Advance SQL

Part-1

Unit - I

1

BASIC STRUCTURED QUERY LANGUAGE – I

Unit Structure

1. OBJECTIVES

After going through this chapter, you will be able to

- Understand RDBMS
- What SQL can do?
- Classify SQL elements
- Classify SQL Commands
- Understand common data types in SQL
- Create and Manage Tables
- Apply Constraints to Tables

2. INTRODUCTION

The amount of information available to us is literally exploding, and the value of data as an organizational asset is widely recognized. To get the most out of their large and complex datasets, users require tools that simplify the tasks of managing the data and extracting useful information in a timely fashion. Otherwise, data can become a liability, with the cost of acquiring it and managing it far exceeding the value derived from it.

A database management system, or DBMS, is software designed to assist in maintaining and utilizing large collections of data. The collection of data, usually referred to as database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

A **data model** is a collection of high-level data description constructs that hide many low-level storage details. A DBMS allows a user to define the data to be stored in terms of a data model. Most database management systems today are based on the **relational data model**. A relational database consists of a collection of tables (mathematical concept of relation). A row in a table represents a relationship among a set of values. Informally, a table is an entity set, and a row is an entity.

The relational model is very simple and elegant: a database is a collection of one or more *relations,* where each relation is a table with rows and columns. This simple tabular representation enables even novice users to understand the contents of a

database, and it permits the use of simple, high-level languages to query the data.

2. What is RDBMS?

RDBMS stands for Relational Database Management System. RDBMS is the basis of SQL, and for all modern database system like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access. It is a software system that manages the storage of data contained in a database. The relational model was first proposed by Dr. E. F. Codd in 1970. The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

1.2.1 Concepts of Relational Database

Relational Database is a structure consisting of a set of objects related together and data integrity methods which aims in the efficient storage of electronic data. In other words, it is a place to store the data, a way to create and retrieve the data, and a way to make sure that the data is logically consistent. The objects can be of various types like tables, indexes, queries etc.

3. INTRODUCTION TO SQL

SQL stands for "**Structured Query Language**" and can be pronounced as "SQL" or "sequel – (Structured English Query Language)". It is a query language used for accessing and modifying information in the database. It has become a Standard Universal Language used by most of the relational database management systems (RDBMS). SQL is tied very closely to the relational model. Few of the SQL commands used in SQL programming are SELECT Statement, UPDATE Statement, INSERT INTO Statement, DELETE Statement, WHERE Clause, ORDER BY Clause, GROUP BY Clause, ORDER Clause, Joins, Views, GROUP Functions, Indexes etc.

In a simple manner, SQL is a non-procedural, English-like language that processes data in groups of records rather than one record at a time. Few functions of SQL are:

- store data
- modify data
- retrieve data
- delete data
- create tables and other database objects

The SQL language has several parts:

Data-Definition Language (DDL): The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

Interactive Data-Manipulation Language (DML): The SQL DML includes a query language based on both the relational algebra and the tuple relational calculus. It also includes commands to insert into tuples, delete tuples from, and modify tuples in the database.

View Definition: The SQL DDL includes commands for defining views.

Transaction Control: SQL includes commands for specifying the beginning and ending of transactions.

Embedded SQL and Dynamic SQL: Embedded and Dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, Java, PL/I, Cobol, Pascal, and Fortran.

Integrity: The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.

Authorization: The SQL DDL includes commands for specifying access rights to relations and views.

1.4 WHAT CAN SQL DO?

SQL can execute queries against a database

1. SQL can retrieve data from a database
2. SQL can insert records in a database
3. SQL can update records in a database
4. SQL can delete records from a database
5. SQL can create new databases
6. SQL can create new tables in a database
7. SQL can create stored procedures in a database
8. SQL can create views in a database
9. SQL can set permissions on tables, procedures, and views

1.5 SQL LANGUAGE ELEMENTS

The SQL language is subdivided into several language elements, including:

- **Clauses**, which are constituent components of statements and queries.

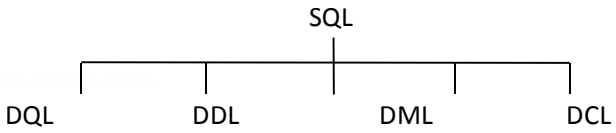
- **Expressions**, which can produce either scalar values or tables consisting of columns and rows of data.
- **Predicates**, which specify conditions that can be evaluated to SQL three-valued logic (3VL) or Boolean (true/false/unknown) truth values and which are used to limit the effects of statements and queries, or to change program flow.
- **Queries**, which retrieve the data based on specific criteria. This is the most important element of **SQL**.
- **Statements**, which may have a persistent effect on schemata and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
  - SQL statements also include the semicolon (";") statement terminator.
- **Insignificant whitespace** is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

The following table shows the most common data types

Data Type	Description
NUMBER	Holds numeric data of any precision. The precision can range from 1 to 38.
CHAR(W)	Holds fixed length alphanumeric data up to w width
VARCHAR2(W)	Holds variable length alphanumeric data up to w width. A varchar2 value can contain up to 4000 bytes of data
DATE	DATE is a data type used to store date and time values in a 7-byte structure
RAW(size)	Raw binary data of varying length(size) bytes. Maximum size is 2000 bytes
TIMESTAMP	Timestamp is an extension of the Date data type that can store date and time data

1.6 CLASSIFICATION OF SQL COMMANDS

In order to learn SQL we will have to learn various commands or statements supported by SQL language. All the SQL commands can be broadly classified into five categories as shown in the figure below.



1. Data Query Language (DQL)

These commands are used to view records from tables. The actual data in the form of tables is stored onto the hard disk and DQL commands enable us to view the information/records stored in the database, for example, using the SELECT command.

2. Data Definition Language (DDL)

DDL is a subset of SQL commands that allows you to make changes to the structure of the database. The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specifies links between tables, and imposes constraints between tables. Table structure refers to the number of columns and the data types of columns and constraints which can be applied or revoked as desired, using commands like CREATE, DROP, ALTER.

3. Data Manipulation Language (DML)

DML describes the portion of SQL that allows you to manipulate or control your data. For adding new records, removing existing records or changing values of existing records we will use DML statements like INSERT, DELETE, UPDATE.

4. Data Control Language (DCL)

Database Management Systems are multi user oriented. Each user has his own assigned area where his data is stored. By default one user cannot view, modify or delete the information stored in another user's area. The owner can grant the privilege of modifying the data owned by him to another user. These commands control how and to what extent one user can view, modify and delete information in another user's area using commands like GRANT, REVOKE.

5. Transaction Control Language (TCL)

Each operation which is done on database is called a Transaction. Transaction can be addition of new record, modification in values of existing records or deletion of existing records. Each transaction can be done permanently or can be

undone by using the transaction control statements, like COMMIT, ROLLBACK, SAVEPOINT.

1.7 CREATING AND MANAGING TABLES

Structured Query Language (SQL) is the language used to manipulate relational databases. In the relational model data is stored in structures called relations or tables. A table is the fundamental building block of a database application. Tables are used to store data on various entities like employees, products, customers, orders, sales etc.

There are a few things that one should note before typing and executing SQL commands:

- Commands may be on a single line, or many lines.
- For the sake of readability, place different clauses on separate lines and make use of tabs and indents.
- SQL command words cannot be split or abbreviated.
- SQL commands are not case sensitive.
- Place a semicolon(;) at the end of the last clause.

1.7.1 How to create a table?

Data Definition Language statements mentioned in the previous chapter allows you to make changes to the structure of the database. The create table statement is used to create a table object.

Syntax:

CREATE TABLE <tablename>

(<attribute1 name> <data type> (size),

<attribute2 name> <data type> (size),);

For example to create a STUDENT table, the command will be as follos:

```
CREATE TABLE student
        (
          rollno char(4),
          name varchar2(10),
          date_of_birth date,
          marks number(3)
        );
```

Rules for creating tables are as follows:

- The name of the table can be of 30 characters at the maximum.
- Alphabets from A to Z and 0 to 9 are allowed.
- Special characters like under score (_) are allowed.
- SQL reserved keywords like SELECT, WHERE etc are not allowed.

1.8 APPLYING DATA CONSTRAINTS

To maintain database integrity we need to enforce some rules on data being stored in a table. There are several ways of controlling what kind of data can be input into a table. The various controls/constraints are as follows:

- PRIMARY KEY
- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- FOREIGN KEY

1.8.1 Classification of Constraints

Constraints can be broadly classified as:

- Column Level
- Table Level

Column Level Constraints

When a constraint is applied to a single column then it is called a Column Level Constraint. This constraint will affect the values being entered for that particular column only irrespective of values of other columns.

Table Level Constraints

When a single constraint is applied to more than one column then it is called Table Level Constraint. These constraints impact the values being entered for a combination of column values, for example, salary can never be less than the commission is a table level constraint.

1.8.2 Primary Key

Primary key can be defined as the single column or combination of columns which uniquely identify a row in a table, for example, the rollno column in the “student” table is the primary key. No two students can have the same rollno. The following are the features of a primary key:

- A table can have one and only one primary key.
- Primary key implies UNIQUE as well as NOT NULL values,
- UNIQUE implies duplicate values cannot be entered.
- NOT NULL implies value cannot be unknown (NULL) for any of the records.
- Primary key columns are automatically indexed.
- A Primary Key can contain more than 1 column known as a Composite Key.

Primary Key = UNIQUE + NOT NULL + Automatic Indexed

An example of Primary key constraint:

```
CREATE TABLE student
    (
```

```
  rollno CHAR(4)      PRIMARY KEY,
  name VARCHAR2(10),
  date_of_birth DATE,
  marks NUMBER(3)
);
```

OR

```
CREATE TABLE student
(
  rollno char(4),
  name varchar2(10),
  date_of_birth date,
  marks number(3),
  CONSTRAINT stud_pk PRIMARY KEY (rollno)
);
```

An example of **Composite key constraint:**

```
CREATE TABLE student
(
  rollno char(4),
  name varchar2(10),
  date_of_birth date,
  marks number(3),
  CONSTRAINT stud_ck PRIMARY KEY (rollno, name)
);
```

1.8.3 NOT NULL

Specifying NOT NULL as constraint means that NULL values cannot be inserted for that particular field although duplicate

values for that column can be entered. It is a type of Domain Integrity Constraint.

An example of NOT NULL constraint:

```
CREATE TABLE student
        (
          rollno char(4),
          name varchar2(10)          NOT NULL,
          date_of_birth date NOT NULL, marks
          number(3)
        );
```

1.8.4 UNIQUE

UNIQUE implies duplicate values for the same field cannot be entered but NULL values can be inserted for that column. Since two NULL values cannot be compared, UNIQUE constraint does not prevent from supplying NULL values. It is a type of Domain Integrity Constraint.

An example of UNIQUE constraint:

```
CREATE TABLE student
        (
          rollno char(4)         UNIQUE,
          name varchar2(10)          NOT NULL,
          date_of_birth date NOT NULL, marks
          number(3)
        );
```

1.8.5 DEFAULT

DEFAULT implies that if we do not specify a value for a column in the INSERT statement, then the value specified in the Default clause will get inserted.

An example of DEFAULT constraint:

```
CREATE TABLE student
    (
      rollno char(4)        UNIQUE,
      name varchar2(10)         NOT NULL,
      date_of_birth date NOT NULL,  marks
      number(3)   DEFAULT 0
    );
```

6. CHECK

The CHECK constraint allows verifying the values being supplied against specified conditions. For example, marks cannot be negative and also cannot exceed the maximum marks (i.e. marks should be between 0 and 100).

An example of CHECK constraint:

```
CREATE TABLE student
  (
    rollno char(4)      UNIQUE,
    name varchar2(10)       NOT NULL,
    date_of_birth date      NOT NULL,
    marks number(3)CHECK (marks >= 0 AND marks <= 100)
      );
```

7. FOREIGN KEY

The FOREIGN KEY constraint is used to define a foreign key which represents relationships between tables. The foreign key is used to enforce referential integrity. Features of foreign key are:

- A foreign key can refer to a primary key.

- A table can have multiple foreign keys referring to different tables for different columns.
- Defining foreign key establishes a PARENT/CHILD relationship between the two tables.
- Defining foreign key implies two conditions:
    - We can insert only those records in the CHILD table for which corresponding records exist in the PARENT table.
    - We can delete only those records from the PARENT table for which there are no corresponding records in the CHILD table.

An example of FOREIGN KEY constraint:

```
CREATE TABLE stud_detail
        (
          rollno char(4)        PRIMARY KEY,
          name varchar2(10)          NOT NULL,
          address date        NOT NULL,
          phone_no number(3) UNIQUE NOT NULL,
          CONSTRAINT std_fk FOREIGN KEY (rollno)
REFERENCES student (rollno)
        );
```

If you want to delete records from the PARENT table for which there are corresponding records in the CHILD table then you need to specify the ON DELETE CASCADE option while defining the Foreign Key relation. If you do not specify this option then the database server will not allow you to delete data if other table references it. If you specify this option then the corresponding records of the CHILD table will also be deleted with the records from the PARENT table.

An example of FOREIGN KEY constraint using ON DELETE CASCADE option:

```
CREATE TABLE stud_detail
```

```
(
  rollno char(4)          PRIMARY KEY,
  name varchar2(10)       NOT NULL,
  address date            NOT NULL,
  phone_no number(3) UNIQUE NOT NULL,
  CONSTRAINT std_fk FOREIGN KEY (rollno)
REFERENCES student (rollno) ON DELETE CASCADE
);
```

1.8.8 Adding, Removing and Altering Constraints

The constraints we applied at the time of table creation can also be specified after the table has been created (without constraints).

```
CREATE TABLE student
(
  rollno char(4),
  name varchar2(10),
  date_of_birth date,
  marks number(3)
);
```

The above table has been created without any constraints. An example for adding constraints after table creation:

```
ALTER TABLE student
    ADD PRIMARY KEY (rollno, name);
ALTER TABLE student
    ADD CONSTRAINT def_mark marks DEFAULT 0;
```

Suppose we do not want roll no and name to be the primary key in the above mentioned table. The constraint can be removed using

the ALTER TABLE command. An example for removing constraints:

```
ALTER TABLE student
        DROP PRIMARY KEY;
```

1.8.9 Enabling and Disabling Constraints

It is not a good programming practice to remove constraints. Removing constraints can lead to problems in the table like data redundancy. Instead of dropping or removing constraints we can temporarily disable constraints. We can disable constraints when we need to and then enable the constraints when required. Consider the following example about the table STUDENT which has the primary key as ROLL NO. We can disable the primary key constraint using the DISABLE CONSTRAINT command with the ALTER TABLE statement and in order to enable the constraint we need to use the ENABLE CONSTRAINT command with the ALTER TABLE statement.

```
CREATE TABLE student
        (
          rollno char(4),
          name varchar2(10),
          date_of_birth date,
          marks number(3),
          CONSTRAINT stud_pk PRIMARY KEY (rollno)
        );
```

Disabling Constraints

```
ALTER TABLE student DISABLE CONSTRAINT stud_pk;
```

Enabling Constraints

```
ALTER TABLE student ENABLE CONSTRAINT stud_pk;
```

1.9 SUMMARY

- A database management system, or DBMS, is software designed to assist in maintaining and utilizing large collections of data.
- The collection of data, usually referred to as database, contains information relevant to an enterprise.
- A relational database consists of a collection of tables.
- Relational Database is a structure consisting of a set of objects related together and data integrity methods which aims in the efficient storage of electronic data.
- SQL stands for “Structured Query Language” and is a query language used for accessing and modifying information in the database.
- SQL can execute queries against a database to store, retrieve, modify and delete data.
- SQL commands can be categorized into Data Query Language (DQL), Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL) and Transaction Control Language (TCL).
- A table is the fundamental building block of a database application. Tables are used to store data on various entities like employees, products, customers, orders, sales etc.
- The CREATE TABLE statement is used to create a table object.
- The various constraints that can be applied on a table are as follows:
  - PRIMARY KEY
  - NOT NULL
  - UNIQUE
  - DEFAULT
  - CHECK
  - FOREIGN KEY
- The structure of a table can be changed after its creation using the ALTER TABLE statement.
- Constraints can also be applied to table after its creation using the ALTER TABLE statement.
- Constraints can be enabled or disabled using the ENABLE/DISABLE CONSTRAINT statement.

1.10 REVIEW QUESTIONS

(1) What is SQL? What can be done using SQL?
(2) List and Explain the common data types in SQL.
(3) List and Explain the different elements in SQL language.
(4) Explain the parts of SQL language?
(5) Explain the classification of SQL statements?
(6) List the rules for creating a table? How can you create a table?
(7) List and Explain the different types of constraints with examples?
(8) Explain the commands used to alter table?

1.11 LAB ASSIGNMENT

1. Create table DEPT with the following columns and constraints

Column name	Data type	Size	Constraint
DEPTNO	NUMBER	2	PRIMARY KEY
DNAME	VARCHAR2	10	UNIQUE + NOT NULL
LOCATION	VARCHAR2	10	UNIQUE + NOT NULL

2. Create table EMPLOYEE with the following columns and constraints

Column name	Data type	Size	Constraint
EMPNO	CHAR	4	PRIMARY KEY
ENAME	VARCHAR2	10	NOT NULL
JOB	VARCHAR2	10	
MGR	CHAR	4	
HIREDATE	TIMESTAMP		NOT NULL
GENDER	CHAR	1	'M' OR 'F' ONLY
SAL	NUMBER	8,2	DEFAULT 0
COMM	NUMBER	8,2	DEFAULT 0
DEPTNO	NUMBER	2	FOREIGN KEY REFERRING TO DEPTNO of DEPT table

3. Insert 5 records in both the tables.

4. Add table level constraint such that commission cannot be greater than 30% of salary after the table has been created. Assign the constraint name COMM_30_SAL.

5. Add new constraint with the name DEPT_CHK_LOCATION to DEPT table such that LOCATION can be any one of the following cities MUMBAI, PUNE, BENGALURU, LONDON, SAN FRANSISCO only.

6. Remove the UNIQUE constraint from the LOCATION column.

1.12 BIBLIOGRAPHY, REFERENCES AND FURTHER READING

- Database Management Systems, Third Edition by RamaKrishnan, Gehre. McGraw Hill
- Database System Concepts, Fifth Edition by Silberschatz, Korth, Sudarshan. McGraw Hill
- Murach's Oracle SQL and PL/SQL by Joel Murach. Shroff Publishers & Distributors
- Oracle Database 11g by Satish Asnani. PHI Learning Private Limited
- Oracle 11g: PL/SQL Reference Oracle Press.
- Expert Oracle PL/SQL, By: Ron Hardman,Michael McLaughlin, Tata McGraw-Hill
- SQL, PL/SQL The programming language of Oracle, Bayross Ivan, BPB Publications
- Fundamentals of Database Systems, Elmasri Ramez and Navathe B. Shamkant, Pearson

1.13 ONLINE REFERENCES

Wikipedia Link
http://en.wikipedia.org/wiki/SQL

Oracle Database PL/SQL language Reference 11g Release 2 (11.2), part number E25519-05
http://docs.oracle.com/cd/E11882_01/appdev.920/a96590/adg09dyn.htm

2

BASIC STRUCTURED QUERY LANGUAGE – II

Unit Structure

1. OBJECTIVES

At the end of this chapter, you will be able to

- Retreive data using SELECT statement
- Restrict and Sort Data
- Manipulate Data using Single Row functions
- Display Data from Multiple Tables using Joins

2. INTRODUCTION

In the previous chapter, we created tables using CREATE TABLE statement and managed tables. In this chapter, we will manipulate data from a single table and also retreive data and join multiple tables.

3. BASIC DATA RETREIVAL

The **SELECT** statement retreives data from the database and returns in the form of query results.

The syntax is:

SELECT [ALL/DISTINCT] select-item

FROM table-specification

[WHERE search-condition]

[GROUP BY grouping-column]

[HAVING search-condition]

[ORDER BY sort-specification]

The SELECT and FROM clauses are required and remaining four clauses are optional.

SELECT statement forms a part of the Data Query Language (DQL). SELECT statement is used to query the database in order to find data. SELECT is the most complex statement in SQL, with optional keywords and clauses that include:

- The FROM clause which indicates the table(s) from which data is to be retrieved. The FROM clause can include optional JOIN subclauses to specify the rules for joining tables.

- The WHERE clause includes a comparison predicate, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set for which the comparison predicate does not evaluate to true.
- The GROUP BY clause is used to project rows having common values into a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregation functions or to eliminate duplicate rows from a result set. The WHERE clause is applied before the GROUP BY clause.
- The HAVING clause includes a predicate used to filter rows resulting from the GROUP BY clause. Because it acts on the results of the GROUP BY clause, aggregation functions can be used in the HAVING clause predicate.
- The ORDER BY clause identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending). Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

To illustrate the SQL commands in this chapter we consider the records in the following tables

Student Table:

Rollno	Name	Course_no	Grade
201	Rohit	SC01	A
209	Raj	SC01	B
325	Rita	COM02	A
355	Parag	SC02	A
365	Mohini	SC03	C

Stud_details Table:

Rollno	Name	Age	Address
201	Rohit	25	MG road, Goregaon
209	Raj	27	Thakur City, Kandivali
310	Manisha	24	Vasai(W)
325	Rita	26	Borivali(W)
355	Parag	23	Thane(E)
365	Mohini	22	Aarey Colony, Goregaon

Course Table:

Course_no	Course_name	Major
SC01	B.Sc IT	Information Technology
SC02	B.Sc CS	Computer Science
SC03	B.Sc Maths	Mathematics
COM01	B.Com	Accounts
COM02	B.M.S	Management Studies

We need to execute the following queries

•To display all student records from table student
SELECT * FROM Student;

•To display name and address of the Students
SELECT Name, Address FROM Stud_Details;

•To display all course details
SELECT * FROM Course;

1. Column Aliases

The column headings are by default based on the column name, but sometimes we may want a customized column heading. It can be done using Column Aliases.

Syntax:

SELECT <column name1> “column alias1”, <column name2> “column alias2”, <column name3> “column alias3”.....

FROM <tablename>;

Rules for declaring Column Aliases

10. Column aliases must not contain any whitespace.
11. The case is ignored.
12. The Alias must be enclosed in double quotes.

2. Duplicate Rows

Many columns in a table can contain duplicate data. When querying data the duplicate data can come more than once in the query solution. For example, in an EMPLOYEE table the names of a few employees can same or repeated. Duplicate rows can be avoided by simply using the keyword **DISTINCT.**

Syntax:

SELECT DISTINCT <column name> FROM <table name>;

For Example:

SELECT DISTINCT Course_no, Name “Student Name”

FROM Student;

3. RESTRICTING AND SORTING DATA

A simple **SELECT** statement will return all rows from a particular table, but if we need to list a record on the basis of some conditions we need to use the **WHERE** clause.

Rules of using a WHERE clause:

- The WHERE clause MUST appear after the FROM clause.
- WHERE clause consists of the keyword WHERE followed by a search condition that specifies the rows to be retrieved.
- Character strings and date values are enclosed with single quote.
- Character values are case sensitive and date values are format sensitive.
- An column alias cannot be used in the WHERE clause.

Consider the following queries

- To display all records of students whose grade = ’A’
  SELECT * FROM Student WHERE Grade = ’A’;

- To display records of students who have enrolled for course ’SC02’
  SELECT * FROM Student

  WHERE Course_no = ’SC02’;

Apart from using the WHERE clause for equality conditions, we can use it for other conditions like:

Comparison Test (<>, <, >, <=, >=)

Compares the value of one expression to the value of another expression.

Syntax: SELECT <list> FROM <table name> WHERE <column name> comparison-operator <value>;

Range Test (BETWEEN)

Tests whether the value of an expression falls within a specified range of values.

Syntax: SELECT <list> FROM <table name> WHERE <column name> BETWEEN <lower-limit> AND <higher-limit>

Set Membership Test (IN)

Tests whether a data value matches one of a list of target values.

Syntax: SELECT <list> FROM <table name> WHERE <column name> IN (list of constants separated by comma);

NULL Value Test (IS NULL)

Used to check explicitly for NULL values in a search condition.

Syntax: SELECT <list> FROM <table name> WHERE <column name> IS NULL;

Pattern Matching Test (LIKE)

Checks to see whether the data value in a column matches a specified pattern. The pattern is a string that may include one or more wildcard characters.

Symbol	Represents
%	It matches any sequence of zero or more characters.
_	The under score matches any single character.
[]	Any single character within the specified range ([a-f]).
[^]	Any single character not within the specified range ([^a-f]).

2.3.1 Ordering Data

When SELECT statement is executed, the order of rows returned in the output is undefined. To define a specific order in the output, we need to use the **ORDER BY** clause with the SELECT statement. Using the ORDER BY clause, records can be arranged

in a specified sequence may be alphabetically or by value. The ORDER BY clause needs to be added to the end of the SELECT statement.

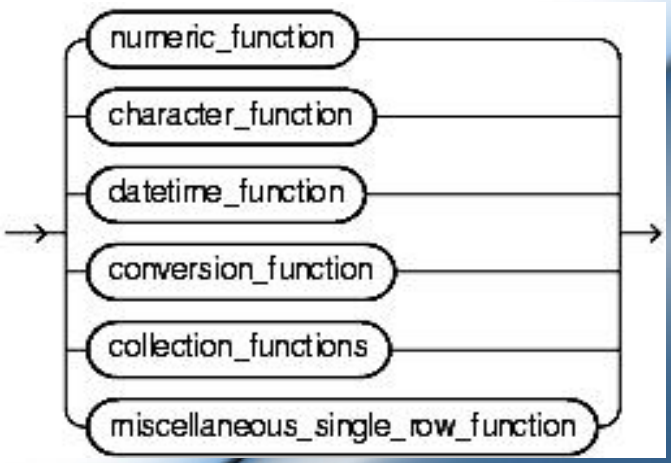
Syntax: SELECT <list> FROM <table name> ORDER BY <column name> [ASC | DESC]

4. DUAL TABLE

The Dual table is a special one-row table present by default in all Oracle database installations. It is suitable for use in selecting a pseudo-column such as SYSDATE or USER. The table has a single VARCHAR2(1) column called DUMMY that has a value of "X". The Dual table can also be used to understand the functioning of various single row functions. In the next section we will learn about single row functions using the DUAL table.

5. SINGLE ROW FUNCTIONS

Single row functions, as the name suggests, operates on single row and returns a single result row for every row of a queried table. These functions can appear in SELECT lists, WHERE clause, and other SQL statements. There are various types of single row functions as shown in the diagram below.



2.5.1 Numeric Functions:

Numeric functions accept numeric values as input and return numeric values as output.

ABS: The ABS function calculates the absolute value of an expression. Since, the absolute value of a real number is its numeric value without regard to its sign (for example, 3 is the absolute value of both 3 and -3), this function always returns a positive value.

Syntax: ABS(*expression*)

Usage: SELECT ABS (expression/column name) FROM <table name>;

CEIL: The CEIL function returns the smallest whole number greater than or equal to a specified number.

Syntax: CEIL(*n*)

Usage: SHOW CEIL(15.7);

RESULT: 16

FLOOR: The FLOOR function returns the largest whole number equal to or less than a specified number.

Syntax: FLOOR(*n*)

Usage: SHOW FLOOR(15.7);

RESULT: 15

ROUND: When a number is specified as an argument, the ROUND function returns the number rounded to the nearest multiple of a second number you specify or to the number of decimal places indicated by the second number.

Syntax: ROUND(*number_exp, roundvalue*)

Usage: SHOW ROUND(2/3, .1);

RESULT: 0.70

TRUNC: When you specify a number as an argument, the TRUNCATE function truncates a number to a specified number of decimal places.

Syntax: TRUNC (*number*, *truncvalue*)

Usage: SHOW TRUNC (15.79, 1);

RESULT: 15.7

2.5.2 Character Functions:

Character functions accept character values as input and can return character or numeric values as output.

LOWER: The LOWER function converts all alphabetic characters in a text expression into lowercase.

Syntax: LOWER(*text-expression*)

UPPER: The UPPER function converts all alphabetic characters in a text expression into uppercase.

Syntax: UPPER (*text-expression*)

CONCAT: CONCAT returns *char1* concatenated with *char2*. Both *char1* and *char2* can be any of the data types CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The string returned is in the same character set as *char1*. Its data type depends on the data types of the arguments.

Syntax: CONCAT (char1, char2)

SUBSTR: The SUBSTR function return a portion of *char*, beginning at character *position*, *substring_length* characters long.

Syntax: SUBSTR (char, position, substring_length)

LENGTH: The LENGTH functions return the length of *char*.

Syntax: LENGTH (char)

2.5.3 DateTime Functions:

DateTime functions operate on date, timestamp and interval values.

CURRENT_DATE: CURRENT_DATE returns the current date in the session time zone, in a value in the Gregorian calendar of data type DATE.

Syntax: SELECT CURRENT_DATE FROM dual;

CURRENT_TIMESTAMP: CURRENT_TIMESTAMP returns the current date and time in the session time zone, in a value of data type TIMESTAMP WITH TIME ZONE.

Syntax: SELECT CURRENT_TIMESTAMP FROM dual;

Note: DateTime Functions have been covered in more detail in later chapters.

2.5.4 Conversion Functions:

Conversion function converts a value from one datatype to another datatype.

TO_CHAR: This function converts number or date data type to character data type.

Syntax: TO_CHAR (value, [format mask])

TO_DATE: This function converts string data type to date data type.

Syntax: TO_DATE (string, [format mask])

TO_NUMBER: This function converts string data type to number data type.

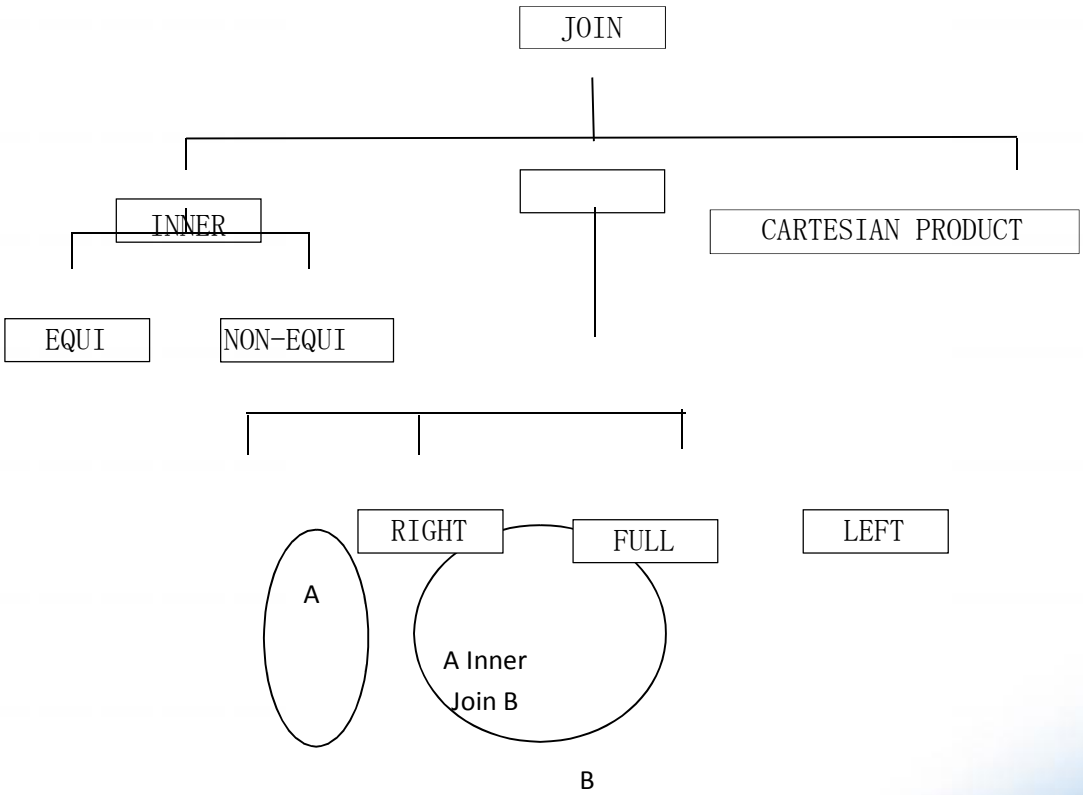
Syntax: TO_NUMBER (string, [format mask])

Note: For more information about different functions and the various categories of functions please refer to the Oracle Documentation available on the Oracle web site.

2.6 JOINS

Till now we have been selecting records from a single table at a time. However, in real life we do not work only on a single table but we need to select columns from two or more tables at a time for a single query. In order to combine two or more tables for a single query we have to use a concept called "**JOIN**" to acheive the desired result.

Joins can be classified as shown in the figure.



1. INNER EQUI JOIN

When two tables are joined using the EQUAL TO operator then that join is called Inner Equi Join.

2. INNER NON-EQUI JOIN

When two tables are joined using any comparison operator (<, >, <=, >=, !=) other than EQUAL TO operator then that join is called Inner Non-Equi Join.

Example for Inner Equi Join

We want to display the course_name and major subject alongwith the name and rollno of the student from the tables mentioned above as reference.

SELECT student.rollno, student.name, course.course_name, course.major

FROM student, course

WHERE student.course_no = course.course_no;

Rollno	Name	Course_name	Major
201	Rohit	B.Sc IT	Information Technology
209	Raj	B.Sc IT	Information Technology
325	Rita	B.M.S	Management Studies
355	Parag	B.Sc CS	Computer Science
365	Mohini	B.Sc Maths	Mathematics

Notice the “WHERE” clause specifying the join of two tables based on the matching course_no column is known as the JOIN condition. The Join condition may involve more than one column.

Above query can also be written as

SELECT student.rollno, student.name, course.course_name, course.major

FROM student INNER JOIN course

ON student.course_no = course.course_no;

Aliasing

Aliasing is the process whereby we can assign a short name to the tables being joined and use those alias names for preceding the column names. The above given Inner Equi Join can be written in the following manner:

SELECT s.rollno, s.name, c.course_name, c.major

FROM student s, course c

WHERE s.course_no = c.course_no;

Example for Inner Non-Equi Join

We want to display the course_name and major subject alongwith the name and rollno of the student whose age is greater than 25 from the tables mentioned above as reference.

SELECT s.rollno, s.name, c.course_name, c.major

FROM student s, course c

WHERE s.course_no = c.course_no AND s.age > 25;

Rollno	Name	Course_name	Major
209	Raj	B.Sc IT	Information Technology
325	Rita	B.M.S	Management Studies

3. Joining a Table to itself using Self Join

When a table is joined with its own then it is called "Self-join". Although self joins are rare, some queries are best solved using self joins. In a self join instead of duplicating the table, SQL lets you refer to it by a different table alias.

Example of Self Join

SELECT s1.rollno, s1.name, s2.age, s2,address

FROM stud_details s1, stud_details s2 WHERE

s1.rollno = s2.rollno;

Note: Although the same table has been used, SQL considers them as two different tables because separate aliases have been used.

4. Outer Joins

An outer join extends the result of an inner join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition.

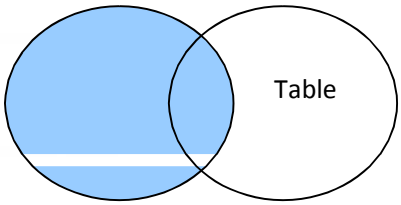
Outer Joins can be classified as

(9) Left Outer Join
(10) Right Outer Join
(11) Full Outer Join

2.6.5 LEFT OUTER JOIN

A query that performs a join on two tables A and B, and returns all rows from table A and only matching rows from table B is

called as LEFT OUTER JOIN. The result set contains all rows from the first or the left table and only matching records from the second table.



Syntax:

SELECT <list>

FROM <table name1> LEFT OUTER JOIN <table name2>

ON JOIN CONDITION

Example for Left Outer Join

Suppose we want to display the list of names and addresses of all students alongwith their course_no and grade.

SELECT sd.rollno, sd.name, sd.address, s.course_no, s.grade

FROM stud_details st LEFT OUTER JOIN student s

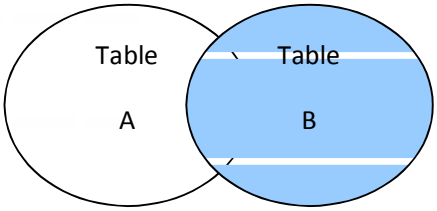
ON sd.rollno = s.rollno;

Rollno	Name	Address	Course_no	Grade
201	Rohit	MG road, Goregaon	SC01	A
209	Raj	Thakur City, Kandivali	SC01	B
310	Manisha	Vasai(W)	-	-
325	Rita	Borivali(W)	COM02	A
355	Parag	Thane(E)	SC02	A
365	Mohini	Aarey Colony, Goregaon	SC03	C

In the above result, all columns from the stud_details table have been listed and only matching columns from the student table have been listed.

2.6.6 RIGHT OUTER JOIN

A query that performs a join on two tables A and B, and returns all rows from table B and only matching rows from table A is called as RIGHT OUTER JOIN. The result set contains all rows from the second or the right table and only matching records from the first table.



Syntax:

SELECT <list>

FROM <table name1> RIGHT OUTER JOIN <table name2>

ON JOIN CONDITION

Example for Right Outer Join

Suppose we want to display the list of names of all students alongwith their course_name and major subject.

SELECT s.rollno, s.name, c.course_no, c.course_name, c.major

FROM student s RIGHT OUTER JOIN course c

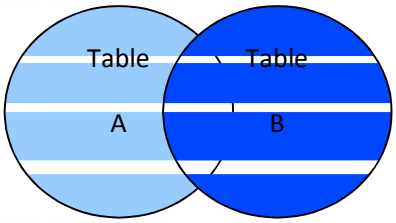
ON s.course_no = c.course_no;

Rollno	Name	Course_no	Course_name	Major
201	Rohit	SC01	B.Sc IT	Information Technology
209	Raj	SC01	B.Sc IT	Information Technology
325	Rita	COM02	B.M.S	Management Studies
355	Parag	SC02	B.Sc CS	Computer Science
365	Mohini	SC03	B.Sc Maths	Mathematics
-	-	COM01	B.Com	Accounts

In the above result, all columns from the course table have been listed and only matching columns from the student table have been listed.

2.6.7 FULL OUTER JOIN

A query that performs a join on two tables A and B, and returns all rows from table A and B alonwith the non-matching rows is called as FULL OUTER JOIN. The result set contains all rows from both the tables.



Syntax:

SELECT <list>

FROM <table name1> FULL OUTER JOIN <table name2>

ON JOIN CONDITION

2.6.8 CARTESIAN PRODUCT

A join in which each record from table 1 gets associated with each and every record of table 2 is called as Cartesian Product. A Cartesian product always generates many rows and is rarely useful. For example, the Cartesian product of two tables, each with 100 rows, has 10,000 rows.

Syntax:

SELECT <list>

FROM <table name1>, <table name2>

Note: In a cartesian product, no WHERE clause is used. Cartesian Product can also be called as CROSS JOIN.

7. SUMMARY

- The SELECT statement retreives data from the database and returns in the form of query results.

- Duplicate rows can be avoided by simply using the keyword DISTINCT.

- We need to use the WHERE clause in order to list a record on the basis of some conditions.

- WHERE clause can be used for
  - Equality conditions
  - Comparison Test
  - Range Test
  - Set Membership Test
  - NULL Value Test
  - Pattern Matching Test

- To define a specific order in the output, we need to use the ORDER BY clause with the SELECT statement.
- Single row functions operate on a single row and returns a single result row for every row of a queried table.There are various types of single row functions

  - Numeric Functions
  - Character Functions
  - DateTime Functions
  - Conversion Functions

- In order to combine two or more tables for a single query we have to use a concept called “JOIN” to acheive the desired result.

- Joins can be classified as
  - Inner - Equi & Non-Equi
  - Self
  - Outer - Left, Right, & Full
  - Cartesian Product

8. REVIEW QUESTIONS

- Explain the Where clause. List and explain the conditions under which you can use it.
- What is a single row function? Explain and give examples.
- What is a join? Explain the various categories of joins that can be used to join two tables?

9. LAB ASSIGNMENT

1. Create a CUSTOMER table with the following columns and constraints

Column name	Data type	Size	Constraint
CUSTOMER_ID	CHAR	6	PRIMARY KEY. MUST BEGIN WITH 'C'
CUSTOMER_NAME	VARCHAR2	20	NOT NULL
ADDRESS	VARCHAR2	20	UNIQUE
CITY	VARCHAR2	20	
PINCODE	NUMBER	6	
STATE	VARCHAR2	20	
BALANCE_DUE	NUMBER	8,2	

2. Create a PRODUCT table with the following columns and constraints

Column name	Data type	Size	Constraint
PRODUCT_CODE	CHAR	6	PRIMARY KEY
PRODUCT_NAME	VARCHAR2	20	UNIQUE
QTY_AVAIL	NUMBER	5	
COST_PRICE	NUMBER	8,2	
SELLING_PRICE	NUMBER	8,2	

3. Create a ORDER table with the following columns and constraints

Column name	Data type	Size	Constraint
ORDER_NO	CHAR	6	
ORDER_DATE	TIMESTAMP		
CUSTOMER_ID	CHAR	6	
PRODUCT_CODE	CHAR	6	
QUANTITY	NUMBER	5	

PRIMARY KEY = ORDER_NO + ORDER_DATE + CUSTOMER_ID + PRODUCT_CODE

- Insert 5-10 records in all the tables.
- Apply UNIQUE constraint on CUSTOMER_ID+ PRODUCT_CODE on the ORDER table.
- Define a foreign key on CUSTOMER_ID of ORDER table referring to CUSTOMER_ID of CUSTOMER table.
- Define a foreign key on PRODUCT_CODE of ORDER table referring to PRODUCT_CODE of PRODUCT table.
- List the customer names whose balance due is more than 4000/-.
- For all products display the product name along with Net Profit as Selling price – Cost price.
- Calculate the profit earned on each order.
- Display the orders placed during 2008.

[Hint: TO_CHAR(ORDER_DATE, 'YYYY') = '2008')

- Arrange the customers first by STATE and then by NAME.
- Display the customers from CUSTOMER table such that the customer with maximum due balance is displayed at the top.
- Find out the product names and their quantity which have been delivered in the month of February.
- Find the product name, customer name and total quantity purchased by various customers.
- Display the total sale amount and quantity for each customer. The report should display customer name instead of customer id.

10. BIBLIOGRAPHY, REFERENCES AND FURTHER READING

- Database Management Systems, Third Edition by RamaKrishnan, Gehre. McGraw Hill
- Database System Concepts, Fifth Edition by Silberschatz, Korth, Sudarshan. McGraw Hill
- Murach's Oracle SQL and PL/SQL by Joel Murach. Shroff Publishers & Distributors
- Oracle Database 11g by Satish Asnani. PHI Learning Private Limited
- Oracle 11g: PL/SQL Reference Oracle Press.
- Expert Oracle PL/SQL, By: Ron Hardman,Michael McLaughlin, Tata McGraw-Hill
- SQL, PL/SQL The programming language of Oracle, Bayross Ivan, BPB Publications
- Fundamentals of Database Systems, Elmasri Ramez and Navathe B. Shamkant, Pearson

2.11 ONLINE REFERENCES

Wikipedia Link

http://en.wikipedia.org/wiki/SQL

Oracle Database PL/SQL language Reference 11g Release 2 (11.2), part number E25519-05

http://docs.oracle.com/cd/E11882_01/appdev.920/a96590/adg09dyn.htm

3

ADVANCED QUERIES AND DATABASE OBJECTS

Unit Structure

1. OBJECTIVES

At the end of this chapter you will be able to:

- Aggregate Data using Group Functions
- Aggregate Data using Group By Having clause
- Create Database Objects like Views, Sequences, Indexes and Synonyms
- Use Subqueries

1. INTRODUCTION

In the previous chapter, you were introduced to scalar (single row) functions, which operate on a single value and return a single value. In this chapter, you'll learn how to code queries that summarize data.

2. AGGREGATE FUNCTIONS

Aggregate functions operate on a series of values and return a single summary value. Aggregate functions allow you to do jobs like calculate averages, summarize totals, or find the highest value for a given column. Aggregate functions can appear in select lists and in ORDER BY and HAVING clauses. They are commonly used with the GROUP BY clause in a SELECT statement, where the rows of a queried table are divided into groups.

Many (but not all) aggregate functions that take a single argument accept these clauses:

- DISTINCT – causes an aggregate function to consider only distinct values of the argument expression.
- ALL – causes an aggregate function to consider all values, including all duplicates.

For example, the DISTINCT average of 1, 1, 1, and 3 is 2. The ALL average is 1.5. If you specify neither, then the default is ALL. Null values are always excluded from these functions.

The table given below presents the syntax of the most common aggregate functions.

Syntax	Description
AVG ([ALL | DISTINCT] expression)	Returns the average of non-null values in the expression.
SUM ([ALL | DISTINCT] expression)	Returns the total of non-null values in the expression.
MIN ([ALL | DISTINCT] expression)	Returns the lowest non-null value in the expression.
MAX ([ALL | DISTINCT] expression)	Returns the highest non-null value in the expression.

COUNT ([ALL | DISTINCT] expression)	Returns the number of non-null values in the expression.
COUNT(*)	Returns the number of rows selected by the query.

Since the purpose of these functions are self-explanatory, we'll focus mainly on how to use them.

Assume records in the EMP table as shown below.

EMPNO	ENAME	HIREDATE	DEPTNO	GENDER	SALARY	COMM
111	Satish	19-DEC-2008	10	M	10,000	1000
222	Rashmi	01-JAN-1987	20	F	8000	550
333	Rishi	05-JUN-1976	10	M	7000	450
444	Anil	16-APR-1967	10	M	12,000	2000
555	Anita	-	30	F	8000	1000
666	Nilesh	20-MAY-1987	20	M	13,000	-
777	Ruchi	11-JUN-2000	30	F	5000	-
888	Sarika	-	20	F	4000	-

- Find the number of distinct departments from emp table.

SELECT COUNT(DISTINCT deptno) FROM emp;

Result:

COUNT(DISTINCT deptno)

3

- Find the number of employees in the emp table.

SELECT COUNT(*) FROM emp;

Result:

COUNT(*)

8

- Find the total salary and the average commission from the emp table.

SELECT SUM(salary), AVG(comm) FROM emp;

Result:

SUM(salary) AVG(comm)

67000 1000

3.3 GROUP BY HAVING CLAUSE

GROUP BY clause forms groups on the specified columns. The GROUP BY clause is used alongwith the aggregate functions to retreive data grouped according to one or more columns. The group by clause should contain all the columns to be displayed except those used alongwith the aggregate functions. The GROUP BY clause groups rows but does not guarantee the order of the result set. To order the groupings, use the ORDER BY clause.

For example,

- Display the number of employees from each department.

SELECT deptno, count(*)

FROM emp

GROUP BY deptno;

DEPTNO	COUNT(*)
10	3
30	2
20	3

- Display department wise total salary from the emp table.

SELECT deptno, sum(salary)

FROM emp

GROUP BY deptno;

DEPTNO	SUM(salary)
10	29,000
30	13,000
20	25,000

The **HAVING** clause to restrict the groups of returned rows to those groups for which the specified *condition* is TRUE. In other words, the HAVING clause is used to filter the records which a GROUP BY clause returns. This is similar to the WHERE clause but is used with the GROUP BY clause. The WHERE clause cannot be used with the GROUP BY clause.

For example,

(12) Display department wise total salary from the emp table such that only those departments are displayed where the total salary is greater than 20,000.

```
SELECT deptno, sum(salary)
FROM emp
GROUP BY deptno
HAVING SUM(salary) > 20000;
```

DEPTNO	SUM(salary)
10	29,000
20	25,000

(13) Display the number of employees from each department where the number of employees is equal to 2.

```
SELECT deptno, count(empno) “empcount”
FROM emp
GROUP BY deptno
HAVING COUNT(empno) = 2;
```

DEPTNO	EMPCOUNT
30	2

3.3.1 Comparing HAVING clause and WHERE clause

A WHERE clause in a SELECT statement that uses grouping & aggregates, the search condition is applied before rows are grouped and aggregates are calculated. That way, only rows that satisfy the condition are grouped. A HAVING clause in a SELECT statement that uses grouping & aggregates, the search condition is applied after rows are grouped and aggregates are calculated. That way, only groups that satisfy the condition are included in the result set.

A WHERE clause can refer to any column in the table. A HAVING clause can only refer to a column included in the SELECT clause.

A WHERE clause cannot contain aggregate functions. Aggregate functions can only be coded in the HAVING clause.

3.4 CREATING OTHER DATABASE OBJECTS

SQL allows you to create various database objects other than table like views, sequences, indexes, synonyms. We will understand these different database objects in this section.

Assume table **PRODUCT** with the following records

product_id	product_name	company_name	unit_price
100	Shampoo	Pantene	180
101	Deospray	Denim	400
102	Tooth paste	Colgate	150
103	Soap	Lux	75
104	Hair gel	Laureal	300

Assume table **ORDER** with the following records

order_id	product_id	total_units	Customer
O1	101	30	Lifestyle
O2	101	5	Shoppers stop
O3	103	25	Spencer
O4	101	10	Food bazaar
O5	103	200	Big bazaar

3.4.1 VIEWS

A **view** is a logical representation of one or more tables. In essence, a view is a stored query. A view derives its data from the tables on which it is based,called base tables. Base tables can be tables or other views. All operations performed on a view actually affect the base tables. You can use views in most places where tables are used. You can query, insert, update and delete from views. Views can be handled as any other table but they do not occupy any space.

Unlike a table, a view is not allocated storage space, nor does a view contain data. Rather, a view is defined by a query that extracts or derives data from the base tables referenced by the view. Because a view is based on other objects, it requires no storage other than storage for the query that defines the view in the **data dictionary**.

Benefits of using Views

Views enable you to tailor the presentation of data to different types of users. Views are often used to:

- Provide an additional level of table security by restricting access to a predetermined set of rows or columns of a table.

- Hide data complexity.

  For example, a single view can be defined with a **join**, which is a collection of related columns or rows in multiple tables. However, the view hides the fact that this information actually originates from several tables. A query might also perform extensive calculations with table information. Thus, users can query a view without knowing how to perform a join or calculations.

- Present the data in a different perspective from that of the base table.

  For example, the columns of a view can be renamed without affecting the tables on which the view is based.

- Isolate applications from changes in definitions of base tables.

  For example, if the defining query of a view references three columns of a four column table, and a fifth column is added to the table, then the definition of the view is not affected, and all applications using the view are not affected.

1. Classification of Views

Views can be classified as updateable views and non-updateable views.

2. Updateable Views

By updateable we mean to say that one can insert, update and delete records from the view. Actually all the DML operations are performed on the base table.

View with the following characteristics is called an updateable view.

- It is created from a single table.
- It includes all PRIMARY KEYS and NOT NULL columns of the base table.
- Aggregate functions like SUM, AVG have not been used.
- It should not have DISTINCT, GROUP BY, HAVING clauses.
- It must not use constants, strings or value expressions like salary * 2.
- It must not any function calls (e.g. RPAD, SUBSTR, etc.).
- If a view is defined from another view then that view must also be updateable.

3. Non-updateable Views

Non-updateable means we cannot insert, update and delete records from that view.

View with the following characteristics is called a non-updateable view.

7. It is created from more than one table.
8. It has DISTINCT, GROUP BY, HAVING clause. Even if view is derived from a single table but contains any of these clauses then it is not updateable.
9. It does not include all the PRIMARY KEYS and NOT NULL columns of base tables.

CREATING VIEWS

Syntax:

CREATE OR REPLACE VIEW <view name> AS

(SELECT query)

[WITH READ ONLY CONSTRAINT <constraint name>];

Create a view showing the details of the products which have been ordered by the customers.

CREATE OR REPLACE VIEW prod_ordered AS

SELECT product_name, company_name, total_units

FROM product INNER JOIN order

ON product.product_id = order.product_id;

To see the details of the products ordered

SELECT * FROM prod_ordered

product_name	company_name	total_units
Deospray	Denim	30
Deospray	Denim	5
Soap	Lux	25
Deospray	Denim	10
Soap	Lux	200

Thank You