**SE - Module: 5 (Database)**

1. Why do you understand By **Database**?

* A database is information that is set up for easy access, management and updating. Computer databases typically store aggregations of data records or files that contain information, such as sales transactions, customer data, financials and product information.

1. What is **Normalization**?

* Normalization is the process of minimizing redundancy (duplicity) from a relation or set of relations.
* Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations.
* Most Commonly used normal forms:

**First Normal Form:**

* First normal form(1NF) Second normal form(2NF) Third normal form(3NF) Boyce & Code normal form (BCNF)
* If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute.
* A relation is in first normal form if every attribute in that relation is singled valued attribute.

**Second Normal Form:**

* To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency.
* relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
* Partial Dependency –If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

**Third Normal Form:**

* A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.
* A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y
* X is a super key.
* Y is a prime attribute (each element of Y is part of some candidate key).
* Transitive dependency –If A->B and B->C are two FDs then A->C is called transitive dependency.

1. What is Difference between **DBMS** and **RDBMS**?

| **DBMS** | **RDBMS** |
| --- | --- |
| DBMS stores data as file. | [RDBMS](https://www.geeksforgeeks.org/rdbms-architecture/) stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present. | Normalization is present. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |
| It stores data in either a navigational or hierarchical form. | It uses a tabular structure where the headers are the column names, and the rows contain corresponding values. |
| It deals with small quantity of data. | It deals with large amount of data. |
| Data redundancy is common in this model. | Keys and indexes do not allow Data redundancy. |
| It is used for small organization and deal with small data. | It is used to handle large amount of data. |
| Not all Codd rules are satisfied. | All 12 Codd rules are satisfied. |
| Security is less | More security measures provided. |
| It supports single user. | It supports multiple users. |
| Data fetching is slower for the large amount of data. | Data fetching is fast because of relational approach. |
| The data in a DBMS is subject to low security levels with regards to data manipulation. | There exists multiple levels of data security in a RDBMS. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| Examples:[XML](https://www.geeksforgeeks.org/xml-basics/), Window Registry, Forxpro, dbaseIIIplus etc. | Examples: [MySQL](https://www.geeksforgeeks.org/architecture-of-mysql/), [PostgreSQL](https://www.geeksforgeeks.org/what-is-postgresql-introduction/), SQL Server, Oracle, Microsoft Access etc. |

1. What do you understand By Data **Redundancy**?

* In DBMS, when the same data is stored in different tables, it causes data redundancy.
* Sometimes, it is done on purpose for recovery or backup of data, faster access of data, or updating data easily. Redundant data costs extra money, demands higher storage capacity, and requires extra effort to keep all the files up to date.
* Sometimes, unintentional duplicity of data causes a problem for the database to work properly, or it may become harder for the end user to access data. Redundant data unnecessarily occupy space in the database to save identical copies, which leads to space constraints, which is one of the major problems.

1. What is **DDL** in Interpreter?

* DDL Interpreter DDL expands to **Data Definition Language**. DDL Interpreter as the name suggests interprets the DDL statements such as schema definition statements like create, delete, etc. The result of this interpretation is a set of a table that contains the meta-data which is stored in the data dictionary.
* Create :- Create a new table, a view of table or other object in database.

Alter :- Modifies an existing database object, such as a table.

Delete:- Delete an entire table, a view of table or other object in database.

1. What is DML Compiler in SQL?

* DML is an abbreviation of **Data Manipulation Language**.
* The DML commands in Structured Query Language change the data present in the SQL database. We can easily access, store, modify, update and delete the existing records from the database using DML commands.
* Insert:- Creates a record
* Update:- Modifies records
* Delete:- Delete records

1. What is **SQL Key Constraints** writing an Example of SQL Key constraints?

* SQL constraints are used to specify rules for the data in a 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.
* Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
* The following constraints are commonly used in SQL:
* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly.

1. What is save **Point**? How to Create Point write a Query?

* A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
* The syntax for a SAVEPOINT command is as shown below.
* SAVEPOINT SAVEPOINT\_NAME;
* This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

1. What is **trigger** and how to create a Trigger in SQL?

* A trigger is a set of SQL statements that reside in system memory with unique names. It is a specialized category of stored procedure that is called automatically when a database server event occurs. Each trigger is always associated with a table.
* A **trigger is called a special procedure** because it cannot be called directly like a stored procedure. The key distinction between the trigger and procedure is that a trigger is called automatically when a data modification event occurs against a table. A stored procedure, on the other hand, must be invoked directly.
* Syntax:**CREATE** **TRIGGER** **schema**.trigger\_name  **ON** table\_name  **AFTER**  {**INSERT**, **UPDATE**, **DELETE**}  [NOT **FOR** REPLICATION]  **AS**  {SQL\_Statements} .

**Task**

1. **Create Table Name : Student and Exam**

* **SQL Query :-**
* **create** table student(Rollno int AUTO\_INCREMENT,

Name varchar(100),

Branch varchar(100),

PRIMARY KEY(Rollno)

);

**Table structure for table student**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Null** | **Default** |
| Rollno | int(11) | No |  |
| Name | varchar(100) | Yes | NULL |
| Branch | varchar(100) | Yes | NULL |

* **insert** into student(Name,Branch)

values("Jay","Computer Science"),

("Suhani","Electronic and Com"),

("Kriti","Electronic and Com")

);

## Dumping data for table student

|  |  |  |
| --- | --- | --- |
| **Rollno** | **Name** | **Branch** |
| 1 | Jay | Computer Science |
| 2 | Suhani | Electronic and Com |
| 3 | Kriti | Electronic and Com |

* **create** table Exam(Rollno int,

S\_code varchar(100),

Marks int,

P\_code varchar(100),

FOREIGN KEY(Rollno) REFERENCES student(Rollno)

);

## **Table structure for table exam**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Null** | **Default** |
| Rollno | int(11) | Yes | NULL |
| S\_code | varchar(100) | Yes | NULL |
| Marks | int(11) | Yes | NULL |
| P\_code | varchar(100) | Yes | NULL |

* **INSERT** INTO exam(Rollno, S\_code,Marks,P\_code)

VALUES (1,"CS11",50,"CS"),

(1,"CS12",60,"CS"),

(2,"EC101",66,"EC"),

(2,"EC102",70,"EC"),

(3,"EC101",45,"EC"),

(3,"EC102",50,"EC");

## **Dumping data for table exam**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rollno** | **S\_code** | **Marks** | **P\_code** |
| 1 | CS11 | 50 | CS |
| 1 | CS12 | 60 | CS |
| 2 | EC101 | 66 | EC |
| 2 | EC102 | 70 | EC |
| 3 | EC101 | 45 | EC |
| 3 | EC102 | 50 | EC |

1. Create table given below: **Employee and Incentive Table**

* **SQL Query :-**
* **CREATE** TABLE Employee(Employee\_id int AUTO\_INCREMENT,

First\_name varchar(100),

Last\_name varchar(100),

Salary int,

Joining\_date varchar(100),

Department varchar(100),

PRIMARY KEY(Employee\_id)

);

**Table structure for table employee**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Null** | **Default** |
| Employee\_id | int(11) | No |  |
| First\_name | varchar(100) | Yes | NULL |
| Last\_name | varchar(100) | Yes | NULL |
| Salary | int(11) | Yes | NULL |
| Joining\_date | varchar(100) | Yes | NULL |
| Department | varchar(100) | Yes | NULL |

* **INSERT** Into employee

(First\_name,Last\_name,Salary,Joining\_date,Department)

VALUES("John","Abraham",1000000,"01-JAN-13 12.00.00 AM","Banking"),

("Michael","Clarke",800000,"01-JAN-13 12.00.00 AM","Insurance"),

("Roy","Thomas",700000,"01-FEB-13 12.00.00 AM","Banking"),

("Tom","Jose",600000,"01-FEB-13 12.00.00 AM","Insurance"),

("Jerry","Pinto",650000,"01-FEB-13 12.00.00 AM","Insurance"),

("Philip","Mathew",750000,"01-JAN-13 12.00.00 AM","Services"),

("TestName1","123",650000,"01-JAN-13 12.00.00 AM","Services"),

("TestName2","Lname%",600000,"01-FEB-13 12.00.00 AM","Insurance");

**Dumping data for table employee**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Employee\_id** | **First\_name** | **Last\_name** | **Salary** | **Joining\_date** | **Department** |
| 1 | John | Abraham | 1000000 | 01-JAN-13  12.00.00  AM | Banking |
| 2 | Michael | Clarke | 800000 | 01-JAN-13  12.00.00  AM | Insurance |
| 3 | Roy | Thomas | 700000 | 01-FEB-13  12.00.00  AM | Banking |
| 4 | Tom | Jose | 600000 | 01-FEB-13  12.00.00  AM | Insurance |
| 5 | Jerry | Pinto | 650000 | 01-FEB-13  12.00.00  AM | Insurance |
| 6 | Philip | Mathew | 750000 | 01-JAN-13  12.00.00  AM | Services |
| 7 | TestName1 | 123 | 650000 | 01-JAN-13  12.00.00  AM | Services |
| 8 | TestName2 | Lname% | 600000 | 01-FEB-13  12.00.00  AM | Insurance |

* **CREATE** table Incentive(Employee\_ref\_id int,

Incentive\_date varchar(100),

Incentive\_amount int

);

**Table structure for table incentive**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Null** | **Default** |
| Employee\_ref\_id | int(11) | Yes | NULL |
| Incentive\_date | varchar(100) | Yes | NULL |
| Incentive\_amount | int(11) | Yes | NULL |

* **INSERT** INTO incentive

(Employee\_ref\_id,Incentive\_date,Incentive\_amount)

VALUES(1,"01-FEB-13",5000),

(2,"01-FEB-13",3000),

(3,"01-FEB-13",4000),

(1,"01-JAN-13",4500),

(2,"01-JAN-13",3500);

**Dumping data for table incentive**

|  |  |  |
| --- | --- | --- |
| **Employee\_ref\_id** | **Incentive\_date** | **Incentive\_amount** |
| 1 | 01-FEB-13 | 5000 |
| 2 | 01-FEB-13 | 3000 |
| 3 | 01-FEB-13 | 4000 |
| 1 | 01-JAN-13 | 4500 |
| 2 | 01-JAN-13 | 3500 |

* **Solve sql query**

1. Get First\_Name from employee table using Tom name “Employee Name”.

* **SELECT** First\_name FROM employee WHERE First\_name="Tom";

1. Get FIRST\_NAME, Joining Date, and Salary from employee table.

* **SELECT** First\_name, Joining\_date, Salary FROM employee;

1. Get all employee details from the employee table order by First\_Name Ascending and Salary descending?

* **SELECT** First\_name, Salary FROM employee order BY First\_name ASC ,salary DESC;

d) Get employee details from employee table whose first name contains ‘J’.

* **SELECT** First\_name FROM employee where First\_name LIKE "j%";

e) Get department wise maximum salary from employee table order by

salary ascending?

* **SELECT** Department, max(Salary) FROM employee GROUP BY Department ORDER BY Salary;

f) Select first\_name, incentive amount from employee and incentives table for those employees who have incentives and incentive amount greater than 3000

* **SELECT** employee.First\_name, incentive.Incentive\_amount FROM employee INNER JOIN incentive ON employee.Employee\_id=incentive.Employee\_ref\_id AND incentive\_amount>3000;

1. Create table given below: **Salesperson and Customer**

* **CREATE** TABLE salesperson(`(PK)SNo` int,

SNAME varchar(100),

CITY varchar(100),

COMM float

);

**Table structure for table salesperson**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Null** | **Default** |
| (PK)SNo | int(11) | Yes | NULL |
| SNAME | varchar(100) | Yes | NULL |
| CITY | varchar(100) | Yes | NULL |
| COMM | float | Yes | NULL |

* **INSERT** INTO salesperson(`(PK)SNo`,SNAME,CITY,COMM)

VALUES(1001,"Peel","London",.12),

(1002,"Serres","San jose",.13),

(1004,"Motika","London",.11),

(1007,"Rafkin","Barcelona",.15),

(1003,"Axelrod","New York",.1);

**Dumping data for table salesperson**

|  |  |  |  |
| --- | --- | --- | --- |
| **(PK)SNo** | **SNAME** | **CITY** | **COMM** |
| 1001 | Peel | London | 0.12 |
| 1002 | Serres | San jose | 0.13 |
| 1004 | Motika | London | 0.11 |
| 1007 | Rafkin | Barcelona | 0.15 |
| 1003 | Axelrod | New York | 0.1 |

* **CREATE** TABLE Customer(`(PK)CNM.` int,

CNAME varchar(100),

CITY varchar(100),

RATING int,

`(FK)SNo` int

);

**Table structure for table customer**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Null** | **Default** |
| (PK)CNM. | int(11) | Yes | NULL |
| CNAME | varchar(100) | Yes | NULL |
| CITY | varchar(100) | Yes | NULL |
| RATING | int(11) | Yes | NULL |
| (FK)SNo | int(11) | Yes | NULL |

* **INSERT** INTO customer (`(PK)CNM.`,CNAME,CITY,RATING,`(FK)SNo`)

VALUES(201,"Hoffman","London",100,1001),

(202,"Giovanne","Roe",200,1003),

(203,"Liu","San Jose",300,1002),

(204,"Grass","Barcelona",100,1002),

(206,"Clemens","London",300,1007),

(207,"Pereira","Roe",100,1004);

**Dumping data for table customer**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **(PK)CNM.** | **CNAME** | **CITY** | **RATING** | **(FK)SNo** |
| 201 | Hoffman | London | 100 | 1001 |
| 202 | Giovanne | Roe | 200 | 1003 |
| 203 | Liu | San Jose | 300 | 1002 |
| 204 | Grass | Barcelona | 100 | 1002 |
| 206 | Clemens | London | 300 | 1007 |
| 207 | Pereira | Roe | 100 | 1004 |

* **Solve sql query**

1. Names and cities of all salespeople in London with commission above 0.12

* **SELECT** SNAME,CITY FROM salesperson WHERE CITY="London" AND COMM>0.12;

1. All salespeople either in Barcelona or in London

* **SELECT** \*FROM salesperson WHERE CITY IN ("Barcelona","London");

c) All salespeople with commission between 0.10 and 0.12. (Boundary values should be excluded).

* **SELECT** \* FROM salesperson WHERE COMM BETWEEN 0.10 AND 0.12;

d) All customers excluding those with rating <= 100 unless they are located in Rome

* **SELECT** \*FROM customer WHERE RATING <=100 OR CITY="Rome";