**DBMS**

**Database Management Systems**

## **What is Database?**

Database is a collection of information’s organized for easy access, management and maintenance.

Example:

* Telephone directory
* Customer data
* Product inventory
* Visitors register
* Weather records

Note: - Let us consider Facebook. It needs to store, update and show related data of members, activities, messages etc

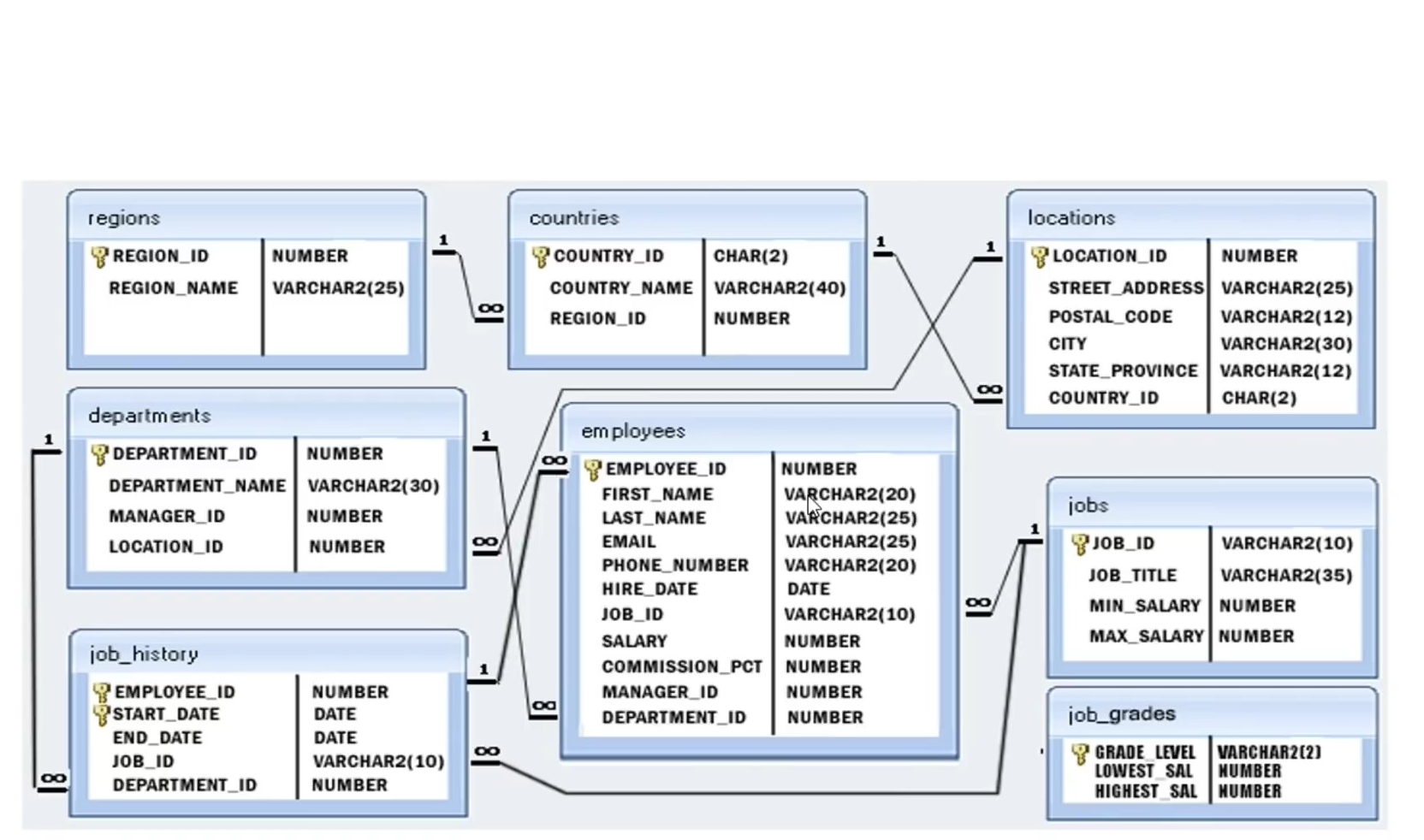
**Types of Data Models**

* **Record based logical model**
  + Hierarchical data model
  + Networks data
  + Relational data
* **Object based logical model**
  + Entity relationship

## **Table of Contents**

* Introduction to DBMS and SQL
* ER- Diagram
* Normalizations
* Subsets of SQL
  + DDL
  + DML
  + DCL
  + TCL
* SQL Operators
* SQL Functions
* SQL Joins
* Lab Sessions
* Class Topics

# Chapter1: Introduction



**DBMS Operations:**

* Adding new files: It means that add value in table
* Inserting data
* Retrieving data : Replace data
* Modifying data : Name changes
* Removing data : Delete data for database
* Removing files: remove database of table

**Advantage of DBMS:**

* Sharing of data across applications
* Enhanced security mechanism
* Enforce integrity constraints
* Better transaction support
* Backup and recovery

**Introduction to RDBMS:**

* A relational database refers to a database that stores data in a structure format, using rows and columns.
* This makes it easier to locate and access specific values within the database.
* It is “relational” because the values within each table are related to each other.
* Tables may also be related to the other tables.
* The relational structure makes it possible to run queries across multiple tables at once.

**Features of RDBMS:**

* Every piece of introduction is stores in the forms of tables.
* Has primary keys for unique identification of rows
* Has foreign keys to ensure data integrity
* Provides SQL for data access
* Uses indexes for faster data retrieval
* Gives access privileges to ensure data security

**RDBMS VS TRADITIONAL APPROACH**

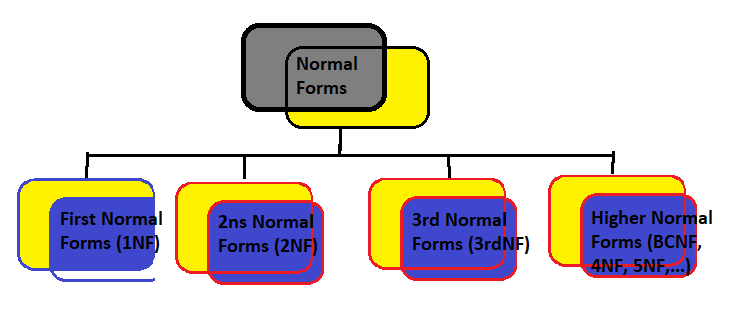
* The Key difference is that RDBMS(Relational Database Management System) applications **store data in a tabular form**, where in traditional approach, **applications store data as files**.
* There can be, but there will be no “relational” relational the tables, like in a RDBMS in traditional approach, data is generally stored in either **a hierarchical form**/navigational form. This means that a single data until will **have 0, 1 or more children’s nodes and 1** parent node.

# Chapter 2: Normalization

**Normalization:**

* **Decompose larger, complex table into simple and smaller once.**
* **Moves from lower normal forms to higher normal forms.**

**Normalization and Normal Forms:**

****

**Need for Normalization**

* In order to produce good database design
* To ensure all operations to be efficiently performed
* Avoid any expensive DBMS operations
* Avoid unnecessary replication of information’s
* **DATABASE of Students but wrong learn next I will explain why wrong.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Students\_Details** | **Course\_details** | **Pre-requisite** | **Results\_details** |
| 101 Shailu 05/01/2002 | J1 Advance Java 301 | Basic Java | 03/08/2022 85 A |
| 102 Sanju 14/02/2002 | P4 Advance Python 304 | Basic Python | 03/08/2022 78 B |
| 103 Annu 05/01/2002 | S Advance SQL 308 | Basic SQL | 03/08/2022 98 A |
| 104 Priya 17/02/2002 | C5 Advance C++ 302 | Basic C++ | 03/08/2022 58 C |

**Note:** Before learn types of normal forms in normalization 1NF, 2NF………we will be most important concepts:

**Functional Dependency:**

* Consider the relation
* Results ( Students#, Course#, CourseName#, Marks#, Grade#)
  + Students# and Course# together exactly one value of marks. Students#, Course#, Marks
  + Students# and Course# determines Marks or Marks is functionally dependent on student# and Course#
* Other Functional dependencies in the relation:
  + Course# - CourseName
  + Marks# - Grade
* Let’s see an example for demo
* In a given relation R, P and Q are attributes. Attributes Q is functionally dependent on attributes P if each of P determines exactly one value of Q

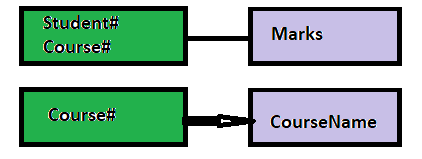
Q

P

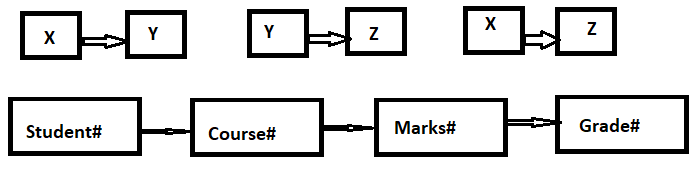
**Functional Dependency Types**

Partial Functional Dependency

Transitive Dependency

1. **Partial Functional Dependency:** 
   1. Attributes Q is partially dependent on attributes on P, if and only if is dependent in the subset of attributes P.
   2. REPORT(Student#, Course#, StudentName#, CourseName#, Marks, Grade)
2. **Transitive Dependency:**

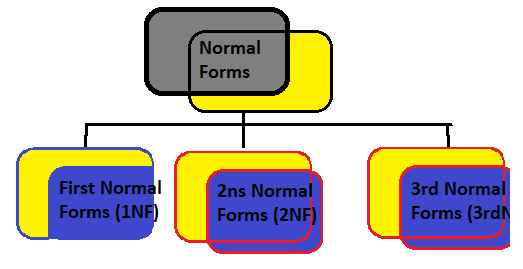
**X, Y,Z are three attributes**

****

## **Types of Normalization**

* We are already discussing about normalization but here now discuss on types only major three:

**Types of Normal Forms**



1. **First Normal Form – (1NF)**

* **A relational schema is in 1NF, if and only if:**
  + **All attributes in the relation are atomic (indivisible value)**
  + **And there are no repeating elements or group of elements**

|  |  |  |  |
| --- | --- | --- | --- |
| **Students\_Details** | **Course\_details** | **Pre-requisite** | **Results\_details** |
| 101 Shailu 05/01/2002 | J1 Advance Java 301 | Basic Java 18 | 03/08/2022 85 A |
| 102 Sanju 14/02/2002 | P4 Advance Python 304 | Basic Python 25 | 03/08/2022 78 B |
| 103 Annu 05/01/2002 | S Advance SQL 308 | Basic SQL 33 | 03/08/2022 98 A |
| 104 Priya 17/02/2002 | C5 Advance C++ 302 | Basic C++ 17 | 03/08/2022 58 C |

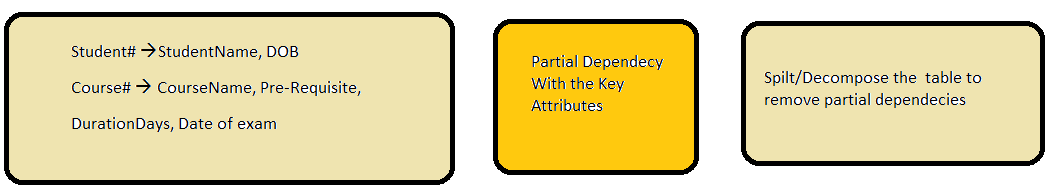
**Note:-** Remembers, I tell you this is wrong table of students information so now explain how to convert in right.

**Student Information Table in 1NF**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student**  **#** | **Student\_**  **Name** | **DOB** | **Course#** | **Course**  **Name** | **Course**  **Id** | **Pre-Requisite** | **Duration in Days** | **Date**  **of Exam** | **Marks** | **Grade** |
| **101** | **Shailu** | **05/01/**  **2002** | **J1** | **Advance Java** | **301** | **Basic Java** | **18** | 03/08  /2022 | **85** | **A** |
| **102** | **Annu** | **07/09/**  **2002** | **P4** | **Advance Python** | **302** | **Basic**  **Python** | **25** | 03/08/  2022 | **78** | **B** |
| **103** | **Priya** | **17/02/**  **20004** | **S2** | **Advance**  **SQL** | **303** | **Basic SQL** | **33** | 03/08/  2022 | **98** | **A** |

1. **Second Normal Form – (2NF):**

* A relation is said to be in 2NF, if and only if:
* It is in 1st Normal form.
* No partial dependency exits between non-key attributes and key attributes**.**
* **Here Now 1NF Divided by 2NF so let’s see now:**
* Students#, Course# 🡪 Marks
* Students#, Course# 🡪 Grade
* Marks🡪 Grade



**Student Information Table in 2NF**

**Student Table**

|  |  |  |
| --- | --- | --- |
| **Student#** | **Student\_Name** | **DOB** |
| 101 | Shailu | 05/01/2002 |
| 102 | Annu | **07/09/2002** |
| 103 | Priya | 17/02/2002 |

**Result Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student#** | **Course#** | **Marks** | **Grade** |
| 101 | J1 | 85 | A |
| 102 | P4 | **78** | **A** |
| 103 | S2 | 98 | B |

**Course Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Course#** | **CourseName** | **Prerequisite** | **DurationDay** | **Date**  **of Exam** |
| J1 | Advance Java | Basic Java | **18** | 03/08  /2022 |
| P4 | **Advance Python** | Basic Python | **25** | 03/08/  2022 |
| S2 | Advance SQL | Basic SQL | **33** | 03/08/  2022 |

1. **Third Normalization –(3NF):**
2. **Result Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student#** | **Course#** | **Marks** | **Grade** |
| 101 | J1 | 85 | A |
| 102 | P4 | **78** | **A** |
| 103 | S2 | 98 | B |

* Student#, Course# 🡪 Marks
* Student#, Course# 🡪 Grade
* Marks 🡪 Grade
* Student#, Course# 🡪 Marks 🡪 Grade: TD -----🡪 Remove

**Result Table**

|  |  |  |
| --- | --- | --- |
| **Student#** | **Course#** | **Marks** |
| 101 | J1 | 85 |
| 102 | P4 | **78** |
| 103 | S2 | 98 |

**Marks Grade Table**

|  |  |
| --- | --- |
| **Marks** | **Course#** |
| 85 | A |
| 73 | B |
| 98 | A |

# Chapter 3:-SQL

Structure Query Language

# What is SQL?

It is a programming language specifically designed for working with Database(DBMS) to….

* CREATE
* DELETE
* REMOVE
* MANIPULATE
* SHARE/ACCESS

# Why is SQL?

SQL is a widely popular because it offers the following advantages:

* Allows users to communicate i.e, **access and manipulate** the database.
* Allows users to retrieve data from a database.
* Allows users to create, update, modify and delete the database

So, SQL is a popular language for defining the structure of a database.

## **SQL Terms**

**Data:** Data is defined as figures, or information that’s stored in or used by a computers

**Database:** A Database is a organized collection of data/ information so that it can be easily accessed, manged and updated.

## **SQL Data Types**

1. Numeric – bit, tin, smaller, int, bigint, decimal, numeric, float, real
2. Character/String – Char, Varchar, Test
3. Date/Time – Date, Time, Datetime, Timestamp, Year
4. Miscellaneous – Json, XML

## **SQL Constraints/ Rules:**

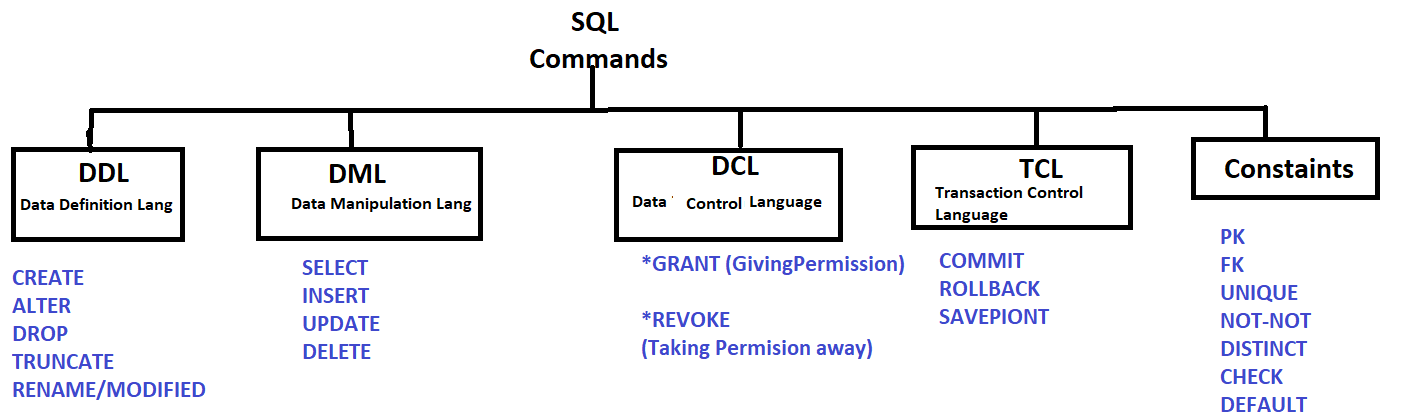
|  |  |
| --- | --- |
| **Constraints** | **Descriptions** |
| Not Null | Ensures that a columns does not have a NULL value. |
| Default | Provides a default value for a columns when none is specified. |
| Unique | Ensures that all the values in a columns are different. |
| Primary | Identifies each row/records in a database tables uniquely. |
| Check | Creates and retrieves data from the database very quickly. |
| Index | Create and retrieves data from the database very quickly. |

## **SQL Command Groups:**

1. **DDL** (Data Definitions Language): creations of objects/structure
2. **DML** (Data Manipulations Language): manipulations of data
3. **DCL** (Data Control Language): assignment of removal of permissions
4. **TCL** (Transaction Control Language): saving and restoring changes to a database.

**Note:----- How to add comments in SQL**

* Same as input python type
* 1-line Comments: # This is one line comments
* 2 or More then line Comments: /\* This is one to more then line comments \*/



## **DDL -** (**Data Definitions Language):**

* **In this case create a structure of table and modified of table parts of rows or columns, take care not delete data in any particular members of information.**

|  |  |
| --- | --- |
| **Command** | **Description** |
| Create | Create objects in the database objects |
| Alter | Alters the structure of the database/ database objects |
| Drop/ Remove | Delete objects from the database |
| Truncate | Removes all records from a table permanently |
| Rename | Renames an object |

**DDL - (Data Definitions Language) – Create Command Syntax of DDL**

CREATE TABLE employess(

    emp\_id INT(10) NOT NULL,

    first\_name VARCHAR(20),

   // last\_name VARCHAR(20) NOT NULL,

    salary int(10) NOT NULL,

    PRIMARY KEY(emp\_id);

)

**Note: When run all code in Workbench SQL Software in your system then create a format of table in put value of employee’s details see in demo.**

**Structure of table**

|  |  |  |  |
| --- | --- | --- | --- |
| **E\_id** | **E\_Name** | **E\_Salary** | **E\_Address** |
| **101** |  |  |  |
| **102** |  |  |  |
| **103** |  |  |  |
| **104** |  |  |  |

Let’s see a video for better understanding on YouTube:

* How to Download and Install Workbench for SQL
* How to create table in MySQL
* Full tutorial on my channel Go and see for learn.

**Note: Following all Steps for Create a Database in Workbench SQL software:**

**Step 1: Create a Database.**

**Command:**

* create database Company;

**Step 2: Create a Table in Database**.

**Command:**

* create table employees(
* emp\_id int not null,
* first\_name varchar(20),
* last\_name varchar(20),
* salary int,
* primary key(emp\_id)
* );

**Step 3: Show the datable format:**

**Command:**

* show select \*from employees;
* SELECT \* FROM registr.sregsnum; // when import data
* describe emp;

**Step 4: Add Extra column in table of employees contact\_num:**

**Command:**

* **alter table employees add column contactnum int;**

**Note : Show data add yes or not.**

**Command:**

* **select \*from employees;**

**Step 5: Show the data which type int, varchar in row format.**

**Command:**

* **describe employees;**

**Step 6: Rename the column contact to mobile num:**

**Command:**

* **alter table employees rename column contactnum to mobilenum;**

**Step 6: Truncate delete all employees data in database;**

**Command:**

* **truncate table employee;**

**Step 7: Delete all data in with database / Delete Table of employees;**

**Command:**

* **Drop table employees;**

So Finally, you are creating table of structure in DDL command using and this is all basic command of DDL same as DML, DCL, etc, command you can use.

## **DML -** (**Data Manipulations Language):**

* **This is 2nd steps of Database in this case insert information in table 4.**

|  |  |
| --- | --- |
| **Command** | **Description** |
| Insert | Insert data into a table |
| Update | Update existing data within a table |
| Delete | Delete specified/ all records from a table |

**DML – Insert Command:**

Using insert command add elements value in table of studs more student information.

* **insert into studs(S\_Roll, S\_Name, S\_Marks, S\_Fees) value (82025, 'Shailu', '458',18500);**
* **insert into studs(S\_Roll, S\_Name, S\_Marks, S\_Fees) value (82089, Annu', '558',18500);**

**Insert element in DDL**

|  |  |  |  |
| --- | --- | --- | --- |
| **S\_Roll** | **S\_Name** | **S\_Marks** | **S\_Fees** |
| **82025** | **Shailu** | **458** | **18500** |
| **82089** | **Annu** | **558** | **18500** |
| **82026** | **Priti** | **489** | **18500** |
| **82045** | **Abhi** | **589** | **18500** |

**DML – update Command:**

Using update command update elements value in table where add primary key.

* update studs set S\_Name='Priya' where S\_Roll=82025**;**
* **update department set dept\_id=3 where dept\_nm='HR'; //update one constants**

**update element in DDL**

|  |  |  |  |
| --- | --- | --- | --- |
| **S\_Roll** | **S\_Name** | **S\_Marks** | **S\_Fees** |
| **82025** | **Priya** | **458** | **18500** |
| **82089** | **Annu** | **558** | **18500** |
| **82026** | **Priti** | **489** | **18500** |
| **82045** | **Abhi** | **589** | **18500** |

**DML –delete Command:**

Using delete command delete elements value in table where add primary key also

* delete from studs where S\_Roll in (82045);

**delete element in DDL**

|  |  |  |  |
| --- | --- | --- | --- |
| **S\_Roll** | **S\_Name** | **S\_Marks** | **S\_Fees** |
| **82025** | **Priya** | **458** | **18500** |
| **82089** | **Annu** | **558** | **18500** |
| **82026** | **Priti** | **489** | **18500** |
|  |  |  |  |

**Complete DDL Command:**

1. **#DDL Command using full in this post**
2. **#insert into studs(S\_Roll, S\_Name, S\_Marks, S\_Fees) value (82089, 'Anuu', '458',18500);**
3. **#update studs set S\_Name='Priya' where S\_Roll=82025;**
4. **#delete from studs where S\_Roll in (82089);**
5. **select \*from studs;**

## **DML -** (**Data Manipulations Language):**

It is used for security purpose, where decide which members given permission for access my database of elements.

|  |  |
| --- | --- |
| **Command** | **Description** |
| Grant | Gives access privileges to database |
| Revoke | Withdraws access privileges given with the grant command |

* **Grant:- Give the permission for data access**

**Example: - GRANT <privilege list /column name> ON <Relation Name/ Table Name> TO <USER>**

* **Revoke:-Take the permission by user**

**REVOKE <privilege list /column name> ON <Relation Name/ Table Name> TO <USER>.**

## **DCL -** (**Data Transaction Language):**

**It is used for security purpose, where if you have given permission by user wrong employees for data access then rollback and modified your problem.**

|  |  |
| --- | --- |
| **Command** | **Description** |
| Commit | Saves the work done |
| Rollback | Restores database to origin state since the last commit |
| Savepoint | Identify a point in a transaction to which you can roll back later. |

# Chapter 4 :- SQL Operators

1. **Where Clause (Where Operators)**:

* Used to specify a condition while fetching the data from a single table or by joining with multiple tables.
* Not only used in the SELECT statements, but it is also used in the UPDATE, DELETE statements, etc.

**Command:**

* **SELECT \*from employees WHERE emp\_id=101;**

Where the example mentioned above extracts all the columns from the table ‘employees’ whose emp\_id=101.

1. **SQL Operators -Logical:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operators** | **IllustrativeExample** | **Condition** | **Results** |
| AND | (10<20) **AND** (20>10) | Both True then Otherwise False | FALSE |
| OR | (10<20) **OR** (20>10) | Any one true | TRUE |
| NOT | NOT(10<20) | Reverse if true the false> false then true | TRUE |

**Simple Example:**

* **#SQL Operators:**
* **Logical :-**
* **#select\* from studs where S\_Name= 'Priya' and S\_Marks=458; # AND : false not show**
* **select \*from studs where S\_Name= 'Shailu' OR S\_Marks=456; # OR : True then show**
* **select \*from studs where NOT S\_Marks != 456; # NOT : True then show**
* **#select \*from studs;**

1. **SQL Operators -Logical:**

|  |  |
| --- | --- |
| **Comparison Operators** | |
| **Symbol** | **Meaning** |
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| < > or != | Not equal to |

**Simple Example:**

* **#SQL Logical Operators:-**
* #select \*from sregsnum where /\*Cl\_Roll\_No = 82025 and \*/ Cl\_Roll\_No <=82089;
* #select \*from sregsnum where Category= 'GEN';
* select \*from sregsnum where Cl\_Roll\_No = 82080;

1. **SQL Operators -Special (vvi):**

|  |  |
| --- | --- |
| **Special Operators** | |
| **Symbol** | **Meaning** |
| BEETWEEN | Check an attribute value within range |
| LIKE | Check an attributes value matches a given string pattern |
| IS NULL | Checks an attributes value is NULL |
| IN | Check an attributes value match any value within a value list |
| DISTINCT | Limits values to unique values |

**Simple Queries:-**

* **#select \*from sregsnum where Cl\_Roll\_No between 82020 and 82070;**
* **#select \*from sregsnum where Nm\_of\_Stud like "Raja Kumar";**
* **select \*from sregsnum where Nm\_of\_Stud like "A%";**
* **#select \*from sregsnum where Sl\_No =null;**
* **#select \*from sregsnum where Cl\_Roll\_No in (82024, 82068, 82020);**
* **#select distinct(Category) from sregsnum;**
* **select distinct(Cl\_Roll\_No) from sregsnum;**

1. **SQL Operators -Aggregations:**

|  |  |
| --- | --- |
| **Aggregations operators** | |
| **Symbol** | **Meaning** |
| Avg() | Returns the average value from specified columns |
| Count() | Returns number of table rows |
| Max() | Returns largest value among the records |
| Min() | Returns smallest value among the records |
| Sum() | Returns the sum of specified columns values |

**Example of Aggregations :**

* **#select avg(Sl\_No) from sregsnum;**
* **#select avg(Study\_Fees) from sregsnum;**
* **#select count(\*) from sregsnum; # Count total line in table =87**
* **select min(Sl\_No) from sregsnum;**
* **select max(Sl\_No) from sregsnum;**
* **select sum(Study\_Fees) from sregsnum;**

# Chapter-5:- SQL Function

# 1.SQL Group BY Clause

* Arrange identical data into groups.

Example: -

SELECT max(salary), dep\_id from empdata Group by dept\_id;

**Practical Example of students table:**

* **alter table studs add column S\_Branch varchar (50);**
* **update studs set S\_Branch="Sales" where S\_Roll=82000;**
* **select \*from studs;**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S\_Roll** | **S\_Name** | **S\_Marks** | **S\_Fees** | **S\_Branch** |
| **82025** | **Priya** | **458** | **18500** | **Sales** |
| **82089** | **Annu** | **558** | **18500** | **Technical** |
| **82026** | **Priti** | **489** | **18500** | **Technical** |
|  |  |  |  |  |

* **select S\_Name, max(S\_Marks),S\_Branch from studs group by S\_Branch;**
* **Output:**

|  | **S\_Name** | **max(S\_Marks)** | **S\_Branch** |
| --- | --- | --- | --- |
|  | Annu | 458 | Sales |
|  | Priya | 558 | Technical |
|  | Ram | 523 | BCA |

# 2.SQL Group BY Clause

* Having with aggregate functions due to its non-performance in the WHERE clause.
* Must follow the GROUP BY clause in a query and must also precede the ORDER By clause if used.
* Example:
* SELECT AVG(S\_Marks),S\_Branch from Studs Group by S\_Branch HAVING count (S\_Branch)>=2;

Students Table for Practical in MySQL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 82000 | Annu | 458 | 18500 | Sales |
| 82025 | Priya | 456 | 18500 | Technical |
| 82036 | Shailesh | 258 | 18500 | Technical |
| 82038 | Ramesh | 523 | 18500 | BCA |
| 82075 | Rohit | 459 | 18500 | BCA |
| 82085 | Priti | 558 | 18500 | Technical |
|  |  |  |  |  |

* **SELECT S\_Name, avg(S\_Marks),S\_Branch from studs group by S\_Branch having count(S\_Branch)>=3;**
* **Can you try check : avg place of [min or max]**
* **Output:**

|  |  |  |
| --- | --- | --- |
| Priya | 424.0000 | Technical |

# 3.SQL ORDER BY Clause

* Used to sort output of select statement
* Default is to sort in ASC (Ascending)
* Can sort in reverse(Descending) Order with “DESC” after the column name
* Example:
* SELECT \*from studs ORDER BY Marks DESC / ASC;
* Can you please try DESC;

Output: ASC ORDER:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 82000 | Annu | 458 | 18500 | Sales |
| 82085 | Priti | 558 | 18500 | Technical |
| 82025 | Priya | 456 | 18500 | Technical |
| 82038 | Ram | 523 | 18500 | BCA |
| 82075 | Rohit | 459 | 18500 | BCA |
| 82036 | Shailesh | 258 | 18500 | Technical |
|  |  |  |  |  |

GERAL CODE:

#use company;

#delete from employee where emp\_id in(100);

#create table Employee( Emp\_Id int not null, First\_Name varchar(30), Last\_Name varchar(30), Birth\_Date bigint, Sex varchar(10), Salary bigint, Super\_Id int, Branch\_Id int, primary key(Emp\_id));

#insert into employee(Emp\_Id,First\_Name, Last\_Name, Birth\_Date, Sex, Salary, Super\_Id, Branch\_Id)value(100, 'David', 'Wallace','1967','M','250000','100' ,1);

#update employee set Emp\_Id= '100' where first\_name='Wallace';

#update employee set emp\_id='101' where Emp\_Id=100;

#into studs(S\_Roll,S\_Name,S\_Marks,S\_Fees,S\_Branch)value(82075,'Sunny','654','18500','BBA');

#select \*from studs;

#desc studs;

#select S\_Name, avg(S\_Marks),S\_Branch from studs group by S\_Branch;

#SELECT S\_Name, avg(S\_Marks),S\_Branch from studs group by S\_Branch having count(S\_Branch)>=1;

#select \*from studs Order by s\_name asc;

# 4.SQL UNION and NION ALL

## UNION

* Used to combine the result-set-of two or more select statement removing duplicates.
* Each SELECT statement within the UNION must have the same number of columns.
* Selected within the UNION must have the same order in each SELECT statement.
* More than two quires can be clubbed using more than one UNION statement.
* It combines the results set from multiple tables and **returns distinct records** into a single result set.

Example : Create two table Product1 and Product2 (Code)

**Product 1 Table No1:**

**Product2 TableNo2**

|  |  |
| --- | --- |
| **Category\_Id** | **Category\_Name** |
| 1 | Samsung |
| 2 | LG |
| 3 | HP |
| 4 | Dell |
| 5 | Apple |
| 6 | Playstation |
| 8 | Xoaimi |
| 9 | Asus |

|  |  |
| --- | --- |
| **Category\_Id** | **Category\_Name** |
| 1 | Nokia |
| 2 | Samsung |
| 3 | HP |
| 4 | Nikon |
| 6 | Redmi |
| 8 | Vivo |

**Union Command:**

**select category\_name from product1**

**union**

**select category\_name from product2;**

**Output:**

|  |
| --- |
| **Category\_Name** |
| |  | | --- | | Nokia | | Samsung | | HP | | Nikon | | Redmi | | Vivo | | LG | | Dell | | Apple | | Playstation | | Xoaimi | | Asus | |

## Union All:

* Used to combine the results of two SELECT statements including duplicates rows.
* The same rules that apply to the UNION clauses will apply to the UNION ALL operators.
* It combines the result set from multiple tables and **return all records** into a single result set

**Union Syntax:**

**select category\_name from product1**

**union all**

**select category\_name from product2;**

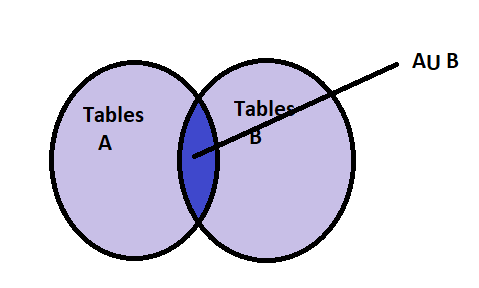
**Output:**

|  |
| --- |
| **Category\_Name** |
| Nokia |
| Samsung |
| HP |
| Nikon |
| Redmi |
| Vivo |
| Samsung |
| LG |
| HP |
| Dell |
| Apple |
| Playstation |
| Xoaimi |
| Asus |

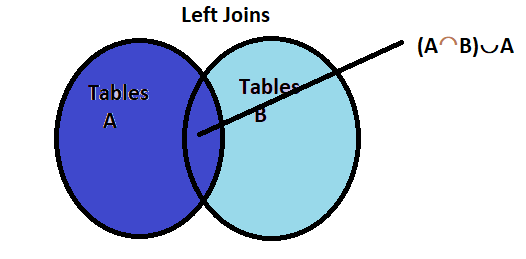
# Chapter-4:- SQL JOINS

Combine rows/columns from two or more tables, based on a related column between them in a databases.

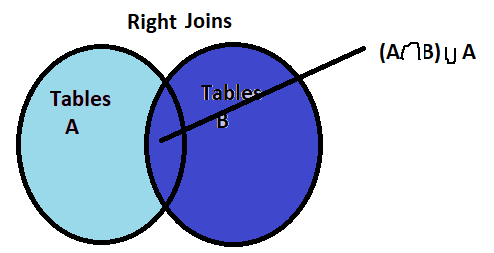
* **INNER JOIN**: Returns rows when there is a match in both tables.



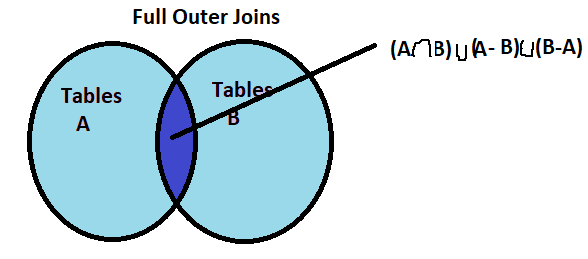
* **LEFT JOIN:** Returns all rows from the left tables, even if there are no matches in the right table.



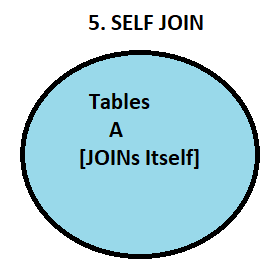
* **RIGHT JOIN :** Returns all rows from the left tables, even if there are no matches in the left table.



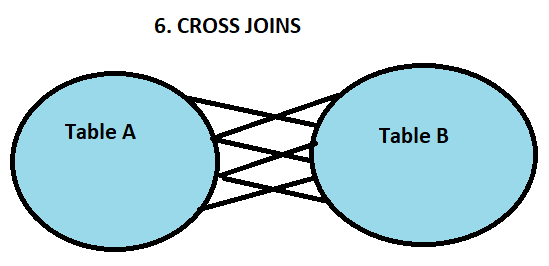
* **FULL OUTER JOIN:** Return rows when there is a match in one of the tables.



* **SELF JOIN:** Used to join a table to itself as if the tables were two tables, temporally at least one tables in the SQL statements.



* **CARTESIAN JOIN[CROSS JOIN] :-** Returns the cartesian product of the set of records from the two or more joined tables.



### **SQL INNER JOIN:**

* The INNER JOINS creates a new results table by combing column values of two tables(table A, table B) based upon the join – predicate.
* The query compares each row of table1 with each rows of table2 to find all pairs of rows which satisfy the join predicate.

**Command of INNER JOIN: -**

**select e.Emp\_id, e.E\_fname,e.E\_lname,**

**d.dept\_id, d. dept\_name**

**from employee e**

**inner join department d**

**on e.dept\_id=d.dept\_id;**

**Example: Create Two Table**

**Table Department Table Employee**

|  |  |  |  |
| --- | --- | --- | --- |
| Dept\_id | Dept\_Name | Manager\_id | Location \_id |
| 10 | IT | 200 | 1700 |
| 11 | Marketing | 201 | 1500 |
| 12 | Resource | 202 | 1800 |
| 13 | Shipping | 203 | 2400 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Emp\_id | E\_Fname | E\_Lname | E\_Salary | D\_i |
| 103 | Harry | Potter | 20000 | 12 |
| 104 | Edwin | Thomas | 15000 | 12 |
| 105 | Steven | Cohen | 10000 | 10 |
| 106 | Erik | John | 12000 | 11 |

**Result of INNER JOIN :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Emp\_id** | **E\_FName** | **E\_LName** | **Dept\_id** | **Dept\_Name** |
| 103 | Harry | Potter | 12 | Resource |
| 104 | Edwin | Thomas | 12 | Resource |
| 105 | Steven | Cohen | 10 | IT |
| 106 | Erik | John | 11 | Marketing |

### **LEFT JOIN:**

The LEFT JOIN return all the values from the left table, plus matched values from the left side table or NULL in case of no matching join predicate.

Example**: Same Example of INNER JOINS:**

**Syntax: -**

**select e.Emp\_id, e.E\_fname,e.E\_lname,e.E\_salary,**

**d.dept\_id, d.dept\_name**

**from employee e**

**left join department d**

**on e.dept\_id=d.dept\_id;**

**Result of LEFT JOIN:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_id** | **E\_FName** | **E\_LName** | **E\_Salary** | **Dept\_Id** | **Dept\_Name** |
| 103 | Harry | Potter | 20000 | 12 | Resource |
| 104 | Edwin | Thomas | 15000 | 12 | Resource |
| 105 | Steven | Cohen | 10000 | 10 | IT |
| 106 | Erik | John | 12000 | 11 | Marketing |

### **RIGHT JOIN:**

The RIGHT JOIN return all the values from the Right table, plus matched values from the left side table or NULL in case of no matching join predicate.

Example**: Same Example of INNER JOINS:**

**Syntax: -**

**select e.Emp\_id, e.E\_fname,e.E\_lname,e.E\_salary,**

**d.dept\_id, d.dept\_name**

**from employee e**

**right join department d # Change this line here**

**on e.dept\_id=d.dept\_id;**

**Result of RIGHT JOIN:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_id** | **E\_FName** | **E\_LName** | **E\_Salary** | **Dept\_Id** | **Dept\_Name** |
| 105 | Steven | Cohen | 10000 | 10 | IT |
| 106 | Erik | John | 12000 | 11 | Marketing |
| 104 | Edwin | Thomas | 15000 | 12 | Resource |
| 103 | Harry | Potter | 20000 | 12 | Resource |
| NULL | NULL | NULL | NULL | 13 | Shipping |

### **4.FULL OUTER JOIN:**

The FULL OUTER JOIN combines the results of both left and right outer joins. The Joined tables contains will all records from both the tables and fill in NULLs for missing matches on either side.

Example**: Same Example of INNER JOINS:**

**Syntax: -**

**select e.Emp\_id, e.E\_fname,e.E\_lname,e.E\_salary,**

**d.dept\_id, d.dept\_name**

**from employee e**

**left join department d** **#step 1 add left join**

**on e.dept\_id=d.dept\_id**

**union #step 2 add union methods**

**select e.emp\_id,e.e\_fname, e.e\_lname, e.E\_salary, d.dept\_id, d.dept\_name**

**from employee e**

**right join department d #step 3 add right join**

**on e.dept\_id=d.dept\_id;**

**Result of FULL OUTER :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_id** | **E\_FName** | **E\_LName** | **E\_Salary** | **Dept\_Id** | **Dept\_Name** |
| 103 | Harry | Potter | 20000 | 12 | Resource |
| 104 | Edwin | Thomas | 15000 | 12 | Resource |
| 105 | Steven | Cohen | 10000 | 10 | IT |
| 106 | Erik | John | 12000 | 11 | Marketing |
| **NULL** | **NULL** | **NULL** | **Null** | 13 | Shipping |

### **CROSS JOIN SQL**

**Definition :** Assume there are 4 records in table1 and 3 records in table 2 find cross joins methods

**Syntax:**

Select \*from table1 CROSS JOIN table2;

**Table1** **Table2**

|  |
| --- |
| **Alphabetic** |
| A |
| B |
| C |
| D |

|  |
| --- |
| **Numerical** |
| 1 |
| 2 |
| 4 |

Explanation of Cross Join : It is just like that multiple into 1 to all (SET- THEORY).

**OUTPUT: Table 1\* Table2**

|  |  |
| --- | --- |
| A | 1 |
| A | 2 |
| A | 3 |
| B | 1 |
| B | 2 |
| B | 3 |
| C | 1 |
| C | 2 |
| C | 3 |
| D | 1 |
| D | 2 |
| D | 3 |

Example : Lets see an Example of Employee and Department value of CROSS JOIN

Command : **select \*from department cross join employee;**

**OUTPUT:-**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Dept\_id** | **Dept\_Name** | **Manager\_id** | **Location \_id** | **Emp\_id** | **E\_Fname** | **E\_Lname** | **E\_Salary** | **EDept\_id** |
| 13 | Shipping | 203 | 2400 | 103 | Harry | Potter | 20000 | 12 |
| 12 | Resource | 202 | 1800 | 103 | Harry | Potter | 20000 | 12 |
| 11 | Marketing | 201 | 1500 | 103 | Harry | Potter | 20000 | 12 |
| 10 | IT | 200 | 1700 | 103 | Harry | Potter | 20000 | 12 |
| 13 | Shipping | 203 | 2400 | 104 | Edwin | Thomas | 15000 | 12 |
| 12 | Resource | 202 | 1800 | 104 | Edwin | Thomas | 15000 | 12 |
| 11 | Marketing | 201 | 1500 | 104 | Edwin | Thomas | 15000 | 12 |
| 10 | IT | 200 | 1700 | 104 | Edwin | Thomas | 15000 | 12 |
| 13 | Shipping | 203 | 2400 | 105 | Steven | Cohen | 10000 | 10 |
| 12 | Resource | 202 | 1800 | 105 | Steven | Cohen | 10000 | 10 |
| 11 | Marketing | 201 | 1500 | 105 | Steven | Cohen | 10000 | 10 |
| 10 | IT | 200 | 1700 | 105 | Steven | Cohen | 10000 | 10 |
| 13 | Shipping | 203 | 2400 | 106 | Erik | John | 12000 | 11 |
| 12 | Resource | 202 | 1800 | 106 | Erik | John | 12000 | 11 |
| 11 | Marketing | 201 | 1500 | 106 | Erik | John | 12000 | 11 |
| 10 | IT | 200 | 1700 | 106 | Erik | John | 12000 | 11 |

**THE END**

**LAB SESSIONS**

**(Part 1)**

**DataBases Used: - cricket1.csv, cricket.csv**

**------**cricket\_1 is the table for cricket test match1.

------cricket\_2 is the table for cricket test match2.

**SOLUTIONS**

**Before find all LAB SESSIONS QUESTIONS, Create two table (Create\_1 and Create\_2) then find.**

* Q1. Find all the players who were present in the test match 1 or test match 2.
* **Syntax:**

**Select \*from Cricket\_1**

**UNION**

**Select \*from Cricket\_2**

* Q2. Write a MySQL query to find the players from the test match1 having popularity higher than the average popularity.
* **Syntax:**

**Select player\_name, popularity from cricket\_1 WHERE popularity> (SELECT AVG(popularity )from Cricket\_1);**

**Command:**

* Q3. Find player\_id and player\_name that are common in the test match 1 and test match 2.
* **Syntax:**

**Select player\_id, player\_name from cricket\_1**

**WHERE cricket\_1.player\_id In (Select player\_id from Cricket\_2);**

* Q4. Retrive player\_id, runs, and player\_name from cricket\_1 table and display list of the players where the runs are more than the average runs.
* **Syntax:**

**Select player\_id, runs, player\_name from cricket\_1 WHERE run>(SELECT avg(runs) from cricket\_1);**

* Q5. Write a query to extract the player\_id, runs, and Player\_name from the table “cricket\_1” where the runs are greater than 50.
* **Syntax:**

**Select player\_id, run , player\_name from cricket\_1, where runs>50;**

* **Q6.** Write a query **to** extract all the columns from cricket\_1, where player\_name starts with “y” and ends with “v”.
* **Syntax:**

**Select \*from cricket\_1 WHERE player\_name LIKE ‘y%v’;**

* Q7. Write a query **to** extract all the columns from cricket\_1, where player\_name does not end with “t”
* **Syntax:**

**Select \*from cricket\_1 WHERE player\_name NOT LIKE ‘%t’;**

* **Q8.** Extract the player\_id and player\_name of the player where the charisma value is NULL.
* **Syntax:**

**SELECT player\_id, player\_name from new\_cricket WHERE charisma IS NULL;**

* **Q9.** Separate the all player\_id into single numeric ids(example PL1=1)
* **Syntax:**

SELECT player\_id, SUBSTR(player\_id,3) from new\_cricket;

* **Q10.** Write a MYSQL query to extract player\_id, player\_name and charisma where the charisma is greater than 20.
* **Syntax:**

SELECT player\_id. Player\_name, charisma from new\_cricket where charisma>25;

**Command all Lab Sessions:**

**#select \*from cricket\_1;**

**#select \*from cricket\_2;**

**#Q1.**

**/\***

**select \*from cricket\_1**

**union**

**select \*from cricket\_2;**

**select player\_name from cricket\_1**

**union**

**select player\_name from cricket\_2;**

**\*/**

**#Q2.**

**#select player\_name, popularity from cricket\_1 where popularity>( select avg(popularity) from cricket\_1);**

**#Q3.**

**#select player\_id, player\_name from cricket\_1 where cricket\_1.player\_id in (select player\_id from cricket\_2);**

**#Q4**

**#select player\_id, runs, player\_name from cricket\_1 where runs>(select avg(runs) from cricket\_1);**

**#Q5**

**#select player\_id,runs,player\_name from cricket\_1 where runs>50;**

**#Q6**

**#select \*from cricket\_1 where player\_name like 'y%v'; #yadav**

**#select \*from cricket\_1 where player\_name like 's%h'; #shailesh**

**#Q7**

**#select \*from cricket\_1 where player\_name not like '%t';**

**#Q8**

**#select player\_id, player\_name from new\_cricket where charisma is Null;**

**#Q9**

**#SELECT player\_id, SUBSTR(player\_id,3) from new\_cricket;**

**#Q10**

**select \*from new\_cricket where charisma >25;**

**(Part 1)**

**Question 1: Create a Databases bank.**

**Solution :** CREATE DATABASES BANK;

Use Bank;

**Question 2: Create a table with the name “bank\_details” with the following columns.**

* **Product** with string data types
* **Quantity** with real number data type
* **Price** with real number data type
* **Purchase\_cost** with decimal data type
* **Estimated\_sale\_price** with data type float

**Solution:**

create table bank\_details(

Product char(10),

Quantity int,

Price real,

Purchase\_cost decimal(6,2),

Estimated\_sale\_price float

);

**Question 3: Display all columns and their datatypes and size in Bank\_Details**

**Solution :**

**Describe bank\_details;**

**Question 4: Insert two records into bank\_details.**

**1st records with values:**

**Product: PayCard**

**Quantity: 3**

**Price: 330**

**Purchase\_Cost: 8008**

**Estimated\_Sale\_Price: 9090**

**Product: PayPoints**

**Quantity: 4**

**Price: 200**

**Purchase\_Cost: 8000**

**Estimated\_Sale\_Price: 6800**

**Solution :**

**insert into bank\_details value('paycard',3,330,8080,9090);**

**insert into bank\_details value('paypoint','4','200',6800,8985);**

**select \*from bank\_details;**

**Question 5: Add a column Locations to the existing Bank\_Details tables with data type varchar and size(20)**

**Solution:**

**alter table bank\_details add column Location varchar(20);**

**Question 6: What is the value of Location for product : “PayCard”**

**Solution:**

**select location from bank\_details where product='PayCard';**

**Question 7: How many characters does the product: “PayCard” have in the Bank\_details table.**

**Solution:**

**SELECT char\_length(product) from bank\_details where product=”PayCard”;**

**Question 8: Alter the product filed from char to varchar in bank\_details.**

**Solution:**

**Alter table bank\_details modify product varchar(10);**

**Questions 9: Reduce the size of the product filed from 10 to 6 check if it possible**

**Solution:**

**Alter table bank\_details modify product varchar(6);**

**#Error because char value max=8,9 so you can’t change the varchar value in (6).**

**(Part 3)**

**Question10. : Create a table named as a bank\_holidays with belows fileds;**

* **Holidays fields which displays only date**
* **Start\_time fiels which displays hours and minutes**
* **End\_time fields also displays hours and minutes and timezone**

**Solution:**

**create table bank\_Holiday(**

**Holiday time,**

**Start\_time datetime,**

**End\_time timestamp**

**);**

**Questions 11:**

* **Step 1: insert today’s date details in all fields in bank\_Holidays.**
* **Step 2: After step 1, perform the below**
* **Step 3: Postpone Holidays to next day by updating the Holidays Fields**

**Solution :**

**Step 1:**

**insert into bank\_holiday value(current\_date(),**

**current\_date(),**

**current\_date()**

**);**

**Step 2:**

**update bank\_holidays set holiday= date\_add(holiday, interval 1 day);**

**Question 12: Update the End\_time with current European time.**

**Solution:**

**Update bank\_holiday set end\_time =utc\_timestamp();**

**Question 13: Display output of product filed as new\_products in bank\_details table**

**Solution :**

**select product as new\_product from bank\_details;**

**Question 14: Display only one records from bank\_details**

**Solution :**

**select \*from bank\_details limit 2;**

**Question 15: Display the first live characters of the Location of Bank\_details;**

**select substr(Location, 1, 5) from bank\_details;**

**Part 2 and Part 3 All Command ;**

/\*use bank;

create table bank\_details(

Product char(10),

Quantity int,

Price real,

Purchase\_cost decimal(6,2),

Estimated\_sale\_price float

);

\*/

#insert into bank\_details value('paycard',3,330,8080,9090);

#insert into bank\_details value('paypoint','4','200',6800,8985);

#alter table bank\_details add column Location varchar(20);

#select location from bank\_details where product='PayCard';

#SELECT char\_length(product) from bank\_details where product="PayCard";

#alter table bank\_details modify product varchar(10);

#Alter table bank\_details modify product varchar(6);

/\*

create table bank\_Holiday(

Holiday date,

Start\_time datetime,

End\_time timestamp

);

\*/

/\*

insert into bank\_Holiday value(

current\_date(),

current\_date(),

current\_date()

);

\*/

#update bank\_holidays set holiday= date\_add(holiday, interval 1 day);

#update bank\_holiday set end\_time =utc\_timestamp();

#select product as new\_product from bank\_details;

#select \*from bank\_details limit 2;

**select substr(Location, 1, 5) from bank\_details;**

**#select \*from bank\_holiday;**

**THE END**

**Class Topics**

**Important Questions**

**Q1. What are the advantages and disadvantages of DBMS?**

**Ans:**

**Advantage of DBMS**

* **Controls Databases redundancy:** All the data is stored in one place, and that recorded in the databases (MySQL) and hence controls the redundancy in the database.
* **Data Sharing:** DBMS allows users with authority to share the data in the database with multiple users.
* **Easy Maintenance:** The centralized nature of the databases helps in the easy maintenance of the data.
* **Reduce Time:** : It reduces the maintenance need and development time.
* **Backup:** It automatically backs up data to maintain its integrity in case of failure.
* **Multiple User Interface:** It offers a number of user interfaces to multiple users.

**Disadvantage of DBMS**

* **Cost of Software and hardware:** It requires a number of highpowered processors and large size memory to run DBMS.
* **Size:** A large amount of storage size is required to run DBMS efficiently.
* **Complexity :** DBMS adds an additional layer complexity to the data.
* **Higher Impact of Failure:** DBMS faces a higher risk of losing the data since all the data is stored at a single location and a catastrophic failure can wipe it all.

**Q2. Explain the various keys in DBMS?**

**Ans:**

**The different types of keys in DMBS are:-**

* **Candidate Key:** The candidate keys in a table are defined as the set of keys that is minimal and can uniquely identify any data row in the table.
* **Primary Key:** - The primary key is selected from one of the candidate keys and becomes the identifying key of a table. It can uniquely identify any data row of the table.
* **Super Key:** - Super Key is the superset of primary key. The super key contains a set of attributes, including the primary key, which can uniquely identify any data row in the table.
* **Composite Key:** - If any single attribute of a table is not capable of being the key i.e. it cannot identify a row uniquely, then we combine two or more attributes to form a key, this is known as a composite key.
* **Secondary Key: -** Only one of the candidate keys is selected as the primary key. The rest of them are known as secondary keys.
* **Foreign Key**: - A foreign key is an attribute value in a table that acts as the primary key in another table. Hence, the foreign key is useful in linking together two tables. Data should be entered in the foreign key column with great care, as wrongly entered data can invalidate the relationship between the two tables.
* **EXAMPLES:**

**Q3. What are advantage of file processing systems over DBMS.**

**Ans:**

A “file system” is the structure and logic rules that are used to organize groups of data and their names. Files can be collectively present inside a directory. Directories can again be present inside another directory giving it a hierarchical structure. Although Windows supports a variety of file systems, NTFS is the most popular in today’s world. Because files are commonly organized in a hierarchy, the file system allows you to browse the files in the current directory.

**Following are some features of the File Systems:**

* **Space Management** – File systems allot space on a device in variable block sizes, usually several physical units. The file system is in charge of organizing files and directories, as well as keeping track of which media areas belong to which file and which do not.
  + When a file is not an exact multiple of the allocation unit, this results in unused space known as slack space. The average unused space for a 512-byte allotment is 256 bytes. The average unused space for 64 KB clusters is 32 KB. When a file system is constructed, the size of the allocation unit is chosen. The quantity of unused space can be reduced by determining the allocation size based on the average size of the files expected to be in the file system.
* **Hierarchical Data Management** – Directories in file systems allow users to organise files into different groupings. This can be accomplished by linking the file name to a table of contents index or an node in a Unix-like file system. Flat (i.e. linear) directory structures are possible, as are hierarchies in which directories can have subdirectories.
* **Metadata Management –** Some extra information pertaining to each file is stored in a file system. A file system keeps all of the metadata connected with a file distinct from the contents of the file, such as the file name, the length of the contents of a file, and the file’s location in the folder hierarchy.
  + The names of all the files in a directory are usually recorded in a single place—the directory table for that directory—which is generally stored like any other file. Many file systems save only a portion of a file’s metadata in the directory table, while the rest is stored in an entirely distinct structure, such as the node.
* **Integrity Management –** One of a file system’s most important responsibilities is to ensure that the file system structures in secondary storage stay consistent regardless of how programs access the file system. This comprises measures executed if a program that modifies the file system stops abnormally or fails to notify the file system that its tasks have been finished. This could include updating metadata, directory entries, and any data that has been buffered but not yet updated on actual storage media.
* **Restricting access to Files –** To control data access, file systems employ a number of approaches. The goal is usually to block a user or group of users from accessing or changing files. Another reason is to ensure that data is modified in a controlled manner, allowing access to a specific program to be restricted. Passwords saved in the file’s metadata or elsewhere, as well as file permissions in the form of permission bits, access control lists, or capabilities, are examples.