SQL

SQLis a database computer language designed for the retrieval and management of data in a relational database.

SQLstands for**Structured Query Language**.

System.

SQL is the standard language for Relational Database

All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Applications of SQL

Allows users to access data in the relational database management systems.

Allows users to describe the data.

Allows users to define the data in a database and manipulate that data.

Allows to embed within other languages using SQL modules, libraries & pre-compilers.

Allows users to create and drop databases and tables.

Allows users to create view, stored procedure, functions in a database.

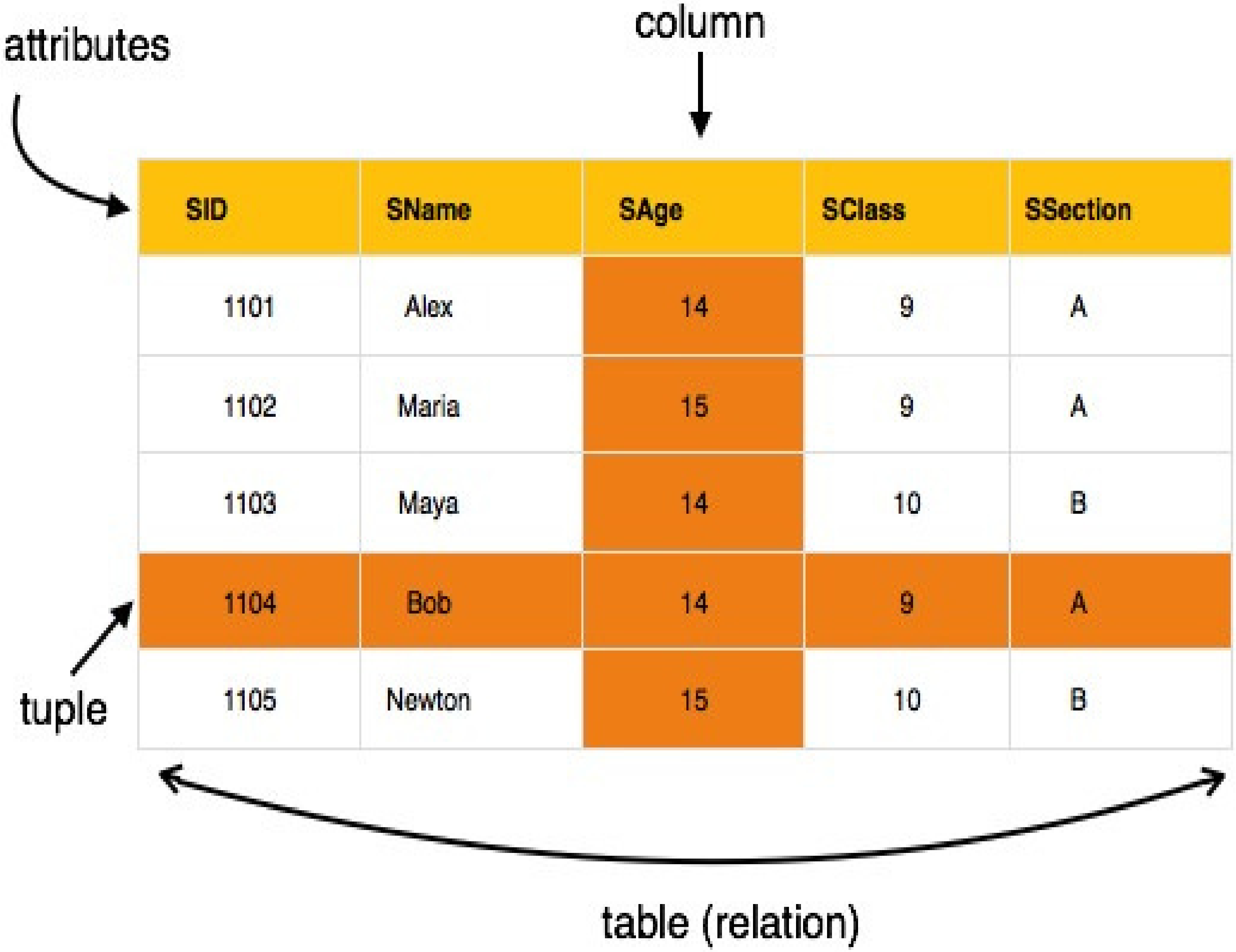
Allows users to set permissions on tables, procedures and view

RDBMS

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

Some important terms:

1. **Table** -The data in an RDBMS is stored in database objects which are called as**tables**. This table is basically a collection of related data entries and it is a collection of columns and rows.



attributes

column

SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

tuple

table (relation)

2. Row/ Tuple

A record is also called as a row of data is each individual entity that exists in a table.

A record is a horizontal entity in a table.

3. Column/Attribute

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

4.Cardinality

The number of rows in a table is called cardinality of table.

Example of DBMS software

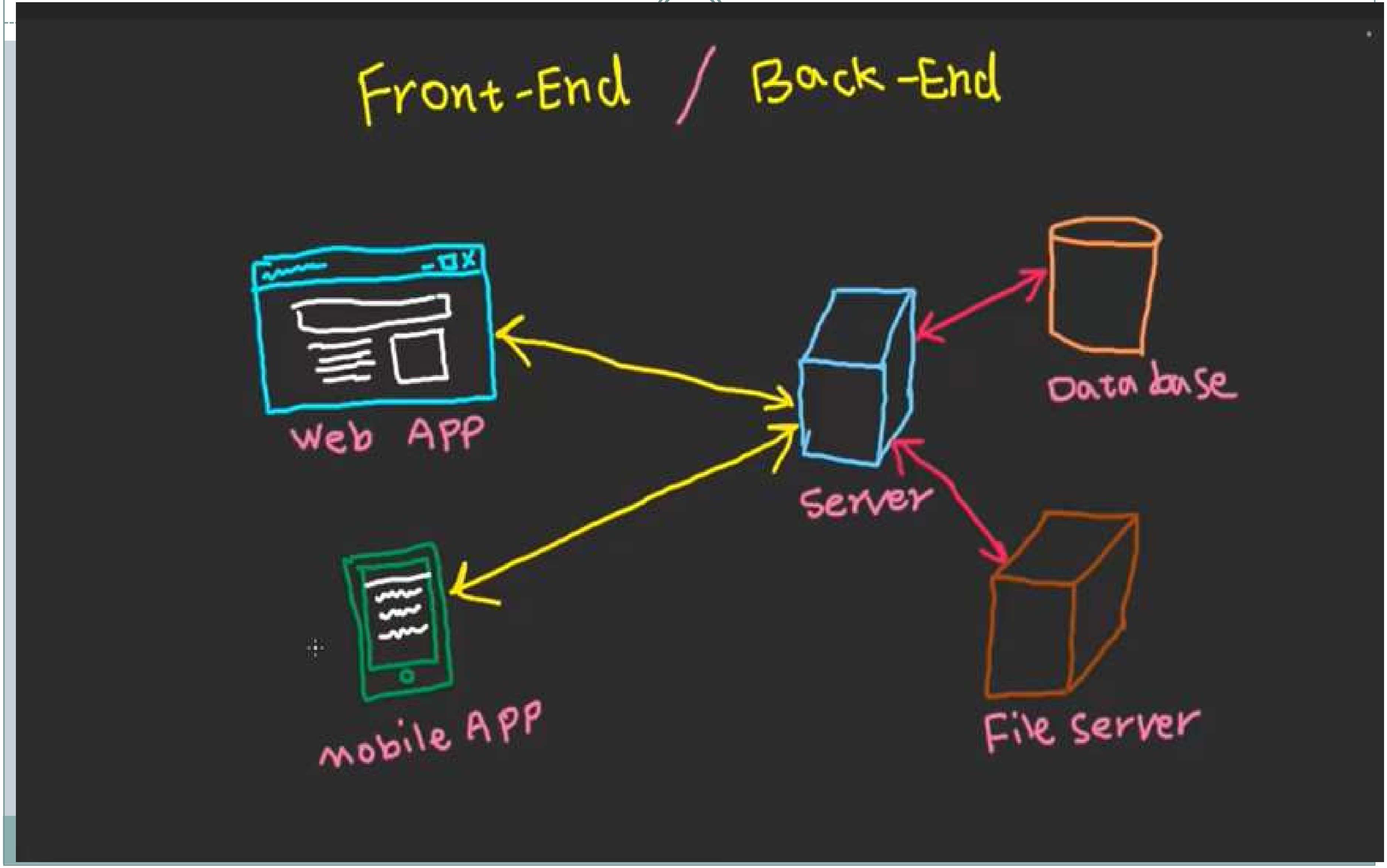
SQL Server 8 , 10 , 12, 16 (By Microsoft)

Oracle 9 ,11, 12 etc. (By Oracle)

My SQL (By Oracle)

DB2 (By IBM)

Front End vs Back End



SQLData Types

The data type of a column defines what value the column can hold: integer, character, date and time and so on.

1. **Char(size)** –Character data type of defined size.

example-name char(15)

2. **Varchar(size)** –size is variable and of character type.

example- fname varchar(15)

* Maximum we can store 15 bytes i.e. 15 characters, but if we store less character then it changes to less size.

3. Integer/int(size)

Signed range is from -2147483648 to 2147483647.
Unsigned range is from 0 to 4294967295.

4. Nchar(size)

5. Nvarchar(size)

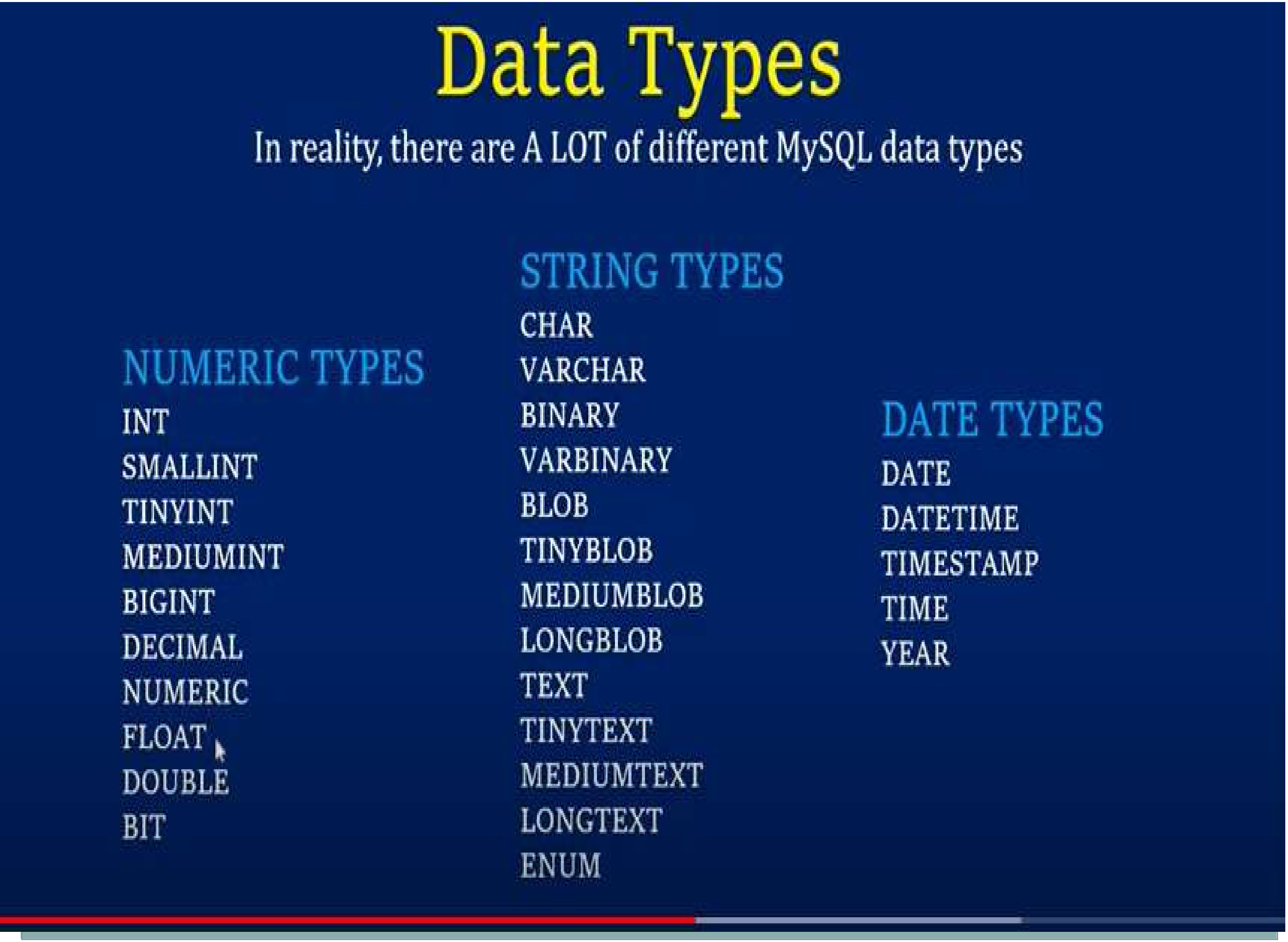
Here 1 char = 2 bytes, so Nchar(15) = 30 bytes

Here also 1 character= 2 bytes

Variable size.

6. **Date**

Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'



Data Types

In reality, there are A LOT of different MySQL data types

NUMERIC TYPES

INT
SMALLINT
TINYINT
MEDIUMINT
BIGINT
DECIMAL
NUMERIC
FLOAT
DOUBLE
BIT

STRING TYPES

CHAR
VARCHAR
BINARY
VARBINARY
BLOB
TINYBLOB
MEDIUMBLOB
LONGBLOB
TEXT
TINYTEXT
MEDIUMTEXT
LONGTEXT
ENUM

DATE TYPES

DATE
DATETIME
TIMESTAMP
TIME
YEAR

- **Database contains multiple tables.**
- **Table contains multiple records.**
- **Records are stored in the table in the form of rows and column.**

Basic Commands in Mysql

1. To create database –create <dbname>;

2 To Select any database –use <dbname>;

. To view available databases –show databases;

3 To view tables inside any databases –show tables;

5. To structure/schema of any table– desc <tablename>;

4 describe <tablename>;

.

Create table Command in SQL

Syntax

```
create table <table name>
(
 attribute1 data type constraint,
  attribute1         data          type
  constraint, :
   :
);
```

Sample code of Create Command

```
Create            table
student (
    sid  char(5)  primary  key,  -attribute  level  sname
    varchar(15),
    fname
    varchar(15),
    marks integer(3) //primary key(sid) -table level
);
```

Table name - Student

Sid	Sname	Fname	Marks

Insert Command

The INSERT INTO statement is used to insert new records in a table.

Insert Into Syntax

It is possible to write the INSERT INTO statement in two ways.

1. insertintotable_name(column1,column2,column3, ...) values(value1,value2,value3, ...);
2. insertintotable_name values(value1,value2,value3, ...);

Example

1. insertintostudent (sid, sname, scity, smarks) values (‘s101’, ‘akash’, ‘raipur’, ‘95’);
2. insertintostudent values (‘s102’, ‘amit', ‘raipur’, ’95’);
3. insertintostudent (sid, sname, smarks) values (‘s103’, ‘rahul’, ’95’);

SQL SELECT Statement

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

The data returned is stored in a result table, called the re

Syntax

SELECT*column1*,*column2, ...*FROM<*table_name>;*

If you want to select all the fields available in the table, use the following syntax:

SELECT*FROM*table_name;*

Example

```
Select * from student;

Select Sid , Sname from student;
```

SQLSELECT DISTINCTStatement

The **select distinct** statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

selectdistinct*column1*,*column2, ...*
from*table_name*;

The SQL 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.

WHERE Syntax

select*column1*,*column2, ...* from*table_name*
where*condition*;

Example

```
Select * from student;
   Select Sid , Snamefrom student;
   Select * from student where marks >70;
   Select * from student where Sname= ‘Rahul’;
   Select Sname, marks from student where marks<40;
```

Sequence

```
Select <attribute list>
From <tablename>
[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

Like Operator

Two Symbols:

1 % = Any number of characters

. _ = Single number of character

2

Q1 –Find out records of those students whose name starts with letter ‘A’.

Answers-select * from student where name like ‘A%’

2. Find out records of those students whose name ends with letter ‘A’.

Answers-select * from student where name like ‘ %A’;

Raman, Karan, Aman, Ananaya

3. Find out records of those students whose second letter is A.

Answer: select * from student where name like ‘_A%’;

4. Find out records of those students whose first letter is A and last letter is I .

Answers-select * from student where name like ‘A%I’; 5. Find out records of those students whose name is
having atleasttwo times A .

Answer: select * from student where name like ‘%A%A%’;

Rahul, karan, vamsi, mohan,

6. Find out records of those students whose name length is 5.

Answer: select * from students where name like ‘_____’

IN OPERATOR

In operator is used to find out selected values.

Ques: Find out the records of those student whose marks are either 90 or 80 or 10;

Query : select * from student where marks IN(90,80,10);

BETWEEN

It is used for any range.

Question: Find out the records of those students whose marks are between 60 to 90;

Query : select * from student where marks between 60 and 90;

Logical Expression

AND

OR

NOT

Example

```
Select * from student where marks >40 AND marks <80;
Select * from student where marks 40 or marks <80;
   Select * from student where not marks=50;
```

SQLORDER BYKeyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
selectcolumn1,column2, ...
fromtable_name
orderbycolumn1, column2, ...asc|desc;
```

ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column: select * from customers orderbycountry;

ORDER BY Several Columns Example

select * from customers orderbycountry, customername;

select * from customers orderbycountryasc, customernamedesc;

SQLNULL Values

A field with a NULL value is a field with no value.

A NULL value is different from a zero value or a field that contains spaces.

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

select *column_names*
from *table_name*
where *column_name* is
null;

IS NOT NULL Syntax

select *column_names*
from *table_name*
where *column_name* is
not null;

Types of SQL Commands

These SQL commands are mainly categorized into four categories as:

1. DDL –Data Definition Language

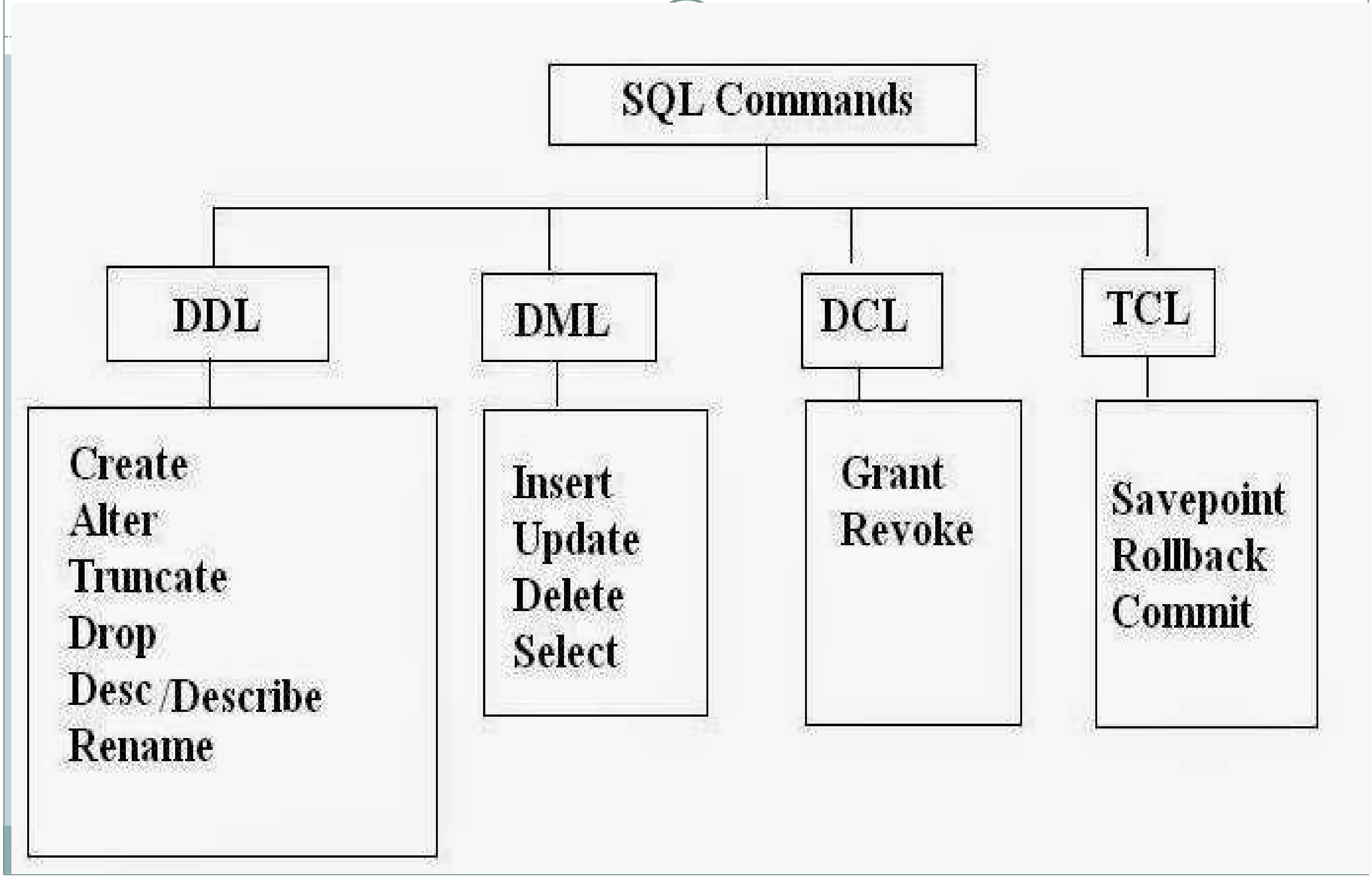
2 DML –Data Manipulation
. Language DCL –Data Control

3. Language

4. TCL –Transaction Control Language

.

Types of SQL Commands



SQL Commands

DDL

Create
Alter
Truncate
Drop
Desc /Describe
Rename

DML

Insert
Update
Delete
Select

DCL

Grant
Revoke

TCL

Savepoint
Rollback
Commit

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

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

SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

<u>DELETE Syntax</u>

DELETEFROM*table_name*WHERE*condition;*

<u>Example -</u>

```
DELETEFROM S3 WHEREsid= 3;
```

SQLDROP DATABASEStatement

The DROP DATABASE statement is used to drop an existing SQL database.

Syntax

DROP DATABASE *database_name*;

SQLDROP TABLEStatement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

DROPTABLE*table_name*;

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

TRUNCATETABLE*table_name*;

TRUNCATE *table_name*;

SQLALTER TABLEStatement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE -ADD Column

To add a column in a table, use the following syntax:

```
ALTERTABLEtable_name
ADD<column> column_namedatatype;
```

Example -The following SQL adds an "Email" column to the "Customers" table:

```
ALTER TABLE Customers
ADD<column> Email varchar(255);
```

ALTER TABLE -DROP COLUMN

To delete a column in a table, use the following syntax.

ALTER TABLE *table_name*
DROP<COLUMN>*column_name;*

Example -The following SQL deletes the "Email" column from the "Customers" table:

ALTERTABLECustomers
DROP <COLUMN> Email;

ALTER TABLE -ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

```
ALTERTABLEtable_name
MODIFY<COLUMN>column_namedatatype;
```

SQL Constraints

SQL constraints are used to specify rules for data in a table.

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table

is created with the ALTER TABLE statement.

Syntax

```
CREATE TABLE table_name (
column1
column2 datatypeconstraint,
datatypeconstraint,
....
column3
```

SQL Constraints(cont.)

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.

The following constraints are commonly used in SQL:

1) **NOT NULL**-Ensures that a column cannot have a NULL value

2 **UNIQUE**-Ensures that all values in a column are different

) **PRIMARY KEY**-A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

3

4)) **CHECK**-Ensures that all values in a column satisfies

a

5) **DEFAULT**-Sets a default value for a column when no value is

specific condition

specified

SQLNOT NULLConstraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

Syntax

```
1. CREATETABLEPersons ( ID intNOTNULL,
LastName varchar(255)NOTNULL,
FirstName varchar(255)NOTNULL,
Age int );
2. ALTERTABLEPersons
      MODIFYAge intNOTNULL;
```

SQLUNIQUEConstraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Syntax

```
    CREATETABLEPersons (
ID intNOTNULL,
     LastName varchar(255)NOTNULL,
   FirstName varchar(255),
     Age int,
     UNIQUE(ID) );
  ALTER TABLE Persons
ADD UNIQUE (ID);
```

SQLPRIMARY KEYConstraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

SQL PRIMARY KEY on CREATE TABLE

```
CREATETABLEPersons (
   ID intNOTNULL,
      LastName varchar(255)NOTNULL,

   FirstName varchar(255),
Age int,

          PRIMARYKEY(ID) );

   ALTER TABLE Persons

   ADD  PRIMARY  KEY  (ID);
   ALTER TABLE Persons

   DROP PRIMARY KEY;
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

Example

```
CREATETABLEPersons (
   ID intNOTNULL,
    LastName varchar(255) NOT NULL,

    FirstName varchar(255),

   Age int,
CHECK(Age>=18) );
   ALTER TABLE Persons

    ADD CHECK (Age>=18);
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

Example

```
CREATETABLEPersons (
   ID intNOTNULL,
  LastName varchar(255) NOT NULL,
   FirstNamevarchar(255),
Age int,
   City varchar(255)DEFAULT'Sandnes');
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQLAUTO INCREMENTField

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Syntax

```
CREATETABLEPersons (
Personid int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
PRIMARY KEY (Personid));
```

AUTO INCREMENT

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

ALTERTABLEPersons AUTO_INCREMENT=100;

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

LIKE Syntax

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

Examples:

1. Selects all customers with a CustomerName starting with "a":

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

2. Selects all customers with a CustomerNameending with "a":

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

3. Selects all customers with a CustomerName that have “ a " in any position:

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

4. Selects all customers with a CustomerName that have "r" in the second position:

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

5. selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

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

6. selects all customers with a ContactName that starts with "a" and ends with “h":

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

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.

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

Example

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

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.

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

Example-

SELECT * FROM Students
WHEREMarks IN(80,60,50);

Aggregate functions in SQL

In database management an aggregate function is a function where the values of multiple rows are groupe together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions

1 Min()

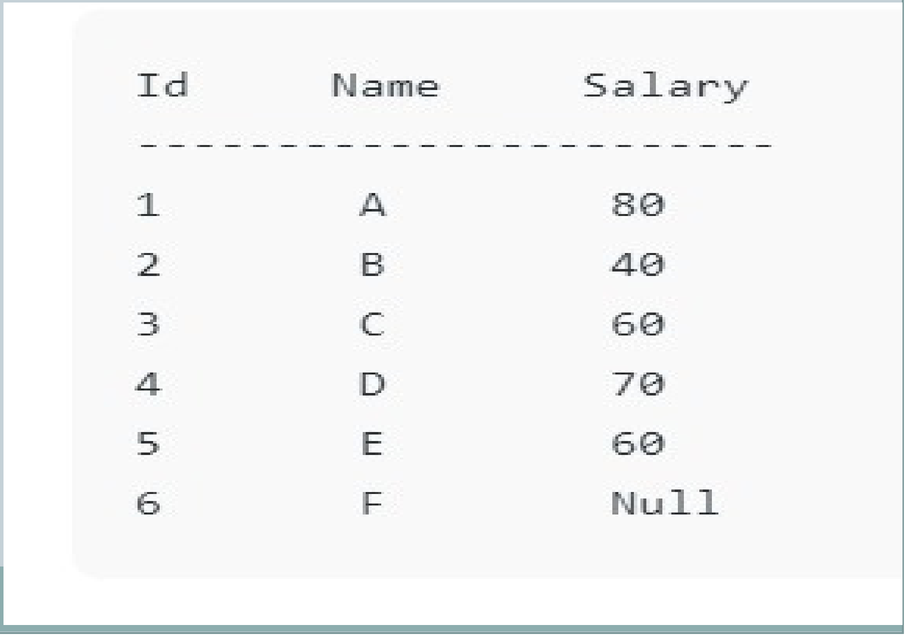
. Max()

2 Count()

4. Sum(

3.)

5. Avg()



Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

The SQL MIN() and MAX() Functions

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

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

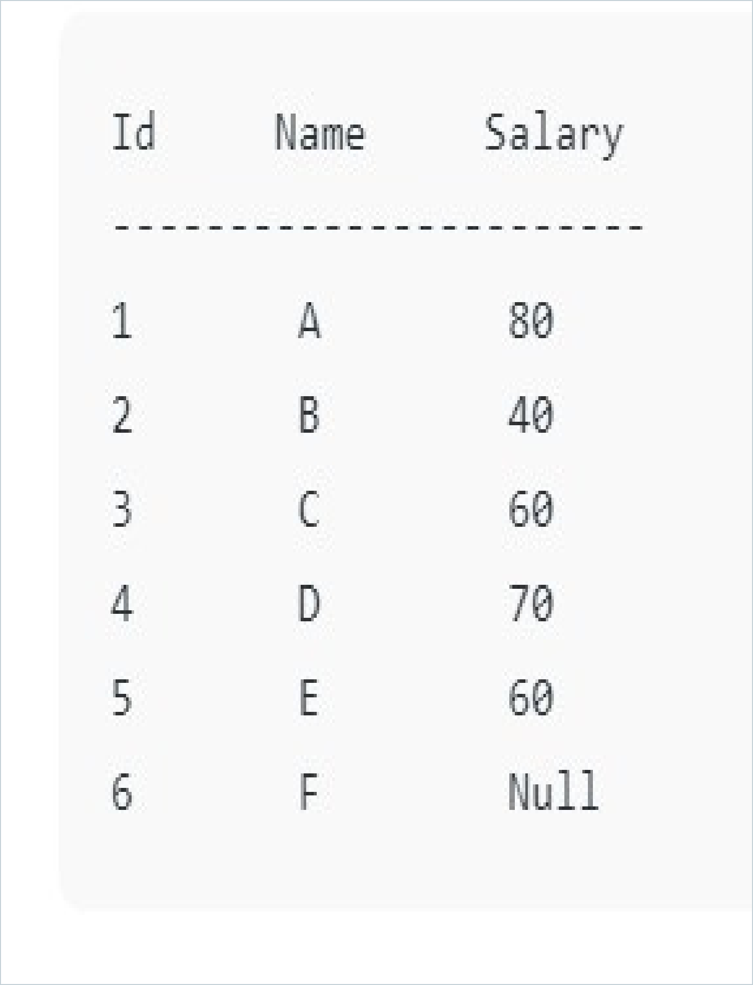
The MAX() function returns the largest value of the selected column.

```
SELECT MAX(column_name)
FROMtable_name
WHERE condition;
```

Example of Min() & Max()

Min(salary): Minimum value in the salary column except NULL i.e., 40.

Max(salary): Maximum value in the salary i.e., 80.



Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

The SQL COUNT( ) Functions

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

SELECT COUNT(*column_name*)
FROM*table_name*
WHERE *condition*;

Count(*):Returns total number of records .

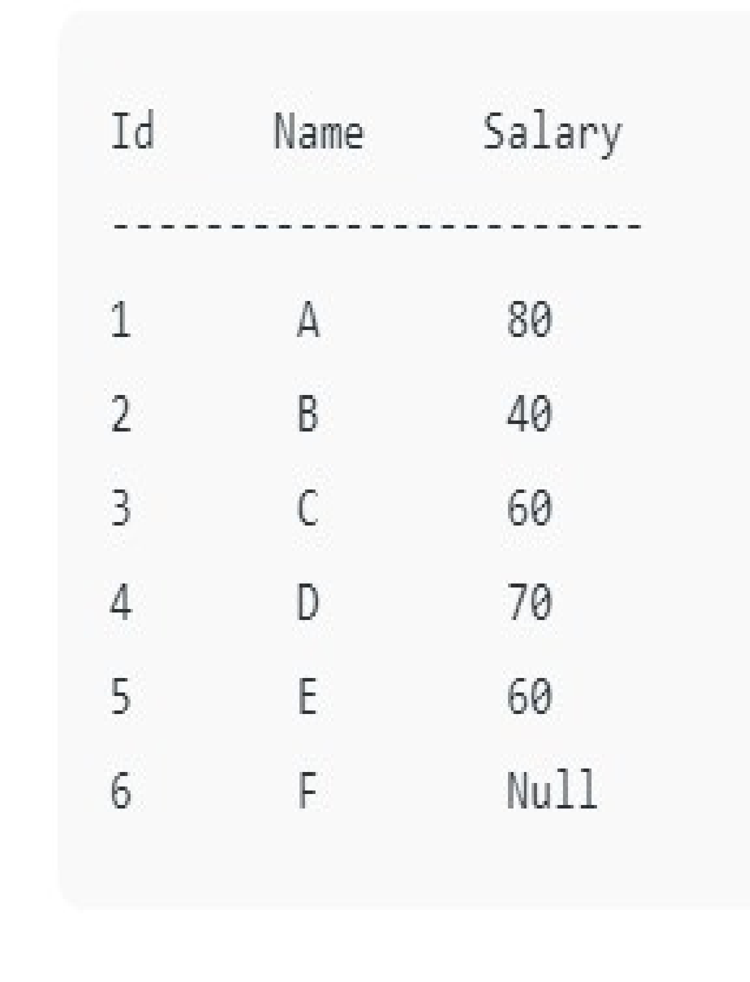
Count(column_name):Return number of Non Null values
that column.
over ***Count(Distinct column_name):***Return number of
Null values over that column.
distinct Non

Example of count

Count(*):Returns total number of records .i.e6.

Count(salary):Return number of Non Null values over the column salary. i.e5.

Count(Distinct Salary):Return number of distinct Non Null values over the column salary .i.e4



```
Id      Name      Salary
------------------------
1        A         80
2        B         40
3        C         60
4        D         70
5        E         60
6        F         Null
```

The SQL SUM( ) Functions

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

SELECT SUM(*column_name*)
FROM*table_name*
WHERE *condition;*

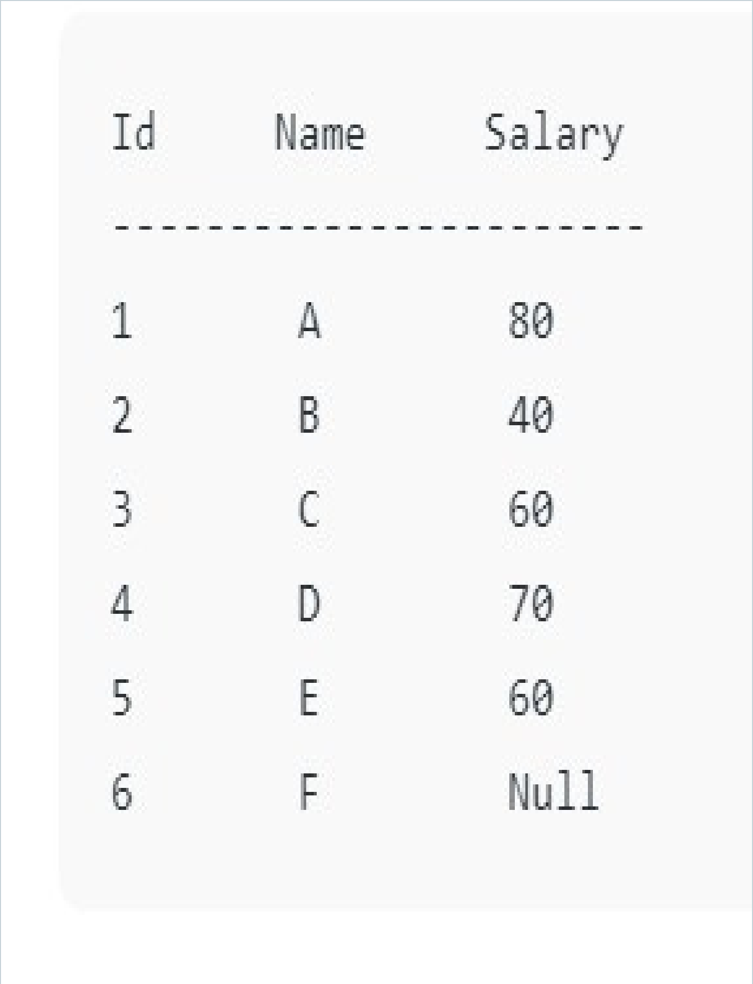
Sum (colum n_name)

Sum(Distinct column_name)

Example ofSum():

sum(salary):Sum all Non Null values of Column salary i.e., 310.

sum(Distinct salary):Sum of all distinct Non-Null values i.e., 250.



Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

The SQL AVG( )Functions

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

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

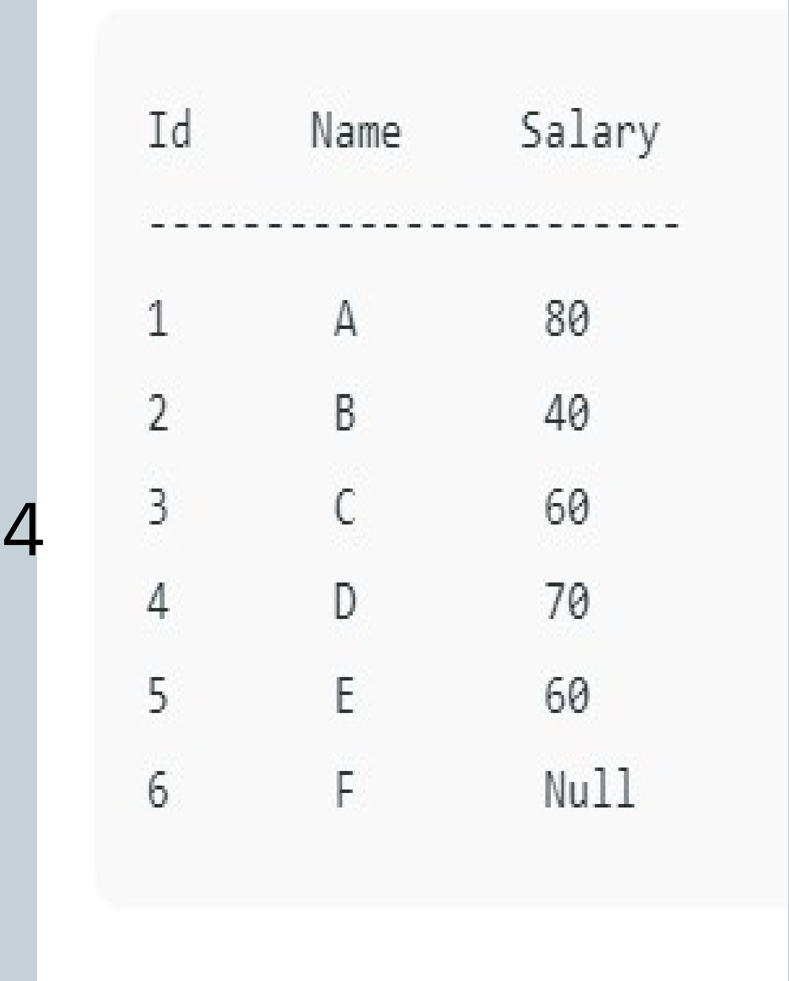
Avg(column_name)

Avg(Distinct column_name)

Example ofAVG():

Avg(salary)= Sum(salary) / count(salary) = 310/5

Avg(Distinct salary)= Sum(Distinct Salary) / count(Distinct salary)) = 250/4



```
Id      Name      Salary
------------------------
1        A         80
2        B         40
3        C         60
4        D         70
5        E         60
6        F         Null
```

General Syntax of SQL statements

```
    SELECT [Distinct] <attribute list>
FROM <TABLE NAME>
[WHERE <condition>]
     [GROUP BY (Attribute) [ having Condition]]
  [ORDER BY (Attribute) [ASC/Desc]];
```

The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

```
 SELECT column_name(s)
FROMtable_name
WHEREcondition
GROUP  BY  column_name(s)
ORDERBYcolumn_name(s);
```

SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

```
SELECTCOUNT(CustomerID), Country
FROMCustomers
GROUP BYCountry;
```

The following SQL statement lists the number of customers in each country, sorted high to low:

```
SELECTCOUNT(CustomerID), Country
FROMCustomers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

HAVING Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
HAVING*condition*
ORDER BY *column_name(s);*

SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECTCOUNT(CustomerID), Country
FROMCustomers
GROUP BY Country
HAVINGCOUNT(CustomerID) >5;
```

SQL Views

In SQL, a view is a virtual table based on the result-set of an SQL statement.

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.

Syntax

```
CREATEVIEWview_nameAS
SELECTcolumn1,column2, ...
FROMtable_name;
```

Creating a table from another table

Syntax

```
Create table <table name>
As < Select Query>;
```

Example

```
Create table chotastudent
as select sid, name, marks from student;
```

Inserting records from another table

Syntax

Insert into <table name> <a select Query>;

Example-

```
1. Insert into chotastudent
     select sid,name,marksfrom student;
2. Insert into chotastudent
     select sid,name,marksfrom student
      where marks>35;
```

Join

A join clause is used to fetch data from two or more tables, based on join condition.

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

STUDENT

SID	Sname	Marks	Dept No
s1	A	30	D1
S	B	4	D2
2	C	0	D1
DS4 3		45 5	D2

DEPARTMENT

Dno	Dname	Location
D1	CS	RAIPUR
D2	CHEM	DURG
D3	PHY	RAIPUR

Question 1: Retrieve the sid and department no. of students whose marks <40;

Select sid, dept no from student where marks<40;

SID	DEPT NO
S1	D1
S4	D2

Question 2: Retrieve the sid and department no. of students whose marks <40;

Cartesian Product

Student X Department

SID	SNAME	MARKS DEPTNO DNO		30 D1	DNAME LOCATION	
S1	A	D1			CS RAIPUR	
S1	A	30 D1 D2			CHEM DURG	
S1	A	30 D1 D3			PHY RAIPUR	
S2	B	40 D2 D1			CS RAIPUR	
S2	B	40 D2 D2			CHEM DURG	

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.

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.

Alias Column Syntax

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

Alias for Columns Examples

```
      SELECTSnameASsn, marks ASm
FROM student;
```

Alias Table Syntax

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

Alias for Tables Example

```
SELECTs.sname, s.marks, d.dname
FROMStudentASs, DepartmentASd
WHERE s.sname=‘A' AND s.deptno=d.dno;
```

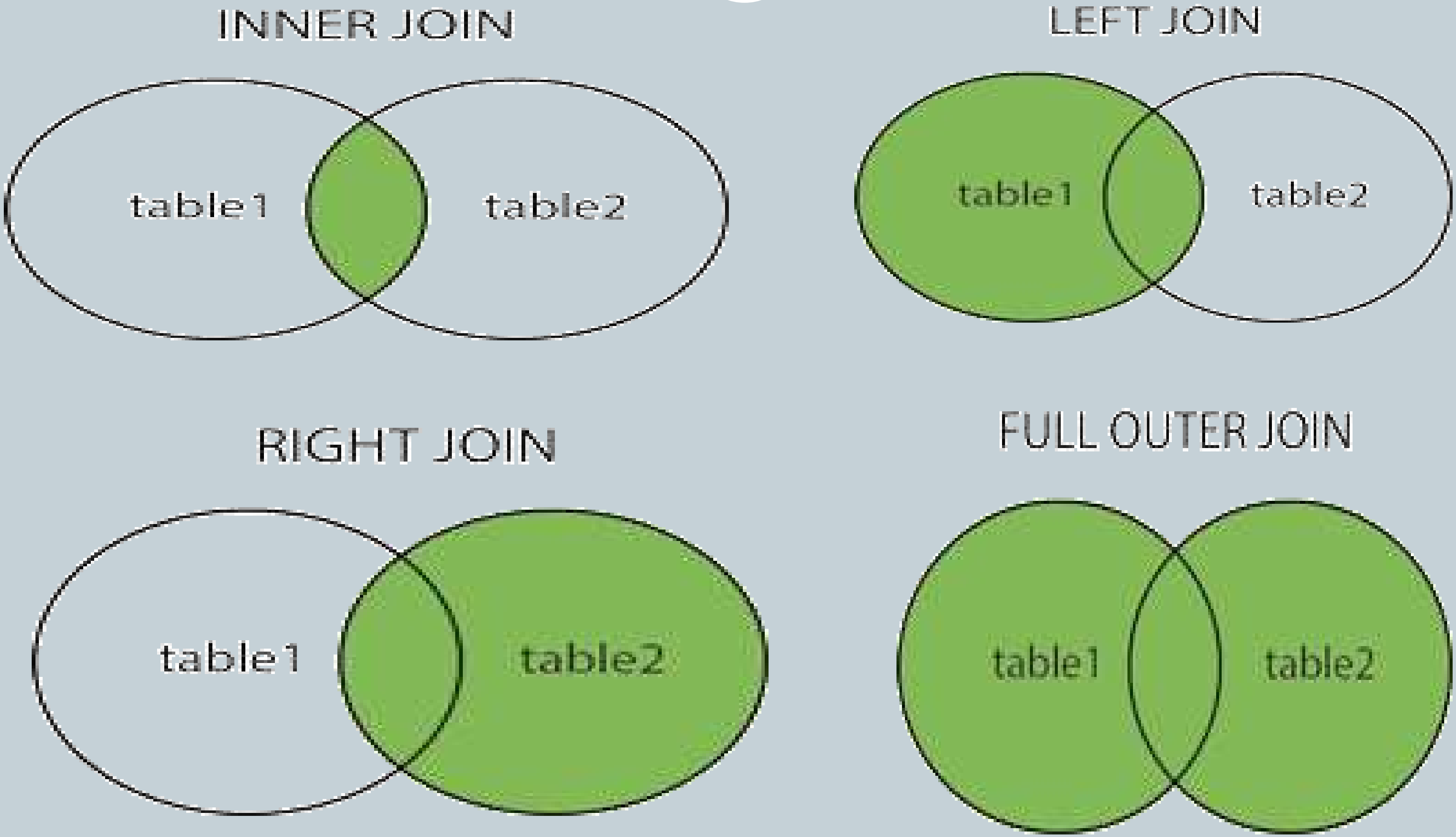
```
SELECTStudent.sname, Student.marks, department.dname
FROM Student,department
WHERE Student.sname =‘Ram' AND
    student.deptno=department.dno;
```

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

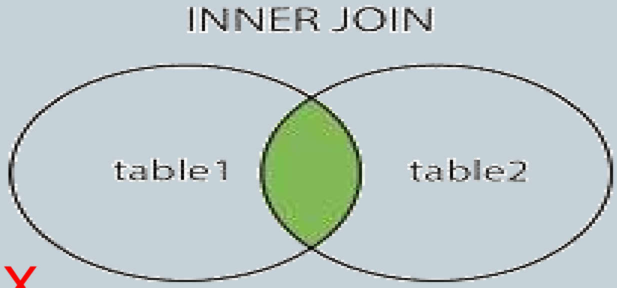
1. **(INNER) JOIN**: Returns records that have matching values in both tables.
2. **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table.
3. **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table.
4. **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table.

Different Types of SQL JOINs



SQL INNER JOIN Keyword

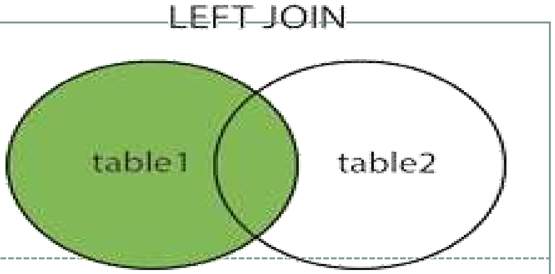
The INNER JOIN keyword selects records that have matching values in both tables.



INNER JOIN Syntax

```
SELECT column_name(s)
FROMtable1
INNER JOIN table2
   ON table1.column_name = table2.column_name;
```

SQL LEFT JOIN Keyword

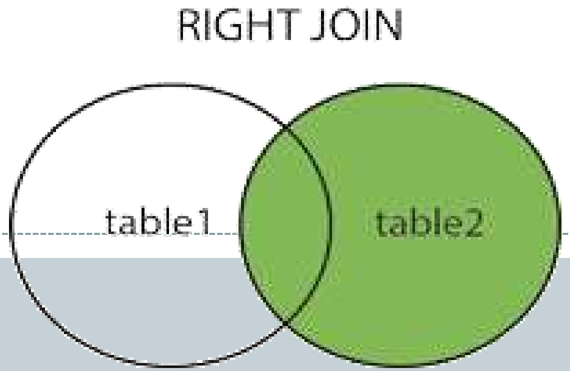


The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFTJOINtable2
   ON table1.column_name = table2.column_name;
```

SQL RIGHT JOIN Keyword

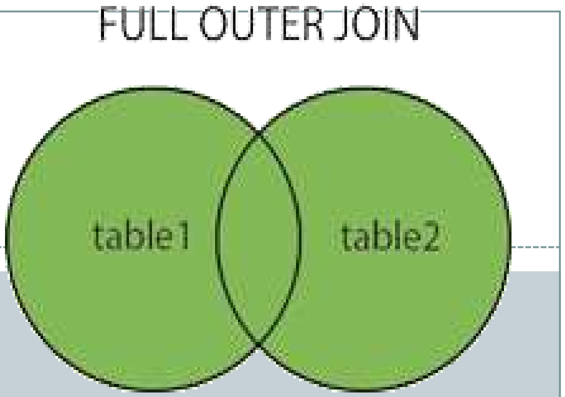


The RIGHT JOIN keyword returns all records from the right table (table2), and the matched from the left table (table1). The result is NULL from records the left side, when there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
   ON table1.column_name = table2.column_name;
```

SQL FULL OUTER JOIN Keyword



The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULLOUTER JOINtable2
   ON table1.column_name = table2.column_name
WHEREcondition;
```

SQL Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

Self JOIN Syntax

```
SELECT column_name(s)
FROMtable1 T1, table1 T2
WHERE condition;
```

*T1*and*T2*are different table aliases for the same table.

SQLFOREIGN KEYConstraint

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

"Persons" table:

PersonI	LastNam	FirstName	Ag
D 1	e Hansen	Ola	e
2	Svendson	Tove	30 23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	4467	3
3	8	2
4	22456 24562	1

6

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

actions that would destroy links between tables.

The FOREIGN KEY constraint is used to prevent

The FOREIGN KEY constraint also prevents invalid

data from being inserted into the foreign key

because it has to be one of the values contained in the column, table it points to.

Functional dependency

In a relation R, let these be two attributes X and Y, the we can say X->Y (X determines Y or Y functionally determined by X) exists for any two tuples t1 and t2 only if they satisfy following conditions.

If t1. X = t2. X then
t1. Y = t2. Y

	X	Y
t1	X1	Y
t2	X1	1

Y
1

	X	Y
t1	X1	Y
t2	X1	1

Y
2

TABLE 1 Is Functionally Dependent but table 2 is not Functionally Dependent

Find the Functional Dependency in following table

A	B	C
2	3	8
1	1	3
3	3	2
1	1	6
2	3	1
2	3	5
1	1	4

Check Whether
1.A -> B
2.A ->C
3.B -> A
4.C ->A
5.C ->B
6.AB ->C
7.AC ->B
8.BC ->A
9.B -> C
10. C->B
11.A ->BC
12. B->AC
13.C ->AB

Attribute Closure (X^{+})

This is the set of attributes determined by X.

Let R(A,B,C,D)

A + = { A,B,C,D}

A defines itself.

Trivial Dependency

Trivial means whose answer is already known to us. like What is your name Ram?

Example : A -> A

AB -> A BC -> C

Inference Rules

1.Reflexive Rules

if X>= Y then X -> y (Trivial)

2. Augmentation Rules –

if X -> Y exists then XZ -> YZ will also exists.

3. Transitive Rules

if X -> Y and Y->Z then X -> Z will also exist.

4. Decomposition Rules

X -> YZ then X -> Y and X -> Z also .

5. Additive Rules

if X -> Y AND X -> Z THEN X -> YZ