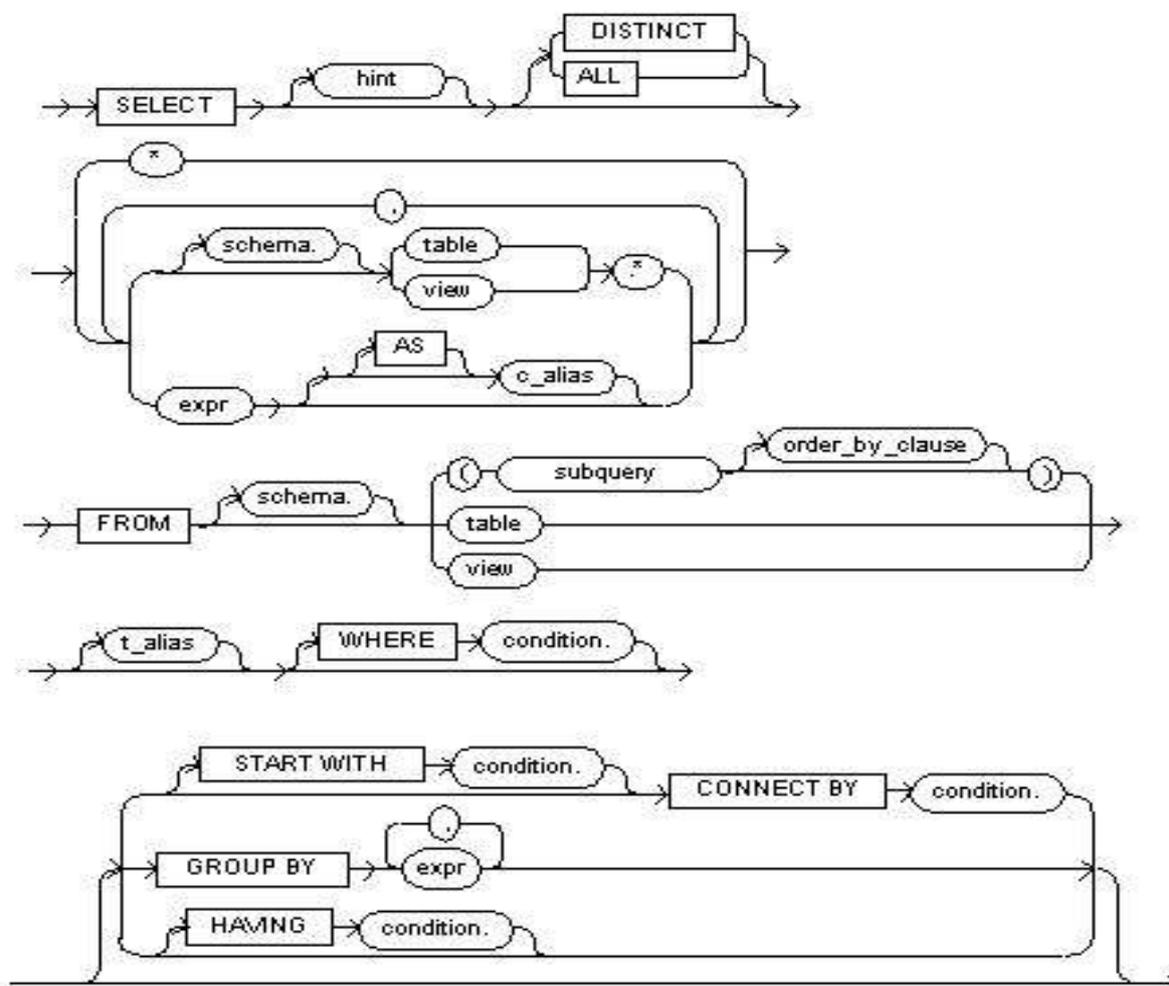


CHAPTER 14

SQL Commands**OBJECTIVES**

- **To understand SQL commands usage.**
- **Learning the data types, expressions, operators.**
- **The use of syntax and constrains for SQL.**
- **Commands for DDI and DML.**
- **Various built in functions SQL.**



14.1 Introduction:

Structured Query Language helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database

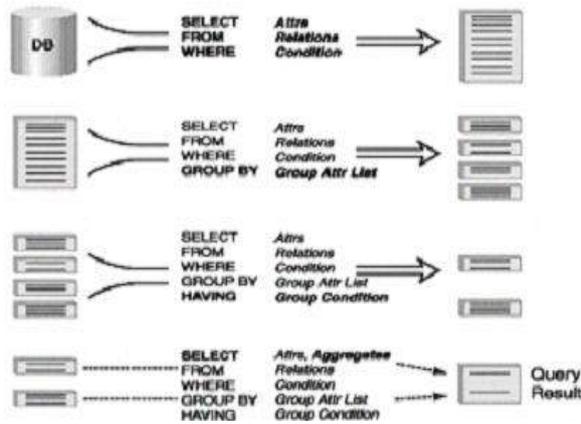


Fig 14.1 Database to Query

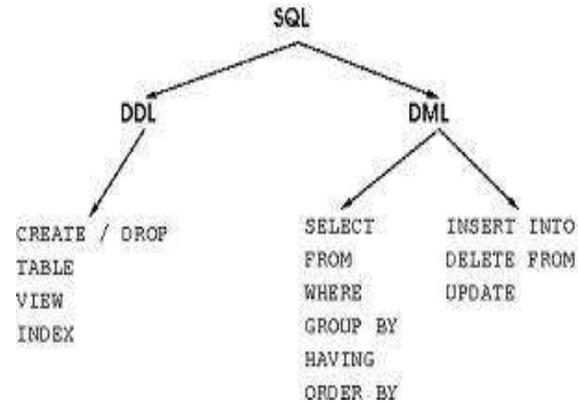


Fig 14.2 SQL languages classifications

creation, deletion, fetching rows and modifying rows etc.

SQL is an ANSI (American National Standards Institute) standard but there are many different versions of the SQL language.

SQL is Structured Query Language, which is a dbase language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgresql and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

History:

- 1970:** Dr. E. F. “Codd” of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974:** Structured Query Language appeared.
- 1978:** IBM worked to develop Codd’s ideas and released a product named System/R.
- 1986:** IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

14.1.1 SQL ARCHITECTURE:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

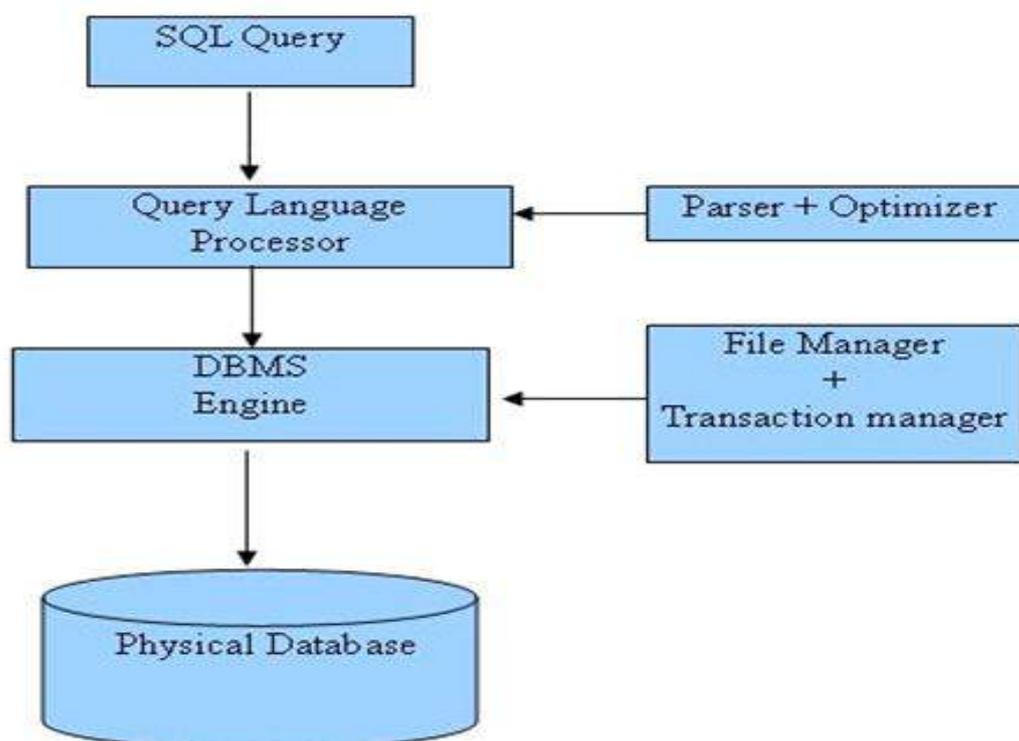


Fig 14.3 SQL Architecture with different layers

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

<p>MySQL MySQL is an open source SQL database, which is developed by Swedish company MySQL AB. MySQL is pronounced "my ess-que-ell," in contrast with SQL, pronounced "sequel." MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X. MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server.</p>	<p>MS SQL Server MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are: T-SQL, ANSI SQL.</p>	<p>ORACLE It is a very large and multi-user database management system. Oracle is a relational database management system developed by 'Oracle Corporation'. Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network. It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.</p>	<p>MS ACCESS This is one of the most popular Microsoft products. Microsoft Access is an entry-level database management software. MS Access database is not only an inexpensive but also powerful database for small-scale projects. MS Access uses the Jet database engine, which utilizes a specific SQL language dialect (sometimes referred to as Jet SQL). MS Access comes with the professional edition of MS Office package. MS Access has easy-to-use intuitive graphical interface.</p>
--	--	---	---

<p>History: Development of MySQL by Michael Widenius & David Axmark beginning in 1994. First internal release on 23 May 1995. Windows version was released on 8 January 1998 for Windows 95 and NT. Version 3.23: beta from June 2000, production release January 2001. Version 4.0: beta from August 2002, production release March 2003 (unions). Version 4.01: beta from August 2003, Jyoti adopts MySQL for database tracking. Version 4.1: beta from June 2004, production release October 2004. Version 5.0: beta from March 2005, production release October 2005. Sun Microsystems acquired MySQL AB on 26 February 2008. Version 5.1: production release 27 November 2008.</p>	<p>History: 1987 - Sybase releases SQL Server for UNIX. 1988 - Microsoft, Sybase, and Aston-Tate port SQL Server to OS/2. 1989 - Microsoft, Sybase, and Aston-Tate release SQL Server 1.0 for OS/2. 1990 - SQL Server 1.1 is released with support for Windows 3.0 clients. Aston-Tate drops out of SQL Server development. 2000 - Microsoft releases SQL Server 2000. 2001 - Microsoft releases XML for SQL Server Web Release 1 (download). 2002 - Microsoft releases SQLXML 2.0 (renamed from XML for SQL Server). 2002 - Microsoft releases SQLXML 3.0. 2005 - Microsoft releases SQL Server 2005 on November 7th, 2005.</p>	<p>History: Oracle began in 1977 and celebrating its 32 wonderful years in the industry (from 1977 to 2009). 1977 - Larry Ellison, Bob Miner and Ed Oates founded Software Development Laboratories to undertake development work. 1979 - Version 2.0 of Oracle was released and it became first commercial relational database and first SQL database. The company changed its name to Relational Software Inc. (RSI). 1981 - RSI started developing tools for Oracle. 1982 - RSI was renamed to Oracle Corporation. 1983 - Oracle released version 3.0, rewritten in C language and ran on multiple platforms. 1984 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc. 1985 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc. 2007 - Oracle has released Oracle11g. The new version focused on better partitioning, easy migration etc.</p>	<p>History: 1992 - Access version 1.0 was released. 1993 - Access 1.1 released to improve compatibility with inclusion the Access Basic programming language. The most significant transition was from Access 97 to Access 2000. 2010 - Access 2010, a new database format was introduced ACCDB which supports complex data types such as multi valued and attachment fields.</p>
--	---	---	--

<p>Features: High Performance. High Availability. Scalability and Flexibility Run anything. Robust Transactional Support. Web and Data Warehouse Strengths. Strong Data Protection. Comprehensive Application Development. Management Ease. Open Source Freedom and 24 x 7 Support. Lowest Total Cost of Ownership.</p>	<p>Features: High Performance. High Availability. Database mirroring. Database snapshots. CLR integration. Service Broker. DDL triggers. Ranking functions. Row version-based isolation levels. XML integration. TRY...CATCH. Database Mail.</p>	<p>Features: Concurrency Read Consistency Locking Mechanisms Quiescent Database Portability Self-managing database SQL*Plus ASM Scheduler Resource Manager Data Warehousing Materialized views Bitmap indexes Table compression Parallel Execution Analytic SQL Data mining Partitioning</p>	<p>Features: Users can create tables, queries, forms and reports and connect them together with macros. The import and export of data to many formats including Excel, Outlook, ASCII, dBase, Paradox, FoxPro, SQL Server, Oracle, ODBC, etc. There is also the Jet Database format (MDB or ACCDB in Access 2007), which can contain the application and data in one file. This makes it very convenient to distribute the entire application to another user, who can run it in disconnected environments. Microsoft Access offers parameterized queries. These queries and Access tables can be referenced from other programs like VB6 and .NET through DAO or ADO. The desktop editions of Microsoft SQL Server can be used with Access as an alternative to the Jet Database Engine. Microsoft Access is a file server-based database. Unlike client-server relational database management systems (RDBMS), Microsoft Access does not implement database triggers, stored procedures, or transaction logging.</p>
--	---	---	---

14.2 SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

14.2.1 DDL - Data Definition Language:

DDL defines the conceptual schema providing a link between the logical (the way the user views the data) and the physical (the way in which the data is stored physically) structures of the database. The logical structure of a database is a schema. A subschema is the way a specific application views the data form the database.

Following are the functions of the Data Definition Language (DDL):-

1. DDL define the physical characteristics of each record, filed in the record, field's data type, field's length, field's logical name and also specify relationships among those records.
2. DDL describes the schema and subschema.
3. DDL indicates the keys of the records.
4. DDL provides means for associating related records or fields.
5. DDL provides data security measures.
6. DDL provides for the logical and physical data independence.

Few of the basic commands for DDL are :-

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

14.2.2 DML - Data Manipulation Language:

1. DML provides the data manipulation techniques like selection, insertion, deletion, update, modification, replacement, retrieval, sorting and display of data or records.
2. DML facilitates use of relationship between the records.
3. DML enables the user and application program to be independent of the physical data structures and database structures maintenance by allowing to process data on a logical and symbolic basis rather than a physical on a physical location basis.
4. DML provide for independence of programming languages by supporting several high-level programming languages like COBOL,PL/1 and C++.

Few of the basic commands for DML are :-

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

14.3 Data types in SQL

SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.

You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement.

SQL Server offers six categories of data types for your use:

14.3.1 Exact Numeric Data Types:

DATA TYPE	FROM	TO
Int	-2,147,483,648	2,147,483,647
numeric	$-10^{38} + 1$	$10^{38} - 1$

14.3.2 Floating point numeric Data Types:

DATA TYPE	FROM	TO
Float	$-1.79E + 308$	$1.79E + 308$
Real	$-3.40E + 38$	$3.40E + 38$

14.3.3 Date and Time Data Types:

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
Date	Stores a date like MARCH 26, 2014	
Time	Stores a time of day like 12:30 P.M.	

Note: Here, datetime has 3.33 milliseconds accuracy whereas small datetime has 1 minute accuracy.

14.3.4 Character, Strings Data Types:

DATA TYPE	FROM	TO
Char	char	Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
varchar	varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).

14.4 Operator in SQL

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

14.4.1 SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

14.4.2 Comparison Operators:

Assume variable a holds 10 and variable b holds 20, then:

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

14.4.3 Logical Operators:

Here is a list of all the logical operators available in SQL.

Show Examples

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.

IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

14.5 SQL expression

SQL EXPRESSIONS are like formulas and they are written in query language. You can also use them to query the database for specific set of data. An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value.

Syntax:

Consider the basic syntax of the SELECT statement as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION | EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below:

14.5.1 SQL - Boolean Expressions:

SQL Boolean Expressions fetch the data on the basis of matching single value. Following is the syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

Consider the EMPLOYEES table having the following records:

Here is simple example showing usage of SQL Boolean Expressions:

```
SQL> SELECT * FROM EMPLOYEES WHERE age =45;
+---+-----+-----+-----+-----+
| ID | NAME      | AGE | ADDRESS | SALARY |
+---+-----+-----+-----+-----+
| 3 | Srinivas  | 45 | Mangalore | 37000.00 |
+---+-----+-----+-----+-----+
1 row inset(0.00 sec)
```

14.5.2 Numeric Expression:

This expression is used to perform any mathematical operation in any query. Following is the syntax:

```
SELECT numerical_expression as OPERATION_NAME
[FROM table_name
WHERE CONDITION];
```

Here numerical_expression is used for mathematical expression or any formula. Following is a simple examples showing usage of SQL Numeric Expressions:

```
SQL> SELECT (15+6) AS ADDITION
+-----+
| ADDITION |
+-----+
| 21 |
+-----+
1 row inset(0.00 sec)
```

There are several built-in functions like avg(), sum(), count() etc., to perform what is known as aggregate data calculations against a table or a specific table column.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM EMPLOYEES;
+-----+
| RECORDS |
+-----+
| 7 |
+-----+
1 row inset(0.00 sec)
```

14.5.3 Date Expressions:

Date Expressions return current system date and time values:

```
SQL> SELECT CURRENT_TIMESTAMP;
+-----+
| Current_Timestamp |
+-----+
| 2014-03-21 06:40:23 |
+-----+
1 row inset(0.00 sec)
```

Another date expression is as follows:

```
SQL> SELECT GETDATE();
+-----+
| GETDATE          |
+-----+
| 2014-01-14 12:07:18.140 |
+-----+
1 row inset(0.00 sec)
```

14.6 SQL Constraints:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL:

The constraints available in SQL are Foreign Key, Not Null, Unique, Check. Constraints can be defined in two ways

- 1) The constraints can be specified immediately after the column definition. This is called column-level definition.
- 2) The constraints can be specified after all the columns are defined. This is called table-level definition.

14.6.1 SQL Primary key:

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

Syntax to define a Primary key at column level:

```
column name datatype [CONSTRAINT constraint_name] PRIMARY KEY
```

Syntax to define a Primary key at table level:

```
[CONSTRAINT constraint_name] PRIMARY KEY
```

```
(column_name1,column_name2,..)
```

column_name1, column_name2 are the names of the columns which define the primary Key.

The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

For Example: To create an employee table with Primary Key constraint, the query would be like.

Primary Key at column level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  city char(10)
);
```

or

```
CREATE TABLE employee
( id number(5) CONSTRAINT emp_id_pk PRIMARY KEY,
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  city char(10)
);
```

Primary Key at column level:

```
CREATE TABLE employee
( id number(5),
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  city char(10),
  CONSTRAINT emp_id_pk PRIMARY KEY (id)
);
```

Primary Key at table level:

```
CREATE TABLE employee
( id number(5), NOT NULL,
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  city char(10),
  ALTER TABLE employee ADD CONSTRAINT PK_EMPLOYEE_ID PRIMARY KEY
(id) );
```

14.6.2 Foreign key or Referential Integrity :

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between different tables. For a column to be defined as a Foreign Key, it should be defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.

Syntax to define a Foreign key at column level:

```
[CONSTRAINT constraint_name] REFERENCES
Referenced_Table_name(column_name)
```

Syntax to define a Foreign key at table level:

```
[CONSTRAINT constraint_name] FOREIGN KEY(column_name) REFERENCES
referenced_table_name(column_name);
```

For Example:

1) Lets use the “sports” table and “order_items”.

Foreign Key at column level:

```
CREATE TABLE product
( product_id number(5) CONSTRAINT pd_id_pk PRIMARY KEY,
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```



```
SQL> select o.order_id,p.product_name,p.unit_price,p.supplier_name
  2  from sports p, order_items o
  3  where o.product_id=p.product_id;
```

ORDER_ID	PRODUCT_NAME	UNIT_PRICE	SUPPLIER_NAME
20141	BAT	15000	SUNNY

```
SQL>
SQL>
SQL>
SQL>
```

14.6.3 Not Null Constraint :

This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.

Syntax to define a Not Null constraint:

```
[CONSTRAINT constraint_name] NOT NULL
```

For Example: To create a employee table with Null value, the query would be like

```
CREATE TABLE employee
( id number(5),
name char(20) CONSTRAINT nm_nn NOT NULL,
dept char(10),
age number(2),
salary number(10),
CITY char(10)
);
```

14.6.4 Unique Key:

This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.

Syntax to define a Unique key at column level:

```
[CONSTRAINT constraint_name] UNIQUE
```

Syntax to define a Unique key at table level:

```
[CONSTRAINT constraint_name] UNIQUE(column_name)
```

For Example: To create an employee table with Unique key, the query would be like,

14.6.5 Check Constraint :

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax to define a Check constraint:

```
[CONSTRAINT constraint_name] CHECK (condition)
```

For Example: In the employee table to select the gender of a person, the query would be like

Check Constraint at column level:

14.7 Implementation of SQL commands:

In this chapter the SQL commands are explained along with the syntax and example, which are worked out in SQL 8.1i. The example is taken as employees table and all the syntax like create, alter, drop are illustrated with the example and DML commands like insert,select,where,order,group etc. are given.

14.7.1 **CREATE TABLE** statement is used to create a new table.

Basic syntax of CREATE TABLE statement is as follows:

```
CREATE TABLE table_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    .....  
    columnN datatype,  
    PRIMARY KEY( one or more columns )  
);
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below.

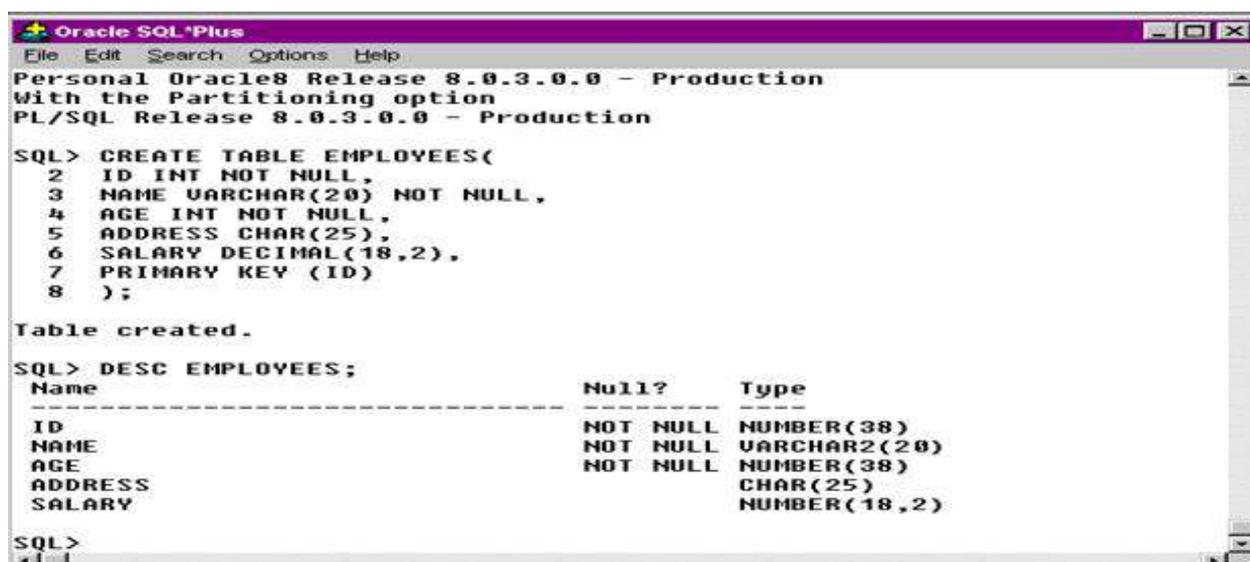
A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.

Following is an example, which creates a EMPLOYEES table with ID as primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table:

You can check complete details at [Create Table Using another Table.](#)

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use **DESC** command as follows:

Now, you have EMPLOYEES table available in your database which you can use to store required information related to EMPLOYEES.



```

Oracle SQL*Plus
File Edit Search Options Help
Personal Oracle8 Release 8.0.3.0.0 - Production
With the Partitioning option
PL/SQL Release 8.0.3.0.0 - Production

SQL> CREATE TABLE EMPLOYEES(
  2  ID INT NOT NULL,
  3  NAME VARCHAR(20) NOT NULL,
  4  AGE INT NOT NULL,
  5  ADDRESS CHAR(25),
  6  SALARY DECIMAL(18,2),
  7  PRIMARY KEY (ID)
  8  );

Table created.

SQL> DESC EMPLOYEES;
Name                               Null?      Type
-----
ID                                  NOT NULL  NUMBER(38)
NAME                                NOT NULL  VARCHAR2(20)
AGE                                  NOT NULL  NUMBER(38)
ADDRESS                             CHAR(25)
SALARY                              NUMBER(18,2)

SQL>

```

14.7.2 Alter statement The table can be modified or changed by using the Alter Command. The command is ALTER table Table Tablename(columnname datatype(size));

```
SQL> alter table employee modify salary number(15,2);
```

Table altered.

* Drop statement:

The SQL **DROP TABLE** statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

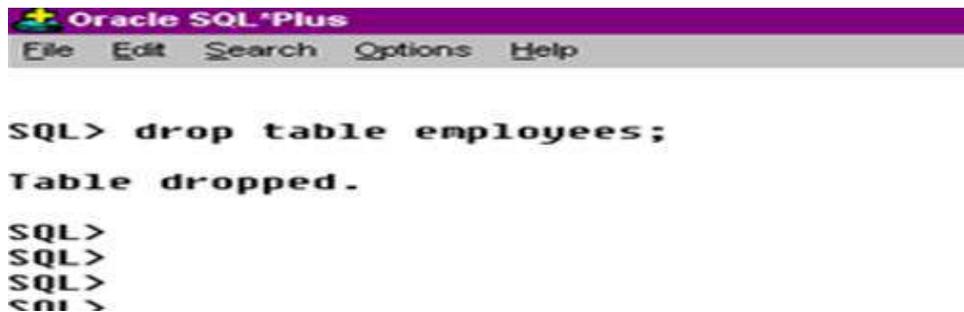
NOTE: You have to be careful while using this command because once a table is deleted then the table along with information available in the table would also be lost forever.

Syntax: Basic syntax of DROP TABLE statement is as follows:

```
DROP TABLE table_name;
```

Example: Let us first verify EMPLOYEES table and then we would delete it from the database:

This means EMPLOYEES table is available in the database, so let us drop it as follows:



```

Oracle SQL*Plus
File Edit Search Options Help

SQL> drop table employees;
Table dropped.

SQL>
SQL>
SQL>
cni >

```

14.7.3 Insert: The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax:

There are two basic syntaxes of INSERT INTO statement as follows:

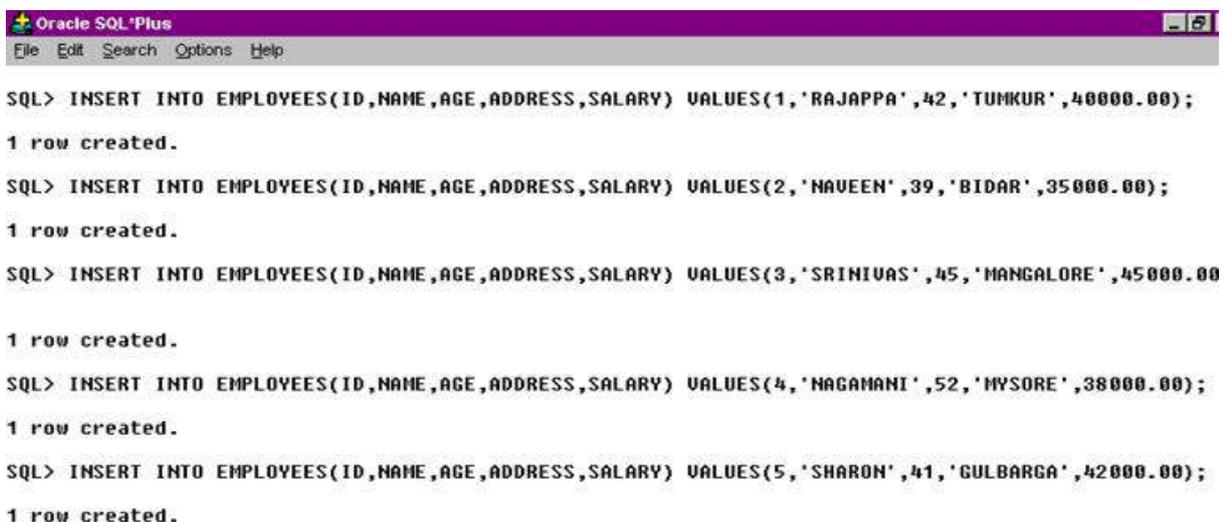
```

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);

```

Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows:



```

Oracle SQL*Plus
File Edit Search Options Help

SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(1,'RAJAPPA',42,'TUMKUR',40000.00);
1 row created.

SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(2,'HAVEEN',39,'BIDAR',35000.00);
1 row created.

SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(3,'SRINIIVAS',45,'MANGALORE',45000.00);
1 row created.

SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(4,'NAGAMANI',52,'MYSORE',38000.00);
1 row created.

SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(5,'SHARON',41,'GULBARGA',42000.00);
1 row created.

```

Populate one table using another table:

You can populate data into a table through select statement over another table provided another table has a set of fields, which are required to populate first table. Here is the syntax.

```
INSERT INTO first_table_name [(column1, column2,... columnN)]
  SELECT column1, column2,...columnN
  FROM second_table_name
  [WHERE condition];
```

14.7.4 SELECT : Select statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Syntax:

The basic syntax of SELECT statement is as follows:

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2, ... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax:

```
SELECT * FROM table_name;
```

Example:

Consider the EMPLOYEES table having the following records:

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(4,'NAGAHANI',52,'MYSORE',38000.00);
1 row created.
SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(5,'SHARON',41,'GULBARGA',42000.00);
1 row created.
SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(6,'XAVIO',40,'BANGALORE',33000.00);
1 row created.
SQL> INSERT INTO EMPLOYEES(ID,NAME,AGE,ADDRESS,SALARY) VALUES(7,'RAVINDRA',42,'SAGAR',43000.00);
1 row created.
SQL>
SQL>
SQL>
SQL> SELECT * FROM EMPLOYEES;
-----
ID NAME                AGE ADDRESS                SALARY
-----
1  RAJAPPA              42 TUMKUR                  40000
2  NAUEEN               39 BIDAR                   35000
3  SRINIIVAS            45 MANGALORE               45000
4  NAGAHANI             52 MYSORE                  38000
5  SHARON               41 GULBARGA                42000
6  XAVIO                40 BANGALORE               33000
7  RAVINDRA             42 SAGAR                   43000
-----
7 rows selected.
SQL> |
```

The SQL **WHERE** clause is used to specify a condition while fetching the data

from single table or joining with multiple tables.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement etc., which we would examine in subsequent chapters.

Syntax:

The basic syntax of SELECT statement with WHERE clause is as follows:

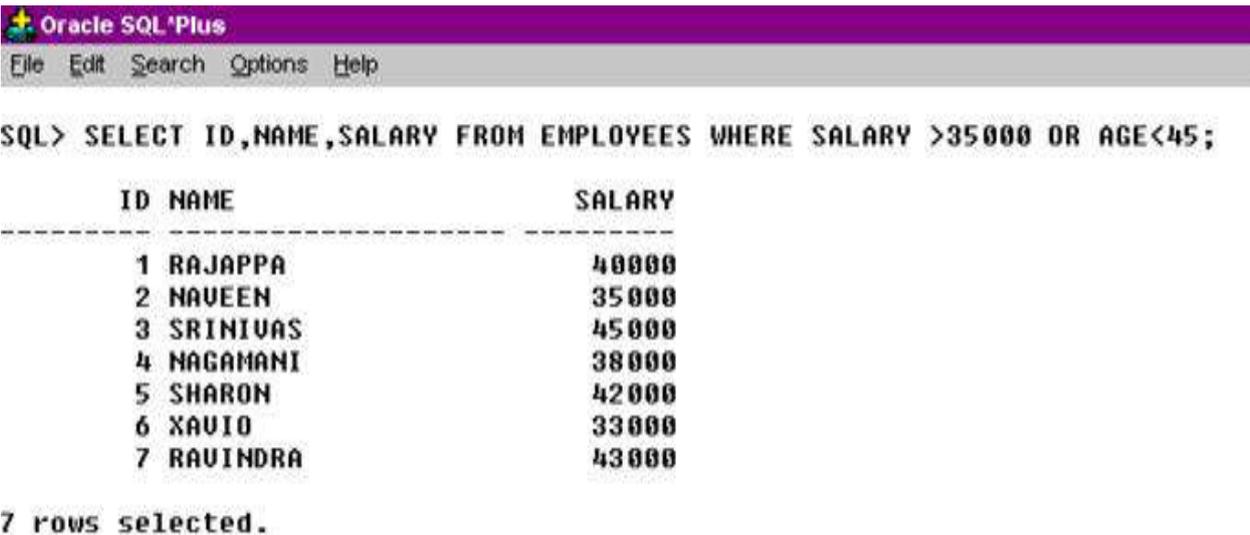
```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using comparison or logical operators like >, <, =, LIKE, NOT, etc. Below examples would make this concept clear.

Example:

Consider the EMPLOYEES table having the following records:

Following is an example which would fetch ID, Name and Salary fields from the EMPLOYEES table where salary is greater than 35000:



The screenshot shows the Oracle SQL*Plus interface with a menu bar (File, Edit, Search, Options, Help) and a command prompt. The command entered is: `SQL> SELECT ID,NAME,SALARY FROM EMPLOYEES WHERE SALARY >35000 OR AGE<45;`. The output is a table with three columns: ID, NAME, and SALARY. The data rows are: (1, RAJAPPA, 40000), (2, NAVEEN, 35000), (3, SRINIVAS, 45000), (4, NAGAMANI, 38000), (5, SHARON, 42000), (6, XAVIO, 33000), and (7, RAVINDRA, 43000). Below the table, it says "7 rows selected."

```
Oracle SQL*Plus
File Edit Search Options Help

SQL> SELECT ID,NAME,SALARY FROM EMPLOYEES WHERE SALARY >35000 OR AGE<45;

   ID NAME                SALARY
-----
    1 RAJAPPA                40000
    2 NAVEEN                 35000
    3 SRINIVAS              45000
    4 NAGAMANI              38000
    5 SHARON                42000
    6 XAVIO                 33000
    7 RAVINDRA             43000

7 rows selected.
```

Following is an example, which would fetch ID, Name and Salary fields Naveen. Here, it is important to note that all the strings should be given inside single quotes (") whereas numeric values should be given without any quote as in above example:

```
SQL> SELECT ID, NAME, SALARY
FROM EMPLOYEES
WHERE NAME ='Naveen';
```

This would produce the following result:

```
+----+-----+-----+
| ID | NAME      | SALARY |
+----+-----+-----+
| 2 | Naveen   | 35000.00 |
```

The SQL **AND** and **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

14.7.5 AND Operator:

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax:

The basic syntax of AND operator with WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE.

Example: Consider the EMPLOYEES table having the following records:

Following is an example, which would fetch ID, Name and Salary fields from the EMPLOYEES table where salary is greater than 2000 AND age is less than 25 years:

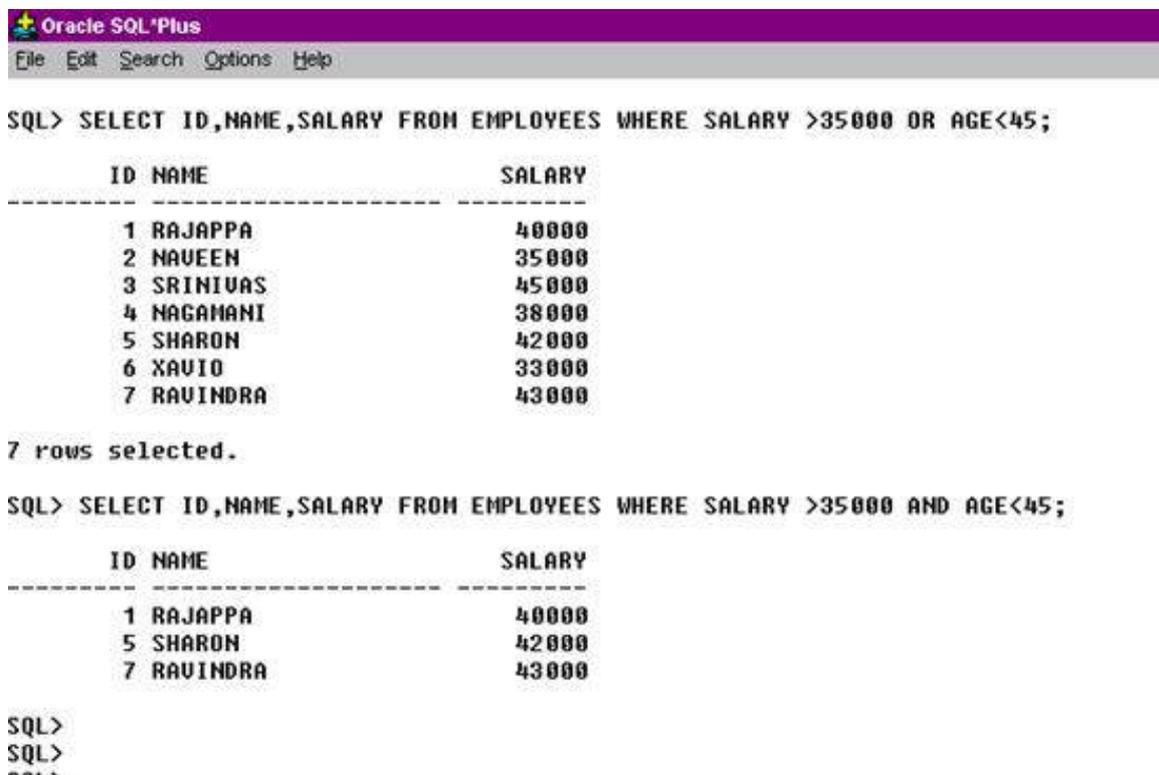
14.7.6 OR Operator:

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax: The basic syntax of OR operator with WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, only any ONE of the conditions separated by the OR must be TRUE.



```
Oracle SQL*Plus
File Edit Search Options Help

SQL> SELECT ID,NAME,SALARY FROM EMPLOYEES WHERE SALARY >35000 OR AGE<45;

   ID NAME                SALARY
-----
    1 RAJAPPA                40000
    2 NAVEEN                  35000
    3 SRINIVAS                45000
    4 NAGAMANI                 38000
    5 SHARON                   42000
    6 XAVIO                    33000
    7 RAVINDRA                 43000

7 rows selected.

SQL> SELECT ID,NAME,SALARY FROM EMPLOYEES WHERE SALARY >35000 AND AGE<45;

   ID NAME                SALARY
-----
    1 RAJAPPA                40000
    5 SHARON                   42000
    7 RAVINDRA                 43000

SQL>
SQL>
```

14.7.7 Update: The SQL **UPDATE** Query is used to modify the existing records

in a table.

You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected.

Syntax: The basic syntax of UPDATE query with WHERE clause is as follows:

```
UPDATE table_name
SET column1 = value1, column2 = value2..., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example:

Consider the EMPLOYEES table having the following records:

Following is an example, which would update ADDRESS for a customer whose ID is 6:

```
SQL> UPDATE EMPLOYEES SET ADDRESS = 'Bengaluru' WHERE ID =6;
```

```
Oracle SQL*Plus
File Edit Search Options Help
SQL>
SQL> SELECT * FROM EMPLOYEES;
```

ID	NAME	AGE	ADDRESS	SALARY
1	RAJAPPA	42	TUNKUR	40000
2	NAVEEN	39	BIDAR	35000
3	SRINIVAS	45	MANGALORE	45000
4	NAGAMANI	52	MYSORE	38000
5	SHARON	41	GULBARGA	42000
6	XAVIO	40	BANGALORE	33000
7	RAVINDRA	42	SAGAR	43000

7 rows selected.

```
SQL> UPDATE EMPLOYEES SET ADDRESS='BENGALURU' WHERE ID=6;
```

1 row updated.

```
SQL> SELECT * FROM EMPLOYEES;
```

ID	NAME	AGE	ADDRESS	SALARY
1	RAJAPPA	42	TUNKUR	40000
2	NAVEEN	39	BIDAR	35000
3	SRINIVAS	45	MANGALORE	45000
4	NAGAMANI	52	MYSORE	38000
5	SHARON	41	GULBARGA	42000
6	XAVIO	40	BENGALURU	33000
7	RAVINDRA	42	SAGAR	43000

7 rows selected.

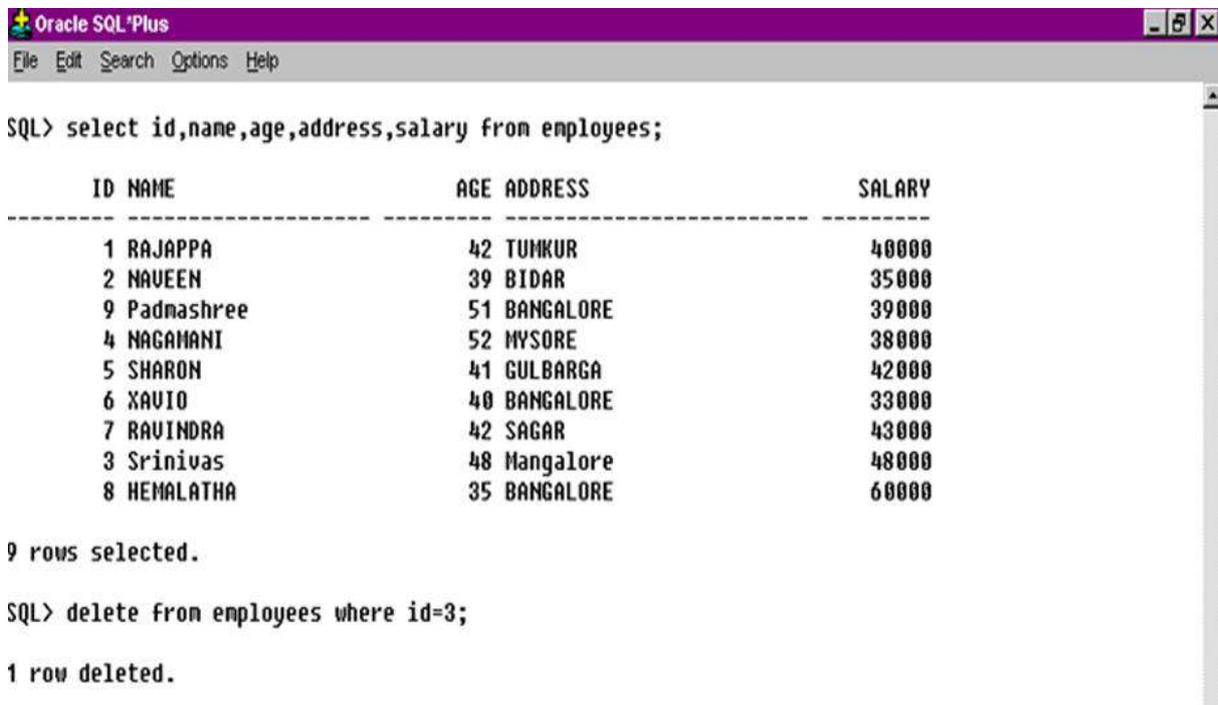
14.7.8 DELETE Query is used to delete the existing records from a table.

You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax: The basic syntax of DELETE query with WHERE clause is as follows:

```
DELETE FROM table_name
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators. Following is an example, which would DELETE a customer, whose ID is 3:



```
Oracle SQL*Plus
File Edit Search Options Help

SQL> select id,name,age,address,salary from employees;

   ID NAME                AGE ADDRESS                SALARY
-----
   1 RAJAPPA                42 TUMKUR                  40000
   2 NAVEEN                  39 BIDAR                   35000
   9 Padmashree              51 BANGALORE               39000
   4 NAGAMANI                52 MYSORE                   38000
   5 SHARON                  41 GULBARGA                42000
   6 XAVIO                    40 BANGALORE               33000
   7 RAVINDRA                42 SAGAR                    43000
   3 Srinivas                 48 Mangalore               48000
   8 HEMALATHA               35 BANGALORE               60000

9 rows selected.

SQL> delete from employees where id=3;

1 row deleted.
```

```
SQL> SELECT * FROM EMPLOYEES
WHERE ROWNUM <= 3;
```

This would produce the following result:

```
+---+-----+-----+-----+-----+
| ID | NAME          | AGE | ADDRESS | SALARY |
+---+-----+-----+-----+-----+
| 1 | Rajappa      | 42 | Tumkur  | 40000.00 |
| 2 | Naveen       | 39 | Bidar   | 35000.00 |
| 3 | Srinivas     | 45 | Mangalore | 32000.00 |
```

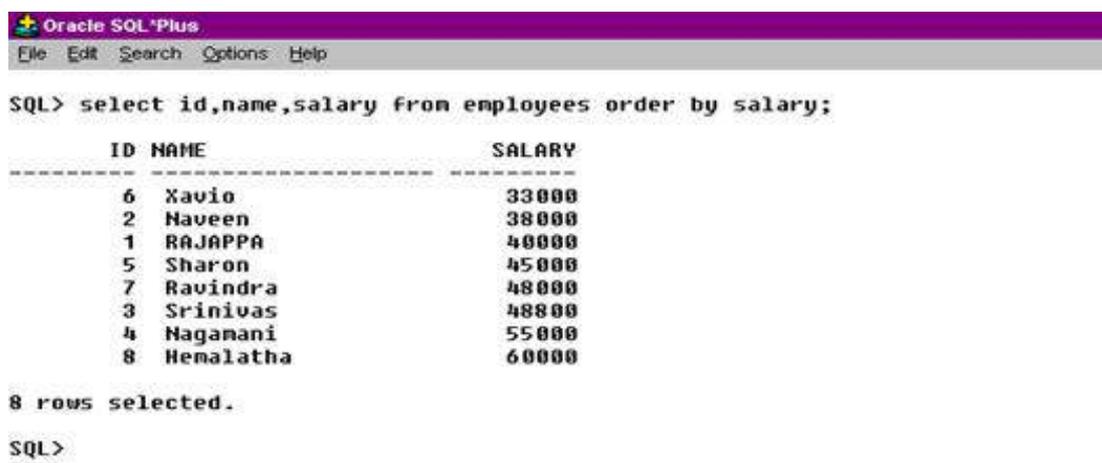
14.7.9 ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax: The basic syntax of ORDER BY clause is as follows:

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2,.. columnN][ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example: Consider the EMPLOYEES table having the following records:



```
Oracle SQL*Plus
File Edit Search Options Help

SQL> select id,name,salary from employees order by salary;

   ID NAME          SALARY
-----
    6 Xavio             33000
    2 Naveen           38000
    1 RAJAPPA          40000
    5 Sharon          45000
    7 Ravindra        48000
    3 Srinivas         48800
    4 Naganani         55000
    8 Hemalatha       60000

8 rows selected.

SQL>
```

14.7.10 GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax: The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: **COUNT, MAX, MIN, AVG, SUM, DISTINCT**

SQL COUNT (): This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.

For Example: If you want the number of employees in a particular department, the query would be:

```
SELECT COUNT (*) FROM employee
```

```
WHERE dept = 'Computer Science';
```

If you want the total number of employees in all the department, the query would take the form:

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

SQL DISTINCT(): This function is used to select the distinct rows.

For Example: If you want to select all distinct department names from employee table, the query would be:

```
Select Distinct dept FROM employee;
```

To get the count of employees with unique name, the query would be:

```
SELECT COUNT (DISTINCT name) FROM employee;
```

SQL MAX(): This function is used to get the maximum value from a column.

To get the maximum salary drawn by an employee, the query would be:

```
SELECT MAX (salary) FROM employee;
```

SQL MIN(): This function is used to get the minimum value from a column.

To get the minimum salary drawn by an employee, he query would be:

```
SELECT MIN (salary) FROM employee;
```

SQL AVG(): This function is used to get the average value of a numeric column.

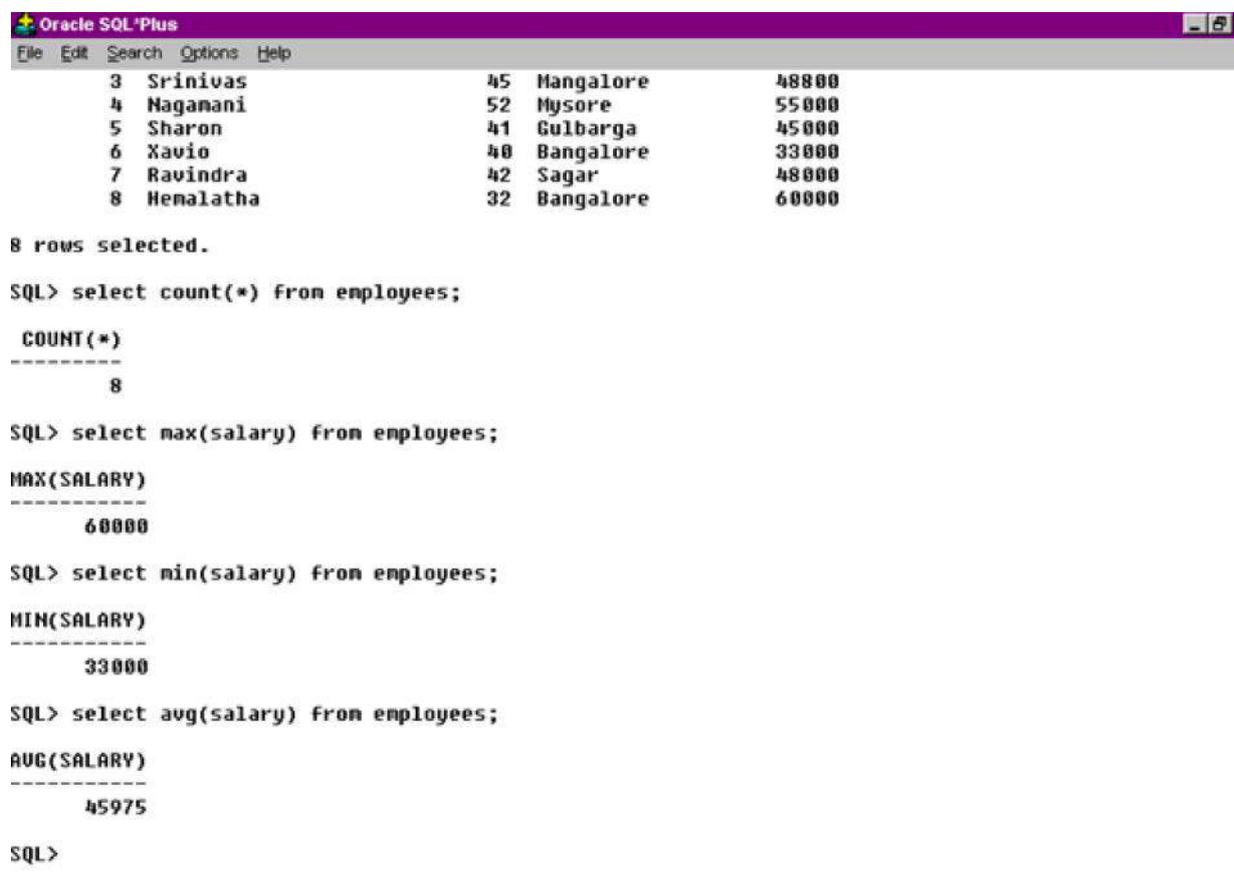
To get the average salary, the query would be

```
SELECT AVG (salary) FROM employee;
```

SQL SUM(): This function is used to get the sum of a numeric column

To get the total salary given out to the employees,

Example: Consider the EMPLOYEES table is having the following records:



```

Oracle SQL*Plus
File Edit Search Options Help
 3 Srinivas          45 Mangalore      48800
 4 Naganani          52 Mysore          55000
 5 Sharon            41 Gulbarga       45000
 6 Xavio              40 Bangalore     33000
 7 Ravindra          42 Sagar           48000
 8 Hemalatha         32 Bangalore     60000

8 rows selected.

SQL> select count(*) from employees;

COUNT(*)
-----
      8

SQL> select max(salary) from employees;

MAX(SALARY)
-----
      60000

SQL> select min(salary) from employees;

MIN(SALARY)
-----
      33000

SQL> select avg(salary) from employees;

AUG(SALARY)
-----
      45975

SQL>

```

```
SELECT SUM (salary) FROM employee;
```

Now again, if you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

```
SQL> SELECT NAME, SUM(SALARY) FROM EMPLOYEES
GROUP BY NAME;
```

DISTINCT keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax: The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

First, let us see how the following SELECT query returns duplicate salary records:

This would produce the following result where salary 45000 is coming twice which is a duplicate record from the original table.

```
SELECT DISTINCT column1, column2, .....columnN
FROM table_name
WHERE [condition]
```

ID	NAME	AGE	ADDRESS	SALARY
1	Rajappa	42	Tumkur	40000.00
2	Naveen	39	Bidar	35000.00
3	Srinivas	45	Mangalore	32000.00
4	Nagamani	52	Myosre	38000.00
5	Sharon	41	Gulbarga	42000.00
6	Xavio	40	Bangalore	33000.00
7	Ravindra	44	Sagar	43000.00

```
SQL> SELECT SALARY FROM EMPLOYEES
ORDER BY SALARY;
```



```
SQL> select id,name,salary from employees order by salary;
```

ID	NAME	SALARY
6	Xavio	33000
2	Naveen	38000
1	RAJAPPA	40000
5	Sharon	45000
7	Ravindra	48000
3	Srinivas	48800
4	Nagamani	55000
8	Hemalatha	60000

```
8 rows selected.
```

```
SQL>
```

14.7.12 Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables, (a) sports table is as follows:

(b) Another table is ORDER_items as follows:

Now, let us join these two tables in our SELECT statement as follows:

This would produce the following result:

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

SQL Join Types:

There are different types of joins available in SQL:

- **INNER JOIN:** returns rows when there is a match in both tables.
- **LEFT JOIN:** returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN:** returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN:** returns rows when there is a match in one of the tables.
- **SELF JOIN:** is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- **CARTESIAN JOIN:** returns the Cartesian product of the sets of records from the two or more joined tables.

In order to experiment the join commands, we are creating two tables one called the sports and the oder_items for the sports items. While preparing the join operations, we can use alias so that it become easy. This is illustrated in the given example. p is the sports table and o is the order items table.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create table sports(
  2 product_id number(5) constraint Pd_id_pk PRIMARY KEY,
  3 product_name char(20),
  4 supplier_name varchar(20),
  5 unit_price number(10)
  6 );

```

Table created.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> create table order_items(
  2 order_id number(5),
  3 product_id number(5),
  4 product_name char(20),
  5 supplier_name varchar(20),
  6 unit_price number(10),
  7 constraint od_id_pk primary key(order_id),
  8 constraint pd_id_fk foreign key (product_id) references sports(product_id)
  9 );

```

Table created.

```

SQL>
SQL>

```

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into order_items values(20141,101,'BAT', 'GPUC',3);

```

1 row created.

```

SQL>
SQL>
SQL>
SQL>

```

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select o.order_id,p.product_name,p.unit_price,p.supplier_name
  2 from sports p, order_items o
  3 where o.product_id=p.product_id;

```

ORDER_ID	PRODUCT_NAME	UNIT PRICE	SUPPLIER_NAME
20141	BAT	15000	SUNNY

```

SQL>
SQL>
SQL>

```

This would produce the following result:

There are two other clauses (i.e., operators), which are very similar to UNION clause:

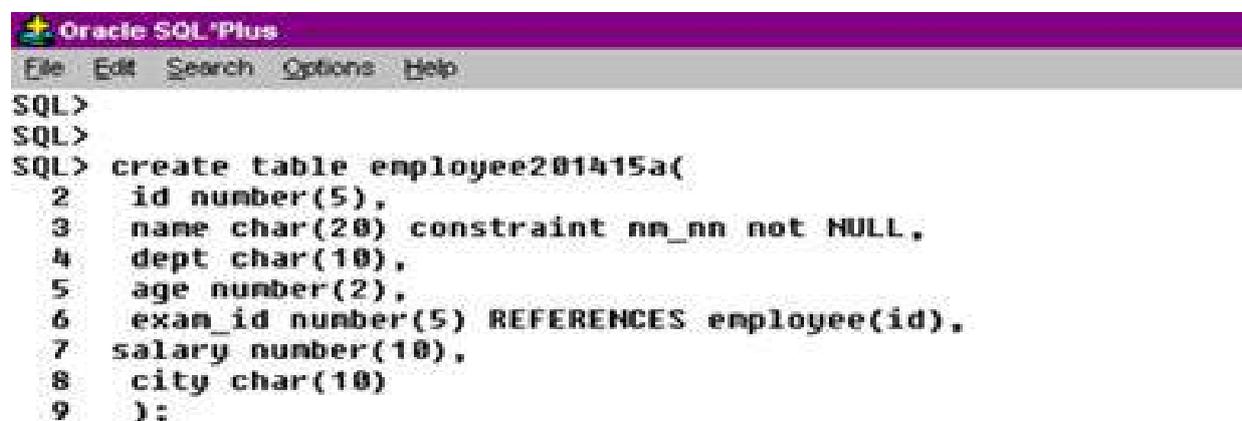
- SQL INTERSECT Clause: is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.
- SQL EXCEPT Clause : combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

14.7.13 NULL: The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax:

The basic syntax of **NULL** while creating a table:



```
Oracle SQL*Plus
File Edit Search Options Help
SQL>
SQL>
SQL> create table employee201415a(
  2   id number(5),
  3   name char(20) constraint nn_nn not NULL,
  4   dept char(10),
  5   age number(2),
  6   exam_id number(5) REFERENCES employee(id),
  7   salary number(10),
  8   city char(10)
  9   );
```

Table created.

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.

A field with a NULL value is one that has been left blank during record creation.

Example:

The NULL value can cause problems when selecting data, however, because when comparing an unknown value to any other value, the result is always unknown and not included in the final results.

You must use the **IS NULL** or **IS NOT NULL** operators in order to check for a NULL value.

Now, following is the usage of **IS NOT NULL** operator:

```
SQL> SELECT ID, NAME, AGE, ADDRESS, SALARY
      FROM EMPLOYEES
      WHERE SALARY IS NOT NULL;
```

This would produce the following result:

Now, following is the usage of **IS NULL** operator:

```
SQL> SELECT ID, NAME, AGE, ADDRESS, SALARY
      FROM EMPLOYEES
      WHERE SALARY IS NULL;
```

This would produce the following result:

You can rename a table or a column temporarily by giving another name known as alias.

```
SQL>
SQL> rename employee to employee2014;
```

Table renamed.

```
SQL> desc employee2014;
```

Name	Null?	Type
ID		NUMBER(5)
NAME		CHAR(20)
DEPT		CHAR(10)
AGE		NUMBER(2)
SALARY		NUMBER(15,2)
CITY		CHAR(10)

```
SQL>
SQL>
```

The use of table aliases means to rename a table in a particular SQL statement.

14.8 Creating Views:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example:

Consider the EMPLOYEES table having the following records:

```
SQL > CREATE VIEW EMPLOYEES_VIEW AS
SELECT name, age
FROM EMPLOYEES;
```

```
SQL > SELECT * FROM EMPLOYEES_VIEW;
```

14.9 The COMMIT Command:

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for COMMIT command is as follows:

```
COMMIT;
```

14.10 DCL commands are used to enforce database security in a multiple user database environment. Two types of DCL commands are GRANT and REVOKE. Only Database Administrator's or owners of the database object can provide/remove privileges on a database object.

14.10.1 GRANT Command

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

The Syntax for the GRANT command is:

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

▮ **privilege_name** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.

▮ **object_name** is the name of an database object like TABLE, VIEW, stored proc and SEQUENCE.

▮ **user_name** is the name of the user to whom an access right is being granted.

▮ **user_name** is the name of the user to whom an access right is being granted.

▮ **PUBLIC** is used to grant access rights to all users.

▮ **ROLES** are a set of privileges grouped together.

▮ **WITH GRANT OPTION** - allows a user to grant access rights to other users.

For Example: GRANT SELECT ON employee TO user1; This command grants a SELECT permission on employee table to user1. You should use the WITH GRANT option carefully because for example if you GRANT SELECT privilege on employee table to user1 using the WITH GRANT option, then user1 can GRANT SELECT privilege on employee table to another user, such as user2 etc. Later, if you REVOKE the SELECT privilege on employee from user1, still user2 will have SELECT privilege on employee table.

14.10.2 REVOKE Command:

The REVOKE command removes user access rights or privileges to the database objects.

The Syntax for the REVOKE command is:

```
REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}
```

For Example: REVOKE SELECT ON employee FROM user1; This command will REVOKE a SELECT privilege on employee table from user1. When you REVOKE SELECT privilege on a table from a user, the user will not be able to SELECT data from that table anymore. However, if the user has received SELECT privileges on that table from more than one users, he/she can SELECT from that table until everyone who granted the permission revokes it. You cannot REVOKE privileges if they were not initially granted by you.

Privileges and Roles:

Privileges: Privileges defines the access rights provided to a user on a database object. There are two types of privileges.

1) System privileges - This allows the user to CREATE, ALTER, or DROP Database objects.

2) Object privileges - This allows the user to EXECUTE, SELECT, INSERT, UPDATE, or Delete data from database objects to which the privileges apply.

Few CREATE system privileges are listed below:

System Privileges	Description
CREATE object	allows users to create the specified object in their own schema.
CREATE ANY object	allows users to create the specified object in any schema.

The above rules also apply for ALTER and DROP system privileges.

Few of the object privileges are listed below:

Object Privileges	Description
INSERT	allows users to insert rows into a table.
SELECT	allows users to select data from a database object.
UPDATE	allows user to update data in a table.
EXECUTE	allows user to execute a stored procedure or a function.

System Role	Privileges Granted to the Role
CONNECT	CREATE TABLE, CREATE VIEW, CREATE SYNONYM, CREATE SEQUENCE, CREATE SESSION etc.
RESOURCE	CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER etc. The primary usage of the RESOURCE role is to restrict access to database objects.
DBA	ALL SYSTEM PRIVILEGES

14.11 SQL built-in functions

There are two types of functions in Oracle sql version.

14.11.1 Single Row Functions: Single row or Scalar functions return a value for every row that is processed in a query.

14.11.2 Group Functions: These functions group the rows of data based on the values returned by the query. This is discussed in SQL GROUP Functions. The group functions are used to calculate aggregate values like total or average, which return just one total or one average value after processing a group of rows.

There are four types of single row functions. They are:

- 1) **Numeric Functions:** These are functions that accept numeric input and return numeric values.
- 2) **Character or Text Functions:** These are functions that accept character input and can return both character and number values.
- 3) **Date Functions:** These are functions that take values that are of datatype DATE as input and return values of datatype DATE, except for the MONTHS_BETWEEN function, which returns a number.
- 4) **Conversion Functions:** These are functions that help us to convert a value in one form to another form. For Example: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE etc.

You can combine more than one function together in an expression. This is known as nesting of functions.

DUAL Table in Oracle

This is a single row and single column dummy table provided by oracle. This is used to perform mathematical calculations without using a table.

Select * from DUAL

Output:

DUMMY

X

Select 777 * 888 from Dual

Output:

777 * 888

689976

Function Name	Return Value
ABS (x)	<u>Absolute value</u> of the number 'x'
CEIL (x)	Integer value that is Greater than or equal to the number 'x'
FLOOR (x)	Integer value that is Less than or equal to the number 'x'
TRUNC (x, y)	Truncates value of number 'x' up to 'y' <u>decimal places</u>
ROUND (x, y)	Rounded off value of the number 'x' up to the number 'y' decimal places

Function Name	Examples	Return Value
ABS (x)	ABS (1)	1
	ABS (-1)	-1
CEIL (x)	CEIL (2.83)	3
	CEIL (2.49)	3
	CEIL (-1.6)	-1
FLOOR (x)	FLOOR (2.83)	2
	FLOOR (2.49)	2
	FLOOR (-1.6)	-2
TRUNC (x, y)	ROUND (125.456, 1)	125.4
	ROUND (125.456, 0)	125
	ROUND (124.456, -1)	120
ROUND (x, y)	TRUNC (140.234, 2)	140.23
	TRUNC (-54, 1)	54
	TRUNC (5.7)	5
	TRUNC (142, -1)	140

These functions can be used on database columns.

For Example: Let's consider the product table used in sql joins. We can use ROUND to round off the unit_price to the nearest integer, if any product has prices in fraction.

```
SELECT ROUND (unit_price) FROM product;
```

2) Character or Text Functions:

Character or text functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.

Few of the character or text functions are as given below:

Function Name	Return Value
LOWER (string_value)	All the letters in ' <i>string_value</i> ' is converted to lowercase.
UPPER (string_value)	All the letters in ' <i>string_value</i> ' is converted to uppercase.
INITCAP (string_value)	All the letters in ' <i>string_value</i> ' is converted to mixed case.
LTRIM (string_value, trim_text)	All occurrences of ' <i>trim_text</i> ' is removed from the left of ' <i>string_value</i> '.
RTRIM (string_value, trim_text)	All occurrences of ' <i>trim_text</i> ' is removed from the right of ' <i>string_value</i> '.
TRIM (trim_text FROM string_value)	All occurrences of ' <i>trim_text</i> ' from the left and right of ' <i>string_value</i> ', ' <i>trim_text</i> ' can also be only one character long.
SUBSTR (string_value, m, n)	Returns ' <i>n</i> ' number of characters from ' <i>string_value</i> ' starting from the ' <i>m</i> ' position.
LENGTH (string_value)	Number of characters in ' <i>string_value</i> ' is returned.
LPAD (string_value, n, pad_value)	Returns ' <i>string_value</i> ' left-padded with ' <i>pad_value</i> '. The length of the whole string will be of ' <i>n</i> ' characters.
RPAD (string_value, n, pad_value)	Returns ' <i>string_value</i> ' right-padded with ' <i>pad_value</i> '. The length of the whole string will be of ' <i>n</i> ' characters.

For Example, we can use the above UPPER() text function with the column value as follows.

```
SELECT UPPER (product_name) FROM product;
```

The following examples explain the usage of the above character or text functions

Function Name	Examples	Return Value
LOWER(string_value)	LOWER('Good Morning')	good morning
UPPER(string_value)	UPPER('Good Morning')	GOOD MORNING
INITCAP(string_value)	INITCAP('GOOD MORNING')	Good Morning
LTRIM(string_value, trim_text)	LTRIM ('Good Morning', 'Good')	Morning
RTRIM (string_value, trim_text)	RTRIM ('Good Morning', 'Morning')	Good
TRIM (trim_text FROM string_value)	TRIM ('o' FROM 'Good Morning')	Gd Mrning
SUBSTR (string_value, m, n)	SUBSTR ('Good Morning', 6, 7)	Morning
LENGTH (string_value)	LENGTH ('Good Morning')	12
LPAD (string_value, n, pad_value)	LPAD ('Good', 6, '*')	**Good
RPAD (string_value, n, pad_value)	RPAD ('Good', 6, '*')	Good**

3) Date Functions:

These are functions that take values that are of datatype DATE as input and return values of datatypes DATE, except for the MONTHS_BETWEEN function, which returns a number as output.

Few date functions are as given below.

Function Name	Return Value
ADD_MONTHS (date, n)	Returns a date value after adding 'n' months to the date 'x'.
MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date 'x' rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.
TRUNC (x, date_format)	Returns the date 'x' lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.
NEXT_DAY (x, week_day)	Returns the next date of the 'week_day' on or after the date 'x' occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date 'x' specified.
SYSDATE	Returns the systems current date and time.
NEW_TIME (x, zone1, zone2)	Returns the date and time in zone2 if date 'x' represents the time in zone1.

Function Name	Examples	Return Value
ADD_MONTHS ()	ADD_MONTHS ('14-Feb-14', 9)	14-Nov-14
MONTHS_BETWEEN()	MONTHS_BETWEEN ('16-Sep-14', '16-Dec-14')	3
NEXT_DAY()	NEXT_DAY ('20-Mar-2014', 'Thursday')	21-Mar-2014
LAST_DAY()	LAST_DAY ('01-Jun-14')	30-Jun-14
NEW_TIME()	NEW_TIME ('01-Jun-18', 'IST', 'EST')	31-May-14

4) Conversion Functions:

These are functions that help us to convert a value in one form to another form. For Ex: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE.

Few of the conversion functions available in oracle are:

Function Name	Return Value
---------------	--------------

Function Name	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by 'date_format'.
NVL (x, y)	If 'x' is NULL, replace it with 'y'. 'x' and 'y' must be of the same datatype.
DECODE (a, b, c, d, e, default_value)	Checks the value of 'a', if $a = b$, then returns 'c'. If $a = d$, then returns 'e'. Else, returns <i>default_value</i> .

The below table provides the examples for the above functions

Function Name	Examples	Return Value
TO_CHAR ()	TO_CHAR (3000, '\$9999')	\$3000
	TO_CHAR (SYSDATE, 'Day, Month YYYY')	WEDNESDAY, MARCH 2014
TO_DATE ()	TO_DATE ('19-MAR-2014')	19-MAR-14
NVL ()	NVL (null, 1)	1

Summary

- >Sql -Structured Query Language(SQL)
- >SQL ARCHITECTURE:
- >SQL languages: DDL,DML,DCL,
- > Data access and retrieval
- >SQL built-in function.

Review questions

One mark questions

1. Expand SQL.
2. Give the syntax for create command in SQL.
3. What is drop command in SQL.
4. Give the command to display all the details in the table.
5. What is update command?
6. What is commit command?

Two marks questions

1. Classify Numeric and Character string data types in SQL.
2. Classify various SQL operators.
3. Which are the logical operators in SQL.
4. How do you modify the column name and width for existing table?
5. Write the syntax for distinct command in SQL.
6. What is the use of NULL value?
7. What is create view command?
8. What is dual table?

Three marks questions

1. Explain the features of SQL?
2. List the components of SQL architecture.
3. Explain DDL commands with example.
4. Explain DML commands with example.
5. Explain with an example Boolean expression in SQL.
6. Explain AND operator using where in SQL.
7. List the built-in functions associated with Group functions in SQL.
8. What is the use of join command?
9. What are privileges and rules?
10. Classify various built-in functions in SQL.

Five marks questions

1. Explain SQL constraints with example.
2. Explain with example to create details of employees and give the minimum and maximum in the salary domain.
3. Write the differences between order by and group by with example.