

1. What do you understand By Database:

A database is a systematic collection of data. They support electronic storage and manipulation of data. Databases make data management easy.

Let us discuss a database example: An online telephone directory uses a database to store data of people, phone numbers, and other contact details. Your electricity service provider uses a database to manage billing, client-related issues, handle fault data, etc.

Let us also consider Facebook. It needs to store, manipulate, and present data related to members, their friends, member activities, messages, advertisements, and a lot more. We can provide a countless number of examples for the usage of databases.

2. What is Normalization?

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

3. What is Difference between DBMS and RDBMS?

DBMS	RDBMS
DBMS stores data as file.	RDBMS stores data in tabular form.

DBMS	RDBMS
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.

DBMS	RDBMS
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 , Window Registry, Forxpro, dbaseIIIplus etc.	Examples: MySQL , PostgreSQL , SQL Server, Oracle, Microsoft Access etc.

4. What is MF Cod Rule of RDBMS Systems?

Rule zero

This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information rule

All information(including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Rule 2: Guaranteed Access

Each unique piece of data(atomic value) should be accesible by : **Table Name + Primary Key(Row) + Attribute(column)**.

Rule 3: Systematic treatment of NULL

Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever. Expression on **NULL** must give null.

Rule 4: Active Online Catalog

Database dictionary(catalog) is the structure description of the complete **Database** and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.

Rule 5: Powerful and Well-Structured Language

One well structured language must be there to provide all manners of access to the data stored in the database. Example: **SQL**, etc. If the database allows access to the data without the use of this language, then that is a violation.

Rule 6: View Updation Rule

All the view that are theoretically updatable should be updatable by the system as well.

Rule 7: Relational Level Operation

There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Rule 8: Physical Data Independence

The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not effect the application.

Rule 9: Logical Data Independence

If there is change in the logical structure(table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10: Integrity Independence

The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. This also make **RDBMS** independent of front-end.

Rule 11: Distribution Independence

A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of **distributed database**.

Rule 12: Nonsubversion Rule

If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of locking or encryption.

5. What do you understand By Data Redundancy?

Data redundancy is when an organization stores the same data in multiple places at the same time. It may occur within many fields in one database or across multiple technological platforms.

Redundancy is common in businesses that don't use a central database or insular management system for data storage. An example of data redundancy is when a company replicates customer information across separate storage systems in multiple departments in a business. Data managers classify data redundancy into two categories, which are:

Positive data redundancy: This is intentional and occurs when an organization creates compressed versions of data to access as a backup. Intentional data

redundancy promotes uniformity and protects data, and it safeguards data in different places to ensure the company's data remain sustainable.

Wasteful data redundancy: This is an unintentional replication of data in a company, which can result from complicated data processes and inefficient coding. It can be difficult to assess which data to update or use when unintentional storage of the same data occurs, but an organization can follow certain practices to reduce this problem.

6. What is DDL Interpreter?

Data Definition Language (DDL) is used to create and modify the structure of objects in a database using predefined commands and a specific syntax. These database objects include tables, sequences, locations, aliases, schemas and indexes.

Data Definition Language explained

DDL is a standardized language with commands to define the storage groups (stogroups), different structures and objects in a database. DDL statements create, modify and remove database objects, such as tables, indexes and stogroups. DDL is also used in a generic sense to refer to any language that describes data.

DDL includes Structured Query Language (SQL) statements to create and drop databases, aliases, locations, indexes, tables and sequences. It also includes statements to alter these objects and impose or drop certain constraints on tables, such as the following:

UNIQUE

PRIMARY

FOREIGN KEY

CHECK

These constraints are used to enforce uniqueness, referential or domain integrity.

When a DDL statement is executed, it takes effect immediately in the database.

DDL is sometimes known as Data Description Language since its statements can also be used to describe, comment on and place labels on database objects.

DDL vs. SQL vs. DML vs. DQL

Since DDL includes SQL statements to define changes in the database schema, it is considered a subset of SQL. SQL uses normal English verbs to modify database objects, and DDL does not appear as a different language in a SQL database.

In Data Manipulation Language (DML), commands are used to modify data in a database. DML statements control access to the database data. In contrast, DDL commands are used to create, delete or alter the structure of objects in a database but not its data. DDL deals with descriptions of the database schema and is useful for creating new tables, indexes, sequences, stogroups, etc. and to define the attributes of these objects, such as data type, field length and alternate table names (aliases).

Structured Query Language (SQL) queries vs. Data Manipulation Language (DML) statements vs. Data Definition Language (DDL) commands

How Data Definition Language (DDL) commands, Data Manipulation Language (DML) statements and Structured Query Language (SQL) queries compare

Data Query Language (DQL) is used to get data within the schema objects of a database and also to query it and impose order upon it. Like DDL, DQL is also a subset of SQL. One of the most common commands in DQL is SELECT. It lets

users get data from a database table and perform some operation on it. When the statement is executed, the result is compiled into a temporary table and displayed by the front-end program or application.

Common Data Definition Language commands

General application users -- i.e., users who are not authorized to work directly with a database -- do not use DDL commands. These general users can and should only access the database indirectly via the application.

The most common command types in DDL are CREATE, ALTER and DROP. All three types have a predefined syntax that must be followed for the command to run and changes to take effect.

1. CREATE

Syntax

CREATE TABLE [table name] ([column definitions]) [table parameters];

The semicolon at the end of the command is used to process every command before it.

The CREATE group of DDL commands includes the following:

CREATE DATABASE defines a logical database under the active location root directory. The database normally consists of a subdirectory of the same name that holds the physical table and index files. Users can use stogroups to implicitly specify different storage directories for individual database objects.

CREATE TABLE creates a table by defining its columns and each column's data type and field length. The command can also be used to create primary and foreign keys for the table.

CREATE STOGROUP creates a Db2-style stogroup to define a physical directory area for storing database objects. A stogroup is associated with a specific directory path.

CREATE TABLESPACE creates a Db2-style tablespace to store tables from the same logical database in multiple directory paths. The tablespace is used with a stogroup.

CREATE ALIAS defines an alias for an existing table or view. The alias may be described in a different location as the table or view. This command also records the alias definition in the catalog tables at the current location.

CREATE SYNONYM can also be used to create an alternate name for an existing table or view at the current location.

CREATE INDEX creates an index on one or more columns of a table for faster data retrieval and to enforce uniqueness constraints on the columns.

CREATE LOCATION creates a new XDB server location in a user-specified subdirectory.

CREATE SEQUENCE must be used to create a sequence at the application server.

CREATE VIEW defines a virtual table that restricts data retrieval and updates to a subset of columns and rows from single- or multibase tables.

CREATE GLOBAL TEMPORARY can be used to create a temporary table's description at the current server.

2. DROP

DDL also includes several DROP commands to delete objects in a database. DROP commands cannot be undone, so once an object is deleted, it cannot be recovered.

Syntax

DROP object type object name;

The most common DROP commands are the following:

DROP DATABASE does the exact opposite of the CREATE DATABASE. It deletes a database defined at a certain location, along with all the objects logically associated with it. It also deletes the database subdirectory even if it is empty and contains no objects logically associated with the database.

DROP STOGROUP deletes a stogroup by severing the logical connection between data objects defined using the stogroup and the directory path specified in the deleted stogroup definition. It doesn't delete the objects associated with the stogroup. As with other DROP commands, the DROP STOGROUP command should be used sparingly and with caution.

DROP TABLE deletes a database table and all associated indexes, views and synonyms built on it.

DROP TABLESPACE deletes a tablespace defined in the current location and all tables logically associated within it.

DROP ALIAS can be used to delete an alternate name for a table or view in a location's system catalog.

DROP SYNONYM can also be used to delete an alternate name for a table or view.

DROP INDEX deletes an index at the current location but only if the index was not created as the result of a UNIQUE, PRIMARY or FOREIGN KEY. To drop/delete such indexes, the existing constraint must first be dropped with the ALTER command.

DROP LOCATION deletes a user-defined XDB server location and the catalog tables, directory structure and objects associated with it.

DROP VIEW deletes a view and all other views defined on it from the system catalog of the current location.

3. ALTER

The third group of DDL commands is ALTER. These commands are used to make modifications to database objects, such as indexes, locations and stogroups.

Syntax

ALTER object type object name parameters;

The most common ALTER commands are the following:

ALTER DATABASE modifies the information parameters of a database under the current XDB server location.

ALTER STOGROUP modifies the specifications of a stogroup defined at the current XDB server location.

ALTER TABLE adds, removes or alters columns and their data types. It can also enforce referential and domain integrity by creating or dropping UNIQUE, PRIMARY, FOREIGN KEY, and CHECK constraints in XDB mode. In Db2 mode, the command can also be used to enforce uniqueness constraints.

ALTER TABLESPACE changes the specifications of a tablespace within the current XDB server location.

ALTER VIEW uses an existing view definition at the current server to regenerate a view.

ALTER SEQUENCE changes sequence attributes at the current server.

ALTER INDEX modifies the configuration of an existing index. The XDB server syntactically supports the command to ensure compatibility with Db2.

4. Other commands

Apart from the CREATE, DROP and ALTER commands, DDL includes other commands:

COMMENT ON is used to add a single-line, multiline or inline comment about an object in the catalog tables at the current location.

LABEL ON is used to add or change descriptive text labels describing tables, views, aliases or columns to the catalog tables.

RENAME is used to modify the name of a database table.

TRUNCATE is used to quickly remove all records from a table, while preserving its full structure so it can be reused later.

Explore the difference between DDL and DML.

This was last updated in June 2022

Continue Reading About Data Definition Language (DDL)

What is the difference between DDL and SQL?

SQL vs. NoSQL vs. NewSQL: How do they compare?

Developing an enterprise data strategy: 10 steps to take

When and how to adopt a data-centric architecture

SQL Server database design best practices and tips for DBAs

Related Terms

conformed dimension

In data warehousing, a conformed dimension is a dimension that has the same meaning to every fact with which it relates. See complete definition

file extension (file format)

In computing, a file extension is a suffix added to the name of a file to indicate the file's layout, in terms of how the data ... See complete definition

tuple

A tuple, pronounced TUH-pul, is an ordered and finite list of elements in various fields of interest, including computing. See complete definition

7. What is DML Compiler in SQL?

DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

SELECT - retrieve data from a database

INSERT - insert data into a table

UPDATE - updates existing data within a table

DELETE - Delete all records from a database table

MERGE - UPSERT operation (insert or update)

CALL - call a PL/SQL or Java subprogram

EXPLAIN PLAN - interpretation of the data access path

LOCK TABLE - concurrency Control

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

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are:

NOT NULL: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.

UNIQUE: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.










PRIMARY KEY: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.



















FOREIGN KEY: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.

CHECK: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.

9. DEFAULT: This constraint specifies a default value for the column when no value is specified by the user

Create Table Name : Student and Exam

				roll_no	name	branch
<input type="checkbox"/>		Edit		Copy		Delete
				3	jay	computer
<input type="checkbox"/>		Edit		Copy		Delete
				4	suhani	electronics and com
<input type="checkbox"/>		Edit		Copy		Delete
				5	kriti	electronics and com

					roll_no	s_code	marks	p_code
<input type="checkbox"/>		Edit		Copy		Delete		
					1	CS11	50	CS
<input type="checkbox"/>		Edit		Copy		Delete		
					2	CS12	60	CS
<input type="checkbox"/>		Edit		Copy		Delete		
					3	EC101	66	EC
<input type="checkbox"/>		Edit		Copy		Delete		
					4	EC102	70	EC
<input type="checkbox"/>		Edit		Copy		Delete		
					5	EC101	45	EC
<input type="checkbox"/>		Edit		Copy		Delete		
					6	EC102	50	EC

10. Create table given below: Employee and Incentive Table Name: Employee

INSERT into employee

(First_name,Last_name,salary,Joining_date,Department)

VALUES('John','Abraham','1000000','01-JAN-13 12:00:00 AM','Banking');

INSERT into employee

(First_name,Last_name,salary,Joining_date,Department)

VALUES('Michael','clarke','8000000','01-JAN-13 12:00:00 AM','Insurance');

INSERT into employee


```
(First_name,Last_name,salary,Joining_date,Department)
VALUES('Roy','Thomas','7000000','01-FEB-13 12:00:00 AM','Banking');
```

```
INSERT into employee
```

```
(First_name,Last_name,salary,Joining_date,Department)
VALUES('Tom','Jose','6000000','01-JAN-13 12:00:00 AM','Insurance');
```

```
INSERT into employee
```

```
(First_name,Last_name,salary,Joining_date,Department)
VALUES('Jerry','Pinto','6500000','01-JAN-13 12:00:00 AM','Insurance');
```

```
INSERT into employee
```

```
(First_name,Last_name,salary,Joining_date,Department)
VALUES('Philip','Mathew','7500000','01-JAN-13 12:00:00 AM','Services');
```

```
INSERT into employee
```

```
(First_name,Last_name,salary,Joining_date,Department)
VALUES('TestName1','123','6500000','01-JAN-13 12:00:00 AM','Services');
```

```
INSERT into employee
```

```
(First_name,Last_name,salary,Joining_date,Department)
VALUES('Testname2','Lname%', '6000000','01-FEB-13 12:00:00
AM','Insurance');
```

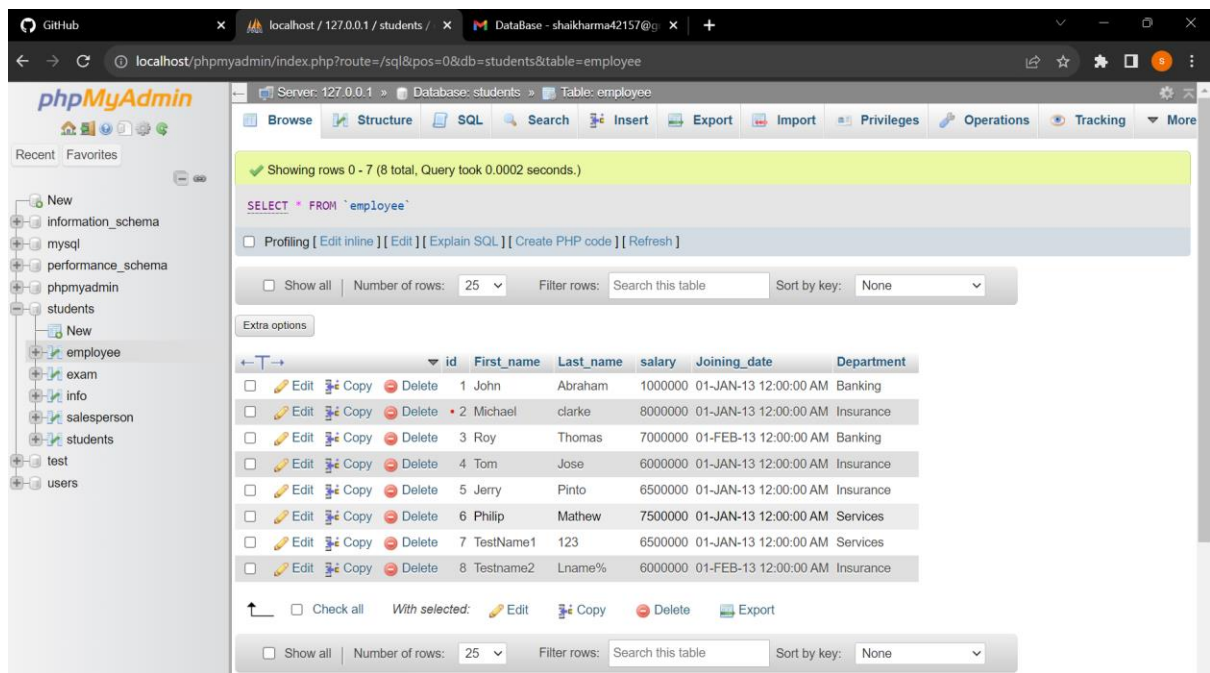


Table Name: Incentive:

CREATE TABLE Incentive(

Employee_Ref_ID INT,

Incentive_Date VARCHAR (20),

Incentive_Amount INT,

FOREIGN KEY (Employee_Ref_ID) REFERENCES employee
(Employee_ID)

);

localhost/phpmyadmin/index.php?route=/sql&pos=0&db=students&table=incentive

Server: 127.0.0.1 » Database: students » Table: incentive

Browse Structure SQL Search Insert Export Import Privileges Op

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

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
1	01-JAN-13	3500

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results, operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

```
INSERT INTO incentive (Employee_Ref_ID, Incentive_Date,
Incentive_Amount)
```

```
VALUES(1,'01-FEB-13',5000);
```

```
INSERT INTO incentive (Employee_Ref_ID, Incentive_Date,
Incentive_Amount)
```

```
VALUES(2,'01-FEB-13',3000);
```

```
INSERT INTO incentive (Employee_Ref_ID, Incentive_Date,
Incentive_Amount)
```

```
VALUES(3,'01-FEB-13',4000);
```

```
INSERT INTO incentive (Employee_Ref_ID, Incentive_Date,
Incentive_Amount)
```

```
VALUES(1,'01-JAN-13',4500);
```

```
INSERT INTO incentive (Employee_Ref_ID, Incentive_Date,
Incentive_Amount)
```

```
VALUES(1,'01-JAN-13',3500);
```

a) Get First_Name from employee table using name “Employee Name”.

ans. `SELECT First_Name FROM employee;`

b) Get FIRST_NAME, Joining Date, and Salary from employee table.

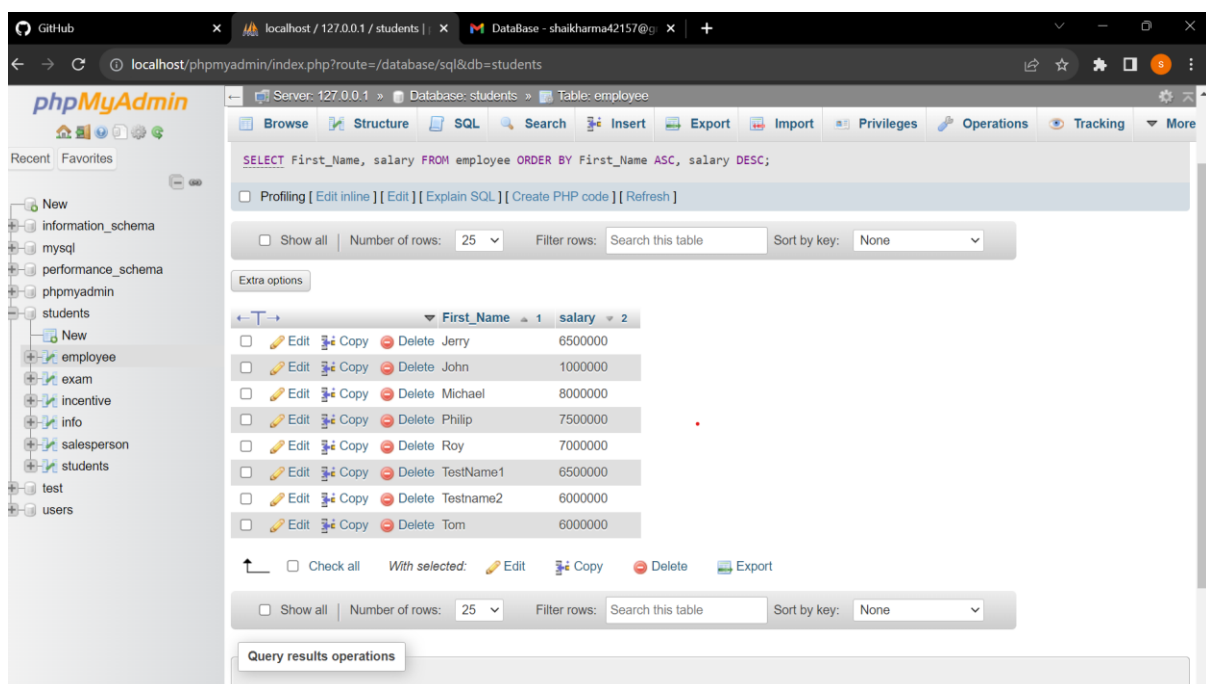
ans. `SELECT First name, Joining Date, Salary FROM employee;`

c) Get all employee details from the employee table order by First_Name
Ascending and Salary

descending?

ans. `SELECT First Name, Salary FROM employee ORDER BY First_Name
ASC,`

`Salary DESC;`



The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** students
- Table:** employee
- SQL Query:** `SELECT First_Name, salary FROM employee ORDER BY First_Name ASC, salary DESC;`
- Query Results:** A table with 2 columns: First_Name and salary. It contains 8 rows of data.

	First_Name	salary
<input type="checkbox"/>	Jerry	6500000
<input type="checkbox"/>	John	1000000
<input type="checkbox"/>	Michael	8000000
<input type="checkbox"/>	Philip	7500000
<input type="checkbox"/>	Roy	7000000
<input type="checkbox"/>	TestName1	6500000
<input type="checkbox"/>	Testname2	6000000
<input type="checkbox"/>	Tom	6000000

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

ans. `SELECT First_Name FROM employee WHERE First_Name LIKE ‘J%’;`

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	First_Name
<input type="checkbox"/> Edit Copy Delete	John
<input type="checkbox"/> Edit Copy Delete	Jerry

☐ Check all | With selected: Edit Copy Delete Export

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results operations

11. Create table given below: Salesperson and Customer

Recent Favorites

- New
- information_schema
- mysql
- performance_schema
- phpmyadmin
- students
 - New
 - employee
 - exam
 - incentive
 - info
 - salesperson
 - students
- test
- users

⚠ Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

✓ Showing rows 0 - 4 (5 total, Query took 0.0003 seconds.)

```
SELECT * FROM `salesperson`
```

☐ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

☐ Show all | Number of rows: 25

Extra options

S_no	S_name	city	Com
1001	peel	London	0
1002	serres	san jose	0
1004	motika	London	0
1007	Rafkin	Barcelona	0
1003	axelrod	New York	0

☐ Show all | Number of rows: 25

- **CREATE TABLE** customer
(CNM INT PRIMARY KEY, CNAME VARCHAR (20), CITY VARCHAR (20), RATING INT, SNo INT,
FOREIGN KEY (SNo) REFERENCES salesperson (SNo));
- **INSERT INTO** customer (CNM, CNAME, CITY, RATING, SNo)
VALUES ('201', 'Hoffman', 'London', '100', '1001'), ('202', 'Giovanne',
'Rome', '200', '1003'), ('203', 'Liu', 'San Jose', '300', '1002'), ('204', 'Grass',
'Barcelona', '100', '1002'), ('206', 'Clemens', 'London', '300', '1007'), ('207',
'Pereira', 'Rome', '100', '1004');

Retrieve the below data from above table

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

					CNM	CNAME	CITY	RATING	SNo
<input type="checkbox"/>	Edit	Copy	Delete		201	Hoffman	London	100	1001
<input type="checkbox"/>	Edit	Copy	Delete		202	Giovanne	Rome	200	1003
<input type="checkbox"/>	Edit	Copy	Delete		203	Liu	San Jose	300	1002
<input type="checkbox"/>	Edit	Copy	Delete		204	Grass	Barcelona	100	1002
<input type="checkbox"/>	Edit	Copy	Delete		206	Clemens	London	300	1007
<input type="checkbox"/>	Edit	Copy	Delete		207	Pereira	Rome	100	1004

↑ ☐ Check all | With selected: Edit Copy Delete

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Console | **Results operations**

- a) All orders for more than \$1000.
- Ans. **SELECT ***
FROM customer
WHERE SNo>1000;

`SELECT * FROM salesperson WHERE COMM>.12;`

☐ Profiling [\[Edit inline \]](#) [\[Edit \]](#) [\[Explain SQL \]](#) [\[Create PHP code \]](#) [\[Refresh \]](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	SNo	SNAME	CITY	COMM
<input type="checkbox"/> Edit Copy Delete	1002	Serres	San Jose	.13
<input type="checkbox"/> Edit Copy Delete	1007	Rafkin	Barcelona	.15

☐ Check all | With selected: [Edit](#) [Copy](#) [Delete](#) [Export](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results operations

[Print](#) [Copy to clipboard](#) [Export](#) [Display chart](#) [Create view](#)

[Bookmark this SQL query](#)

Console

b) Names and cities of all salespeople in London with commission above 0.12

Ans. `SELECT * FROM salesperson WHERE COMM>.12;`

c) All salespeople either in Barcelona or in London

ans. `SELECT * FROM salesperson WHERE CITY='Barcelona' OR CITY='London';`

Extra options

	SNo	SNAME	CITY	COMM
<input type="checkbox"/> Edit Copy Delete	1001	Peel	London	.12
<input type="checkbox"/> Edit Copy Delete	1004	Motika	London	.11
<input type="checkbox"/> Edit Copy Delete	1007	Rafkin	Barcelona	.15

☐ Check all | With selected: [Edit](#) [Copy](#) [Delete](#) [Export](#)

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

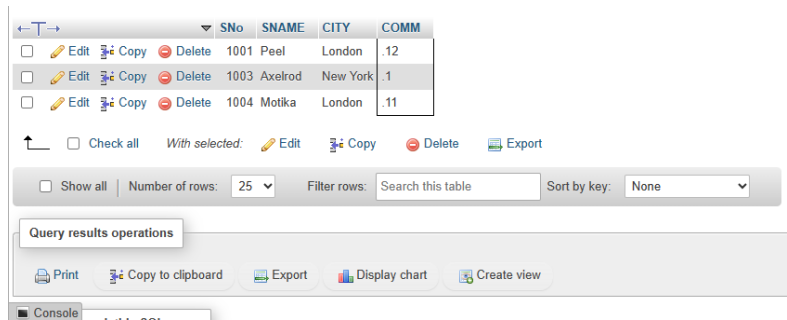
Query results operations

[Print](#) [Copy to clipboard](#) [Export](#) [Display chart](#) [Create view](#)

Console

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

Ans. `SELECT * FROM salesperson WHERE COMM BETWEEN .10 AND 0.12;`

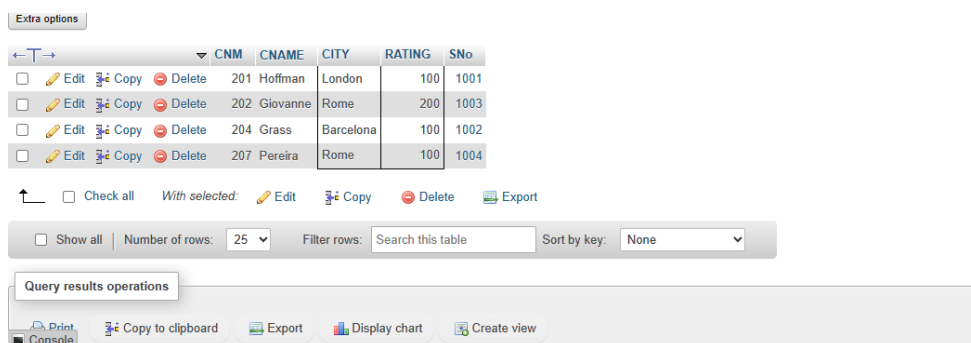


The screenshot shows a database query result interface. At the top, there's a toolbar with navigation and action icons. Below it is a table with columns: SNo, SNAME, CITY, and COMM. The table contains three rows of data. Below the table, there are controls for row selection, editing, deleting, and exporting. Further down, there are filters for showing all rows, setting the number of rows to 25, a search filter, and a sort key dropdown. At the bottom, there's a 'Query results operations' section with buttons for Print, Copy to clipboard, Export, Display chart, and Create view. A 'Console' tab is visible at the very bottom.

SNo	SNAME	CITY	COMM
1001	Peel	London	.12
1003	Axelrod	New York	.1
1004	Motika	London	.11

e) All customers excluding those with rating ≤ 100 unless they are located in Rome

Ans. `SELECT * FROM customer WHERE RATING \leq 100 OR CITY='Rome';`



The screenshot shows a database query result interface. At the top, there's a toolbar with navigation and action icons. Below it is a table with columns: CNM, CNAME, CITY, RATING, and SNo. The table contains four rows of data. Below the table, there are controls for row selection, editing, deleting, and exporting. Further down, there are filters for showing all rows, setting the number of rows to 25, a search filter, and a sort key dropdown. At the bottom, there's a 'Query results operations' section with buttons for Print, Copy to clipboard, Export, Display chart, and Create view. A 'Console' tab is visible at the very bottom.

CNM	CNAME	CITY	RATING	SNo
201	Hoffman	London	100	1001
202	Giovanne	Rome	200	1003
204	Grass	Barcelona	100	1002
207	Pereira	Rome	100	1004