SQL(Structured Query language) chp 5 & 6

Prepared by: Er.Simanta Kasaju

Introduction

- SQL is a database computer language designed for the retrieval and management of data in a relational database.
- All task related to relational data management creating tables, querying the database for information, modifying the data in the database ,deleting them, granting access to users and so on can be done using SQL.
- SQL is based on relational algebra and tuple relational calculus.
- SQL stands for Structured Query language, pronounced as "S-Q-L" or sometimes as "See-Quel"... Relational databases like MySQL Database, Oracle, MS SQL Server, Sybase, etc. use ANSI SQL.

SQL is required(Characteristic)

- To create new databases, tables and views
- To insert records in a database
- To update records in a database
- To delete records from a database
- To retrieve data from a database
- To set permissions on tables, procedures, and views

SQL Syntax

- SQL is not case sensitive. Generally SQL keywords are written in uppercase.
- SQL statements are dependent on text lines. We can place a single SQL statement on one or multiple text lines.
- You can perform most of the action in a database with SQL statements.
- SQL depends on relational algebra and tuple relational calculus.

SQL statement

- SQL statements are started with any of the SQL commands/keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP etc. and the statement ends with a semicolon (;).
- Example of SQL statement:
- SELECT "column_name" FROM "table_name";
- Why semicolon is used after SQL statements:
- Semicolon is used to separate SQL statements. It is a standard way to separate SQL statements in a database system in which more than one SQL statements are used in the same call.

SQL Data Types

- Data types are used to represent the nature of the data that can be stored in the database table. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.
- Data types mainly classified into three categories for every database.
- String Data types
- Numeric Data types
- Date and time Data types

MySQL String Data Types

CHAR(Size)	It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1.
VARCHAR(Size)	It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
BINARY(Size)	It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.
VARBINARY(Size)	It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes.
TEXT(Size)	It holds a string that can contain a maximum length of 255 characters.
TINYTEXT	It holds a string with a maximum length of 255 characters.
MEDIUMTEXT	It holds a string with a maximum length of 16,777,215.
LONGTEXT	It holds a string with a maximum length of 4,294,967,295 characters.
ENUM(val1, val2, val3,...)	It is used when a string object having only one value, chosen from a list of possible values. It contains 65535 values in an ENUM list. If you insert a value that is not in the list, a blank value will be inserted.
SET(val1,val2,val3,....)	It is used to specify a string that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values at one time in a SET list.
BLOB(size)	It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes.

MySQL Numeric Data Types

BIT(Size)	It is used for a bit-value type. The number of bits per value is specified in size. Its size can be 1 to 64. The default value is 1.
INT(size)	It is used for the integer value. Its signed range varies from -2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. The size parameter specifies the max display width that is 255.
INTEGER(size)	It is equal to INT(size).
FLOAT(size, d)	It is used to specify a floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal point is specified by **d** parameter.
FLOAT(p)	It is used to specify a floating point number. MySQL used p parameter to determine whether to use FLOAT or DOUBLE. If p is between 0 to24, the data type becomes FLOAT (). If p is from 25 to 53, the data type becomes DOUBLE().
DOUBLE(size, d)	It is a normal size floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal is specified by d parameter.
DECIMAL(size, d)	It is used to specify a fixed point number. Its size parameter specifies the total number of digits. The number of digits after the decimal parameter is specified by **d** parameter. The maximum value for the size is 65, and the default value is 10. The maximum value for **d** is 30, and the default value is 0.
DEC(size, d)	It is equal to DECIMAL(size, d).
BOOL	It is used to specify Boolean values true and false. Zero is considered as false, and nonzero values are considered as true.

MySQL Date and Time Data Types

DATE	It is used to specify date format YYYY-MM-DD. Its supported range is from '1000-01-01' to '9999-12-31'.
DATETIME(fsp)	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1000-01-01 00:00:00' to 9999-12-31 23:59:59'.
TIMESTAMP(fsp)	It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
TIME(fsp)	It is used to specify the time format. Its format is hh:mm:ss. Its supported range is from '-838:59:59' to '838:59:59'
YEAR	It is used to specify a year in four-digit format. Values allowed in four digit format from 1901 to 2155, and 0000.

SQL Operators

- SQL statements generally contain some reserved words or characters that are used to perform operations such as comparison and arithmetical operations etc. These reserved words or characters are known as operators.
- Generally there are three types of operators in SQL:

- SQL Arithmetic Operators
- SQL Comparison Operators
- SQL Logical Operators

SQL Arithmetic Operators:
Let's assume two variables "a" and "b". Here "a" is valued 50 and "b" valued 100

Operators	Descriptions	Examples
+	It is used to add containing values of both operands	a+b will give 150
-	It subtracts right hand operand from left hand operand	a-b will give -50
*	It multiply both operand's values	a*b will give 5000
/	It divides left hand operand by right hand operand	b/a will give 2
%	It divides left hand operand by right hand operand and returns reminder	b%a will give 0

SQL Comparison Operators:

Operator	Description	Example
=	Examine both operands value that are equal or not,if yes condition become true.	(a=b) is not true
!=	This is used to check the value of both operands equal or not,if not condition become true.	(a!=b) is true
< >	Examines the operand's value equal or not, if values are not equal condition is true	(a<>b) is true
>	Examine the left operand value is greater than right Operand, if yes condition becomes true	(a>b) is not true
<	Examines the left operand value is less than right Operand, if yes condition becomes true	(a
>=	Examines that the value of left operand is greater than or equal to the value of right operand or not,if yes condition become true	(a>=b) is not true
<=	Examines that the value of left operand is less than or equal to the value of right operand or not, if yes condition becomes true	(a<=b) is true
!<	Examines that the left operand value is not less than the right operand value	(a!
!>	Examines that the value of left operand is not greater than the value of right operand	(a!>b) is true

SQL Logical Operators:

Operator	Description
ALL	this is used to compare a value to all values in another value set.
AND	this operator allows the existence of multiple conditions in an SQL statement.
ANY	this operator is used to compare the value in list according to the condition.
BETWEEN	this operator is used to search for values, that are within a set of values
IN	this operator is used to compare a value to that specified list value
NOT	the NOT operator reverse the meaning of any logical operator
OR	this operator is used to combine multiple conditions in SQL statements
EXISTS	the EXISTS operator is used to search for the presence of a row in a specified table
LIKE	this operator is used to compare a value to similar values using wildcard operator

SET operators

Operator	Definition
UNION	Returns all distinct rows from both queries
INTERSECT	Returns common rows selected by both queries
MINUS	Returns all distinct rows that are in the first query, but not in second one.

Fig. 3.36 Set operators

➢ **Examples :**

Consider following two relations -

Permanent_Emp {Emp_Code, Name, Salary}

Temporary_emp = {Emp_Code, Name, Daily_wages}

1) *Display name of all employees.*

⇒ select Name
from Permanent_Emp
Union
select Name
from Temporary_Emp ;

SQL literals

- There are four kinds of literal values supported by SQL.They are
- Character string
- Bit string
- Exact numeric
- Approximate numeric

Character string

- Character string are written as a sequence of characters enclosed in single quotes. The single quote character is represented within a character string by a two single quotes. Some of character strings are
- ‘Computer Engineering’
- ‘Structured query language’

Bit string

- A bit string is written as a sequence of 0’s and 1’s enclosed in single quotes ad preceded by the letter ‘B’ or as a sequence of hexadecimal digits enclosed in single quotes and preceded by the letter ‘X’ some examples are given below:
- B ‘1011011’
- B ‘1’
- B ‘0’
- X ‘A5’
- X ‘1’

Exact numeric

- These literals are written as a signed or unsigned decimal number possible with a decimal point. Examples
- 9
- 90
- 90.00
- 0.9
- +99.99
- -99.99

Approximate numeric

- Approximate numeric literals are written as exact numeric literals followed by the letter ‘E’ followed by a signed or unsigned integer.Some example
- 5E5
- 55.5E5
- +55E-5
- 0.55E
- -O.55E-9

Types of SQL Commands

- SQl provides set of commands for a varety of tasks including the following:

❖Querying data

❖Updating ,inserting and deleting data

❖Creating , modifying and deleting database objects

❖Controlling access to the database

❖Providing for data integrity and consistency

- For example using SQL statements you can create tables, modify them, delete the tables,query the data in the tables ,insert data into the tables ,modify and delete the data ,decide who can see the data and so on.

Continue

- The SQL statement is a set of instructions to the RDMBS to perform an action.
- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.
- SQL statements are divided into the following categories:

❖Data definition language(DDL)

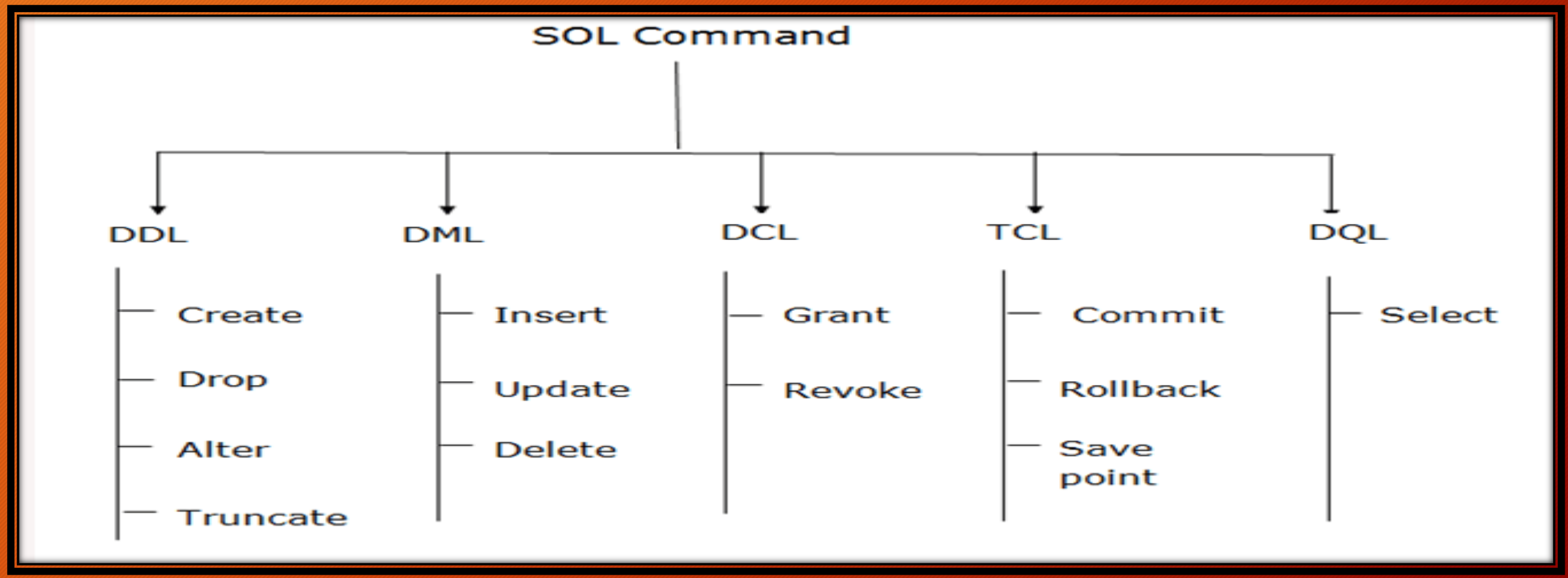
❖Data Manipulation language(DML)

❖Data Query language(DQL)

❖Data control Language(DCL)

❖Transcation control language(TCL)

Types of SQL Commands



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.
- Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

CREATE

- It is used to create a new table or new database in the database.
- **CREATE DATABASE** database_name;
- CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
- CREATE TABLE EMPLOYEE(Name VARCHAR(20), Email VARCHAR(100), DOB DATE);
- Example

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

ALTER

- It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.
- To add a new column in the table

```
ALTER TABLE table_name
ADD column_name COLUMN-definition;
```

- To modify existing column in the table:

```
ALTER TABLE MODIFY(COLUMN DEFINITION....);
ALTER TABLE Customers
ADD Email varchar(255);
ALTER TABLE STU_DETAILS MODIFY NAME
VARCHAR(20) ;
```

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

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

DROP

- It is used to delete both the structure and record stored in the table.
- `DROP DATABASE databasename;`
- `DROP TABLE table_name;`

```
Drop table student;
```

TRUNCATE

- It is used to delete all the rows from the table and free the space containing the table.
- `TRUNCATE TABLE table_name;`

```
TRUNCATE TABLE EMPLOYEE;
```

S.No	DDL Commands	Description	Sample Query
1.	CREATE	Used to create tables or databases.	CREATE table student;
2.	ALTER	Used to modify the values in the tables.	ALTER table student add column roll_no int;
3.	RENAME	Used to rename the table or database name.	RENAME student to student_details;
4.	DROP	Deletes the table from the database.	DROP table student_details;
5.	TRUNCATE	Used to delete a table from database.	TRUNCATE table student_details;

2. Data Manipulation Language (DML)

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.
- Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

INSERT

- The INSERT statement is a SQL query. It is used to insert data into the row of a table.
- INSERT INTO TABLE_NAME

(col1, col2, col3,..... col N)

VALUES (value1, value2, value3, ... valueN);

- Or
- INSERT INTO TABLE_NAME

VALUES (value1, value2, value3, ... valueN);

```
INSERT INTO Customers (CustomerName,
ContactName, Address, City, PostalCode,
Country)
VALUES ('Cardinal', 'Tom B.
Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

It is also possible to only insert data in specific columns.

UPDATE

- This command is used to update or modify the value of a column in the table.

```
UPDATE table_name
SET column1 = value1, column2 = value2,
...
WHERE condition;
UPDATE Customers
SET ContactName = 'Alfred Schmidt',
City= 'Frankfurt'
WHERE CustomerID = 1;
```

- **Note**: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

DELETE

- It is used to remove one or more row from a table.
- `DELETE FROM table_name WHERE condition;`

```
DELETE FROM Customers WHERE
 CustomerName='Alfreds
Futterkiste’;
```

- **Note:** Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

S.No	DML Command	Description	Sample Query
1.	**INSERT**	**Used to insert new rows in the tables.**	**INSERT into student(roll_no, name) values(1, Anoop);**
2.	**DELETE**	**Used to delete a row or entire table.**	**DELETE table student;**
3.	**UPDATE**	**Used to update values of existing rows of tables.**	**UPDATE students set s_name = 'Anurag' where s_name like 'Anoop';**
4.	**LOCK**	**Used to lock the privilege as either read or write.**	**LOCK tables student read;**
5.	**MERGE**	**Used to merge two rows of existing tables in database.**	

3. Data Control Language (DCL)

- DCL commands are used to grant and take back authority from any database user.
- Here are some commands that come under DCL:
- Grant
- Revoke

Grant

- It is used to give user access privileges to a database.
- GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

```
GRANT SELECT ON Users TO'Tom'@'localhost;
```

REVOKE

- It is used to take back permissions from the user.
- REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

```
REVOKE SELECT, UPDATE ON student FROM BCA, MCA;
```

S.No	DCL Commands	Description	Sample Query
1.	GRANT	Used to provide access to users.	GRANT CREATE table to user1;
2.	REVOKE	Used to take back the access privileges from the users.	REVOKE CREATE table from user1;

4. Transaction Control Language (TCL)

- TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.
- These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.
- Here are some commands that come under TCL:
- COMMIT
- ROLLBACK
- SAVEPOINT

COMMIT

- Commit command is used to save all the transactions to the database.
- Commit;
- Example

```
DELETE FROM Students
WHERE RollNo =25;
COMMIT;
```

ROLLBACK

- Rollback command is used to undo transactions that have not already been saved to the database.
- ROLLBACK;
- Example

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
ROLLBACK;
```

SAVEPOINT

- It is used to roll the transaction back to a certain point without rolling back the entire transaction.
- SAVEPOINT SAVEPOINT_NAME;
- Example

```
SAVEPOINT
RollNo;
```

S.No	TCL Commands	Description	Sample Query
1.	ROLL BACK	Used to cancel or UNDO the changes made in the database.	ROLLBACK;
2.	COMMIT	Used to deploy or apply or save the changes in the database.	COMMIT;
3.	SAVEPOINT	Used to save the data on temporary basis in the database.	SAVEPOINT roll_no;

5. Data Query Language (DQL)

- DQL is used to fetch the data from the database.
- It uses only one command:
- SELECT

SELECT

- This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

```
SELECT expressions
FROM TABLES
WHERE conditions;
Example
SELECT emp_name
FROM employee
WHERE age > 20;
```

S.No	DQL Command	Description	Sample Query
1.	SELECT	Used to fetch data from tables/database.	SELECT * from student_details;

SELECT col1,col2 FROM tablename;

SQL Constraints

- SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.
- 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.

Column level

- If the constraints are defined along with the column definition ,it is called a column level constraints.
- Column level constraints can be applied to any column.
- If the constraints spans across multiple columns the user has to use table constraints.

Table level

- If the constraints attached to a specific column in a table references the contents of another column in the table then the user will have to use table level constraints.
- Examples of different constraints that can be applied on the table are a follows:

❖Null value
❖Primary key
❖Unique Key
❖Default value
❖Foreign key
❖Check integrity

1.Null value

1) Null Value Concept

While creating tables, if a row lacks a data value for a particular column, that value is said to be null. Columns of any data types may contain null values unless the column was defined as not null when the table was created.

➢ **Principles of NULL Values**

- Setting a null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will evaluate to null in any expression. Example : null multiplied by 10 is null.
- When a column name is defined as not null, then that column becomes a mandatory column. It implies that the user is forced to enter data into that column.

Example

```
CREATE TABLE persons (
id INT NOT NULL,
name VARCHAR(30) NOT NULL,
birth_date DATE,
phone VARCHAR(15) NOT NULL );
```

Note: A null value or NULL is different from zero (0), blank, or a zero-length character string such as ''. NULL means that no entry has been made.

2.Primary key

2) Primary Key Concept

A primary key is one or more columns in a table used to uniquely identify each row in the table. Primary key values must not be null and must be unique across the column.

A multicolumn primary key is called a *composite primary key*.

➢ **Examples :**

1) Column level primary key constraint

```
SQL> CREATE TABLE student
        (roll_no number(5) PRIMARY KEY,
        name varchar(25) NOT NULL,
        address varchar(25) NOT NULL,
        ph_no varchar(15));
Table created.
```

2) Table level primary key constraint.

```
SQL> CREATE TABLE student1
        (roll_no number(5),
        name varchar(25) NOT NULL,
        address varchar(25) NOT NULL,
        ph_no varchar(15),
        PRIMARY KEY(roll_no));
Table created.
```

```
CREATE TABLE persons (
id INT PRIMARY KEY,
name VARCHAR(30) NOT NULL,
birth_date DATE,
phone VARCHAR(15) NOT NULL
);
```

3.Unique Key

3) Unique Key Concepts

Unique key is used to ensure that the information in the column for each record is unique, as with license number. A table may have many unique keys.

- **Example :**

```
CREATE TABLE special_customer
(customer_code number(5) PRIMARY KEY,
customer_name varchar(25) NOT NULL,
customer_address varchar(30) NOT NULL,
license_no varchar(15) constraint uk_ license_no UNIQUE);
```

Example

```
CREATE TABLE persons (
id INT PRIMARY KEY,
name VARCHAR(30) NOT NULL,
birth_date DATE,
phone VARCHAR(15) NOT NULL UNIQUE);
```

Note: Multiple UNIQUE constraints can be defined on a table, whereas only one PRIMARY KEY constraint can be defined on a table. Also, unlike PRIMARY KEY constraints, the UNIQUE constraints allow NULL values.

4.Default value

4) Default Value Concepts

At the time of column creation a 'default value' can be assigned to it. When the user is loading a record with values and leaves this column empty, the DBA will automatically load this column with the default value specified. The data type of the default value should match the data type of the column. You can use the default clause to specify any default value you want.

➢ **Example**

```
SQL> CREATE TABLE employee
     (emp_code number(5),
     emp_name varchar(25) NOT NULL,
     emp_address varchar(30) NOT NULL,
     ph_no varchar(15),
     married char(1) DEFAULT 'M',
     PRIMARY KEY(emp_code));
Table created.
```

```sql
CREATE TABLE persons (
id INT NOT NULL PRIMARY KEY,
name VARCHAR(30) NOT NULL,
birth_date DATE,
phone VARCHAR(15) NOT NULL UNIQUE,
country VARCHAR(30) NOT NULL DEFAULT
'Australia' );
```

5.Foreign key

5) Foreign Key Concepts

Foreign key represents relationship between tables. The existence of a foreign key implies that the table with the foreign key is related to the primary key table from which the foreign key is derived.

The foreign key/references constraint

- rejects an INSERT or UPDATE of a value, if a corresponding value does not currently exist in the primary key table.
- rejects a DELETE, if it would invalidate a REFERENCES constraint.
- must reference a PRIMARY KEY or UNIQUE column in primary key table.
- will reference the PRIMARY KEY of the primary key table if no column or group of columns is specified in the constraint.
- must reference a table, not a view or cluster.
- requires that the FOREIGN KEY column(s) and the CONSTRAINT column(s) have matching data types.

➢ **Example**

```
SQL> CREATE TABLE book
       (ISBN varchar(15) PRIMARY KEY,
       title varchar(25), pub_year varchar(4),
       unit_price number(4),
       author_name varchar(25) references author
       publisher_name varchar(25) references pub
Table created.
```

```
CREATE TABLE employees ( emp_id INT NOT
NULL PRIMARY KEY, emp_name VARCHAR(55)
NOT NULL, hire_date DATE NOT NULL, salary
INT, dept_id Int, FOREIGN KEY (dept_id)
  REFERENCES departments(dept_id) );
```

6.Check integrity

6) Check Integrity Constraints

The CHECK contraint defines a condition that every row must satisfy. There can be more than one CHECK constraint on a column, and the CHECK constraint can be defined at column as well as the table level.

At the column level, the constraint is defined by,

```
DeptID Number (2) CONSTRAINT ck-deptID
CHECK (( DeptID >= 10) and (DeptID <= 99)),
```

and at the table level by

```
CONSTRAINT ck-deptId
CHECK ((DeptID >= 10) and (DeptID <= 99))
```

➢ **Following are a few examples of appropriate CHECK constraints.**

- Add CHECK constraint on the Emp-Code column of the Employee so that every Emp_Code should start with 'E'.
- Add CHECK constraint on the City column of the Employee so that only the cities 'Mumbai', 'Pune', 'Nashik', 'Solapur' are allowed.

- **Example**

```
SQL> CREATE TABLE  employee
        (emp_code number(5)CONSTRAINT ck_empcodeCHECK (emp_code like 'E%'),
        emp_name varchar(25) NOT NULL,
        city varchar(30) CONSTRAINT ck_city
        CHECK (city IN('Mumbai', 'Pune', 'Nashik', 'Solapur' )),
        salary number(5),
        PRIMARY KEY(emp_code));
Table created.
```

```sql
CREATE TABLE employees (
emp_id INT NOT NULL PRIMARY KEY,
emp_name VARCHAR(55) NOT NULL,
hire_date DATE NOT NULL, salary INT NOT NULL CHECK
(salary >= 3000 AND salary <= 10000), dept_id INT,
FOREIGN KEY (dept_id) REFERENCES departments(dept_id) );
```

➢ **Restrictions on CHECK constraints**

- The condition must be a Boolean expression that can be evaluated using the values in the row being inserted or updated.
- The condition cannot contain subqueries or sequences.
- The condition cannot include the SYS DATE, UID, USER or USER ENV SQL functions.

➢ **Defining integrity constraints in the ALTER TABLE command :**

You can also define integrity constraints using the constraint clause in the ALTER TABLE command.

Consider following existing tables :

1) Student with definition :

```
SQL> CREATE TABLE student
     (roll_no number(3),
     name varchar(25));
```

2) Test with definition :

```
SQL> CREATE TABLE test
     (roll_no number(3),
     subject_ID number(2),
     marks number(2));
```

3) Subject-info with definition :

```
SQL> CREATE TABLE subject_info
     (subject_ID number(2) PRIMARY KEY,
     subject_name varchar(15));
```

The following examples show the definitions of several integrity constraints.

1) *ADD PRIMARY KEY constraints on column roll_no in student table.*

```
SQL> ALTER TABLE student
       ADD PRIMARY KEY(roll_no);
Table altered.
```

2) *Modify column marks to include NOT NULL constraint.*

```
SQL> ALTER TABLE test
       MODIFY (marks number(3) NOT NULL);
```

➢ **Dropping integrity constraints in the ALTER TABLE command**

You can drop an integrity constraint if the rule that it enforces is no longer true or if the constraint is no longer needed. Drop the constraint using the ALTER TABLE command with the DROP clause.

The following examples illustrate the dropping of the integrity constraints.

1) *Drop the primary key of student table.*

```
SQL> ALTER TABLE student
        DROP PRIMARY KEY;
Table altered.
```

2) *Drop unique key constraint on column license-no in table customer.*

```
SQL> ALTER TABLE customer
        DROP CONSTRAINT license_ukey;
```

7) Renaming a Table

You can rename a table provided you are the owner of the table. The general syntax is :

```
RENAME old table name TO new table name ;
```

➢ **Example :**

```
SQL>  RENAME test TO test_info;
Table renamed.
```

Advantages of SQL

- There are the following advantages of SQL:

❖High speed

- Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

❖No coding needed

- In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

❖Well defined standards

- Long established are used by the SQL databases that are being used by ISO and ANSI.

Continue

❖Portability

- SQL can be used in laptop, PCs, server and even some mobile phones.

❖Interactive language

- SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

❖Multiple data view

- Using the SQL language, the users can make different views of the database structure.

SQL Comments

- Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.
- Single Line Comments
- Single line comments start with --.
- Any text between -- and the end of the line will be ignored (will not be executed).
-
```
--Select all:
SELECT * FROM Customers;
```
- Multi-line Comments
- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored.
-
```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

SQL Wildcard Characters

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

Example

- `SELECT * FROM Customers[] WHERE City LIKE 'ber%’;`
- `SELECT * FROM Customers WHERE City LIKE '%es%’;`
- `SELECT * FROM Customers WHERE City LIKE '_ondon’;`
- `SELECT * FROM Customers WHERE City LIKE 'L_n_on’;`
- `SELECT * FROM Customers WHERE City LIKE '[bsp]%’;`
- `SELECT * FROM Customers WHERE City LIKE '[a-c]%';`

The following SQL statement selects all customers with a City starting with "ber":

The following SQL statement selects all customers with a City with a City containing the pattern "es":

The following SQL statement selects all customers with a City starting with any character, followed by "ondon":

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

Example

- `SELECT * FROM Customers WHERE City LIKE '[!bsp]%';`

The two following SQL statements select all customers with a City NOT starting with "b", "s", or "p":

SQL keywords

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table
ALTER TABLE	Adds, deletes, or modifies columns in a table
ALL	Returns true if all of the subquery values meet the condition
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition
AS	Renames a column or table with an alias
ASC	Sorts the result set in ascending order

Continue

BACKUP DATABASE	Creates a back up of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table
CONSTRAINT	Adds or deletes a constraint
CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database
CREATE INDEX	Creates an index on a table (allows duplicate values)
CREATE OR REPLACE VIEW	Updates a view
CREATE TABLE	Creates a new table in the database

Continue

CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column
DELETE	Deletes rows from a table
DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values
DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table
DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint

Continue

DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view
EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from
FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table
GROUP BY	Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG)
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause

Continue

INDEX	Creates or deletes an index in a table
INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table
INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty values
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables
LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table
LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set
NOT	Only includes rows where a condition is not true
NOT NULL	A constraint that enforces a column to not accept NULL values

Continue

OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order
OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure
RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set
SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table

Continue

TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself
UNION	Combines the result set of two or more SELECT statements (only distinct values)
UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)
UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement
VIEW	Creates, updates, or deletes a view
WHERE	Filters a result set to include only records that fulfill a specified condition

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,OR,NOT

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

Eg:

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

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

Eg:

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

- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE NOT condition;
  ```

Eg:

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

SELECT CLAUSE

- The SELECT statement is used to select data from a database.

```
1.SELECT column1, column2, ...
FROM table_name;


2.SELECT * FROM table_name;


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

WHERE CLAUSE

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.
- The WHERE clause is not only used in SELECT statement, it is also used in UPDATE, DELETE statement, etc.!

- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition;
  ```
  ```

Operators in The WHERE Clause

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

FROM CLAUSE

- This clause specifies for SELECT, UPDATE and DELETE for the target tables.
- It defines the cartesian product of of the relation in the clause.
- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition;
  ```
  ```

GROUP BY

- (Aggregate functions often need an added GROUP BY statement.)

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

Eg:

```
SELECT Shippers.ShipperName,COUNT(Orders.OrderID)FROM Orders
LEFT JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID
GROUP BY ShipperName;
```

HAVING

- (The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.)
-
```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Eg:

-
```
SELECT Employees.LastName, COUNT(Orders.OrderID) FROM (Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- ```
  SELECT column1, column2, ...
  FROM table_name
  ORDER BY column1, column2, ... ASC|DESC;
  ```

- ```
  SELECT * FROM Customers
  ORDER BY Country;
  ```

BETWEEN and NOT BETWEEN

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE column_name BETWEEN value1 AND value2;
  ```
- *Eg:*
- ```
  SELECT * FROM Products
  WHERE Price BETWEEN 10 AND 20;
  ```

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

SQL function(Aggregate function)

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

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

Eg: SELECT AVG(Price) FROM Products;

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

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

SELECT COUNT(*) FROM table_name;

SELECT COUNT(DISTINCT column_name) FROM table_name;

Eg: SELECT COUNT(CustomerID) FROM Orders
WHERE CustomerID=7;

Continue

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

SELECT MAX(column_name) FROM table_name;

Eg: SELECT MAX(Price) FROM Products;

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

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

Eg: SELECT MIN(Price) FROM Products;

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

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

Continue

- **ROUND() (**The ROUND() function is used to round a numeric field to the number of decimals specified.**)**

```
SELECT ROUND(column_name,decimals) FROM table_name;
```

```
Eg:SELECT ProductName,ROUND(Price,0) FROM Products;
```

- **UPPER() (**The UCASE() function converts the value of a field to uppercase.**)**

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

```
Eg: SELECT UPPER(CustomerName)FROM Customers;
```

LOWER() (The LCASE() function converts the value of a field to lowercase.**)**

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

```
Eg: SELECT LOWER(CustomerName)FROM Customers;
```

Continue

- **LEN() (**The LEN() function returns the length of the value in a text field.**)**

SELECT LEN(column_name) FROM table_name;

Eg: SELECT CustomerName,LEN(Address) FROM Customers;

- **MID() (**The MID() function is used to extract characters from a text field.**)**

```
SELECT MID(column_name,start,length) FROM table_name;
```

- **FIRST() (**The FIRST() function returns the first value of the selected column)

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

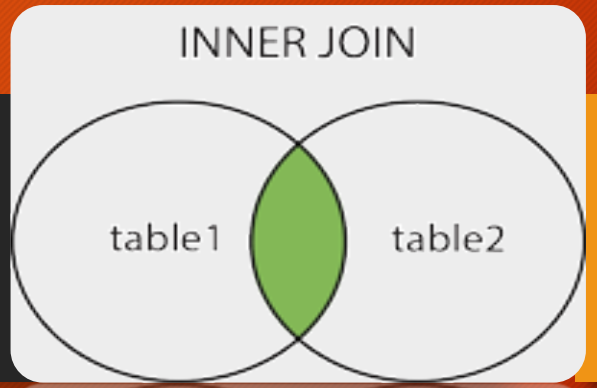
- **LAST() (**The LAST() function returns the last value of the selected column**)**

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

SQL using Join

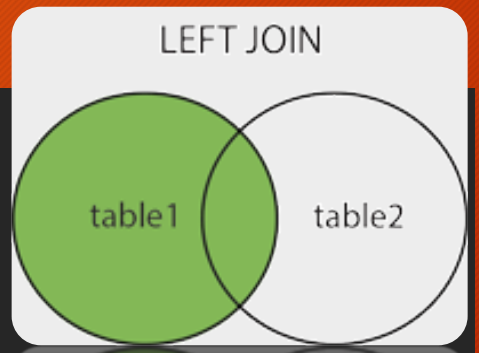
- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Different Types of SQL JOINs
- **(INNER) JOIN:** Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN:** Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN:** Returns all records when there is a match in either left or right table

(INNER) JOIN



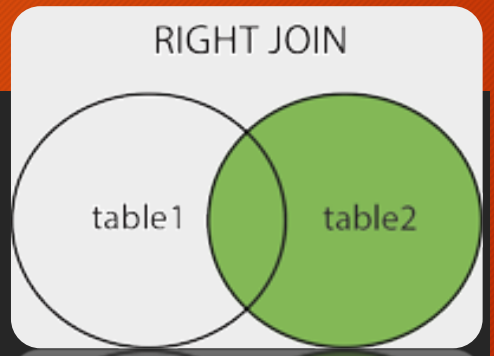
- The INNER JOIN keyword selects records that have matching values in both tables.
- SELECT *column_name(s)*
  FROM *table1*
  INNER JOIN *table2*
  ON *table1.column_name* = *table2.column_name*;
- Eg:
- SELECT Orders.OrderID, Customers.CustomerName
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
-

LEFT (OUTER) JOIN:



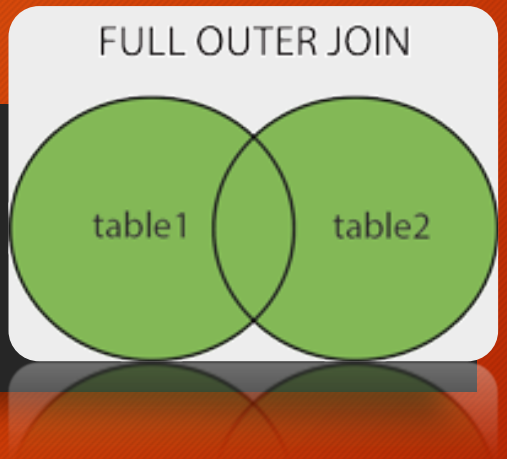
- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.
- ```
  SELECT column_name(s)
  FROM table1
  LEFT JOIN table2
  ON table1.column_name = table2.column_name;
  ```
- Eg:
- ```
  SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
  ORDER BY Customers.CustomerName;
  ```

RIGHT (OUTER) JOIN:



- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.
- ```
  SELECT column_name(s)
  FROM table1
  RIGHT JOIN table2
  ON table1.column_name = table2.column_name;
  ```
- Eg:
- ```
  SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
  ORDER BY Orders.OrderID;
  ```

FULL (OUTER) JOIN:



- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
- ```
  SELECT column_name(s)
  FROM table1
  FULL OUTER JOIN table2
  ON table1.column_name = table2.column_name
  WHERE condition;
  ```
- Eg:
- ```
  SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerI
  D
  ORDER BY Customers.CustomerName;
  ```

SELF JOIN

- A self JOIN is a regular join, but the table is joined with itself.
- ```
  SELECT column_name(s)
  FROM table1 T1, table1 T2
  WHERE condition;
  ```
- Eg:
- ```
  SELECT A.CustomerName AS CustomerName1,
  B.CustomerName AS CustomerName2, A.City
  FROM Customers A, Customers B
  WHERE A.CustomerID <> B.CustomerID
  AND A.City = B.City
  ORDER BY A.City;
  ```

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

ANS

```
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

```
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

continue

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

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

```
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 operation

- UNION

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

Continue

- INTERSECT
- ```
  SELECT column_name(s) FROM table1
  INTERSECT
  SELECT column_name(s) FROM table2;
  ```

- EXCEPT
- ```
  SELECT column_name(s) FROM table1
  EXCEPT
  SELECT column_name(s) FROM table2;
  ```

Subquery

- A subquery is a SQL query nested inside a larger query.
- Subqueries are embedded queries inside another query. The embedded query is known as the inner query and the container query is known as the outer query.
- Sub queries are easy to use, offer great flexibility and can be easily broken down into single logical components making up the query which is very useful when Testing and debugging the queries.

Eg:

```
SELECT c.contact_id, c.last_name
FROM contacts c
WHERE c.site_name IN
(SELECT a.site_name
FROM address_book a
WHERE a.address_book_id < 50);
```

Most often, the subquery will be found in the WHERE clause.
These subqueries are also called nested subqueries.

Continue

```
SELECT contacts.last_name, subquery1.total_size
FROM contacts,
(SELECT site_name, SUM(file_size) AS total_size
FROM pages
GROUP BY site_name) subquery1
WHERE subquery1.site_name = contacts.site_name;
```

A subquery can also be found in the FROM clause. These are called inline views.

```
SELECT p1.site_name,
  (SELECT MAX(file_size)
   FROM pages p2
   WHERE p1.site_id = p2.site_id) subquery2
FROM pages p1;
```

A subquery can also be found in the SELECT clause.

VIEWS

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- It is a logical representation of data.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.
- It allows users to

➢ Structure data in a way that users want

➢ Restrict access to data such that user can see and sometimes modify exactly what they need and no more.

➢ Summerize data from various tables which can be used to generate reports.

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

- Eg:

```
1.CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

To create view

```
2.SELECT * FROM viewname;
```

To query view

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

To update view

UPDATE Viewname SET column1=val1

-

Continue

- `DROP VIEW view_name;` To drop view