In MySQL, a popular relational database management system, there is a wide range of data types available to store various types of data. Here are the commonly used data types in My:

**Numeric Data Types:**

INT or INTEGER: Used for storing whole numbers.

SMALLINT: Used for smaller whole numbers.

BIGINT: Used for larger whole numbers.

DECIMAL or NUMERIC: Used for precise decimal numbers.

FLOAT: Used for approximate floating-point numbers.

DOUBLE: Used for larger floating-point numbers.

**Character Data Types:**

CHAR: Used for fixed-length strings.

VARCHAR: Used for variable-length strings.

TEXT: Used for large text blocks.

**Date and Time Data Types:**

DATE: Used for storing dates (YYYY-MM-DD format).

TIME: Used for storing times (HH:MM:SS format).

DATETIME: Used for storing dates and times together (YYYY-MM-DD HH:MM:SS format).

TIMESTAMP: Used for storing dates and times, often used for tracking changes.

**Boolean Data Type:**

BOOLEAN or BOOL: Used for storing true/false values.

**Binary Data Types:**

BINARY: Used for storing binary data with a fixed length.

VARBINARY: Used for storing binary data with a variable length.

BLOB: Used for storing large binary objects like images, documents, etc.

**Enumerated Data Type:**

ENUM: Used for storing a set of predefined values.

**JSON Data Type:**

JSON: Used for storing JSON (JavaScript Object Notation) data.

**Spatial Data Types:**

GEOMETRY, POINT, LINESTRING, POLYGON, etc.: Used for storing spatial data.

**DDL and DML COMMANDS IN**

1. CREATE DATABASE
2. CREATE TABLE
3. SELECT
4. INSERT
5. UPDATE
6. DELETE
7. WHERE
8. ORDER BY
9. GROUP BY
10. JOIN
11. ALTER TABLE
12. DROP TABLE

The syntax for the commands:

CREATE DATABASE:

CREATE DATABASE database\_name;

CREATE TABLE:

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

...

PRIMARY KEY (column\_name)

);

INSERT:

INSERT INTO table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

UPDATE:

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

DELETE:

DELETE FROM table\_name

WHERE condition;

SELECT:

SELECT column1, column2, ...

FROM table\_name;

WHERE:

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

ORDER BY:

SELECT column1, column2, ...

FROM table\_name

ORDER BY column\_name [ASC|DESC];

GROUP BY:

SELECT column1, COUNT(\*)

FROM table\_name

GROUP BY column1;

JOIN:

SELECT column1, column2, ...

FROM table1

JOIN table2 ON table1.column\_name = table2.column\_name;

ALTER TABLE:

ALTER TABLE table\_name

ADD column\_name datatype;

DROP TABLE:

DROP TABLE table\_name;

**EXAMPLES**

Step 1: CREATE DATABASE

CREATE DATABASE Company;

Step 2: CREATE TABLE

USE Company;

CREATE TABLE Employee (

id INT PRIMARY KEY,

name VARCHAR(50),

age INT,

position VARCHAR(50),

salary DECIMAL(10, 2)

);

Step 3: INSERT

INSERT INTO Employee (id, name, age, position, salary)

VALUES (1, 'John Doe', 30, 'Manager', 5000.00);

INSERT INTO Employee (id, name, age, position, salary)

VALUES (2, 'Jane Smith', 25, 'Developer', 4000.00);

INSERT INTO Employee (id, name, age, position, salary)

VALUES (3, 'Mark Johnson', 35, 'Analyst', 4500.00);

Step 4: UPDATE

UPDATE Employee

SET position = 'Senior Developer', salary = 5500.00

WHERE id = 2;

Step 5: DELETE

DELETE FROM Employee

WHERE id = 3;

Step 6: SELECT

SELECT \* FROM Employee;

Step 7: WHERE

SELECT name, position, salary FROM Employee

WHERE age > 28;

Step 8: ORDER BY

SELECT \* FROM Employee

ORDER BY salary DESC;

Step 9: GROUP BY

SELECT position, COUNT(\*) AS count

FROM Employee

GROUP BY position;

Step 10: JOIN (Example with another table called "Department")

CREATE TABLE Department (

id INT PRIMARY KEY,

name VARCHAR(50),

location VARCHAR(50)

);

INSERT INTO Department (id, name, location)

VALUES (1, 'IT', 'New York');

SELECT Employee.name, Department.name AS department

FROM Employee

JOIN Department ON Employee.id = Department.id;

Step 11: ALTER TABLE

Step 12: DROP TABLE

DROP TABLE Employee;

**Table: Employees**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **emp\_age** | **emp\_position** | **emp\_salary** |
| 1 | John Doe | 30 | Manager | 5000 |
| 2 | Jane Smith | 25 | Developer | 4000 |
| 3 | Mark Johnson | 35 | Analyst | 4500 |
| 4 | Sarah Brown | 28 | Designer | 3500 |
| 5 | Michael Clark | 32 | Engineer | 4800 |
| 6 | Emily Davis | 27 | Developer | 4200 |
| 7 | David Lee | 29 | Manager | 5200 |
| 8 | Lisa Wilson | 31 | Analyst | 3800 |
| 9 | Andrew Taylor | 33 | Designer | 3600 |
| 10 | Jessica Allen | 26 | Engineer | 4700 |

**DDL Query:**

**Create a new table called "Departments" with columns: dept\_id (integer) and dept\_name (varchar).**

**DML Queries:**

1. **Insert a new employee with emp\_id 11, emp\_name "Robert Johnson," emp\_age 34, emp\_position "Manager," and emp\_salary 5500.00.**
2. **Update the salary of the employee with emp\_id 3 to 4800.00.**
3. **Delete the employee with emp\_id 8 from the table.**
4. **Select all employees whose emp\_age is greater than 30.**
5. **Select the emp\_name and emp\_position of employees whose emp\_salary is above 4000.00.**
6. **Select the average emp\_salary of all employees.**
7. **Group employees by emp\_position and count the number of employees in each position.**
8. **Join the Employees table with a new table called "Departments" on emp\_id and dept\_id and select emp\_name and dept\_name.**

**Select all columns for all employees:**

**SELECT \* FROM Employees;**

**Select employees whose age is greater than 30:**

**SELECT \* FROM Employees WHERE emp\_age > 30;**

**Select employees who are developers and have a salary above 4000.00:**

**SELECT \* FROM Employees WHERE emp\_position = 'Developer' AND emp\_salary > 4000.00;**

**Select employees whose names start with "J":**

**SELECT \* FROM Employees WHERE emp\_name LIKE 'J%';**

**Select employees sorted by salary in descending order:**

**SELECT \* FROM Employees ORDER BY emp\_salary DESC;**

**Select the average salary of all employees:**

**SELECT AVG(emp\_salary) AS average\_salary FROM Employees;**

**Select the total number of employees in each position:**

**SELECT emp\_position, COUNT(\*) AS total\_employees FROM Employees GROUP BY emp\_position;**

**Select the youngest and oldest employee:**

**SELECT \* FROM Employees WHERE emp\_age = (SELECT MIN(emp\_age) FROM Employees)**

**UNION**

**SELECT \* FROM Employees WHERE emp\_age = (SELECT MAX(emp\_age) FROM Employees);**

**Select employees whose salaries are above the average salary:**

**SELECT \* FROM Employees WHERE emp\_salary > (SELECT AVG(emp\_salary) FROM Employees);**

**Select employees with their position and a calculated column for 10% salary increment:**

**SELECT emp\_name, emp\_position, emp\_salary, emp\_salary \* 1.1 AS new\_salary FR**

**Query 1:**

**Find the names of all employees.**

**SELECT emp\_name FROM Employees;**

**Query 2:**

**What is the total count of employees?**

**SELECT COUNT(\*) AS total\_employees FROM Employees;**

**Query 3:**

**Retrieve the details of employees whose age is between 25 and 35.**

**SELECT \* FROM Employees WHERE emp\_age BETWEEN 25 AND 35;**

**Query 4:**

**Which employees have a position of "Manager" or "Developer"?**

**SELECT \* FROM Employees WHERE emp\_position IN ('Manager', 'Developer');**

**Query 5:**

**Find the average salary of all employees.**

**SELECT AVG(emp\_salary) AS average\_salary FROM Employees;**

**Query 6:**

**List the employees in descending order of their salaries.**

**SELECT \* FROM Employees ORDER BY emp\_salary DESC;**

**Query 7:**

**Retrieve the employees with the highest salary.**

**SELECT \* FROM Employees WHERE emp\_salary = (SELECT MAX(emp\_salary) FROM Employees);**

**Query 8:**

**Find the employees whose names contain the letter "a".**

**SELECT \* FROM Employees WHERE emp\_name LIKE '%a%';**

**Query 9:**

**Show the employees grouped by their positions along with the count of employees in each position.**

**SELECT emp\_position, COUNT(\*) AS count\_employees FROM Employees GROUP BY emp\_position;**

**Query 10:**

**Retrieve the employees hired before the age of 30.**

**SELECT \* FROM Employees WHERE emp\_age < 30;**

**INSERT Queries:**

Insert a new employee with specified values into the "Employees" table:

INSERT INTO Employees (emp\_id, emp\_name, emp\_age, emp\_position, emp\_salary)

VALUES (11, 'Amy Johnson', 29, 'Developer', 4500.00);

Insert multiple employees at once using a single INSERT statement:

INSERT INTO Employees (emp\_id, emp\_name, emp\_age, emp\_position, emp\_salary)

VALUES (12, 'Robert Smith', 32, 'Analyst', 4000.00),

(13, 'Emily Brown', 27, 'Designer', 3500.00),

(14, 'Daniel Wilson', 31, 'Engineer', 4800.00);

Insert data into a table from the result of a SELECT query:

INSERT INTO NewEmployees (emp\_name, emp\_age, emp\_position, emp\_salary)

SELECT emp\_name, emp\_age, emp\_position, emp\_salary

FROM Employees WHERE emp\_age > 30;

Insert rows into a table with default values for specific columns:

INSERT INTO Employees (emp\_id, emp\_name, emp\_age)

VALUES (15, 'Michelle Lee', 28),

(16, 'William Davis', 33),

(17, 'Olivia Johnson', 26);

Insert data into a table by selecting values from another table:

INSERT INTO Employees (emp\_id, emp\_name, emp\_age, emp\_position, emp\_salary)

SELECT emp\_id, emp\_name, emp\_age, 'Manager', emp\_salary

FROM TemporaryEmployees;

**UPDATE Queries:**

Update the salary of an employee based on their emp\_id:

UPDATE Employees

SET emp\_salary = 5200.00

WHERE emp\_id = 5;

Update multiple columns for a specific employee:

UPDATE Employees

SET emp\_position = 'Senior Developer', emp\_salary = 5500.00

WHERE emp\_id = 6;

Update a column with an expression based on its current value:

UPDATE Employees

SET emp\_salary = emp\_salary \* 1.1

WHERE emp\_position = 'Manager';

Update a column based on a subquery:

UPDATE Employees

SET emp\_salary = (SELECT AVG(emp\_salary) FROM Employees)

WHERE emp\_position = 'Analyst';

Update rows in a table based on values from another table using a JOIN:

UPDATE Employees

SET emp\_salary = source.emp\_salary

FROM NewSalaries AS source

WHERE Employees.emp\_id = source.emp\_id;

**DELETE Queries:**

Delete a specific employee from the "Employees" table based on emp\_id:

DELETE FROM Employees

WHERE emp\_id = 9;

Delete employees older than 35 years:

DELETE FROM Employees

WHERE emp\_age > 35;

Delete all rows from a table:

DELETE FROM Employees;

Delete duplicate rows from a table based on a specific column:

DELETE FROM Employees

WHERE emp\_id NOT IN (

SELECT MIN(emp\_id)

FROM Employees

GROUP BY emp\_name

);

Delete rows from a table using a subquery with conditions:

DELETE FROM Employees

WHERE emp\_id IN (

SELECT emp\_id

FROM TemporaryEmployees

WHERE emp\_age < 25

);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Department Table: |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | **dept\_id** | **dept\_name** | **location** |  |  |  |
|  | 1 | IT | New York |  |  |  |
|  | 2 | Sales | London |  |  |  |
|  | 3 | HR | Paris |  |  |  |
|  |  |  |  |  |  |  |
|  | Employee Table: |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | **emp\_id** | **emp\_name** | **dept\_id** | **emp\_position** | **emp\_salary** |  |
|  | 1 | John Doe | 1 | Manager | 5000 |  |
|  | 2 | Jane Smith | 1 | Developer | 4000 |  |
|  | 3 | Mark Johnson | 2 | Analyst | 4500 |  |
|  | 4 | Sarah Brown | 1 | Designer | 3500 |  |
|  | 5 | Michael Clark | 3 | Engineer | 4800 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Project Table: |  |  |  |  |  |
|  | **project\_id** | **project\_name** | **dept\_id** | **start\_date** | **end\_date** |  |
|  | 1 | Project X | 1 | 01-01-2022 | 30-06-2022 |  |
|  | 2 | Project Y | 2 | 15-03-2022 | 30-09-2022 |  |
|  | 3 | Project Z | 1 | 01-05-2022 | 31-12-2022 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Employee\_Project Table: |  |  |  |  |  |
|  | **emp\_id** | **project\_id** |  |  |  |  |
|  | 1 | 1 |  |  |  |  |
|  | 1 | 2 |  |  |  |  |
|  | 2 | 1 |  |  |  |  |
|  | 3 | 2 |  |  |  |  |
|  | 4 | 3 |  |  |  |  |
|  | 5 | 1 |  |  |  |  |
|  | 5 | 3 |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Manager Table: |  |  |  |  |  |
|  | **emp\_id** | **manager\_name** |  |  |  |  |
|  | 1 | Michael Cole |  |  |  |  |
|  | 2 | Emily Davis |  |  |  |  |
|  | 3 | John Smith |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | SalaryHistory Table: |  |  |  |  |  |
|  | **history\_id** | **emp\_id** | **salary** | **effective\_date** |  |  |
|  | 1 | 1 | 5000 | 01-01-2022 |  |  |
|  | 2 | 1 | 5500 | 01-07-2022 |  |  |
|  | 3 | 2 | 4000 | 01-01-2022 |  |  |
|  | 4 | 3 | 4500 | 01-01-2022 |  |  |
|  | 5 | 4 | 3500 | 01-01-2022 |  |  |
|  | 6 | 5 | 4800 | 01-01-2022 |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**DML Queries:**

1. **Insert a new department named "Marketing" with dept\_id 4 and location "San Francisco".**
2. **Insert a new employee with emp\_id 6, emp\_name "Laura Anderson", dept\_id 1, emp\_position "Developer", and emp\_salary 4200.00.**
3. **Update the salary of employee with emp\_id 2 to 4500.00.**
4. **Delete the project with project\_id 3 from the Project table.**
5. **Insert a new entry in the Employee\_Project table to assign employee with emp\_id 3 to project with project\_id 1.**
6. **Update the manager\_name of employee with emp\_id 3 to "Michelle Roberts" in the Manager table.**
7. **Delete all employees from the Sales department (dept\_id 2) from the Employee table.**
8. **Insert a new salary record for employee with emp\_id 4, set the salary as 3800.00, and the effective\_date as the current date.**
9. **Update the salary of all employees in the IT department (dept\_id 1) by adding a 10% increment to their current salaries.**
10. **Delete all salary history records for employees with emp\_id 5 from the SalaryHistory table.**

**DDL Queries/Questions:**

1. **Create a table called "Customers" with columns cust\_id (INT), cust\_name (VARCHAR), and city (VARCHAR).**
2. **Add a new column "phone" with datatype VARCHAR(15) to the Employee table.**
3. **Create a foreign key constraint on the Employee table to enforce referential integrity with the Department table.**
4. **Drop the Manager table from the database.**
5. **Create an index on the emp\_name column in the Employee table for faster retrieval.**
6. **Rename the column "emp\_position" in the Employee table to "job\_title".**
7. **Add a primary key constraint to the Project table on the project\_id column.**
8. **Alter the data type of the salary column in the SalaryHistory table to DECIMAL(12, 2).**
9. **Create a view named "EmployeeDetails" that displays the employee name, department name, and project name for each employee.**
10. **Create a stored procedure named "GetEmployeeCountByDepartment" that accepts a department ID as input and returns the count of employees in that department.**

**Join Queries/Questions:**

1. **Retrieve the employee name, department name, and salary for all employees.**
2. **Retrieve the employee names and project names for all employees assigned to projects.**
3. **Retrieve the employee names and manager names for all employees who have a manager.**
4. **Retrieve the employee names, department names, and project names for all employees in the Sales department.**
5. **Retrieve the employee names and their respective salary history records.**
6. **Retrieve the employee names, project names, and start dates for all employees assigned to projects in the IT department.**
7. **Retrieve the employee names and project names for all employees who are not assigned to any project.**
8. **Retrieve the employee names, department names, and average salary for each department.**
9. **Retrieve the employee names and project names for all employees who are assigned to more than one project.**
10. **Retrieve the employee names and their respective manager names for all employees in the IT department.**

**APPENDIX : POSTGRE**

In Postgre, a powerful open-source relational database management system, there is a wide range of data types available to store different types of data. Here are some commonly used data types in Postgre:

**Numeric Data Types:**

INTEGER: Used for storing whole numbers.

SMALLINT: Used for smaller whole numbers.

BIGINT: Used for larger whole numbers.

DECIMAL or NUMERIC: Used for precise decimal numbers.

REAL: Used for single-precision floating-point numbers.

DOUBLE PRECISION: Used for double-precision floating-point numbers.

**Character Data Types:**

CHAR: Used for fixed-length character strings.

VARCHAR: Used for variable-length character strings.

TEXT: Used for large blocks of text.

**Date and Time Data Types:**

DATE: Used for storing dates (YYYY-MM-DD format).

TIME: Used for storing times (HH:MI:SS format).

TIMESTAMP: Used for storing dates and times together (YYYY-MM-DD HH:MI:SS format).

INTERVAL: Used for storing intervals of time.

**Boolean Data Type:**

BOOLEAN: Used for storing true/false values.

**Binary Data Types:**

BYTEA: Used for storing binary data.

BLOB: Used for storing large binary objects.

**Enumerated Data Type:**

ENUM: Used for storing a predefined list of values.

**JSON and JSONB Data Types:**

JSON: Used for storing JSON (JavaScript Object Notation) data as text.

JSONB: Used for storing JSON data in a binary format, providing efficient storage and indexing capabilities.

**Array Data Types:**

ARRAY: Used for storing arrays of any data type.

**UUID Data Type:**

UUID: Used for storing universally unique identifiers.

**Geometric Data Types:**

POINT, LINE, POLYGON, etc.: Used for storing geometric data.