standard. Most implementations of SQL have extensions to these aggregate functions, some of which are covered here. Some implementations may use different names for these functions.

Aggregate Functions

These functions are also referred to as group functions. They return a value based on the values in a column. The examples in this section use the table EMPLOYEE.

1. **COUNT**

The function COUNT returns the number of rows that satisfy the condition in the WHERE clause.

Example 33:

Find the total number of employees earning more than 20000 per month?

```
SELECT COUNT (*)
  FROM EMPLOYEE
  WHERE SALARY>20000;
```

If you use COUNT without a WHERE clause, it returns the number of records in the table.

Example 34:

```
SELECT COUNT (*)
  FROM EMPLOYEE;
```

2. SUM

SUM does just what it means. It returns the sum of all values in a column.

Example 35:

Find out the total salary of all the employees?

```
SELECT SUM (SALARY) TOTAL_SALARY
  FROM EMPLOYEE;
```

SUM works only with numbers. If you try it on a non-numerical field, you get an error. The error message is logical because you cannot sum a group of names.

```
SELECT SUM (NAME)
  FROM EMPLOYEE;
```

3. AVG

The AVG function computes the average of a column.

Example 36:

Find out the average salary of all the employees?

SELECT AVG (SALARY) AVERAGE_SALARY

FROM EMPLOYEE;

Like the SUM function, AVG works only with numbers.

4. MAX

If you want to find the largest value in a column, use MAX.

Example 37:

Find the highest salary being paid?

SELECT MAX (salary)

FROM EMPLOYEE;

Can you find out who has the highest salary?

SELECT NAME

FROM EMPLOYEE

WHERE salary = MAX (salary);

In Oracle the error will be like:

ERROR at line 3:

ORA-00934: group function is not allowed here

Unfortunately, you can't. The error message is a reminder that this group function (remember that *aggregate functions* are also called *group functions*) does not work in the WHERE clause. Don't worry, Sub-queries can be used to do this.

What happens if you try a non numerical column?

SELECT MAX (NAME)

FROM EMPLOYEE;

Here's something new. MAX returns the highest (closest to Z) string.

5. MIN

MIN does the expected thing and works like MAX except it returns the lowest member of a column.

Example 38:

Find out the Minimum salary of all the employees?

SELECT MIN (salary)

FROM EMPLOYEE;

The following statement returns the name closest to the beginning of the alphabet:

SELECT MIN (NAME)

FROM EMPLOYEE;

You can combine MIN with MAX to give a range of values. For example:

SELECT MIN (SALARY), MAX (SALARY)

FROM EMPLOYEE;

This sort of information can be useful when using statistical functions.

6. VARIANCE

VARIANCE produces the square of the standard deviation, a number vital to many statistical calculations. It works like this:

Example 39:

SELECT VARIANCE (AGE)

FROM EMPLOYEE;

If you try a string

SELECT VARIANCE (NAME)

FROM EMPLOYEE;

DMS.KU

ERROR:

ORA-01722: invalid number

Note that VARIANCE is another function that works exclusively with numbers.

7. STDDEV

The final group function, STDDEV, finds the standard deviation of a column of numbers, as demonstrated by this example:

Example 40:

```
SELECT STDDEV (age)
   FROM EMPLOYEE;
```

It also returns an error when confronted by a string:

```
SELECT STDDEV (NAME)
  FROM EMPLOYEE;
```

ERROR:

ORA-01722: invalid number

These aggregate functions can also be used in various combinations:

```
SELECT COUNT (SALARY),
   AVG (SALARY),
   MIN (SALARY),
   MAX (SALARY),
   STDDEV (SALARY),
   VARIANCE (SALARY),
   SUM (SALARY)
   FROM EMPLOYEE;
```

DMS.KU

Date and Time Functions

We live in a civilization governed by times and dates, and most major implementations of SQL have functions to cope with these concepts. This section uses the table PROJECT, shown below, to demonstrate the time and date functions.

PROJECT		
PROJ_NAME	**START_DATE**	**END_DATE**
Info. system	01-APR-2010	30-APR-2010
Children survey	02-APR-2010	01-MAY-2010
Bank analysis	15-MAY-2010	30-MAY-2010
Software comparison	01-JUN-2010	30-JUN-2010
Population analysis	03-SEP-2010	20-JAN-2011

1. **ADD_MONTHS**

This function adds a number of months to a specified date. For example, say something extraordinary happened, and the preceding project slipped to the right by two months. You could make a new schedule by typing

Example 41:

```
SELECT PROJ_NAME,
   START_DATE,
   END_DATE ORIGINAL_END,
   ADD_MONTHS (END_DATE,2)
   FROM PROJECT;
```

Not that a slip like this is possible, but it's nice to have a function that makes it so easy. ADD_MONTHS also works outside the SELECT clause like

```
SELECT PROJ_NAME PROJECTS_SHORTER_THAN_ONE_MONTH
   FROM PROJECT
   WHERE ADD_MONTHS ( STARTDATE,1) > ENDDATE;
```

You will find that all the functions in this section work in more than one place. However, ADD MONTHS does not work with other data types like character or number without the help of functions TO_CHAR and TO_DATE, which are discussed later.

2. LAST_DAY

LAST_DAY returns the last day of a specified month. If, for example, you need to know what the last day of the month is in the column END_DATE, you would type
Example 42:

```
SELECT END_DATE, LAST_DAY(END_DATE)
  FROM PROJECT;
```

3. MONTHS_BETWEEN

If you need to know how many months fall between month X and month Y, use MONTHS_BETWEEN like this:
Example 43:

```
SELECT PROJ_NAME, START_DATE, END_DATE, MONTHS_BETWEEN
(START_DATE, END_DATE)
   DURATION
  FROM PROJECT;
```

Wait a minute--that doesn't look right. Try this:

```
SELECT PROJ_NAME, STARTDATE, ENDDATE,
  MONTHS_BETWEEN (ENDDATE,STARTDATE) DURATION
  FROM PROJECT;
```

That's better. You see that MONTHS_BETWEEN is sensitive to the way you order the months. Negative months might not be bad. For example, you could use a negative result

to determine whether one date happened before another. For example, the following statement shows all the tasks that started before May 19, 2010:

```
SELECT *
   FROM PROJECT
   WHERE MONTHS_BETWEEN ('19 MAY 2010', STARTDATE) > 0;
```

4. NEXT_DAY

NEXT_DAY finds the name of the first day of the week that is equal to or later than another specified date.

Example 42:

For example, to send a report on the Friday following the first day of each event, you would type

```
SELECT START_DATE,
   NEXT_DAY (START_DATE, 'FRIDAY')
   FROM PROJECT;
```

5. SYSDATE

SYSDATE returns the system time and date:

Example 43:

```
SELECT DISTINCT SYSDATE
  FROM PROJECT;
```

If you wanted to see where you stood today in a certain project, you could type

```
SELECT *
  FROM PROJECT
  WHERE STARTDATE > SYSDATE;
```

DMS.KU

Arithmetic Functions

Many of the uses you have for the data you retrieve involve mathematics. Most implementations of SQL provide arithmetic functions similar to the functions covered here. The examples in this section use the NUMBERS table shown below.

NUMBERS	
X	**Y**
3.1415	4
-45	.707
5	9
-57.667	42
15	55
-7.2	5.3

1. **ABS**

The ABS function returns the absolute value of the number you point to.

Example 44:

```
SELECT ABS (X) ABSOLUTE_VALUE
  FROM NUMBERS;
```

OUTPUT IN ORACLE:

```
ABSOLUTE_VALUE
--------------
      3.1415
          45
           5
      57.667
          15
         7.2
```

ABS changes all the negative numbers to positive and leaves positive numbers alone.

2. CEIL and FLOOR

CEIL returns the smallest integer greater than or equal to its argument. FLOOR does just the reverse, returning the largest integer equal to or less than its argument.

Example 45:

```
SELECT Y, CEIL (Y) CEILING
  FROM NUMBERS;
```

OUTPUT IN ORACLE:

```
         Y    CEILING
---------- ---------
         4          4
      .707          1
         9          9
        42         42
        55         55
       5.3          6
```

```
SELECT X, FLOOR (X) FLOOR
  FROM NUMBERS;
```

```
         X      FLOOR
---------- ---------
    3.1415          3
       -45        -45
         5          5
   -57.667        -58
        15         15
      -7.2         -8
```

DMS,KU

3. COS, COSH, SIN, SINH, TAN, and TANH

The COS, SIN, and TAN functions provide support for various trigonometric concepts. They all work on the assumption that **n** is in radians. The following statement returns some unexpected values if you don't realize COS expects X to be in radians.

Example 46:

```
SELECT X, COS (X)
  FROM NUMBERS;
```

OUTPUT:

X	COS (X)
3.1415	-1
-45	.52532199
5	.28366219
-57.667	.437183
15	-.7596879
-7.2	.60835131

You would expect the COS of 45 degrees to be in the neighborhood of .707, not .525. To make this function work the way you would expect it to in a degree-oriented world, you need to convert degrees to radians. (360 degrees = 2 pi radians)

Example 47:

```
SELECT X, COS (X* 0.01745329251994)
  FROM NUMBERS;
```

OUTPUT:

X	COS (X*0.01745329251994)

3.1415	.99849724
-45	.70710678
5	.9961947
-57.667	.5348391
15	.96592583
-7.2	.9921147

Note that the number 0.01745329251994 is radians divided by degrees. The trigonometric functions work as follows:

SELECT X, COS (X*0.017453), COSH (X*0.017453)
FROM NUMBERS;

X	COS (X*0.017453)	COSH (X*0.017453)
3.1415	.99849729	1.0015035
-45	.70711609	1.3245977
5	.99619483	1.00381
-57.667	.53485335	1.5507072
15	.96592696	1.0344645
-7.2	.99211497	1.0079058

SELECT X, SIN (X*0.017453), SINH (X*0.017453)
FROM NUMBERS;

X	SIN (X*0.017453)	SINH (X*0.017453)
3.1415	.05480113	.05485607
-45	-.7070975	-.8686535
5	.08715429	.0873758

-57.667	-.8449449	-1.185197
15	.25881481	.26479569
-7.2	-.1253311	-.1259926

SELECT X, TAN (X*0.017453), TANH (X*0.017453)
FROM NUMBERS;

X	TAN (X*0.017453)	TANH (X*0.017453)
3.1415	.05488361	.05477372
-45	-.9999737	-.6557867
5	.08748719	.08704416
-57.667	-1.579769	-.7642948
15	.26794449	.25597369
-7.2	-.1263272	-.1250043

4. EXP

EXP enables you to raise *e* (*e* is a mathematical constant used in various formulas) to a power. Here's how EXP raises *e* by the values in column X:

Example 48:

SELECT X, EXP (X)
FROM NUMBERS;

OUTPUT:

X	EXP (X)
3.1415	23.138549
-45	2.863E-20
5	148.41316

57.667 9.027E-26

15 3269017.4

-7.2 .00074659

5. LN and LOG

These two functions generate logarithmic values. LN returns the natural logarithm of its argument.

Example 49:

SELECT X, LN (X)

FROM NUMBERS;

OUTPUT:

ERROR:

ORA-01428: argument '-45' is out of range

We FORGOT to mention that the argument had to be positive.

SELECT X, LN (ABS (X))

FROM NUMBERS;

X	LN (ABS (X))
3.1415	1.1447004
-45	3.8066625
5	1.6094379
-57.667	4.0546851
15	2.7080502
-7.2	1.974081

Notice how you can embed the function ABS inside the LN call. The other logarithmic function, LOG, takes two arguments, returning the logarithm of the first argument in the base of the second. The following query returns the logarithms of column Y in base 10.

SELECT Y, LOG (Y, 10)
FROM NUMBERS;

B	LOG(B,10)
4	1.660964
.707	-6.640962
9	1.0479516
42	.61604832
55	.57459287
5.3	1.3806894

6. MOD

The ANSI standard for the modulo operator % is sometimes implemented as the function MOD.

Example 50:

Here's a query that returns a table showing the remainder of A divided by B:

SELECT X, Y, MOD (X, Y)
FROM NUMBERS;

OUTPUT:

X	Y	MOD (X, Y)
3.1415	4	3.1415
-45	.707	-.459

5	9	5
-57.667	42	-15.667
15	55	15
-7.2	5.3	-1.9

7. POWER

To raise one number to the power of another, use POWER. In this function the first argument is raised to the power of the second:

Example 51:

```
SELECT X, Y, POWER (X, Y)
  FROM NUMBERS;
```

OUTPUT:

ERROR:

ORA-01428: argument '-45' is out of range

At first glance you are likely to think that the first argument can't be negative. But that impression can't be true, because a number like -4 can be raised to a power. Therefore, if the first number in the POWER function is negative, the second must be an integer. You can work around this problem by using CEIL (or FLOOR):

```
SELECT X, CEIL (Y), POWER (X,CEIL (Y))
  FROM NUMBERS;
```

OUTPUT:

X	CEIL(Y)	POWER(X,CEIL (Y))
3.1415	4	97.3976
-45	1	-45
5	9	1953125

-57.667	42	9.098E+73
15	55	4.842E+64
-7.2	6	139314.07

8. SIGN

SIGN returns -1 if its argument is less than 0, 0 if its argument is equal to 0, and 1 if its argument is greater than 0.

Example 52:

```
SELECT X, SIGN (X)
  FROM NUMBERS;
```

OUTPUT:

X	SIGN(X)
3.1415	1
-45	-1
5	1
-57.667	-1
15	1
-7.2	-1

You can also use SIGN in a SELECT WHERE clause like this:

```
SELECT X
  FROM NUMBERS
  WHERE SIGN (X) = 1;
```

OUTPUT:

X

DMS.KU

3.1415

5

15

9. SQRT

The function SQRT returns the square root of an argument. Because the square root of a negative number is undefined, you cannot use SQRT on negative numbers.

Example 53:

```
SELECT X, SQRT (X)
   FROM NUMBERS;
```

ERROR:

ORA-01428: argument '-45' is out of range

However, you can fix this limitation with ABS:

```
SELECT ABS(X), SQRT (ABS (X))
   FROM NUMBERS;
```

ABS (X)	SQRT(ABS (X))
3.1415	1.7724277
45	6.7082039
5	2.236068
57.667	7.5938791
15	3.8729833
7.2	2.6832816

Character Functions

Many implementations of SQL provide functions to manipulate characters and strings of characters. This section covers the most common character functions. The examples in this section use the table CHARACTERS.

CHARACTERS			
LASTNAME	**FIRSTNAME**	**MI**	**CODE**
Lone	**Tariq**	**A**	**32**
Bhat	**Junaid**	**J**	**67**
Darzi	**Tanveer**	**C**	**65**
Wani	**Mudasir**	**M**	**87**
Khan	**Rafi**	**A**	**77**
Zargar	**Shahid**	**G**	**52**

1. CHR

CHR returns the character equivalent of the number it uses as an argument. The character it returns depends on the character set of the database. For this example the database is set to ASCII. The column CODE includes numbers.

Example 54:

```
SELECT CODE, CHR(CODE)
  FROM CHARACTERS;
```

```
 CODE       CH
 ---------  --
   32
   67       C
   65       A
   87       W
   77       M
   52       4
```

The space opposite the 32 shows that 32 is a space in the ASCII character set.

2. CONCAT

The way || (discussed earlier) symbol joins two strings together, so does CONCAT. It works like this:

Example 55:

SELECT CONCAT (FIRSTNAME, LASTNAME) "FULL NAME" FROM CHARACTERS;

OUTPUT:

FULL NAME

Tariq Lone

Junaid Bhat

Tanveer Darzi

Mudasir Wani

Rafi Khan

Shahid Zargar

Quotation marks surround the multiple-word alias FIRST AND LAST NAMES. Again, it is safest to check your implementation to see if it allows multiple-word aliases.

Also notice that even though the table looks like two separate columns, what you are seeing is one column. The first value you concatenated, FIRSTNAME, is 15 characters wide. This operation retained all the characters in the field.

INITCAP

INITCAP capitalizes the first letter of a word and makes all other characters lowercase.

INPUT:

SQL> **SELECT FIRSTNAME BEFORE, INITCAP(FIRSTNAME) AFTER**

2 **FROM CHARACTERS;**

OUTPUT:

BEFORE AFTER

---------- ----------

KELLY Kelly

CHUCK Chuck

LAURA Laura

FESTER Fester

ARMANDO Armando

MAJOR Major

6 rows selected.

LOWER and UPPER

As you might expect, LOWER changes all the characters to lowercase; UPPER does just the reverse.

The following example starts by doing a little magic with the UPDATE function (you learn more about this next week) to change one of the values to lowercase:

INPUT:

SQL> **UPDATE CHARACTERS**

2 **SET FIRSTNAME = 'kelly'**

3 **WHERE FIRSTNAME = 'KELLY';**

OUTPUT:

1 row updated.

INPUT:

SQL> **SELECT FIRSTNAME**

2 **FROM CHARACTERS;**

OUTPUT:

FIRSTNAME

---

kelly

CHUCK

LAURA

FESTER

ARMANDO

MAJOR

6 rows selected.

Then you write

INPUT:

SQL> **SELECT FIRSTNAME, UPPER(FIRSTNAME), LOWER(FIRSTNAME)**

2 **FROM CHARACTERS;**

OUTPUT:

FIRSTNAME UPPER(FIRSTNAME LOWER(FIRSTNAME

--- --- ---

kelly KELLY kelly

CHUCK CHUCK chuck

LAURA LAURA laura

FESTER FESTER fester

ARMANDO ARMANDO armando

MAJOR MAJOR major

6 rows selected.

Now you see the desired behavior.

LPAD and RPAD

LPAD and RPAD take a minimum of two and a maximum of three arguments. The first argument is the character string to be operated on. The second is the number of characters to pad it with, and the optional third argument is the character to pad it with. The third argument defaults to a blank, or it can be a single character or a character string. The following statement adds five pad characters, assuming that the field LASTNAME is defined as a 15-character field:

INPUT:

SQL> **SELECT LASTNAME, LPAD(LASTNAME,20,'*')**

2 **FROM CHARACTERS;**

OUTPUT:

```
LASTNAME       LPAD(LASTNAME,20,'*'
-------------- --------------------
PURVIS         *****PURVIS
TAYLOR         *****TAYLOR
CHRISTINE      *****CHRISTINE
ADAMS          *****ADAMS
COSTALES       *****COSTALES
KONG           *****KONG
6 rows selected.
```

ANALYSIS:

Why were only five pad characters added? Remember that the LASTNAME column is 15 characters wide and that LASTNAME includes the blanks to the right of the characters that make up the name. Some column data types eliminate padding characters if the width of the column value is less than the total width allocated for the column. Check your implementation. Now try the right side:

INPUT:

```
SQL> SELECT LASTNAME, RPAD(LASTNAME,20,'*')
  2  FROM CHARACTERS;
```

OUTPUT:

```
LASTNAME       RPAD(LASTNAME,20,'*'
-------------- --------------------
PURVIS         PURVIS         *****
TAYLOR         TAYLOR         *****
CHRISTINE      CHRISTINE      *****
ADAMS          ADAMS          *****
COSTALES       COSTALES       *****
KONG           KONG           *****
6 rows selected.
```

ANALYSIS:

Here you see that the blanks are considered part of the field name for these operations. The next two functions come in handy in this type of situation.

LTRIM and RTRIM

LTRIM and RTRIM take at least one and at most two arguments. The first argument, like LPAD and RPAD, is a character string. The optional second element is either a character or character string or defaults to a blank. If you use a second argument that is not a blank, these trim functions will trim that character the same way they trim the blanks in the following examples.

INPUT:

```
SQL> SELECT LASTNAME, RTRIM(LASTNAME)
  2  FROM CHARACTERS;
```

OUTPUT:

```
LASTNAME        RTRIM(LASTNAME)
--------------- ---------------
PURVIS          PURVIS
TAYLOR          TAYLOR
CHRISTINE       CHRISTINE
ADAMS           ADAMS
COSTALES        COSTALES
KONG            KONG
6 rows selected.
```

You can make sure that the characters have been trimmed with the following statement:

INPUT:

```
SQL> SELECT LASTNAME, RPAD(RTRIM(LASTNAME),20,'*')
  2  FROM CHARACTERS;
```

OUTPUT:

```
LASTNAME        RPAD(RTRIM(LASTNAME)
--------------- --------------------
```

PURVIS PURVIS**********

TAYLOR TAYLOR**********

CHRISTINE CHRISTINE**********

ADAMS ADAMS**********

COSTALES COSTALES**********

KONG KONG**********

6 rows selected.

The output proves that trim is working. Now try LTRIM:

INPUT:

SQL> **SELECT LASTNAME, LTRIM(LASTNAME, 'C')**

2 **FROM CHARACTERS;**

OUTPUT:

LASTNAME LTRIM(LASTNAME,

--------------- ---------------

PURVIS PURVIS

TAYLOR TAYLOR

CHRISTINE HRISTINE

ADAMS ADAMS

COSTALES OSTALES

KONG KONG

6 rows selected.

Note the missing Cs in the third and fifth rows.

REPLACE

REPLACE does just that. Of its three arguments, the first is the string to be searched. The second is the search key. The last is the optional replacement string. If the third argument is left out or NULL, each occurrence of the search key on the string to be searched is removed and is not replaced with anything.

INPUT:

SQL> **SELECT LASTNAME, REPLACE(LASTNAME, 'ST') REPLACEMENT**

2 **FROM CHARACTERS;**

OUTPUT:

LASTNAME REPLACEMENT

--- ---

PURVIS PURVIS

TAYLOR TAYLOR

CHRISTINE CHRIINE

ADAMS ADAMS

COSTALES COALES

KONG KONG

6 rows selected.

If you have a third argument, it is substituted for each occurrence of the search key in the target string. For example:

INPUT:

SQL> **SELECT LASTNAME, REPLACE(LASTNAME, 'ST','**') REPLACEMENT**

2 **FROM CHARACTERS;**

OUTPUT:

LASTNAME REPLACEMENT

--- ---

PURVIS PURVIS

TAYLOR TAYLOR

CHRISTINE CHRI**INE

ADAMS ADAMS

COSTALES CO**ALES

KONG KONG

6 rows selected.

If the second argument is NULL, the target string is returned with no changes.

INPUT:

SQL> **SELECT LASTNAME, REPLACE(LASTNAME, NULL) REPLACEMENT**

2 **FROM CHARACTERS;**

OUTPUT:

LASTNAME REPLACEMENT

---------- ----------

PURVIS PURVIS

TAYLOR TAYLOR

CHRISTINE CHRISTINE

ADAMS ADAMS

COSTALES COSTALES

KONG KONG

6 rows selected.

SUBSTR

This three-argument function enables you to take a piece out of a target string. The first argument is the target string. The second argument is the position of the first character to be output. The third argument is the number of characters to show.

INPUT:

SQL> **SELECT FIRSTNAME, SUBSTR(FIRSTNAME,2,3)**

2 **FROM CHARACTERS;**

OUTPUT:

FIRSTNAME SUB

---------- ---

kelly ell

CHUCK HUC

LAURA AUR

FESTER EST

ARMANDO RMA

MAJOR AJO

DMS.KU

6 rows selected.

If you use a negative number as the second argument, the starting point is determined by counting backwards from the end, like this:

INPUT:

```
SQL> SELECT FIRSTNAME, SUBSTR(FIRSTNAME,-13,2)
  2  FROM CHARACTERS;
```

OUTPUT:

```
FIRSTNAME       SU
--------------- --
kelly           ll
CHUCK           UC
LAURA           UR
FESTER          ST
ARMANDO         MA
MAJOR           JO
```

6 rows selected.

ANALYSIS:

Remember the character field FIRSTNAME in this example is 15 characters long. That is why you used a -13 to start at the third character. Counting back from 15 puts you at the start of the third character, not at the start of the second. If you don't have a third argument, use the following statement instead:

INPUT:

```
SQL> SELECT FIRSTNAME, SUBSTR(FIRSTNAME,3)
  2  FROM CHARACTERS;
```

OUTPUT:

```
FIRSTNAME       SUBSTR(FIRSTN
--------------- -------------
kelly           lly
CHUCK           UCK
```

LAURA URA

FESTER STER

ARMANDO MANDO

MAJOR JOR

6 rows selected.

The rest of the target string is returned.

INPUT:

SQL> **SELECT * FROM SSN_TABLE;**

OUTPUT:

SSN_____

300541117

301457111

459789998

3 rows selected.

ANALYSIS:

Reading the results of the preceding output is difficult--Social Security numbers usually have dashes. Now try something fancy and see whether you like the results:

INPUT:

SQL> **SELECT SUBSTR(SSN,1,3)||'-'||SUBSTR(SSN,4,2)||'-'||SUBSTR(SSN,6,4) SSN**

2 **FROM SSN_TABLE;**

OUTPUT:

SSN_____

300-54-1117

301-45-7111

459-78-9998

3 rows selected.

DMS.KU

NOTE: This particular use of the substr function could come in very handy with large numbers using commas such as 1,343,178,128 and in area codes and phone numbers such as 317-787-2915 using dashes.

Here is another good use of the SUBSTR function. Suppose you are writing a report and a few columns are more than 50 characters wide. You can use the SUBSTR function to reduce the width of the columns to a more manageable size if you know the nature of the actual data. Consider the following two examples:

INPUT:

SQL> **SELECT NAME, JOB, DEPARTMENT FROM JOB_TBL;**

OUTPUT:

NAME_____

JOB_____DEPARTMENT_____

ALVIN SMITH

VICEPRESIDENT MARKETING

1 ROW SELECTED.

ANALYSIS:

Notice how the columns wrapped around, which makes reading the results a little too difficult. Now try this select:

INPUT:

SQL> **SELECT SUBSTR(NAME, 1,15) NAME, SUBSTR(JOB,1,15) JOB,**
DEPARTMENT
2 **FROM JOB_TBL;**

OUTPUT:

NAME_____JOB_____DEPARTMENT_____

ALVIN SMITH VICEPRESIDENT MARKETING

Much better!

TRANSLATE

The function TRANSLATE takes three arguments: the target string, the FROM string, and the TO string. Elements of the target string that occur in the FROM string are translated to the corresponding element in the TO string.

INPUT:

```
SQL> SELECT FIRSTNAME, TRANSLATE(FIRSTNAME
  2  '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
  3  'NNNNNNNNNNAAAAAAAAAAAAAAAAAAAAAAAA)
  4  FROM CHARACTERS;
```

OUTPUT:

```
FIRSTNAME       TRANSLATE(FIRST
--------------- ---------------
kelly           kelly
CHUCK           AAAAA
LAURA           AAAAA
FESTER          AAAAAA
ARMANDO         AAAAAAA
MAJOR           AAAAA
6 rows selected.
```

Notice that the function is case sensitive.

INSTR

To find out where in a string a particular pattern occurs, use INSTR. Its first argument is the target string. The second argument is the pattern to match. The third and forth are numbers representing where to start looking and which match to report. This example returns a number representing the first occurrence of O starting with the second character:

INPUT:

```
SQL> SELECT LASTNAME, INSTR(LASTNAME, 'O', 2, 1)
  2  FROM CHARACTERS;
```

OUTPUT:

```
LASTNAME        INSTR(LASTNAME,'O',2,1)
```

--- ---

PURVIS 0

TAYLOR 5

CHRISTINE 0

ADAMS 0

COSTALES 2

KONG 2

6 rows selected.

ANALYSIS:

The default for the third and fourth arguments is 1. If the third argument is negative, the search starts at a position determined from the end of the string, instead of from the beginning.

LENGTH

LENGTH returns the length of its lone character argument. For example:

INPUT:

SQL> **SELECT FIRSTNAME, LENGTH(RTRIM(FIRSTNAME))**

2 **FROM CHARACTERS;**

OUTPUT:

FIRSTNAME LENGTH(RTRIM(FIRSTNAME))

--- ---

kelly 5

CHUCK 5

LAURA 5

FESTER 6

ARMANDO 7

MAJOR 5

6 rows selected.

ANALYSIS:

Note the use of the RTRIM function. Otherwise, LENGTH would return 15 for every value.

Conversion Functions

These three conversion functions provide a handy way of converting one type of data to another. These examples use the table CONVERSIONS.

INPUT:

SQL> **SELECT * FROM CONVERSIONS;**

OUTPUT:

```
NAME            TESTNUM
--------------- ---------
40              95
13              23
74              68
```

The NAME column is a character string 15 characters wide, and TESTNUM is a number.

TO_CHAR

The primary use of TO_CHAR is to convert a number into a character. Different implementations may also use it to convert other data types, like Date, into a character, or to include different formatting arguments. The next example illustrates the primary use of TO_CHAR:

INPUT:

SQL> **SELECT TESTNUM, TO_CHAR(TESTNUM)**
2 **FROM CONVERT;**

OUTPUT:

```
 TESTNUM TO_CHAR(TESTNUM)
--------- ----------------
      95              95
      23              23
      68              68
```

Not very exciting, or convincing. Here's how to verify that the function returned a character string:

INPUT:

```
SQL> SELECT TESTNUM, LENGTH(TO_CHAR(TESTNUM))
  2  FROM CONVERT;
```

OUTPUT:

```
  TESTNUM LENGTH(TO_CHAR(TESTNUM))
--------- ------------------------
       95                        2
       23                        2
       68                        2
```

ANALYSIS:

LENGTH of a number would have returned an error. Notice the difference between TO CHAR and the CHR function discussed earlier. CHR would have turned this number into a character or a symbol, depending on the character set.

TO_NUMBER

TO_NUMBER is the companion function to TO_CHAR, and of course, it converts a string into a number. For example:

INPUT:

```
SQL> SELECT NAME, TESTNUM, TESTNUM*TO_NUMBER(NAME)
  2  FROM CONVERT;
```

OUTPUT:

```
NAME           TESTNUM TESTNUM*TO_NUMBER(NAME)
-------------- -------- ----------------------
40                  95                   3800
13                  23                    299
74                  68                   5032
```

ANALYSIS:

This test would have returned an error if TO_NUMBER had returned a character.

Miscellaneous Functions

Here are three miscellaneous functions you may find useful.

GREATEST and LEAST

These functions find the GREATEST or the LEAST member from a series of expressions. For example:

INPUT:

SQL> **SELECT GREATEST('ALPHA', 'BRAVO','FOXTROT', 'DELTA')**

2 **FROM CONVERT;**

OUTPUT:

GREATEST

FOXTROT

FOXTROT

FOXTROT

ANALYSIS:

Notice GREATEST found the word closest to the end of the alphabet. Notice also a seemingly unnecessary FROM and three occurrences of FOXTROT. If FROM is missing, you will get an error. Every SELECT needs a FROM. The particular table used in the FROM has three rows, so the function in the SELECT clause is performed for each of them.

INPUT:

SQL> **SELECT LEAST(34, 567, 3, 45, 1090)**

2 **FROM CONVERT;**

OUTPUT:

LEAST(34,567,3,45,1090)

3

3

3

As you can see, GREATEST and LEAST also work with numbers.

USER

USER returns the character name of the current user of the database.

INPUT:

SQL> **SELECT USER FROM CONVERT;**

OUTPUT:

USER

PERKINS

PERKINS

PERKINS

There really is only one of me. Again, the echo occurs because of the number of rows in the table. USER is similar to the date functions explained earlier today. Even though USER is not an actual column in the table, it is selected for each row that is contained in the table.

Summary

It has been a long day. We covered 47 functions--from aggregates to conversions. You don't have to remember every function--just knowing the general types (aggregate functions, date and time functions, arithmetic functions, character functions, conversion functions, and miscellaneous functions) is enough to point you in the right direction when you build a query that requires a function.

EXERCISE:

1. What are the advantages and disadvantages of SQL?

……

……

2. Create Room, Booking, and Guest tables with the following constraints:

(a) Room type must be one of Single, Double, or Family.

(b) Price must be between Rs.100/- and Rs.1000/-.

(c) Room Number must be between 1 and 100.

(d) Booking "date from" and "date to" must be greater than today's date.

(e) The same room cannot be double booked.

(f) The same guest cannot have overlapping bookings.

……..……
……..……

3. Define the function of each of the clauses in the SELECT statement. What are the restrictions imposed on these clauses?

……..……
……

4. Consider the supplier relations.

SUPPLIER

SNO (SUPPLIER NUMBER)	**SNAME (SUPPLIER NAME)**	**STATUS**	**CITY**
S1	PHI	30	CALICUT
S2	TMHC	30	MUMBAI
S3	WILEY	20	CHENNAI
S4	PEARSON	40	DELHI

S5	THOMPSON	10	DELHI

PARTS

SNO	PART_NO	QUANTITY
S1	P1	300
S1	P2	200
S2	P1	100
S2	P2	400
S3	P2	200
S4	P2	200

a) Get supplier numbers for suppliers with status > 20 and city is Delhi

b) Get Supplier Numbers and status for suppliers in Delhi in descending order of status.

c) Get unique supplier names for suppliers who supply part P2 without using IN operator.

e) Give the same query above by using the operator IN.

f) Get supplier names for suppliers who supply part P1.

g) Suppose for the supplier S5 the value for status is NULL instead of 10. Get supplier numbers for suppliers greater than 25. (use NULL)

h) Get unique supplier numbers supplying parts. (use the built-in function count).

i) For each part supplied, get the part number and the total quantity supplied for that part. (Hint: The query using GROUP BY).

j) Get part numbers for all parts supplied by more than one supplier. (GROUP BY).

k) For all those parts, for which the total quantity supplied is greater than 300, get the part number and the maximum quantity of the part supplied in a single supply. Order the result by descending part number within those maximum quantity values.

l) Double the status of all suppliers in Delhi. (UPDATE Operation).

m) Let us consider the table TEMP has one column, called PNO. Enter into TEMP part numbers for all parts supplied by S2.

n) Add part p7.

o) Delete all the suppliers in Mumbai and also the parts concerned.

p) Add a new column phone number to supplier table

Lesson-12

DATA CONTROL LANGUAGE

The data control basically refers to commands that allow system and data privileges to be passed to various users. These commands are normally available to database administrator. Let us look into some data control language commands:

Create a new user: To create a new user, we use the CREATE USER command. The user has to be assigned a password which is also done by the same command as shown in the syntax below:

CREATE USER < user name > IDENTIFIED BY < Password> ;

Example 28:

CREATE USER MCOM1 IDENTIFIED BY DDE12011;

Granting privileges to the users: The GRANT command is used to provide database access permission to users. The grants are of two types

(1) System level grants/permissions

(2) Object level grants/permissions

The general syntax is:

GRANT <privilege> [ON <object>] TO <user>;

Example 29:

GRANT CREATE SESSION TO MCOM1;

(This command provides system level permission on creating a session)

Example 30:

GRANT SELECT ON EMPLOYEE TO MCOM1;

(Object level permission on table EMPLOYEE)

Example 31:

GRANT SELECT, INSERT, UPDATE, DELETE ON EMPLOYEE TO MCOM1;

Example 32:

GRANT SELECT, UPDATE ON EMPLOYEE TO MCOM1, MCOM2; (Two users)

Example 33:

GRANT ALL ON EMPLOYEE TO PUBLIC;

(All permission to all users, do not use it. It is very dangerous for database)

Revoking Permissions: The REVOKE command is used to cancel the permission granted to users.

Example 34:

REVOKE ALL ON EMPLOYEE FROM MCOM1; (All permissions will be cancelled)

You can also revoke only some of the permissions.

Example 35:

REVOKE DELETE ON EMPLOYEE FROM MCOM1;

DELETING USERS: The DROP command is used to delete users from the system. The syntax is:

DROP USER <user name>;

Example 36:

DROP USER MCOM1;

Accessing information about users: DBMSs generally store information about all database objects in the data dictionary. This data about data is called metadata. In most RDBMSs the data is stored in tables, generally called system tables. The information about users and their permissions is also stored in these tables.

1. **Object level permissions:** With the help of data dictionary you can view the permissions to user. Let us take the table name from oracle. In oracle the name of the table containing these permissions is user_tab_privs.

DESCRIBE USER_TAB_PRIVS ;

The DESCRIBE OR DESC command is used to describe the schema of a table i.e. its name, attributes, data types and constraints. This information helps us in retrieving selective information from the tables.

SELECT * FROM USER_TAB_PRIVS;

This query displays all information about users contained in the table named USER_TAB_PRIVS.

2. **System level permissions:** With the help of data dictionary tables you can see the system level permissions and privileges. One of the tables that contain this information in Oracle is user_sys_privs.

DESCRIBE USER_SYS_PRIVS;

SELECT * FROM USER_SYS_PRIVS;

All these commands are very specific to a data base system and may be different on different DBMSs.

Lesson-13

DATABASE OBJECTS: VIEWS, SEQUENCES, AND INDEXES

Some of the important concepts in a database system are the views and indexes. Views are a mechanism that can support data independence and security. Indexes help in better query responses. Let us discuss about them along with two more concepts sequences and synonyms in more details.

Views: A view is often referred as a virtual table. It is like a window through which data from tables can be viewed or changed. The table(s) on which a view is based is

called Base table. The view is not stored as a table but as a SELECT statement in the data dictionary. When a user wants to retrieve data using view, following steps are followed.

1. View definition is retrieved from data dictionary table. For example, the view definitions in oracle are to be retrieved from table name USER-VIEWS.
2. Checks access privileges for the base table(s) of the view.
3. Converts the view query into an equivalent operation on the underlying base table.

Advantages of view:

- It restricts data access.
- It makes complex queries look easy.
- It allows data independence.
- It helps to present different views of the same data.
- Instead of using join queries on multiple tables, simple queries can be used on views.

Type of views: Views are divided into two categories based on the complexity of the views. These are:

1. **Simple views**
2. **Complex Views**

FEATURE	SIMPLE VIEWS	COMPLEX VIEWS
Number of tables	One	More
Contain functions	No	Yes
Contain data groups	No	Yes
Data manipulation	Allowed	Not always

Figure: Simple views vs. complex views

DMS.KU

Let us look into some of the examples. To see all the details about existing views in Oracle, we can use the USER_VIEWS table:

SELECT* FROM USER_VIEWS;

Creating a view:

- A query can be embedded within the CREATE VIEW STATEMENT.
- A query can contain complex select statements including join, groups and sub-queries
- A query that defines the view cannot contain an order by clause.
- DML operation (delete/modify/add) cannot be applied in the following cases:

Delete: You can't delete if view contains following:

- ❖ Group functions
- ❖ A group by clause
- ❖ A distinct keyword

Modify: you cannot modify if view contains following:

- ❖ Group functions
- ❖ A group by clause
- ❖ A distinct keyword
- ❖ Columns defined by Expressions

Insert: you cannot insert if view contains following:

- ❖ Group functions
- ❖ A group by clause
- ❖ A distinct keyword
- ❖ Columns defined by Expressions
- ❖ There are Not Null Columns in the base tables that are not selected by view.

DMS,KU

Example 37:

Create a view named E_SALARY having minimum, maximum and average salary for each department.

CREATE VIEW E_SALARY (NAME, MINSAL, MAXSAL, AVGSAL) AS

SELECT D.D_NAME,MIN(E.SALARY),MAX(E.SALARY),AVG(E.SALARY)

FROM EMPLOYEE E, DEPARTMENT D

WHERE E.DNO=D.DNO

GROUP BY D.DNAME;

To see the result of the above query you can give the following command:

SELECT * FROM E_SALARY;

You may get some sample output like:

NAME	MINSAL	MAXSA	AVGSAL
TBS	1300	5000	32916.666
BIOTECH	800	3000	32175.323
DDE	950	2850	27566.666

To see the structure of the E_SALARY view, you can give the following command:

DESCRIBE E_SALARY;

Name	Null?	Type

NAME	VARCHAR2 (14)
MINSAL	NUMBER
MAXSAL	NUMBER
AVGSAL	NUMBER

Creating views with check option: This option restricts those updates of data values that cause records to go off the view. The following example explains this in more detail:

Example 38:

To create a view for employees of Department number 20, such that user cannot change the department number from the view:

```
CREATE OR REPLACE VIEW EMPD20 AS
SELECT ENO, ENAME, SALARY
FROM EMP
WHERE DNO=30
WITH CHECK OPTION;
```

Now the user cannot change the department number of any record of view EMPD20. If this change is allowed then the record in which the change has been made will go off the view as the view is only for department-20. This is restricted because of use of WITH CHECK OPTION clause.

Creating views with Read only option: In the view definition this option is used to ensure that no DML operations can be performed on the view.

Creating views with Replace option: This option is used to change the definition of the view without dropping and recreating it or re-granting object privileges previously granted on it.

Sequences: Sequences-

- Automatically generate unique numbers
- Are sharable
- Are typically used to create a primary key value
- Replace application code
- Speed up the efficiency of accessing sequence Values when cached in memory.

The general syntax for creating a sequence is:

CREATE [OR MODIFY] SEQUENCE <seq. name>

START WITH <start value>

INCREMENT BY <Increment>

MAX VALUE <Last value>;

Example 39:

Create a sequence named SEQ12 that starts at 101, has a step of 1 and can take maximum value as 1000.

CREATE SEQUENCE SEQ12

START WITH 101

INCREMENT BY 1

MAX VALUE 1000;

The following sequence of commands demonstrate the use of the sequence SEQ12.

Suppose a table person exists as:

SELECT * FROM PERSON;

CODE	NAME	ADDRESS
100	RAMZAN	SRINAGAR

Now, if we give the command:

INSERT INTO PERSON

VALUES (SEQ12.NEXTVAL, 'Riyaz', 'Srinagar')

On execution of this statement, one record will be inserted into the table PERSON. The value for CODE attribute will be taken from the sequence SEQ12. The keyword NEXTVAL always points to the next value in the sequence. As our sequence starts at 101, therefore this will be the first value generated by the SEQ12.NEXTVAL.

Now, give the following command to see the output:

SELECT * FROM PERSON;

CODE	NAME	ADDRESS
100	RAMZAN	SRINAGAR
101	RIYAZ	SRINAGAR

The descriptions of sequences such as minimum value, maximum value, step or increment are stored in the data dictionary. For example, in oracle it is stored in the table USER_SEQUENCES. You can see the description of sequences by giving the SELECT command on user_sequences table.

Sometimes gaps in sequence values can occur because:

- A rollback occurs that is when a statement changes are not accepted.
- The system crashes
- A sequence is used in another table.

Modifying a sequence: Sequences can be modified to generate different values.

Example 40:

```
ALTER SEQUENCE SEQ12
  INCREMENT 2
  MAXVALUE 2000;
```

Removing a sequence: Sequences can be deleted by the DROP command.

Example 41:

```
DROP SEQUENCE SEQ12;
```

Indexes: Data can be retrieved from a database using two methods. The first method, often called the Sequential Access Method, requires SQL to go through each record looking for a match. This search method is inefficient, but it is the only way for SQL to locate the correct record.

Adding indexes to database enables SQL to use the Direct Access Method. SQL uses a treelike structure to store and retrieve the index's data. Pointers to a group of data are stored at the top of the tree. These groups are called nodes. Each node contains pointers to other nodes. The nodes pointing to the left contain values that are less than its parent node. The pointers to the right point to values greater than the parent node.

The database system starts its search at the top node and simply follows the pointers until it is successful.

Some of the basic properties of indexes are:

- An Index is a schema Object
- Indexes can be created explicitly or automatically
- Indexes are used to speed up the retrieval of records
- Indexes are logically and physically independent of the table. It means they can be created or dropped at any time and have no effect on the base tables or other indexes.
- When a table is dropped corresponding indexes are also dropped.

Creation of Indexes: Indexes can be created in two ways:

1. Automatically or Implicitly
2. Manually or Explicitly

Automatically: When a primary key or Unique constraint is defined in a table definition then a unique index is created automatically.

Manually: User can create non-unique indexes on columns to speed up access time to records.

The general syntax to create an index is as follows:

CREATE INDEX <index name>

ON <table name(column_name1, [column_name2], ...)>;

Example 41:

The following commands create index on employee name and employee name, department number respectively.

CREATE INDEX EMP_ENAME_IDX ON EMPLOYEE (ENAME);

CREATE INDEX EMP_MULTI_IDX ON EMPLOYEE (ENAME, DEPTNO);

Finding details about created indexes: The data dictionary contains the name of index, table name and column names. For example, in Oracle a USER_INDEXES and USER_IND_COLUMNS view contains the details about user created indexes.

Remove an index: The DROP INDEX command is used to remove an index from data dictionary. The syntax is:

DROP INDEX <index name>;

Example 42:

DROP INDEX EMP_ENAME_IDX;

Note:- Indexes cannot be modified.

Lesson-14

THE JOIN OPERATION

In RDBMS more than one table can be handled at a time by using join operation. Join operation is a relational operation that causes two tables with a common domain to be combined into a single table or view. SQL specifies a join implicitly by referring the matching of common columns over which tables are joined in a WHERE clause. Two tables may be joined when each contains a column that shares a common domain with the other. The result of join operation is a single table. Selected columns from all the tables are included. Each row returned contains data from rows in the different input tables where values for the common columns match. An important rule of table handling is that there should be one condition within the WHERE clause for each pair of tables being joined. Thus if two tables are to be combined, one condition would be necessary, but if three tables (X, Y, Z) are to be combined then two conditions would be necessary because there are two pairs of tables (X-Y and Y-Z) OR (X-Z and Y-Z), and so forth. There are several possible types of joins in relational database queries.

The different types of joins are:

a) Equi join
b) Non-equi join
c) Natural join
d) Outer join:
    1. Left uter join
    2. Right outer join
    3. Full outer join
e) Self join

1. **Equi Join:** A join in which the joining condition is based on equality between values in the common columns. Common columns appear (redundantly) in the result table. Consider the following relations:

   CUSTOMER (cust_id, cust_name, ……..) and

   ODER (cust_id, order_id, order_date, …….)

Example 43:

Find the names of customers who have placed orders?

The required information is available in two tables, CUSTOMER and ORDER. The SQL solution requires joining the two table using equi join as shown below.

SELECT CUSTOMER.CUST_ID, ORDER.CUST_ID, CUST_NAME, ORDER_ID

FROM CUSTOMER, ORDER

WHERE CUSTOMER.CUST_ID=ORDER.CUST_ID;

Note that the dot (.) operator is used to remove the ambiguity in selecting the attributes from the two tables.

2. **Natural Join:** The equi join operation results in redundant data as the common joining column from both the tables is selected. The natural join is the same like Equi join except that one of the duplicate columns is eliminated in the result table. The natural join is the most commonly used form of join operation.

Example 44:

SELECT CUSTOMER.CUT_ID, CUST_NAME, ORDER_ID

FROM CUSTOMER, ORDER

WHERE CUSTOMER.CUST_ID=ORDER.CUST_ID;

3. **Outer Join:** The use of Outer Join is that it even joins those tuples that do not have matching values in common columns but are still included in the result table. Outer join places null values in columns where there is not a match between tables. A condition involving an outer join is that it cannot use the IN operator or cannot be linked to another condition by the OR operator. The outer join has three types:
   a. Left outer join:
   b. Right outer join
   c. Full outer join

Example 45: The following is an example of left outer join (which only considers the non-matching tuples of table on the left side of the join expression).

SELECT CUSTOMER.CUTOID, CUSTONAME, ORDERID

FROM CUSTOMER LEFT OUTER JOIN ORDER

WHERE CUSTOMER.CUSTOID = ORDER.CUSTOID;

Output: The following result assumes a CUSTID in CUSTOMER table who have not issued any order so far.

CUSTOID CUSTONAME ORDERID

---------- ---------- ----------

10 Pooja Enterprises 1001

12 Estern Enterprises 1002

3 Impressions 1003

15 South Enterprises NULL

The other types of outer join are the Right outer join or complete outer join.

(4) Self-Join: It is a join operation where a table is joined with itself. Consider the following sample partial data of EMPLOYEE table:

E_ID	ENAME	MGR_ID	……..
101	TARIQ	104	--
102	TANVEER	104	--
103	MUSHTAQ	101	--
104	RAFI	NULL	--
-------	---------	---------	--

Example 46: Find the name of each employee's manager .

SELECT E.ENAME, M.ENAME

FROM EMPLOYEE E, EMPLOYEE M

WHERE E.MGR_ID=M.E_ID;

Check Your Progress

1) Discuss how the Access Control mechanism of SQL works.

……..……

……

2) Consider Hotel schema consisting of the tables Hotel, Booking and Guest and Room

1. CREATE TABLE Hotel

(

hotelNo Number NOT NULL,

hotelName VARCHAR(20) NOT NULL,

city VARCHAR(50) NOT NULL,

PRIMARY KEY (hotelNo)

);

2. CREATE TABLE Booking

(

hotelNo Number NOT NULL,

guestNo Number NOT NULL,

dateFrom Date NOT NULL,

dateTo Date NOT NULL,

roomNo Number NOT NULL,

```
  PRIMARY KEY (hotelNo, guestNo, dateFrom),
  FOREIGN KEY (hotelNo) REFERENCES Hotel
   ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (guestNo) REFERENCES Guest
   ON DELETE NO ACTION ON UPDATE CASCADE,
  FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
   ON DELETE NO ACTION ON UPDATE CASCADE
);
```

3. CREATE TABLE Guest

```
    (
        guestNo Number  PRIMARY KEY,
        guestName  VARCHAR(20)  NOT NULL,
        guestAddress VARCHAR(50)  NOT NULL
      );
```

4. CREATE TABLE Room

```
(
  roomNo  Number  NOT NULL,
  hotelNo  Number  NOT NULL,
```

DMS.KU

RoomType Char NOT NULL DEFAULT 'S'

price Number NOT NULL,

PRIMARY KEY (roomNo, hotelNo),

FOREIGN KEY (hotelNo) REFERENCES Hotel

ON DELETE CASCADE ON UPDATE CASCADE

);

a) Create a view containing the hotel name and the names of the guests staying at the hotel.

……
……

b) Give the users Manager and Director full access to views Hotel_Data and Booking_Out_Today, with the privilege to pass the access on to other users.

NESTED QUERIES

By now we have discussed the basic commands including data definition and data manipulations. Now let us look into some more complex queries in this section.

Sub-queries

Some of the basic issues of sub-queries are:

- A sub-query is a SELECT statement that is embedded in a clause of another SELECT statement. They are often referred to as a NESTED SELECT or SUB SELECT or INNER SELECT.
- The sub-query (inner query) executes first before the main query. The result of the sub-query is used by the main query (outer query).
- Sub-query can be placed in WHERE or HAVING or FROM clauses.

The general syntax of sub queries is:

SELECT<select_list>

FROM<table>

WHERE expr OPERATOR

(SELECT <select_list>

FROM <TABLE>WHERE);

Operator includes a comparison operator (single or multiple row operators)

Single row operators: **>, =, >=, <, <=, <>**

Multiple row operators: **IN, ANY, ALL**

- Order by clause cannot be used in sub-query, if specified it must be the last clause in the main select statement.
- Types of sub-queries:
  - Single row sub-query: It returns only one row from the inner select statement.
  - Multiple row sub-query: It returns more than one row from the inner select statement
  - Multiple column sub-query: It returns more than one column from the inner select statement.
- Single row operators are used with single row sub queries and multiple row operators are used with multiple row sub queries.
- The Outer and Inner queries can get data from different tables.
- Group Functions can be used in sub queries.

Consider the following relation EMP. Let us create some sub-queries for them

ENO ENAME JOB SALARY DNO

1234	Tariq	Manager	30000	10
1234	Tahir	Analyst	21000	10
1236	Rafi	Associate	28000	20
1237	Suhaib	Professor	80000	20
1238	Zuhaib	Professor	82000	20

Example 47: Get the details of the person having the minimum salary.

```
SELECT ENAME, JOB, SALARY
  FROM EMP
  WHERE SALARY =  (SELECT MIN (SALARY)
    FROM EMP);
```

Example 48: Display the employees whose job title is the same as that of employee 1237 and salary is more than the salary of employee 1234.

```
 SELECT ENAME, JOB
FROM EMP
   WHERE JOB =   (SELECT JOB
     FROM EMP
      WHERE ENO = 1237)
    AND SALARY > (SELECT SALARY
     FROM EMP
```

WHERE ENO=1234);

Example 49: To find the minimum salary in those departments whose minimum salary is greater than minimum salary of department number 10.

```
SELECT DNO, MIN(SALARY)
FROM EMP
  GROUP BY DNO
        HAVING MIN(SALARY) >  ( SELECT MIN (SALARY)
  FROM EMP
  WHERE DNO = 10);
```

Example 50: Find the name, department number and salary of employees drawing minimum salary in that department.

```
SELECT ENAME, SALARY, DNO
 FROM EMP
 WHERE SALARY IN (SELECT MIN (SALARY)
 FROM EMP
  GROUP BY DNO);
```

Example 51: Find the salary of employees who are not 'ANALYST' but get a salary less than or equal to any person employed as 'ANALYST'.

```
 SELECT ENO, ENAME, JOB, SALARY
   FROM EMP
    WHERE SALARY <= ANY (SELECT SALARY
```

FROM EMP WHERE JOB = 'ANALYST')

AND JOB<>'ANALYST';

Example 52: Find out the employee who draws a salary more than the average salary of all the departments.

SELECT ENO, ENAME, JOB, SALARY

FROM EMP

WHERE SALARY> ALL (SELECT AVG (SALARY)

FROM EMP

GROUP BY DNO);

Check Your Progress

1. What is the difference between a sub-query and a join? Under what circumstances would you not be able to use a sub-query? ……
……
2. Use the Hotel schema defined in question number 2 (Check Your Progress 2) and answer the following queries:

- List the names and addresses of all guests in Srinagar, alphabetically ordered by name.
- List the price and type of all rooms at the GRAND Hotel.
- List the rooms that are currently unoccupied at the Shangrila Hotel.
- List the number of rooms in each hotel.
- What is the most commonly booked room type for hotels in Srinagar
- Update the price of all rooms by 5%.

SUMMARY

This unit has introduced the SQL language for relational database definition, manipulation and control. Each schema definition that describes the database objects is stored in data dictionary/ system catalog. Information contained in system catalog is maintained by the DBMS itself, rather than by the users of DBMS.

The data definition language commands are used to define a database, including its creation and the creation of its tables, indexes and views. Referential integrity constraints are also maintained through DDL commands. The DML commands of SQL are used to load, update and query the database. DCL commands are used to establish user access to the database. SQL commands directly affect the base tables, which contain the raw data, or they may affect database view, which has been created. The basic syntax of an SQL SELECT statement contains the following keywords: SELECT, FROM, WHERE, ORDER BY, GROUP BY and HAVING.

SELECT determines which attributes will be displayed in the query result table. FROM determines which tables or views will be used in the query. WHERE sets the criteria of the query, including any joins of multiple tables, which are necessary. ORDER BY determines the order in which the result will be displayed. GROUP BY is used to categorize results. HAVING is used to impose condition with GROUP BY.

FURTHER READINGS

Fundamentals of Database Systems; Almasri and Navathe; Pearson Education Limited; Fourth Edition; 2004.

A Practical Approach to Design, Implementation, and Management; Thomas Connolly and Carolyn Begg; Database Systems, Pearson Education Limited; Third Edition; 2004.

The Complete Reference; Kevin Lonely and George Koch; Oracle 9i, Tata McGraw-Hill; Fourth Edition; 2003.

Jeffrey A. Hoffer, Marry B. Prescott and Fred R. McFadden; Modern Database Management; Pearson Education Limited; Sixth Edition; 2004