Module – 5 Database

Q.1 What do you understand By Database.

A database is an organized collection of data, typically stored and accessed electronically. Databases are designed to efficiently manage, store, retrieve, and update data. Here are key aspects and concepts related to databases:

1. **Data Structure**: Databases organize data into tables, which consist of rows and columns. Each row represents a record, and each column represents a field within the record. This structure allows for efficient data retrieval and manipulation.
2. **Database Management System (DBMS)**: This is software that interacts with the user, applications, and the database itself to capture and analyze data. Examples include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.
3. **SQL (Structured Query Language)**: The standard language used to communicate with relational databases. SQL commands are used to perform tasks such as querying data, updating records, and managing database structures.
4. **Types of Databases**:
    - **Relational Databases**: Use tables to represent data and relationships between data. Examples include MySQL, PostgreSQL, and SQLite.
    - **NoSQL Databases**: Designed for unstructured data and often used for large-scale data storage needs. Examples include MongoDB, Cassandra, and Redis.
    - **In-Memory Databases**: Store data in memory for faster access. Examples include Redis and Memcached.

    - **Graph Databases**: Designed to store data in graph structures to represent relationships. Examples include Neo4j and ArangoDB.
5. **Database Models**:
    - **Hierarchical**: Data is organized into a tree-like structure.
    - **Network**: Data is organized in a graph, allowing for more complex relationships.
    - **Relational**: Data is organized in tables, and relationships are defined through foreign keys.
    - **Object-Oriented**: Data is stored in objects, similar to object-oriented programming.
6. **Transactions and ACID Properties**:
    - **Atomicity**: Ensures that all parts of a transaction are completed successfully or none are.
    - **Consistency**: Ensures that a transaction brings the database from one valid state to another.
    - **Isolation**: Ensures that transactions do not interfere with each other.
    - **Durability**: Ensures that the results of a transaction are permanently stored in the database.
7. **Indexes**: Structures that improve the speed of data retrieval operations on a database table at the cost of additional storage space.
8. **Backup and Recovery**: Mechanisms to ensure data can be restored in case of a failure.
9. **Security**: Measures to protect data from unauthorized access and ensure data integrity.

Q.2 What is Normalization?

Ans. Normalization refers to a set of techniques used in various fields to ensure data consistency, efficiency, and integrity. Here's an overview of normalization in different contexts:

1. Database Normalization:

In the context of relational databases, normalization is the process of organizing data to reduce redundancy and improve data integrity. It involves decomposing a database into smaller tables and defining relationships between them to eliminate redundancy and dependency.

Q.3 What is Difference between DBMS and RDBMS?

Ans. Database Management System (DBMS) and Relational Database Management System (RDBMS) are both systems used for managing databases, but they have distinct characteristics and functionalities. Here's a detailed comparison:

A database is a collection of organised or arranged data that can be easily accessed, updated/ modified or controlled. Information within the database can be easily placed into rows and columns, or tables. Meanwhile, database management enables the user to store, manage and access data. Knowing the definitions and the difference between relational database management system and database management system will help candidates to prepare well

Q.4 What do you understand By Data Redundancy?

Ans. Rules for RDBMS

1. **Information Rule:**

- All information in a relational database is represented explicitly at the logical level and in exactly one way: by values in tables.

2. **Guaranteed Access Rule:**
   - Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.
3. **Systematic Treatment of Null Values:**
   - Null values (distinct from empty character strings or a string of blanks and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.
4. **Dynamic Online Catalog Based on the Relational Model:**
   - The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to the regular data.
5. **Comprehensive Data Sublanguage Rule:**
   - A relational system may support several languages and various modes of terminal use (for example, the fill-in-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all the following items:
     - Data definition
     - View definition
     - Data manipulation (interactive and by program)
     - Integrity constraints
     - Authorization
     - Transaction boundaries (begin, commit, and rollback)
6. **View Updating Rule:**

- All views that are theoretically updateable must be updateable by the system.

7. **High-Level Insert, Update, and Delete:**
   - The system must support set-at-a-time insert, update, and delete operators for updating the database.
8. **Physical Data Independence:**
   - Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.
9. **Logical Data Independence:**
   - Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.
10. **Integrity Independence:**
    - Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.
11. **Distribution Independence:**
    - The data manipulation sublanguage of a relational DBMS must enable application programs to remain logically unimpaired whether and whenever data are physically centralized or distributed.
12. **Non-Subversion Rule:**
    - If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level relational language.

Q.5 What do you understand By Data Redundancy?

Ans. **Data redundancy** refers to the unnecessary duplication of data within a database or data storage system. It can occur intentionally for backups and performance improvements, or

unintentionally due to poor database design and manual entry errors.

1. **Poor Database Design:** Lack of proper normalization.
2. **Lack of Coordination:** Different departments maintaining separate databases.
3. **Manual Data Entry:** Human errors leading to duplicate entries.
4. **Legacy Systems:** Integration of older systems with newer ones.

Problems

1. **Increased Storage Costs:** More disk space is consumed.
2. **Data Inconsistency:** Conflicting information due to updates in one place but not another.
3. **Data Integrity Issues:** Challenges in ensuring data accuracy.
4. **Maintenance Overhead:** More effort required for backups, updates, and synchronization.
5. **Decreased Performance:** Slower query performance due to multiple data copies.

Prevention

1. **Normalization:** Organizing data into tables to minimize redundancy.
2. **Best Practices:** Using primary keys, foreign keys, and indexing.
3. **Data Integration:** Consolidating data from different sources.
4. **Automated Data Entry:** Reducing manual input errors.
5. **Regular Audits:** Identifying and eliminating redundant data.

Q.6 What is DDL Interpreter?

Ans. **Data Definition Language** The query processor has the following components. DDL Interpreter - DDL stands for Data Definition Language. The DDL interpreter changes the DDL statements into a specific format to make sense to the storage manager. The DDL also ensures the consistency and validity of the database.

Q.7 What is DML Compiler in SQL?

- Ans. **Data Manipulation Language (DML)** The Data Manipulation Language is the sublanguage responsible for adding, editing or deleting data from a database. In SQL, this corresponds to the INSERT, UPDATE, and DELETE

Q.8 • What is SQL Key Constraints writing an Example of SQL Key Constraints

Ans. SQL key constraints are rules enforced on columns in a table to ensure data integrity and to define relationships between tables. Here are the main types:

1. **Primary Key Constraint:**
    - Ensures each record in a table is unique and not NULL.
    - Typically used to uniquely identify each row in a table.

    - Example
    -
    - CREATE TABLE Employees (
    -
    - EmployeeID INT PRIMARY KEY,

- FirstName VARCHAR(50),
- LastName VARCHAR(50),
- Department VARCHAR(50)
-);

2. **Foreign Key Constraint:**
   - Enforces a link between data in two tables.
   - Ensures referential integrity by requiring that a value in one table corresponds to a primary key value in another table.

   Example:

   CREATE TABLE Departments (DepartmentID INT PRIMARY KEY, DepartmentName VARCHAR(50)

   );

   CREATE TABLE Employees (EmployeeID INT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50),

   DepartmentID INT,

3. **Unique Key Constraint:**
   - Ensures all values in a column or a group of columns are unique.
   - Allows NULL values unless explicitly prohibited.

   Example

   CREATE TABLE Employees (EmployeeID INT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50)

);

4. **Not Null Constraint:**
    - Ensures a column cannot have NULL values.
    - Forces a field to always contain a value, which means you cannot insert a new record or update a record without adding a value to this field.

Example

```
CREATE TABLE Products ( ProductID INT PRIMARY KEY, ProductName VARCHAR(100) NOT NULL, Price DECIMAL(10, 2));
```

Q.9 What is save Point? How to create a save Point write a Query?

Ans. In SQL, a **SAVEPOINT** is a mechanism that allows you to create a point within a transaction to which you can later roll back. This feature is particularly useful when you want to implement a partial rollback in case of errors or other exceptional conditions within a transaction.

Nested Transactions

SAVEPOINT allows you to create nested transactions within a larger transaction. This is helpful when you have multiple steps in a

transaction, and you want to handle errors at a more fine-grained level.

Partial Rollback

It provides the ability to roll back to a specific point within a transaction without affecting the entire transaction. This can be useful if only a portion of the transaction encounters an error, and you want to revert to a known good state.

Error Handling

SAVEPOINT can be used for better error handling. If an error occurs, you can roll back to a savepoint, handle the error, and possibly continue or retry the transaction.

Q10- What is trigger and how to create a Trigger in SQL?

Ans -

A trigger in SQL is a database object that is automatically executed or fired when certain events occur on a particular table or view. These events are typically INSERT, UPDATE, or DELETE operations.

Steps to Create a Trigger in SQL:

1. **Define the Trigger Name**: Give the trigger a unique name.
2. **Specify the Timing**: Determine whether the trigger should be executed BEFORE or AFTER the event.
3. **Specify the Event**: Define the event that will cause the trigger to execute (INSERT, UPDATE, or DELETE).
4. **Specify the Table**: Identify the table to which the trigger applies.
5. **Write the Trigger Logic**: Provide the SQL statements that define the action to be taken when the trigger is fired.

Task1

1. Create Table Name : Student and Exam

Ans:

```
create database studentdb;


use studentdb;


create table  student(rollno integer
primary key auto_increment,name
varchar(20),branch varchar(20));


create table  exam(rollno
integer,foreign key(rollno) references
student(rollno),S_code
varchar(20),Mark integer,p_code
varchar(10));
```

```
insert into student(name,branch)
values('jay','computer science'),
('suhani','Electronic and com'),
('kriti','Electronic and com');

select *from student;
```

OUTPUT

	rollno	name	branch
▶	1	jay	computer science
	2	suhani	Electronic and com
	3	kriti	Electronic and com
*	NULL	NULL	NULL

```
insert into
exam(rollno,S_code,mark,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');

select *from exam;
```

exam table

rollno	S_code	Mark	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

Task 2

. Create table given below

```
create database datadb ;

use datadb;

create table data(id integer primary key
auto_increment,firstname
varchar(20),lastname
varchar(20),address varchar(20),city
varchar(20));
```

```
insert into data(firstname
,lastname,address,city)

values('Mickey','mouse','123 fantasy
Way','anaheim'),

('bat','man','321 cavern ave','gotham'),

('Wonder','Women','987 truth
Way','paradi'),

('donald','DUck','555 Quack
Street','mallard'),

('bugs','bunny','567 carrot
Street','Rascal'),

('Willey','coyote','999  Acme
Way','canyon'),

('cat','Woman','234 purrfect
Street','hairball'),

('Tweety','Bird','543 ','Itotltaw');
```

select *from data;

data table:

output

	id	firstname	lastname	address	city	age
▶	1	Mickey	mouse	123 fantasy Way	anaheim	73
	2	bat	man	321 cavern ave	gotham	54
	3	Wonder	Women	987 truth Way	paradi	39
	4	donald	DUck	555 Quack Street	mallard	65
	5	bugs	bunny	567 carrot Street	Rascal	58
	6	Willey	coyote	999 Acme Way	canyon	61
	7	cat	Woman	234 purrfect Street	hairball	32
	8	Tweetv	Bird	543	Itotltaw	28

Task 3

3. Create table given below: Employee and Incentive.

```
create database empdb1;

use empdb1;

create table  employee(id integer primary key
auto_increment,
firstname varchar(20),
lastname varchar(20),
salary int,
joining_date datetime,

department varchar(20));

insert into
employee(firstname,lastname,salary,joining_date,d
epartment)
values('john','abraham',1000000,'2013-01-01
12:00:00','banking'),
('Michel','clarke',800000,'2013-01-01
12:00:00','insurance'),
('roy','thomas',700000,'2013-02-01
12:00:00','banking'),
```

('tom','jose',600000,'2013-02-01 12:00:00','insurance'),

('jerry','pinto',650000,'2013-02-01 12:00:00','insurance'),

('philip','mathew',750000,'2013-02-01 12:00:00','services'),

('testname1','123',650000,'2013-01-01 12:00:00','services'),

('test','lname%',600000,'2013-02-10 12:00:00','insurance');

select *from employee;

output

id	firstname	lastname	salary	joining_date	department
1	john	abraham	1000000	2013-01-01 12:00:00	banking
2	Michel	clarke	800000	2013-01-01 12:00:00	insurance
3	roy	thomas	700000	2013-02-01 12:00:00	banking
4	tom	jose	600000	2013-02-01 12:00:00	insurance
5	jerry	pinto	650000	2013-02-01 12:00:00	insurance
6	philip	mathew	750000	2013-02-01 12:00:00	services
7	testname1	123	650000	2013-01-01 12:00:00	services
8	test	lname%	600000	2013-02-10 12:00:00	insurance
NULL	NULL	NULL	NULL	NULL	NULL

```
create table incentive(employee_ref_id int, foreign key (employee_ref_id) references employee(id),incentive_date date,incentive_amount int);


insert into incentive(employee_ref_id,incentive_date,incentive_amount)
values(1,'2013-02-01',5000),
(2,'2013-02-01',3000),
(3,'2013-02-01',4000),
(1,'2013-01-01',4500),
(2,'2013-01-01',3500);
```

select *from incentive;

output

employee_ref_id	incentive_date	incentive_amount
1	2013-02-01	5000
2	2013-02-01	3000
3	2013-02-01	4000
1	2013-01-01	4500
2	2013-01-01	3500

select firstname from employee where firstname='tom';

output

firstname
tom

select firstname,joining_date,salary from employee;

output

	firstname	joining_date	salary
▶	john	2013-01-01 12:00:00	1000000
	Michel	2013-01-01 12:00:00	800000
	roy	2013-02-01 12:00:00	700000
	tom	2013-02-01 12:00:00	600000
	jerry	2013-02-01 12:00:00	650000
	philip	2013-02-01 12:00:00	750000
	testname1	2013-01-01 12:00:00	650000
	test	2013-02-10 12:00:00	600000

select *from employee where firstname like '%j%';

output

	id	firstname	lastname	salary	joining_date	department
▶	1	john	abraham	1000000	2013-01-01 12:00:00	banking
	5	jerry	pinto	650000	2013-02-01 12:00:00	insurance
*	NULL	NULL	NULL	NULL	NULL	NULL

select *from employee order by firstname ,salary desc;

output

id	firstname	lastname	salary	joining_date	department
5	jerry	pinto	650000	2013-02-01 12:00:00	insurance
1	john	abraham	1000000	2013-01-01 12:00:00	banking
2	Michel	clarke	800000	2013-01-01 12:00:00	insurance
6	philip	mathew	750000	2013-02-01 12:00:00	services
3	roy	thomas	700000	2013-02-01 12:00:00	banking
8	test	lname%	600000	2013-02-10 12:00:00	insurance
7	testname1	123	650000	2013-01-01 12:00:00	services
4	tom	jose	600000	2013-02-01 12:00:00	insurance
NULL	NULL	NULL	NULL	NULL	NULL

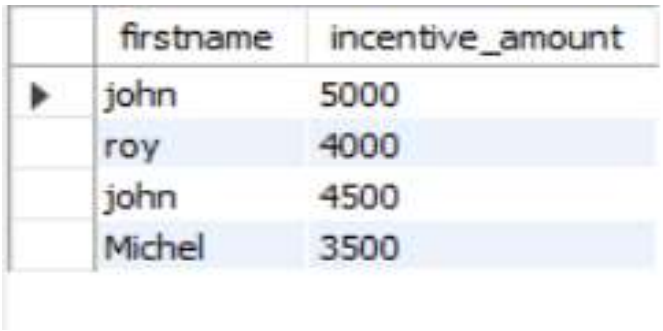
select max(salary),department from employee group by department order by max(salary);

output

max(salary)	department
750000	services
800000	insurance
1000000	banking

select e.firstname,i.incentive_amount from employee as e join incentive as i on e.id=i.employee_ref_id where i.incentive_amount > 3000;

output



	firstname	incentive_amount
▶	john	5000
	roy	4000
	john	4500
	Michel	3500

drop table employee;

drop table incentive;

create table employee_backup(id int primary key auto_increment,firstname varchar(20),

lastname varchar(20),

salary int,

joining_date datetime,

department varchar(20));

delimiter //

create trigger emplog

after insert

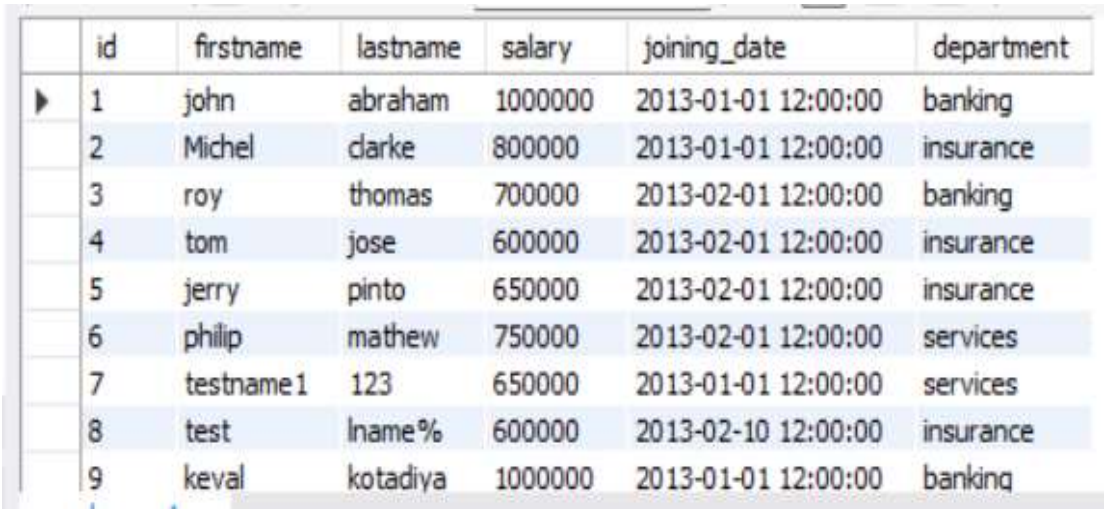
on Employee

```
for each row
begin
insert into
employee_backup(firstname,lastname,salary,joining
_date,department)values
(NEW.firstname,NEW.lastname,NEW.salary,NEW.joi
ning_date,NEW.department);
end //
delimiter ;
```

```
insert into
employee(firstname,lastname,salary,joining_date,d
epartment)
values('keval','kotadiya',1000000,'2013-01-01
12:00:00','banking');
```

```
select  *from employee;
```

output



id	firstname	lastname	salary	joining_date	department
1	john	abraham	1000000	2013-01-01 12:00:00	banking
2	Michel	clarke	800000	2013-01-01 12:00:00	insurance
3	roy	thomas	700000	2013-02-01 12:00:00	banking
4	tom	jose	600000	2013-02-01 12:00:00	insurance
5	jerry	pinto	650000	2013-02-01 12:00:00	insurance
6	philip	mathew	750000	2013-02-01 12:00:00	services
7	testname1	123	650000	2013-01-01 12:00:00	services
8	test	lname%	600000	2013-02-10 12:00:00	insurance
9	keval	kotadiya	1000000	2013-01-01 12:00:00	banking

Task 4

```
create database Salesperson;

use Salesperson;

create table  Salesperson(id integer primary key,
 sname varchar(20),
city varchar(20),
comm real);
```

```
create table Customer(cnm integer primary key
auto_increment,
city varchar(20),
 cname varchar(20),
rating varchar(20),
sno int,foreign key (sno) references Salesperson(id));


insert into Salesperson(id,sname,city,comm)
values(1001,'peel','london',0.12),
(1002,'serres','sanjose',0.13),
(1004,'motika','london',0.11),
(1007,'rafkin','barcelona',0.15),
(1003,'axelrod','newyork',0.01);

select *from Salesperson;
```
output

	id	sname	city	comm
▶	1001	peel	london	0.12
	1002	serres	sanjose	0.13
	1003	axelrod	newyork	0.01
	1004	motika	london	0.11
	1007	rafkin	barcelona	0.15
*	NULL	NULL	NULL	NULL

insert into Customer(cnm,cname,rating,city,sno)
values(201,'Hoffman',100,'london',1001),
(202,'giovanne',200,'roe',1003),
(203,'liu',300,'sanjose',1002),
(204,'grass',100,'barcelona',1002),
(206,'clemens',200,'london',1007),
(207,'pereira',300,'rome',1004);

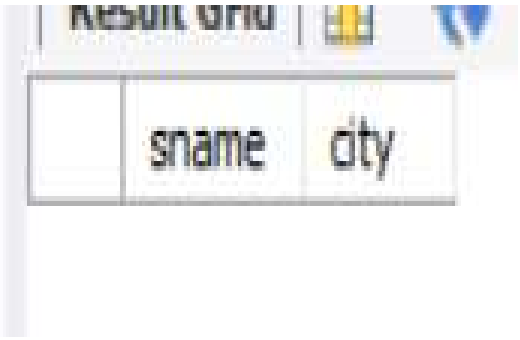
select *from customer;

output

	cnm	city	cname	rating	sno
▶	201	london	Hoffman	100	1001
	202	roe	giovanne	200	1003
	203	sanjose	liu	300	1002
	204	barcelona	grass	100	1002
	206	london	clemens	200	1007
	207	rome	pereira	300	1004
•	NULL	NULL	NULL	NULL	NULL

select sname,city from Salesperson where city='london' and comm > 0.12;

output



select * from Salesperson where city='barcelona'or city='london';

output

	id	sname	city	comm
▶	1001	peel	london	0.12
	1004	motika	london	0.11
	1007	rafkin	barcelona	0.15
*	NULL	NULL	NULL	NULL

select *from Salesperson where comm between 0.10 and 0.12;

output

	id	sname	city	comm
▶	1001	peel	london	0.12
	1004	motika	london	0.11
*	NULL	NULL	NULL	NULL

select *from customer where rating > 100 and city!='rome';

output

	cnm	city	cname	rating	sno
▶	202	roe	giovanne	200	1003
	203	sanjose	liu	300	1002
	206	london	clemens	200	1007
*	NULL	NULL	NULL	NULL	NULL