SQL is a standard language for storing, manipulating and retrieving data in databases.

What is SQL?

SQL stands for Structured Query Language
SQL lets you access and manipulate databases
SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

SQL can execute queries against a database
SQL can retrieve data from a database
SQL can insert records in a database
SQL can update records in a database
SQL can delete records from a database
SQL can create new databases
SQL can create new tables in a database
SQL can create stored procedures in a database
SQL can create views in a database
SQL can set permissions on tables, procedures, and views

Using SQL in Your Web Site

To build a web site that shows data from a database, you will need:
An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
To use a server-side scripting language, like PHP or ASP
To use SQL to get the data you want
To use HTML / CSS to style the page

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- 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.

SQL Table

SQL Table is a collection of data which is organized in terms of rows and columns. In DBMS, the table is known as relation and row as a tuple.

Table is a simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of the **EMPLOYEE** table:

EMP_ID	EMP_NAME	CITY	PHONE_NO
1	Kristen	Washington	7289201223
2	Anna	Franklin	9378282882
3	Jackson	Bristol	9264783838
4	Kellan	California	7254728346
5	Ashley	Hawaii	9638482678

In the above table, "EMPLOYEE" is the table name, "EMP_ID", "EMP_NAME", "CITY", "PHONE_NO" are the column names. The combination of data of multiple columns forms a row, e.g., 1, "Kristen", "Washington" and 7289201223 are the data of one row.

SQL Data Types

SQL data types can be broadly divided into following categories.

- Numeric data types such as int, tinyint, bigint, float, real etc.
- Date and Time data types such as Date, Time, Datetime etc.
- Character and String data types such as char, varchar, text etc.
- Unicode character string data types, for example nchar, nvarchar, ntext etc.
- Binary data types such as binary, varbinary etc.

SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with maximum length of 8,000 characters
VARCHAR	Variable length storage with maximum length of 8,000 characters
VARCHAR(max)	Variable length storage with provided max characters, not supported in MySQL
TEXT	Variable length storage with maximum size of 2GB data

SQL Numeric Data Types

Datatype	From	To
bit	0	1
tinyint	0	255
smallint	-32,768	32,767
int	-2,147,483,648	2,147,483,647
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

SQL Date and Time Data Types

DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS
TIMESTAMP	Stores number of seconds passed since the Unix epoch (‘1970-01-01 00:00:00’ UTC)
YEAR	Stores year in 2 digit or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069.

Operation on Table

- Create table
- Drop table
- Delete table
- Rename table

SQL Create Table

SQL create table is used to create a table in the database. To define the table, you should define the name of the table and also define its columns and column's data type.

Syntax

```
create table "table_name"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
```

Example

```
SQL> CREATE TABLE EMPLOYEE (
EMP_ID INT                    NOT NULL,
EMP_NAME VARCHAR (25) NOT NULL,
PHONE_NO INT                  NOT NULL,
ADDRESS CHAR (30),
PRIMARY KEY (ID)
);
```

If you create the table successfully, you can verify the table by looking at the message by the SQL server. Else you can use DESC command as follows:

SQL> DESC EMPLOYEE;

Field	Type	Null	Key	Default	Extra
EMP_ID	int(11)	NO	PRI	NULL	
EMP_NAME	varchar(25)	NO		NULL	
PHONE_NO	NO	int(11)		NULL	
ADDRESS	YES			NULL	char(30)

Drop table

A SQL drop table is used to delete a table definition and all the data from a table. When this command is executed, all the information available in the table is lost forever, so you have to very careful while using this command.

Syntax

DROP TABLE "table_name";

SQL>DROP TABLE EMPLOYEE;

SQL DELETE table

In SQL, DELETE statement is used to delete rows from a table. We can use WHERE condition to delete a specific row from a table. If you want to delete all the records from the table, then you don't need to use the WHERE clause.

Syntax

DELETE FROM table_name WHERE condition;

Suppose, the EMPLOYEE table having the following records:

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Denzel	Boston	7353662627	100000
4	Angelina	Denver	9232673822	600000
5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

The following query will DELETE an employee whose ID is 2.

```
SQL> DELETE FROM EMPLOYEE
WHERE EMP_ID = 3;
```

Now, the EMPLOYEE table would have the following records.

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
4	Angelina	Denver	9232673822	600000
5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

NOTE: If you don't specify the WHERE condition, it will remove all the rows from the table.

SQL SELECT Statement

In SQL, the SELECT statement is used to query or retrieve data from a table in the database. The returns data is stored in a table, and the result table is known as result-set.

Syntax

SELECT column1, column2, ...
FROM table_name;

Here, the expression is the field name of the table that you want to select data from.

Use the following syntax to select all the fields available in the table:

SELECT * FROM table_name;

Employee

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Angelina	Denver	9232673822	600000
4	Robert	Washington	9367238263	350000
5	Christian	Los angels	7253847382	260000

To fetch the EMP_ID of all the employees, use the following query:

SELECT EMP_ID FROM EMPLOYEE;

Q1. SELECT EMP_ID FROM EMPLOYEE;

EMP_ID
1
2
3
4
5

Q2. SELECT EMP_NAME, SALARY FROM EMPLOYEE;

EMP_NAME	SALARY
Kristen	150000
Russell	200000
Angelina	600000
Robert	350000
Christian	260000

SQL INSERT Statement

The SQL INSERT statement is used to insert a single or multiple data in a table.

In SQL, You can insert the data in two ways:

- Without specifying column name
- By specifying column name

Without specifying column name

If you want to specify all column values, you can specify or ignore the column names.

Syntax

```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value 3, .... Value N);
```

Example: INSERT INTO EMPLOYEE VALUES (6, 'Marry', 'Canada', 600000, 48);

To insert partial column values, you must have to specify the column names.

Syntax

```
INSERT INTO TABLE_NAME
[(col1, col2, col3,.... col N)]
VALUES (value1, value2, value 3, .... Value N);
```

Example:
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, AGE) VALUES (7, 'Jack', 40);

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48
7	Jack	null	null	40

Note: In SQL INSERT query, if you add values for all columns then there is no need to specify the column name. But, you must be sure that you are entering the values in the same order as the column exists.

SQL Update Statement

The SQL UPDATE statement is used to modify the data that is already in the database. The condition in the WHERE clause decides that which row is to be updated.

Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating single record

- Update the column EMP_NAME and set the value to 'Emma' in the row where SALARY is 500000.

Syntax

```
UPDATE table_name
SET column_name = value
WHERE condition;
```

EXAMPLE:

```
UPDATE EMPLOYEE
SET EMP_NAME = 'Emma'
WHERE SALARY = 500000;
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Emma	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating multiple records

If you want to update multiple columns, you should separate each field assigned with a comma.

In the EMPLOYEE table, update the column EMP_NAME to 'Kevin' and CITY to 'Boston' where EMP_ID is 5.

Syntax

UPDATE table_name
SET column_name = value1, column_name2 = value2
WHERE condition;

Query

```
UPDATE EMPLOYEE
SET EMP_NAME = 'Kevin', City = 'Boston'
WHERE EMP_ID = 5;
```

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Kevin	Boston	200000	36
6	Marry	Canada	600000	48

Without use of WHERE clause

If you want to update all row from a table, then you don't need to use the WHERE clause. In the EMPLOYEE table, update the column EMP_NAME as 'Harry'.

Syntax

```
UPDATE table_name
SET column_name = value1;
```

Query

```
UPDATE EMPLOYEE
SET EMP_NAME = 'Harry';
```

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Harry	Chicago	200000	30
2	Harry	Austin	300000	26
3	Harry	Denver	100000	42
4	Harry	Washington	500000	29
5	Harry	Los angels	200000	36
6	Harry	Canada	600000	48

SQL DELETE Statement

The SQL DELETE statement is used to delete rows from a table. Generally, DELETE statement removes one or more records form a table.

Syntax

DELETE FROM table_name WHERE some_condition;

Deleting Single Record

Delete the row from the table EMPLOYEE where EMP_NAME = 'Kristen'.

This will delete the row having name ‘Kristen’..

Query

```
DELETE FROM EMPLOYEE
WHERE EMP_NAME = 'Kristen';
```

Deleting Multiple Record

Delete the row from the EMPLOYEE table where AGE is 30. This will delete two rows(first and third row).

Query

```
DELETE FROM EMPLOYEE WHERE AGE= 30;
```

Delete all of the records

Delete all the row from the EMPLOYEE table. After this, no records left to display. The EMPLOYEE table will become empty.

Syntax

DELETE * FROM table_name;

or

DELETE FROM table_name;

SQL - Constraints

- Constraints are the rules enforced on the data columns of a 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 either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Integrity Constraints

Integrity constraints are used to **ensure accuracy and consistency of the data** in a relational database.

- Data integrity is handled in a relational database through the concept of referential integrity.
- There are many types of integrity constraints that play a role in **Referential Integrity (RI)**. These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints which are mentioned below.

- **NOT NULL Constraint– Ensures that a column cannot have NULL value.**
- **DEFAULT Constraint – Provides a default value for a column when none is specified.**
- **UNIQUE Constraint – Ensures that all values in a column are different.**
- **PRIMARY Key – Uniquely identifies each row/record in a database table.**
- **FOREIGN Key – Uniquely identifies a row/record in any of the given database table.**
- **CHECK Constraint– The CHECK constraint ensures that all the values in a column satisfies certain conditions.**
- **INDEX – Used to create and retrieve data from the database very quickly.**

NOT NULL Constraint–

- By default, a column can hold NULL values. **If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.**
- A NULL is not the same as no data, rather, it represents unknown data.

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, are ID NAME and AGE, In this we specify not to accept NULLs −

CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID));

- If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column in Oracle and MySQL, you would write a query like the one that is shown in the following code block.

ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) NOT NULL;

DEFAULT Constraint

- For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
 ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
 ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2) DEFAULT 5000.00,
PRIMARY KEY (ID) );
```

- If the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query like the one which is shown in the code block below.

```
ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00;
```

Drop Default Constraint

- To drop a DEFAULT constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS ALTER COLUMN SALARY DROP DEFAULT;
```

The UNIQUE Constraint

The UNIQUE Constraint prevents two records from having identical values in a column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having an identical age.

- For example, the following SQL query creates a new table called CUSTOMERS and adds five columns. Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

- If the CUSTOMERS table has already been created, then to add a UNIQUE constraint to the AGE column. You would write a statement like the query that is given in the code block below.

```
CREATE TABLE CUSTOMERS
( ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL UNIQUE,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID) );
```

ALTER TABLE CUSTOMERS MODIFY AGE INT NOT NULL UNIQUE;

You can also use the following syntax, which supports naming the constraint in multiple columns as well.

ALTER TABLE CUSTOMERS ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL query.
ALTER TABLE CUSTOMERS DROP CONSTRAINT myUniqueConstraint;

CHECK Constraint

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered the table.

Example

For example, the following program creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER who is below 18 years.

```
CREATE TABLE CUSTOMERS
( ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL CHECK (AGE >= 18),
ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2),
 PRIMARY KEY (ID) );
```

- If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement like the one given below.

ALTER TABLE CUSTOMERS MODIFY AGE INT NOT NULL CHECK (AGE >= 18);

- **You can also use the following syntax, which supports naming the constraint in multiple columns as well –**

ALTER TABLE CUSTOMERS ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL syntax. This syntax does not work with MySQL.

- ALTER TABLE CUSTOMERS DROP CONSTRAINT myCheckConstraint;

Example Employee Table

- CREATE TABLE EMPLOYEE
- (FNAME VARCHAR2(40), MINIT VARCHAR2(40), LNAME VARCHAR2(40),
- SSN int PRIMARY KEY,
- BDATE DATE,
- ADDRESS VARCHAR2(40),
- **SEX CHAR(1) CONSTRAINT CHK_SEX CHECK (SEX IN ('M','F','m','f')),**
- SALARY int,
- SUPERSSN int REFERENCES EMPLOYEE(SSN),
- DNO int);

- ALTER TABLE EMPLOYEE ADD FOREIGN KEY(DNO) REFERENCES DEPARTMENT (DNUMBER);

Foreign Key

- A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.
- A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
- **The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.**
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Example

Consider the structure of the following two tables.

CUSTOMERS table

```
CREATE TABLE CUSTOMERS(
C_ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2),
PRIMARY KEY (ID) );
```

ORDERS table

```
CREATE TABLE ORDERS
(O_ID INT NOT NULL,
 DATE DATETIME,
CUSTOMER_ID INT references CUSTOMERS(C_ID),
AMOUNT double,
PRIMARY KEY (ID) );
```

- If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

ALTER TABLE ORDERS ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);

DROP a FOREIGN KEY Constraint

- To drop a FOREIGN KEY constraint, use the following SQL syntax.

ALTER TABLE ORDERS DROP FOREIGN KEY;

LIKE OPERATOR

The SQL LIKE Operator:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
There are two wildcards often used in conjunction with the LIKE operator:
% - The percent sign represents zero, one, or multiple characters
_ - The underscore represents a single character

The percent sign and the underscore can also be used in combinations!
LIKE Syntax

SELECT *column1, column2, ...*
FROM *table_name*
WHERE *columnN* LIKE *pattern*;

Tip: You can also combine any number of conditions using AND or OR operators.
Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

- **The following SQL statement selects all customers with a Customer Name starting with "a":**

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

- **The following SQL statement selects all customers with a CustomerName that have "or" in any position:**

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

- **The following SQL statement selects all customers with a CustomerName that have "r" in the second position:**

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Exercise:

1. Retrieve the birthdate and address of the employee whose name is 'Franklin T. Wong'

2. Retrieve **all customers with a CustomerName that starts with "a" and are at least 3 characters in length**

3. Retrieve all customers with a ContactName that starts with "a" and ends with "o“

4. selects all customers with a CustomerName that does NOT start with "a"

- **The following**

SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';

- The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';

- The following SQL statement selects all customers with a CustomerName that does NOT start with "a":

SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';

- Retrieve the birthdate and address of the employee whose name is 'Franklin T. Wong'

SELECT BDATE, ADDRESS FROM EMPLOYEE WHERE FNAME='Franklin' AND MINIT='T' AND LNAME='Wong'

SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of the query.

Alias Column Syntax

SELECT *column_name* AS *alias_name*
FROM *table_name;*

Alias Table Syntax

SELECT *column_name(s)*
FROM *table_name* AS *alias_name;*

Alias for Columns Examples

- The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column:

Example

- SELECT CustomerID AS ID, CustomerName AS CName
  FROM Customers;

The following SQL statement creates two aliases, one for the CustomerName column and one for the ContactName column.

Note: It requires double quotation marks or square brackets if the alias name contains spaces:

Example

- SELECT CustomerName AS Customer, ContactName AS [Contact Person]
  FROM Customers;

The following SQL statement creates an alias named "Address" that combine four columns (Address, PostalCode, City and Country):
Example

Note: To get the SQL statement above to work in MySQL use the following:

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address
FROM Customers;
```

CustomerName	Address
Alfreds Futterkiste	Obere Str. 57, 12209 Berlin, Germany

Alias for Tables Example

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn).

We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;
```

Aliases can be useful when:

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

The SQL IN Operator(Shorthand for multiple OR Conditions)

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

IN Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1*, *value2*, ...);

or:

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*SELECT STATEMENT*);

IN Operator Examples

- **The following SQL statement selects all customers that are located in "Germany", "France" or "UK":**

Example

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

- **The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":**

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

- **The following SQL statement selects all customers that are from the same countries as the suppliers:**

Example

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

The SQL AND, OR and NOT Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition:
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition1* AND *condition2* AND *condition3 ...*;

OR Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition1* OR *condition2* OR *condition3 ...*;

NOT Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE NOT *condition*;

AND Example

- **The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":**

Example

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

Exercise:

- **Retrieve all fields from "Customers" where city is "Berlin" OR "München":**
- **Retrieve fields from "Customers" where country is "Germany" OR "Spain":**

NOT Example

- **Retrieve all fields from "Customers" where country is NOT "Germany“**
- **Retrieve all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions)**
- **Retrieve all fields from "Customers" where country is NOT "Germany" and NOT "USA":**

Example 1.

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

Example 2

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

Example 3

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Example 4

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

Example 5

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

The SQL BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.
BETWEEN Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* BETWEEN *value1* AND *value2;*

BETWEEN Example

- **The following SQL statement selects all products with a price BETWEEN 10 and 20:**

Example
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;

NOT BETWEEN Example

- **To display the products outside the range of the previous example, use NOT BETWEEN:**
- **Example**

SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;

BETWEEN with IN Example

- **The following SQL statement selects all products with a price BETWEEN 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:**
- **Example**

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND NOT CategoryID IN (1,2,3);
```

BETWEEN Text Values Example

- **The following SQL statement selects all products with a ProductName BETWEEN Carnarvon Tigers and Mozzarella di Giovanni:**

Example

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

- **The following SQL statement selects all products with a ProductName BETWEEN Carnarvon Tigers and Chef Anton's Cajun Seasoning:**

Example

```
SELECT * FROM Products
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"
ORDER BY ProductName;
```

SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

Subqueries with the Select Statement

SQL subqueries are most frequently used with the Select statement.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
( SELECT column_name  from table_name WHERE ... );
```

Example

Consider the EMPLOYEE table have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be:

```
SELECT *
   FROM EMPLOYEE
   WHERE ID IN (SELECT ID
   FROM EMPLOYEE
   WHERE SALARY > 4500);
```

Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
INSERT INTO table_name (column1, column2, column3....)
SELECT *
FROM table_name
WHERE VALUE OPERATOR
```

Example

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

- Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYEE_BKP
  SELECT * FROM EMPLOYEE
  WHERE ID IN (SELECT ID
  FROM EMPLOYEE);
```

Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table.

- **The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29**.

```
UPDATE EMPLOYEE
SET SALARY = SALARY * 0.25
 WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
 WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	1625.00
5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

```
DELETE FROM TABLE_NAME
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table.

- **The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.**

```
DELETE FROM EMPLOYEE
  WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
    WHERE AGE >= 29 );
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

Exercise:

Faculty(fid,fname,qualification,depid)

Department(depid,dname)

Example 1. find name of faculty working in ‘it’ department.

Example 2. find name of faculty working in ‘it’ or ‘cs’ department.

Solution:

Example 1:

```
select fname from faculty
where depid in(select depid from department where dname='cs');
```

Example 2:

```
select  fname from faculty
 where depid in(select depid from department where dname in('cs','it'));
```

The SQL EXISTS Operator

- The EXISTS operator is used to test for the existence of any record in a sub-query.
- The EXISTS operator returns true if the sub-query returns one or more records.

EXISTS Syntax
SELECT *column_name(s)*
FROM *table_name*
WHERE EXISTS
(SELECT *column_name* FROM *table_name* WHERE *condition*);

- **The following SQL statement returns TRUE and** will return all records of student table

Example:

- Select *
- from student
- where exists(select * from department where depid=1);

Since inner query is returning value as department table is having one record with depid=1 there fore outer query will return all records of student table

- Select *
- from student
- where exists(select * from department where depid=5);

Here inner query is not returning value as department table is having no record with depid=5 ,therefore outer query will not any records of student table

Not exist is just opposite of Exist i.e. if inner query is true(RETURNING VALUES) the outer query will return empty set and vice versa.

COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

COUNT(*) OR

COUNT([ALL|DISTINCT] expression)

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

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

Output:

10

Example: COUNT with WHERE

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
WHERE RATE>=20;
```

Output:

7

Example: COUNT() with DISTINCT

```
SELECT COUNT(DISTINCT COMPANY)
FROM PRODUCT_MAST;
```

Output:

3

Example: COUNT() with GROUP BY

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
HAVING COUNT(*)>2;
```

Output:

```
Com1    5     Com2    3
```

SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

```
SUM()
or
SUM( [ALL|DISTINCT] expression )
```

Example: SUM()

```
SELECT SUM(COST)
FROM PRODUCT_MAST;
```

Output:

```
670
```

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3;
```

Output:

320

Example: SUM() with GROUP BY

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3
GROUP BY COMPANY;
```

Output:

Com1 150 Com2 170

Example: SUM() with HAVING

```
SELECT COMPANY, SUM(COST)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING SUM(COST)>=170;
```

Output:

Com1 335 Com3 170

AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

or

AVG([ALL|DISTINCT] expression)

Example:

SELECT AVG(COST)
FROM PRODUCT_MAST;

Output:

67.00

MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX() OR

MAX([ALL|DISTINCT] expression)

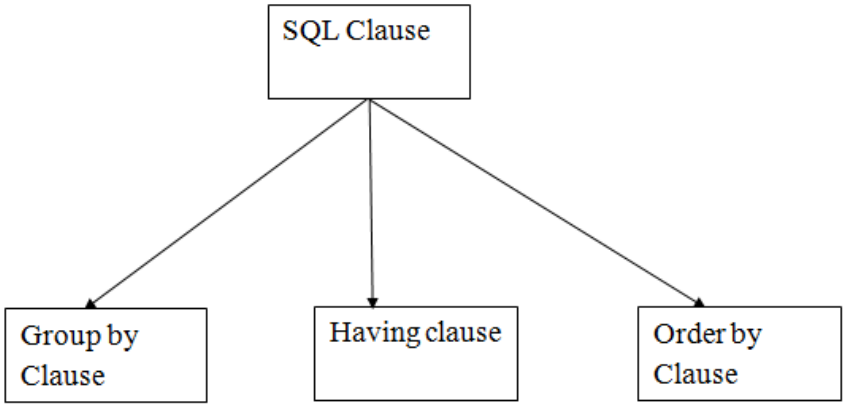
Example:

SELECT MAX(RATE)
FROM PRODUCT_MAST;

30

SQL Clauses

The following are the various SQL clauses:



1**. GROUP BY**

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

The GROUP BY statement is used with aggregation function.

Syntax

SELECT column
FROM table_name
WHERE conditions
GROUP BY column
ORDER BY column

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
```

Output:

Com1 5 Com2 3 Com3 2

Example:

Consider schema :

Student(sid,sname,branch,marks)

- Find total number of students in each branch
- Find total marks of each branch
- Find average marks of each branch having marks branch only CSE or IT
- find average marks of students who are not from department 1 or 3.

Solutions:

Example 1:

```
Select  branch,count(*)
from student
group by branch;
```

Example 2:

```
Select branch,sum(marks)
From student
Group by branch;
```

Example 3:

```
select branch,avg(marks)
 from student
group by branch
having branch in('cs','it');
```

Example 4:

```
Select branch,avg(marks) from student  where branch in(select dname
from department where depid not in(1,3)) group by branch;
```

- **HAVING**
- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause.
- If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE conditions
GROUP BY column1, column2
HAVING conditions
ORDER BY column1, column2;
```

Example:

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

Output:

Com1 5 Com2 3

ORDER BY

The ORDER BY clause sorts the result-set in ascending or descending order.
It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE condition
ORDER BY column1, column2... ASC|DESC;
```

Where

ASC: It is used to sort the result set in ascending order by expression.
DESC: It sorts the result set in descending order by expression.

Example: Sorting Results in Ascending Order

Table:

CUSTOMER

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
23	David	Bangkok
34	Alina	Dubai
45	John	UK
56	Harry	US

Enter the following SQL statement:

```
SELECT *
FROM CUSTOMER
ORDER BY NAME;
```

OUTPUT

CUSTOMER_ID	NAME	ADDRESS
34	Alina	Dubai
23	David	Bangkok
56	Harry	US
45	John	UK
12	Kathrin	US

SQL JOIN

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".
In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

- NATURAL JOIN
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
LEFT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
RIGHT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
FULL JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

SQL Set Operation

The SQL Set operation is used to combine the two or more SQL **SELECT statements.**

Types of Set Operation

Union
UnionAll
Intersect
Minus

Union

The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
The union operation eliminates the duplicate rows from its result set.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union SQL query will be:

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The result set table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example: Using the above First and Second table.
Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The result set table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

Intersect

It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
In the Intersect operation, the number of datatype and columns must be the same.
It has no duplicates and it arranges the data in ascending order by default.

Syntax

SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;

Example:

Using the above First and Second table.

Intersect query will be:

SELECT * FROM First
INTERSECT
SELECT * FROM Second;

ID	NAME
3	Jackson

Minus

It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example

Using the above First and Second table.

Minus query will be:

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

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

Creating View from a single table

In this example, we create a View named **firstview** from the table Student.

Query:

```
create view firstview as
select id, sname from student
where branch='it';
```

Just like table query, we can query the view to view the data.

SELECT * FROM **firstview**;

Creating View from multiple tables

- View from multiple tables can be created by simply include multiple tables in the SELECT statement.
- In the given example, a view is created named MULTIPLETABLEVIEW from two tables Student and Department.

Query:

```
CREATE VIEW MULTIPLETABLEVIEW AS
SELECT ID, SNAME, DNAME FROM STUDENT, DEPARTMENT
 WHERE STUDENT.BRANCH=DEPARTMENT.DNAME;
```

To display data of View MarksView:

```
SELECT * FROM  MULTIPLETABLEVIEW;
```

Deleting View

A view can be deleted using the Drop View statement.

Syntax

DROP VIEW view_name;

Example:

If we want to delete the View FIRSTVIEW, we can do this as:

DROP VIEW FIRSTVIEW;

```
CREATE TABLE EMPLOYEE ( FNAME VARCHAR2(40), MINIT VARCHAR2(40), LNAME
VARCHAR2(40),
SSN NUMBER(5) PRIMARY KEY,
BDATE DATE, ADDRESS VARCHAR2(40),
SEX CHAR(1) CONSTRAINT CHK_SEX CHECK (SEX IN ('M','F','m','f')),
SALARY NUMBER(5),
SUPERSSN NUMBER(5) REFERENCES EMPLOYEE(SSN),
DNO NUMBER(1));


CREATE TABLE DEPARTMENT
(DNAME VARCHAR2(20),
DNUMBER NUMBER(1) PRIMARY KEY,
MGRSSN NUMBER(5) REFERENCES EMPLOYEE(SSN),
MGRSTARTDATE DATE);

CREATE TABLE DEPT_LOCATIONS
(DNUMBER NUMBER(1),
DLOCATION VARCHAR2(40),
PRIMARY KEY(DNUMBER, DLOCATION),
FOREIGN KEY(DNUMBER) REFERENCES DEPARTMENT(DNUMBER));
```

```
CREATE TABLE PROJECT
(PNAME VARCHAR2(40),
PNUMBER NUMBER(2) PRIMARY KEY,
PLOCATION VARCHAR2(40),
DNUM NUMBER(1) REFERENCES DEPARTMENT(DNUMBER));


CREATE TABLE WORKS_ON(ESSN NUMBER(5),
PNO NUMBER(2),
HOURS NUMBER(5),
PRIMARY KEY(ESSN, PNO),
FOREIGN KEY(ESSN) REFERENCES EMPLOYEE(SSN),
FOREIGN KEY(PNO) REFERENCES PROJECT(PNUMBER));

CREATE TABLE DEPENDENT (ESSN NUMBER(5),
DEPENDENT_NAME VARCHAR2(40),
SEX CHAR(1) CONSTRAINT CHK_SEX2 CHECK (SEX IN ('M','F','m','f')),
BDATE DATE,
RELATIONSHIP VARCHAR2(40),
 PRIMARY KEY(ESSN, DEPENDENT_NAME),
FOREIGN KEY(ESSN) REFERENCES EMPLOYEE(SSN));
```

- Q1. Display all the details of all employees working in the company.

```
SELECT * FROM EMPLOYEE;
```

- Q2. Display ssn, lname, fname, address of employees who work in department no 7.

```
SELECT SSN, LNAME, FNAME, ADDRESS FROM EMPLOYEE WHERE DNO=7;
```

- Q3. Retrieve the birthdate and address of the employee whose name is 'Franklin T. Wong'

```
SELECT BDATE, ADDRESS FROM EMPLOYEE WHERE FNAME='Franklin' AND MINIT='T' AND
LNAME='Wong';
```

- Q4. Retrieve the name and salary of every employee

```
SELECT FNAME||' '||MINIT||' '||LNAME AS NAME, SALARY FROM EMPLOYEE;
```

- Q5. Retrieve all distinct salary values

```
SELECT DISTINCT(SALARY) FROM EMPLOYEE;
```

- Q6. Retrieve all employee names whose address is in ‘Bellaire’

```
SELECT FNAME||' '||MINIT||' '||LNAME AS NAME FROM EMPLOYEE WHERE ADDRESS
LIKE '%Bellaire%';
```

- Q7. Retrieve all employees who were born during the 1950s

```
SELECT * FROM EMPLOYEE WHERE EXTRACT(YEAR FROM BDATE)=1950;
```

- Q8. Retrieve all employees in department 5 whose salary is between 50,000 and 60,000(inclusive)

```
SELECT * FROM EMPLOYEE WHERE DNO=5 AND SALARY BETWEEN 50000
AND 60000;
```

- Q9. Retrieve the names of all employees who do not have supervisors

```
SELECT FNAME||' '||MINIT||' '||LNAME AS NAME FROM EMPLOYEE
WHERE SUPERSSN IS NULL;
```

NAME
James E Borg

- Q10. Retrieve SSN and department name for all employees

```
SELECT E.SSN, D.DNAME FROM EMPLOYEE E, DEPARTMENT D WHERE
E.DNO=D.DNUMBER;
```

SSN	DNAME
33344	Research
99988	Administration
98765	Administration
45345	Research
98798	Administration
12345	Research
66688	Research
88866	Headquarters

- Q11. Retrieve the name and address of all employees who work for the 'Research' department

- SELECT E.FNAME||' '||E.MINIT||' '||E.LNAME AS NAME, E.ADDRESS FROM EMPLOYEE E, DEPARTMENT D WHERE E.DNO=D.DNUMBER AND D.DNAME='Research';

NAME	ADDRESS
Franklin T Wong	638, Voss, Houston, TX
Joyce A English	5631, Rice, Houston, TX
John B Smith	731, Fondren, Houston, TX
Ramesh K Narayan	975, Fire Oak, Humble, TX

- Q12. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

SELECT P.PNUMBER, P.DNUM, E.LNAME, E.ADDRESS, E.BDATE FROM PROJECT P, EMPLOYEE E, DEPARTMENT D WHERE P.PLOCATION='Stafford' AND P.DNUM=D.DNUMBER AND D.MGRSSN=E.SSN;

PNUMBER	DNUM	LNAME	ADDRESS	BDATE
10	4	Wallace	291, Berry, Bellaire, TX	06/20/1941
30	4	Wallace	291, Berry, Bellaire, TX	06/20/1941

- Q13. For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
SELECT E.FNAME||' '||E.MINIT||' '||E.LNAME EMPLOYEE, M.FNAME||' '||M.MINIT||'
'||M.LNAME MANAGER FROM EMPLOYEE E, EMPLOYEE M WHERE E.SUPERSSN=M.SSN;
```

EMPLOYEE	MANAGER
Ramesh K Narayan	Franklin T Wong
John B Smith	Franklin T Wong
Joyce A English	Franklin T Wong
Ahmad V Jabbar	Jennifer S Wallace
Alicia J Zelaya	Jennifer S Wallace
Jennifer S Wallace	James E Borg
Franklin T Wong	James E Borg

- Q14. Retrieve all combinations of Employee Name and Department Name

```
SELECT E.FNAME||' '||E.MINIT||' '||E.LNAME NAME, D.DNAME FROM EMPLOYEE E,
DEPARTMENT D WHERE E.DNO=D.DNUMBER;
```

NAME	DNAME
Franklin T Wong	Research
Alicia J Zelaya	Administration
Jennifer S Wallace	Administration
Joyce A English	Research
Ahmad V Jabbar	Administration
John B Smith	Research
Ramesh K Narayan	Research
James E Borg	Headquarters

- Q14. Retrieve all combinations of Employee Name and Department Name

```
SELECT E.FNAME||' '||E.MINIT||' '||E.LNAME NAME, D.DNAME FROM
EMPLOYEE E, DEPARTMENT D WHERE E.DNO=D.DNUMBER;
```

Output –

NAME	DNAME
Franklin T Wong	Research
Alicia J Zelaya	Administration
Jennifer S Wallace	Administration
Joyce A English	Research
Ahmad V Jabbar	Administration
John B Smith	Research
Ramesh K Narayan	Research
James E Borg	Headquarters

- Q15. Make a list of all project numbers for projects that involve an employee whose last name is 'Narayan' either as a worker or as a manager of the department that controls the project.

```
SELECT DISTINCT(PNUMBER) FROM PROJECT WHERE PNUMBER IN(SELECT P.PNUMBER
FROM PROJECT P, EMPLOYEE E, DEPARTMENT D WHERE E.LNAME='Narayan' AND
D.MGRSSN=E.SSN AND P.DNUM=D.DNUMBER) OR PNUMBER IN (SELECT W.PNO FROM
WORKS_ON W, EMPLOYEE E WHERE E.SSN=W.ESSN AND E.LNAME='Narayan');
```

PNUMBER
3

- Q16(a).Increase the salary of all employees working on the 'ProductX' project by 15%. (b)Retrieve employee name and increased salary of these employees.

a)UPDATE EMPLOYEE SET SALARY=SALARY+SALARY*0.15 WHERE SSN IN (SELECT ESSN FROM WORKS_ON WHERE PNO IN (SELECT PNUMBER FROM PROJECT WHERE PNAME='ProductX'));

b) SELECT FNAME||' '||MINIT||' '||LNAME AS NAME, SALARY FROM EMPLOYEE WHERE SSN IN (SELECT ESSN FROM WORKS_ON WHERE PNO IN (SELECT PNUMBER FROM PROJECT WHERE PNAME='ProductX'));

NAME	SALARY
John B Smith	34500
Joyce A English	28750

Q17. Retrieve a list of employees and the project name each works in, ordered by the employee's department, and within each department ordered alphabetically by employee first name.

```
SELECT E.FNAME,E.LNAME,P.PNAME FROM EMPLOYEE E, DEPARTMENT D,
PROJECT P WHERE E.DNO=D.DNUMBER AND D.DNUMBER=P.DNUM ORDER
BY D.DNAME, E.FNAME ASC;
```

FNAME	LNAME	PNAME
Ahmad	Jabbar	Computerization
Ahmad	Jabbar	Newbenefits
Alicia	Zelaya	Computerization
Alicia	Zelaya	Newbenefits
Jennifer	Wallace	Computerization
Jennifer	Wallace	Newbenefits
James	Borg	Reorganization
Franklin	Wong	ProductZ
Franklin	Wong	ProductX
Franklin	Wong	ProductY
John	Smith	ProductZ
John	Smith	ProductX
John	Smith	ProductY
Joyce	English	ProductX
Joyce	English	ProductZ
Joyce	English	ProductY
Ramesh	Narayan	ProductX
Ramesh	Narayan	ProductY
Ramesh	Narayan	ProductZ

Q18. Select the names of employees whose salary does not match with salary of any employee in department 10.

```
SELECT FNAME||' '||MINIT||' '||LNAME NAME FROM EMPLOYEE WHERE SALARY NOT IN(SELECT SALARY FROM EMPLOYEE WHERE DNO=10) ;
```

Q19. Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

```
SELECT E.FNAME||' '||E.MINIT||' '||E.LNAME NAME FROM EMPLOYEE E, DEPENDENT T WHERE E.FNAME=T.DEPENDENT_NAME AND E.SEX=T.SEX AND T.ESSN=E.SSN;
```

Q20. Retrieve the employee numbers of all employees who work on project located in Bellaire, Houston, or Stafford.

```
SELECT SSN FROM EMPLOYEE WHERE SSN IN (SELECT ESSN FROM WORKS_ON WHERE PNO IN(SELECT PNUMBER FROM PROJECT WHERE PLOCATION IN('Bellaire','Houston','Stafford')));
```

Q21. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary. Display with proper headings.

```
SELECT SUM(SALARY) "Total Salary", MAX(SALARY) "Maximum Salary", MIN(SALARY) "Minimum Salary", AVG(SALARY) "Average Salary" FROM EMPLOYEE;
```

Output –

Total Salary	Maximum Salary	Minimum Salary	Average Salary
289250	55000	25000	36156.25

Q22. Find the sum of the salaries and number of employees of all employees of the ‘Marketing’ department, as well as the maximum salary, the minimum salary, and the average salary in this department.

Select count(*)”No Of Emp. In Research Dept.”,sum(salary)”Salary Sum”,max(salary)”Max Salary,min(salary)”Minimum Salary”,avg(salary)”Average Salary” from employee,department Where dname=’marketing’ and dno=dnumber;

Q23. Select the names of employees whose salary is greater than the average salary of all employees in department 10
SELECT FNAME||' '||MINIT||' '||LNAME "NAME FROM EMPLOYEE WHERE SALARY >(SELECT AVG(SALARY) FROM EMPLOYEE WHERE DNO=10) ;

Q24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

SELECT DNO, COUNT(DNO), AVG(SALARY) FROM EMPLOYEE GROUP BY DNO;

DNO	COUNT(DNO)	AVG(SALARY)
1	1	55000
5	4	35312.5
4	3	31000

Q25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

SELECT PNUMBER, PNAME, COUNT(*) FROM PROJECT, WORKS_ON WHERE PNUMBER=PNO GROUP BY PNUMBER, PNAME;

PNUMBER	PNAME	COUNT(*)
20	Reorganization	3
1	ProductX	2
10	Computerization	3
30	Newbenefits	3
2	ProductY	3
3	ProductZ	2

Q26. Change the location and controlling department number for all projects having more than 5 employees to ‘Bellaire’ and 6 respectively.

UPDATE PROJECT SET PLOCATION='Bellaire', DNUM=6 WHERE PNUMBER IN(SELECT PNO FROM WORKS_ON GROUP BY PNO HAVING COUNT(PNO)>5);

Q27. For each department having more than 10 employees, retrieve the department no, no of employees drawing more than 40,000 as salary

SELECT DNUMBER, COUNT(*) FROM DEPARTMENT, EMPLOYEE WHERE DNUMBER=DNO AND SALARY>40000 AND DNO IN(SELECT DNO FROM EMPLOYEE GROUP BY DNO HAVING COUNT(*)>10) GROUP BY DNUMBER;

Q28. Insert a record in Project table which violates referential integrity constraint with respect to Department number. Now remove the violation by making necessary insertion in the Department table

INSERT INTO PROJECT VALUES ('ProductA',4,'Spring',2);

```
ORA-02291: integrity constraint (PRACTICAL 1.SYS_C007378) violated - parent key not found
```

INSERT INTO DEPARTMENT VALUES ('Analysis',2,66688, TO_DATE('3-FEB-1990','DD-MON-YYYY'));

Output –

INSERT INTO PROJECT VALUES ('ProductA',4,'Spring',2);

Output –

Q29. Delete all dependents of employee whose ssn is ‘123456789’.

```
DELETE FROM DEPENDENT WHERE ESSN=12345;
```

Q30. Perform a query using alter command to drop/add field and a constraint in Employee table

```
ALTER TABLE DEPENDENT DROP CONSTRAINT CHK_SEX2;
 ALTER TABLE DEPENDENT ADD CONSTRAINT CHK_SEX2 CHECK (SEX IN ('M','F','m','f'));
ALTER TABLE EMPLOYEE ADD Bonus NUMBER(5);
ALTER TABLE EMPLOYEE DROP COLUMN Bonus;
```