SQL
Structured Query Language

Objectives

- Explore basic commands and functions of SQL
- How to use SQL for data administration (to create tables, indexes, and views)
- How to use SQL for data manipulation (to add, modify, delete, and retrieve data)
- How to use SQL to query a database to extract useful information

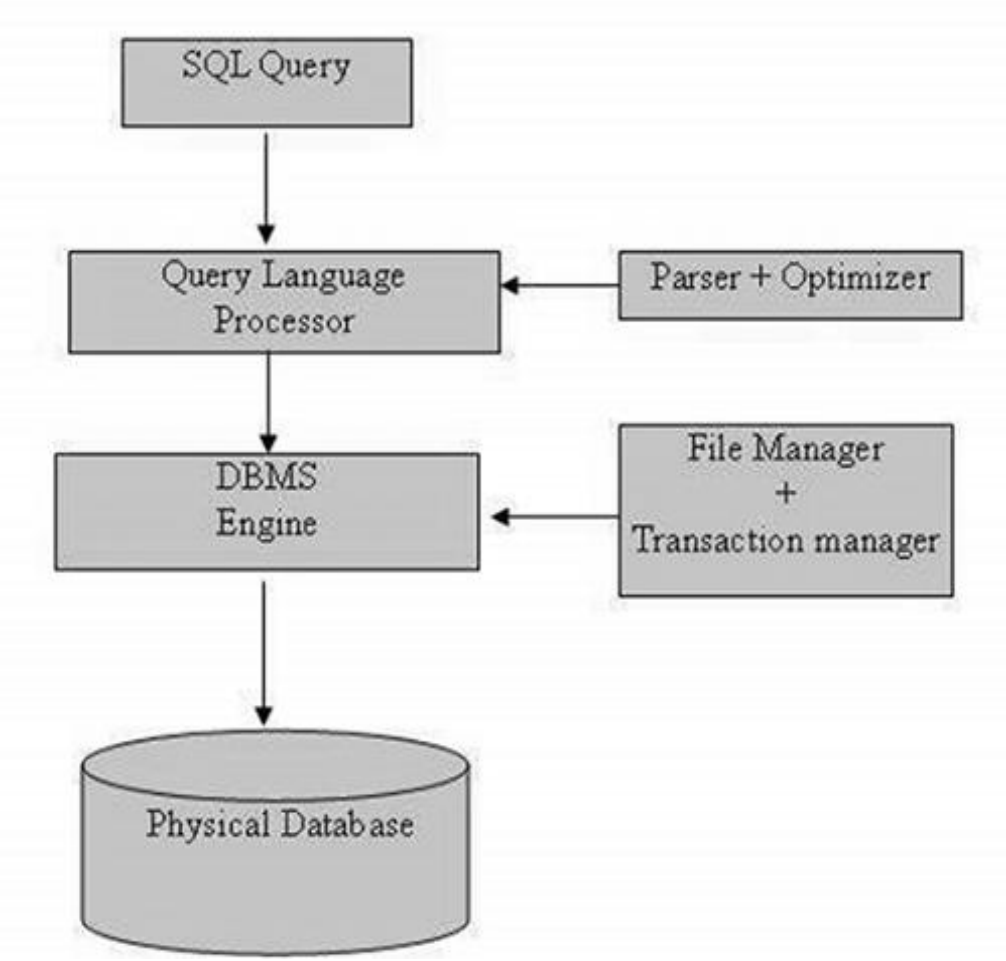
INTRODUCTION

- What is SQL?
- How SQL Process / SQL Architecture
- SQL Classification
  - ✓ DDL - Data Definition Language
  - ✓ DML - Data Manipulation Language
  - ✓ DCL - Data Control Language
- SQL Syntax
- Data Types
  - ✓ Numeric Data Types
  - ✓ Approximate Numeric Data Types
  - ✓ Date and Time Data Types
  - ✓ Character Strings Data Types
  - ✓ Unicode Character Strings Data Types
  - ✓ Binary Data Types
- SQL Operators
  - ✓ Arithmetic Operators
  - ✓ Comparison Operators
  - ✓ Logical Operators

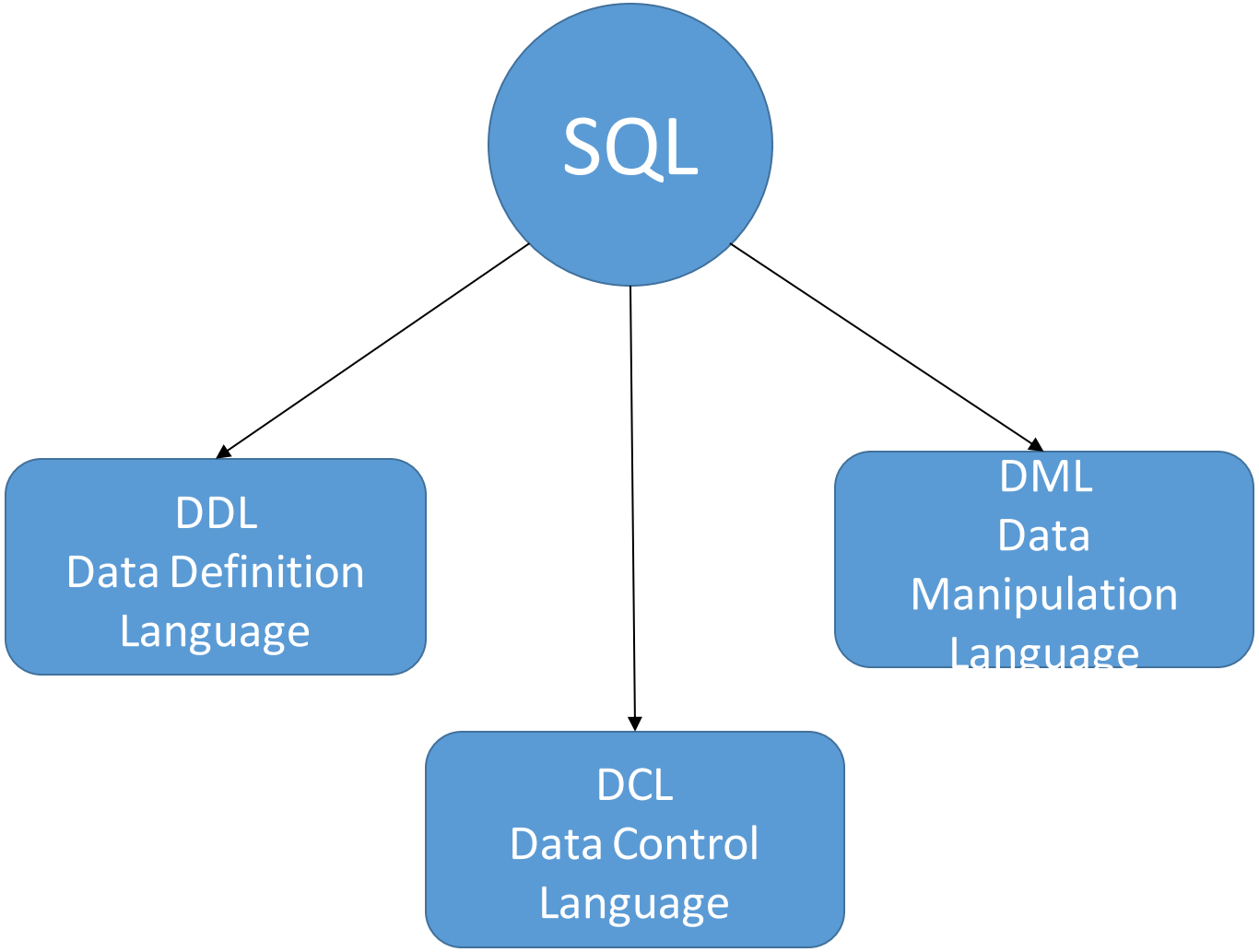
What is SQL?

- SQL is a language to operate databases
- It includes database creation, deletion, fetching rows, modifying rows, etc.
- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.
- like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

How SQL Process / SQL Architecture



SQL Classification



DDL - Data Definition Language

- Any change required in the structure of table or in database, then DDL commands are use

Sr.No.	Command & Description
1	**CREATE** Creates a new table, a view of a table, or other object in the database.
2	**ALTER** Modifies an existing database object, such as a table.
3	**DROP** Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

- DML commands enable us to work with data that goes into the database, such as INSERT, SELECT, UPDATE and DELETE records

Sr.No.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.
2	**INSERT** Creates a record.
3	**UPDATE** Modifies records.
4	**DELETE** Deletes records.

DCL - Data Control Language

- DCL commands allow us to give permission on the particular database or table, such as GRANT, REVOKE

Sr.No.	Command & Description
1	**GRANT** Gives a privilege to user.
2	**REVOKE** Takes back privileges granted from user.

SQL Syntax

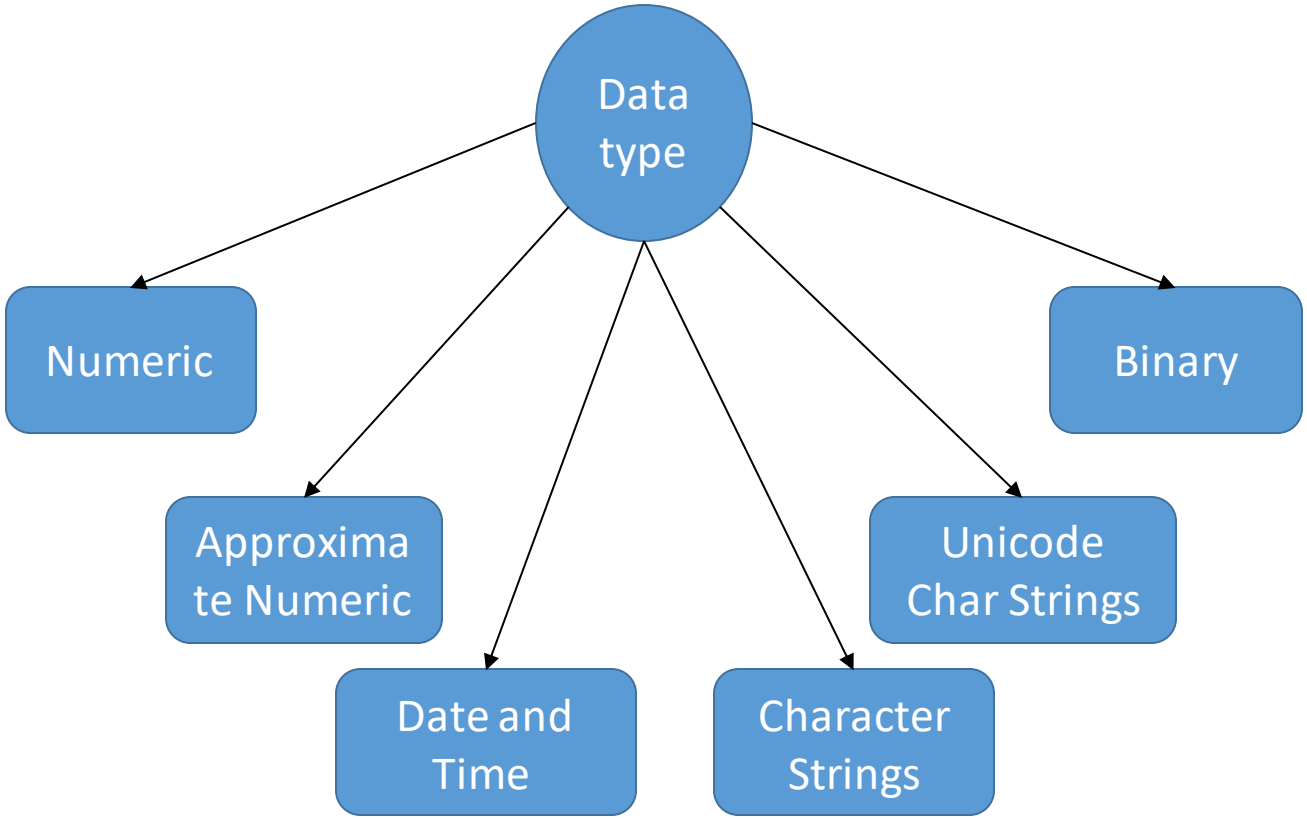
- SQL is followed by a unique set of rules and guidelines called Syntax
- All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

SQL CREATE TABLE syntax

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype);
```

Data Types

- SQL Data Type is an attribute that specifies the type of data of any object.
- Each column, variable and expression has a related data type in SQL.
- We can use these data types while creating our tables.



Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Character Strings Data Types

Sr.No.	DATA TYPE & Description
1	**char** Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
2	**varchar** Maximum of 8,000 characters.(Variable-length non-Unicode data).
3	**varchar(max)** Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
4	**text** Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types

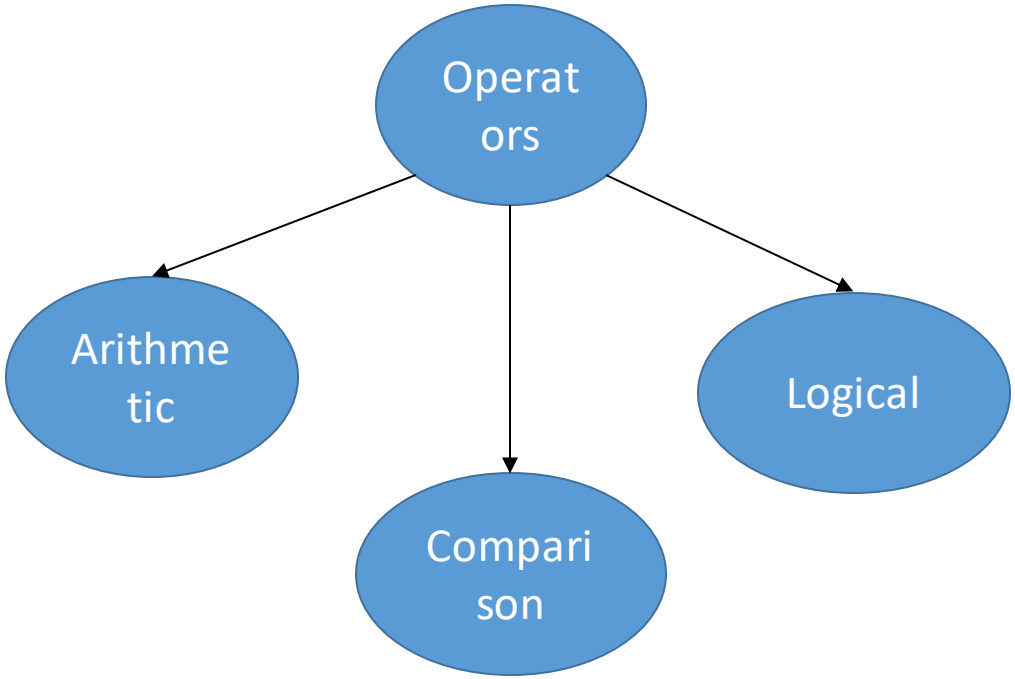
Sr.No.	DATA TYPE & Description
1	**nchar** Maximum length of 4,000 characters.(Fixed length Unicode)
2	**nvarchar** Maximum length of 4,000 characters.(Variable length Unicode)
3	**nvarchar(max)** Maximum length of 231characters (SQL Server 2005 only).(Variable length Unicode)
4	**ntext** Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types

Sr.No.	DATA TYPE & Description
1	**binary** Maximum length of 8,000 bytes(Fixed-length binary data)
2	**varbinary** Maximum length of 8,000 bytes.(Variable length binary data)
3	**varbinary(max)** Maximum length of 231 bytes (SQL Server 2005 only). (Variable length Binary data)
4	**image** Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

SQL Operators

- An operator is a reserved word or a character used in SQL statement's when ever it is required
- These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.



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

Comparison Operators

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.

Logical Operators

Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.
2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	**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.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.

Logical Operators (Cont…)

6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.
8	**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.**
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.
11	**UNIQUE** The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL Queries

CREATE Database

- The SQL **CREATE <u>DATABASE</u>** statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE DatabaseName;
```

Example

```
CREATE DATABASE testDB;
```

SHOW Database

- Once a database is created, you can check it in the list of databases by SQL command **SHOW DATABASE**

Syntax

```
show databases;
```

Example

```
show databases;
```

SELECT Database

- When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.
- The SQL **USE** statement is used to select any existing database.

Syntax

```
USE DatabaseName;
```

Example

```
USE testDB;
```

CREATE Table

- Creating a basic table involves naming the table and defining its columns and each column's data type.The SQL **USE** statement is used to select any existing database.
- The SQL **CREATE TABLE** statement is used to create a new table.

Syntax

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,…);
```

Example

```
CREATE TABLE CUSTOMERS( ID INT , NAME VARCHAR (20) , AGE INT , ADDRESS CHAR (25) , SALARY DECIMAL (18, 2) );
```

DESCRIBE Table Structure

- if your table has been created successfully and want to look its colum and table structure
- The SQL **DESC** statement is used to describe table structure.

Syntax

```
DESC tablename
```

Example

```
DESC CUSTOMERS;
```

INSERT Query

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

Syntax

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

or

INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);

Example

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (1, 'Ramesh ', 32, 'Ahmedabad', 2000.00);

or

INSERT INTO CUSTOMERS VALUES (7, 'Muffy', 24, 'Indore', 10000.00);

SELECT Query

- The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

  Syntax

  ```
  SELECT * FROM table_name;
  ```

  or

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

  Example

  ```
  SELECT * FROM CUSTOMERS;
  ```

  or

  ```
  SELECT ID, NAME, SALARY FROM CUSTOMERS;
  ```

WHERE Clause

- The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- If the given condition is satisfied, then only it returns a specific value from the table.

Syntax

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

condition may be (comparison or logical operators, such as >, <, =, **LIKE**)

Example

SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000;

or

SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE NAME = 'Hardik';

AND Operator

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

Syntax

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

Example

```
SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY
> 2000 AND age <   25;
```

OR Operator

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

Syntax

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

Example

```
SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY
> 2000 OR age <      25;
```

UPDATE Query

- The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax

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

Example

```
UPDATE CUSTOMERS SET ADDRESS = 'Pune' WHERE ID = 6;
```

DELETE Query

- The SQL DELETE Query is used to delete the existing records from a table.
- You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

```
DELETE FROM table_name;
```

or

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

Example

```
DELETE FROM CUSTOMERS;
```

or

```
DELETE FROM CUSTOMERS WHERE ID = 6;
```

ORDER BY Clause

- The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns.
- Some databases sort the query results in an ascending order by default.

Syntax

```
SELECT * FROM table_name [ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

Example

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

or

```
SELECT * FROM CUSTOMERS ORDER BY NAME DESC;
```

Group By Clause

- The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

Syntax

SELECT column1, column2 FROM table_name GROUP BY column1, column2

Example

SELECT NAME, SUM(SALARY) FROM CUSTOMERS GROUP BY NAME;

Distinct Keyword

- The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

Syntax

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

Example

```
SELECT DISTINCT SALARY FROM CUSTOMERS;
```

LIMIT Clause

- The SQL **LIMIT** clause to fetch limited number of records.

Syntax

```
SELECT * FROM table_name LIMIT 3;
```

Example

```
SELECT * FROM CUSTOMERS LIMIT 3;
```

DROP Table

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

Syntax

```
DROP TABLE table_name;
```

Example

```
DROP TABLE CUSTOMERS;
```

DELETE Table

- The SQL **DELETE TABLE** statement is used to remove a table data but all the definations, indexes, triggers, constraints and permission specifications for that table exist.

Syntax

```
DELETE TABLE table_name;
```

Example

```
DELETE TABLE CUSTOMERS;
```

DROP Database

- The SQL **DROP DATABASE** statement is used to drop an existing database in SQL schema.

Syntax

```
DROP DATABASE DatabaseName;
```

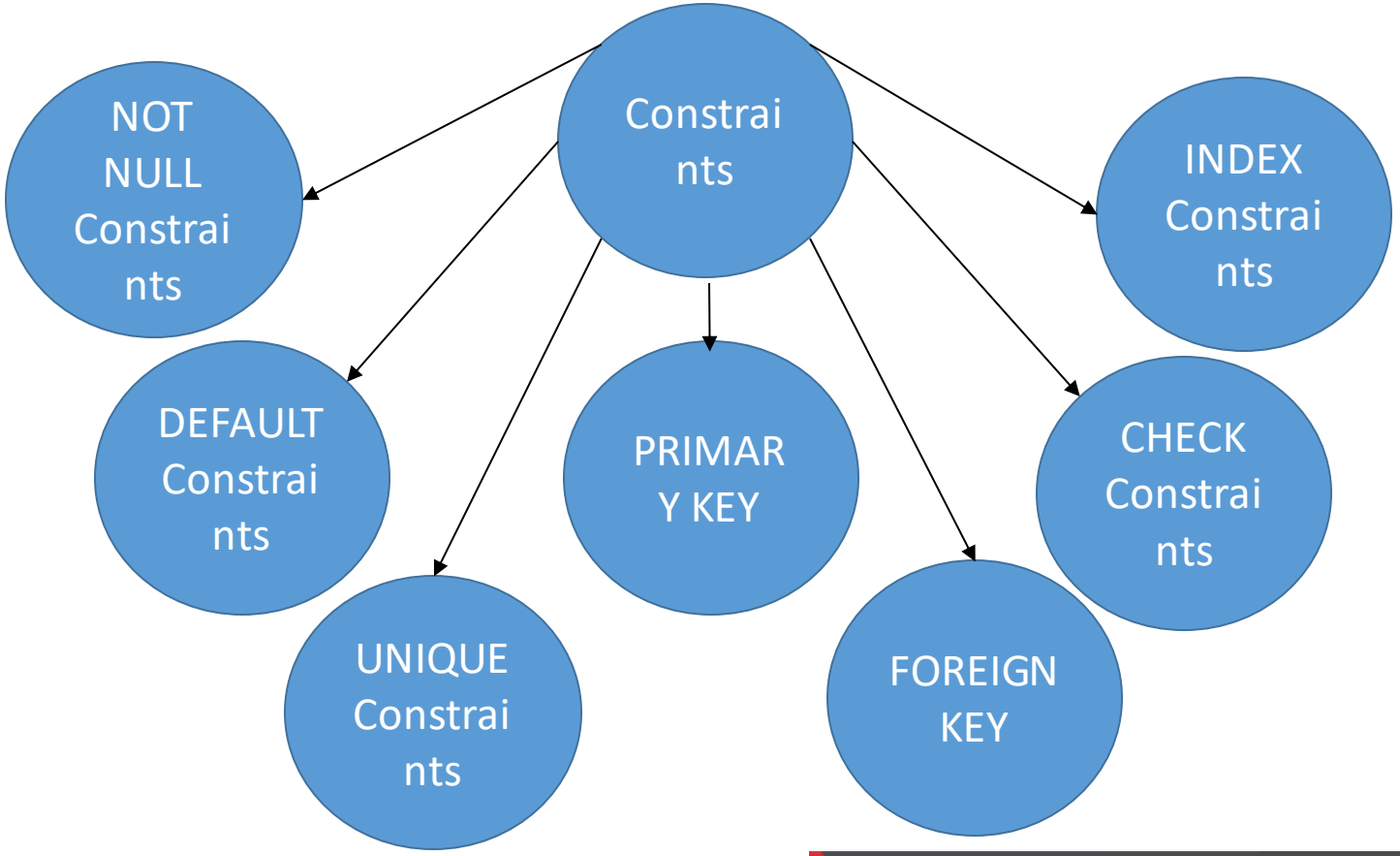
Example

```
DROP DATABASE testDB;
```

Advanced SQL

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..



SQL - NOT NULL Constraint

- By default, a column can hold NULL values. If you do not want a column to have a EMPTY value, then you need to define such a constraint on this column specifying that EMPTY SPACE is now not allowed for that column.

Syntax

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

Example

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

SQL - DEFAULT Constraint

- Provides a default value for a column when none is specified.
- The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Syntax

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

Example

```
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);
```

SQL - UNIQUE Constraint

- Ensures that all values in a column are different.
- The UNIQUE Constraint prevents two records from having identical values in a column.

Syntax

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

Example

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

SQL - Primary Key

- A primary key is a field in a table which uniquely identifies each row/record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot have NULL values.
- A table can have only one primary key.
- 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).

Syntax

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

Example

```
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));
```

SQL - 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 whose values match a Primary Key in a different table.

Example

```
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));

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

For Foreign Key

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

SQL - 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.

Syntax

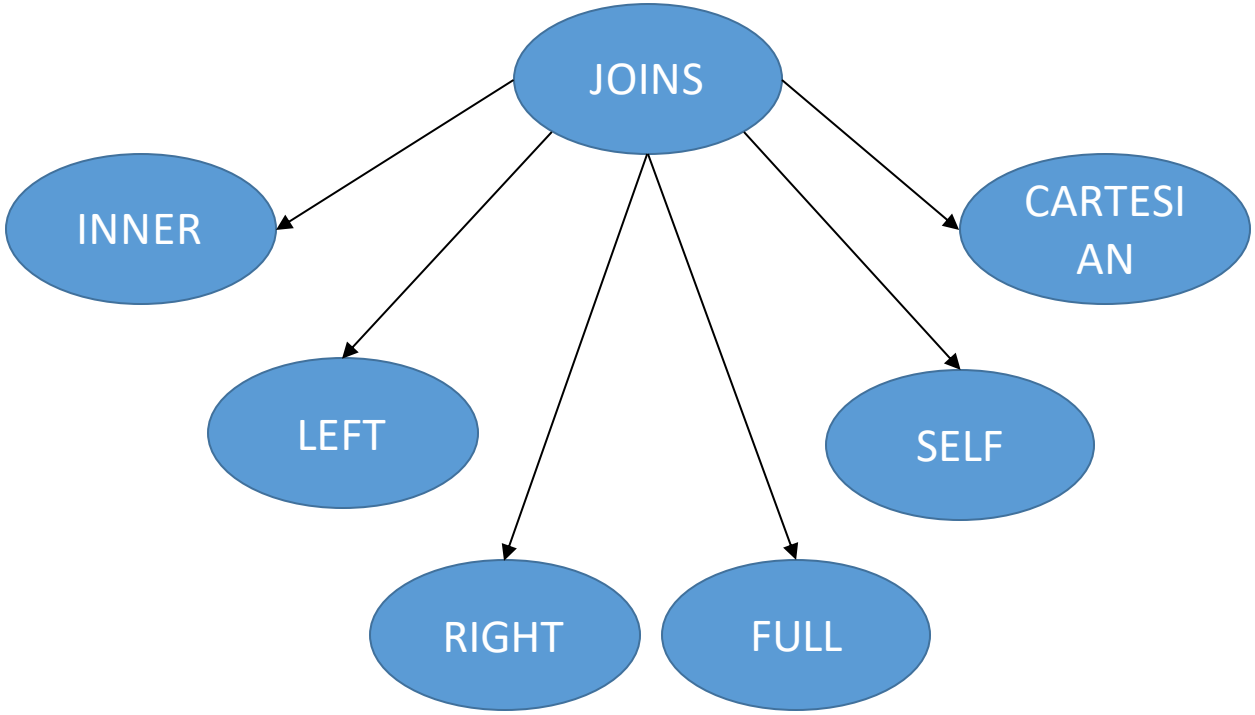
```
CREATE TABLE table_name( ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL CHECK (AGE >= 18), ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID) );
```

Example

```
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) );
```

SQL - USING JOINS

- The SQL **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.



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

SQL - INNER JOINS

- The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate.
- The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.
- When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax

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

Example

```
SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS INNER JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

SQL - LEFT JOINS

- The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table.
- This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax

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

Example

```
SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

SQL - RIGHT JOINS

- The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table.
- This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax

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

Example

```
SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

SQL - FULL JOINS

- The SQL **FULL JOIN** combines the results of both left and right outer joins.
- The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

Syntax

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

Example

```
SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS FULL JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

SQL - CARTESIAN or CROSS JOINS

- The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables.

Syntax

```
SELECT table1.column1, table2.column2... FROM  table1, table2 [, table3 ]
```

Example

```
SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS, ORDERS;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	3000	2009-10-08 00:00:00
1	Ramesh	1500	2009-10-08 00:00:00
1	Ramesh	1560	2009-11-20 00:00:00
1	Ramesh	2060	2008-05-20 00:00:00
2	Khilan	3000	2009-10-08 00:00:00
2	Khilan	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
2	Khilan	2060	2008-05-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
3	kaushik	1560	2009-11-20 00:00:00
3	kaushik	2060	2008-05-20 00:00:00
4	Chaitali	3000	2009-10-08 00:00:00
4	Chaitali	1500	2009-10-08 00:00:00
4	Chaitali	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	3000	2009-10-08 00:00:00
5	Hardik	1500	2009-10-08 00:00:00

SQL - UNIONS CLAUSE

- The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.
- To use this UNION clause, each SELECT statement must have
  - The same number of columns selected
  - The same number of column expressions
  - The same data type and
  - Have them in the same order

Syntax

```
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]

UNION

SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
```

SQL - UNIONS CLAUSE (Cont…)

Example

```
SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID

UNION

SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

SQL - UNION ALL Clause

- The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.
- The same rules that apply to the UNION clause will apply to the UNION ALL operator.

Syntax

```
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]

UNION ALL

SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
```

SQL – UNION ALL CLAUSE (Cont…)

Example

```
SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID

UNION ALL

SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

SQL - INTERSECT Clause

- The SQL **INTERSECT** clause/operator 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.
- This means INTERSECT returns only common rows returned by the two SELECT statements.

Syntax

```
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
INTERSECT
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
```

SQL – INTERSECT Clause (Cont…)

Example

```
SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID

INTERSECT

SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Ramesh	1560	2009-11-20 00:00:00
4	kaushik	2060	2008-05-20 00:00:00

SQL – EXCEPT/MINUS Clause

- The SQL **EXCEPT** clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.
- This means EXCEPT returns only rows, which are not available in the second SELECT statement.

Syntax

```
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]

EXCEPT

SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
```

SQL – EXCEPT/MINUS Clause (Cont…)

Example

```
SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID

EXCEPT

SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN
ORDERS ON     CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

SQL - NULL Values

- 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

```
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));
```

Example

```
SELECT ID, NAME, AGE, ADDRESS, SALARY FROM CUSTOMERS WHERE SALARY IS NOT NULL;
```

or

```
SELECT ID, NAME, AGE, ADDRESS, SALARY FROM CUSTOMERS WHERE SALARY IS NULL;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	
7	Muffy	24	Indore	

SELECT ID, NAME, AGE, ADDRESS, SALARY FROM CUSTOMERS WHERE SALARY IS NOT NULL;

or

SELECT ID, NAME, AGE, ADDRESS, SALARY FROM CUSTOMERS WHERE SALARY IS NULL;

SQL - ALIAS SYNTAX

- You can rename a table or a column temporarily by giving another name known as **Alias**.
- The renaming is a temporary change and the actual table name does not change in the database.

Syntax

```
SELECT column1, column2.... FROM table_name AS alias_name
WHERE [condition];
```

or

```
SELECT column_name AS alias_name FROM table_name WHERE
[condition];
```

Example

```
SELECT C.ID, C.NAME, C.AGE, O.AMOUNT FROM CUSTOMERS AS C,
ORDERS AS O  WHERE  C.ID = O.CUSTOMER_ID;
```

or

```
SELECT  ID AS CUSTOMER_ID, NAME AS CUSTOMER_NAME FROM
CUSTOMERS
WHERE  SALARY IS NOT NULL;
```

ID	NAME	AGE	AMOUNT
3	kaushik	23	3000
3	kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

CUSTOMER_ID	CUSTOMER_NAME
1	Ramesh
2	Khilan
3	kaushik
4	Chaitali
5	Hardik
6	Komal
7	Muffy

SQL - ALTER TABLE Command

- The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table.
- You should also use the ALTER TABLE command to add and drop various constraints on an existing table.

Syntax

```
ALTER TABLE table_name ADD column_name datatype;
```

or

```
ALTER TABLE table_name DROP COLUMN column_name;
```

or

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

Example

```
ALTER TABLE CUSTOMERS ADD GENDER char(1);
```

or

```
ALTER TABLE CUSTOMERS DROP GENDER;
```

ID	NAME	AGE	ADDRESS	SALARY	Gender
1	Ramesh	32	Ahmedabad	2000.00	NULL
2	Ramesh	25	Delhi	1500.00	NULL
3	kaushik	23	Kota	2000.00	NULL
4	kaushik	25	Mumbai	6500.00	NULL
5	Hardik	27	Bhopal	8500.00	NULL
6	Komal	22	MP	4500.00	NULL
7	Muffy	24	Indore	10000.00	NULL

SQL - TRUNCATE TABLE Command

- The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.
- You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

Syntax

```
TRUNCATE TABLE  table_name;
```

Example

```
TRUNCATE TABLE CUSTOMERS;
```

Aggregation Functions

SQL - MIN() Function

- The MIN() function returns the smallest value of the selected column.

Syntax

```
SELECT MIN(column_name) FROM table_name;
```

Example

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

SQL - MAX() Function

- The MAX() function returns the largest value of the selected column.

Syntax

```
SELECT MAX(column_name) FROM table_name;
```

Example

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

SQL - COUNT() Function

- The COUNT() function returns the number of rows that matches a specified

  criteria.

  Syntax

  SELECT COUNT(column_name) FROM table_name;

  or

  SELECT COUNT(*) FROM table_name;

  Example

  SELECT COUNT(salary) FROM CUSTOMERS;

  or

  SELECT COUNT(*) FROM CUSTOMERS;

SQL - AVG() Function

- The AVG() function returns the average value of a numeric column.

Syntax

```
SELECT AVG(column_name) FROM table_name;
```

Example

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

SQL - SUM() Function

- The SUM() function returns the total sum of a numeric column.

Syntax

```
SELECT SUM(column_name) FROM table_name;
```

Example

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

SQL - Sub Queries

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Syntax

```
SELECT column_name [, column_name ] FROM   table WHERE
column_name OPERATOR
(SELECT column_name [, column_name ] FROM table [WHERE])
```

Example

```
 SELECT * FROM CUSTOMERS WHERE
ID =
(SELECT ID FROM CUSTOMERS WHERE SALARY > 4500);
```

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Logical Operators

SQL - IN Operator

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

Syntax

SELECT column_name FROM table_name WHERE column_name IN (value1, value2, ...) / (SELECT Statement);

Example

SELECT * FROM Customers WHERE Address IN (‘Delhi', ‘Kota', ‘MP’);

Or

SELECT * FROM Customers WHERE Address NOT IN (‘Delhi', ‘Kota', ‘MP’);

Or

SELECT * FROM Customers WHERE age IN (SELECT age FROM customers where salary > 2500);

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.

Syntax

```
SELECT column_name FROM table_name WHERE column_name
BETWEEN value1     AND value2;
```

Example

```
SELECT * FROM Customers WHERE age BETWEEN 25 AND 30;
```

Or

```
SELECT * FROM Customers WHERE age NOT BETWEEN 25 AND 30;
```

SQL - EXISTS Operator

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

Syntax

```
SELECT column_name FROM table_name WHERE EXISTS (SELECT
column_name        FROM table_name WHERE condition);
```

Example

```
SELECT * FROM Customers WHERE age EXISTS (SELECT age FROM
customers where     salary > 2500);
```

SQL ANY Operators

- The ANY operator returns true if any of the subquery values meet the condition.

Syntax

```
SELECT * FROM table_name WHERE column _name = ANY (SELECT
column_name        FROM  tablename  where column_name  operator
value);
```

Example

```
SELECT * FROM Customers WHERE age = ANY (SELECT age FROM
customers where     salary > 2500);
```

SQL - ALL Operators

- The All operator returns true if all of the subquery values meet the condition.

Syntax

SELECT * FROM table_name WHERE column _name = ALL (SELECT column_name FROM tablename where column_name operator value);

Example

SELECT * FROM Customers WHERE age = ALL (SELECT age FROM customers where salary > 2500);