**QUESTION:**01 .The difference between *data* and *information* lies primarily in their level of processing, meaning, and usefulness. Here’s a breakdown:

**1. Data**

* **Definition**: Data is raw, unprocessed facts and figures without context. It is typically in the form of numbers, symbols, characters, or observations and lacks any inherent meaning until interpreted.
* **Example**: A list of temperatures recorded throughout the day (e.g., 22, 25, 20, 18, 23).
* **Characteristics**:
  + Collected but not yet organized, analyzed, or interpreted.
  + Often presented in raw format.
  + Requires processing to be useful.

**2. Information**

* **Definition**: Information is processed, structured, or organized data that has been interpreted to add meaning and context, making it useful for decision-making.
* **Example**: An analysis showing that temperatures peak around noon, which suggests the warmest part of the day.
* **Characteristics**:
  + Data that has been refined and contextualized.
  + Helps to answer specific questions, leading to better understanding.
  + Ready for direct application or interpretation.

**In Summary**

* **Data** is the unprocessed input, while **information** is the meaningful output derived from that data.
* Data becomes information when it’s organized, processed, and presented in a way that provides value or insight.

**QUESTION\_02:**

In relational databases, *Primary Key*, *Composite Primary Key*, and *Foreign Key* are essential concepts that help maintain data integrity and establish relationships between tables. Here’s an explanation of each:

**1. Primary Key**

* **Definition**: A primary key is a unique identifier for each record in a table. It ensures that each row in the table is unique and can be referenced individually. A primary key must have unique values and cannot contain NULL values.
* **Example**: In a table called **Students**, the column student\_id could serve as the primary key, as each student has a unique ID.

sql

Copy code

Students Table

+------------+-------------+------------+

| student\_id | first\_name | last\_name |

+------------+-------------+------------+

| 1 | Alice | Johnson |

| 2 | Bob | Smith |

| 3 | Charlie | Lee |

+------------+-------------+------------+

Here, student\_id is the primary key because each ID is unique and identifies each student record.

**2. Composite Primary Key**

* **Definition**: A composite primary key is a primary key that consists of two or more columns to uniquely identify each record in a table. It is used when a single column alone cannot uniquely identify records.
* **Example**: In a table called **Enrollment**, each record could be uniquely identified by the combination of student\_id and course\_id, representing the student’s enrollment in specific courses.

sql

Copy code

Enrollment Table

+------------+-----------+-------------+

| student\_id | course\_id | semester |

+------------+-----------+-------------+

| 1 | 101 | Fall 2023 |

| 1 | 102 | Fall 2023 |

| 2 | 101 | Fall 2023 |

+------------+-----------+-------------+

Here, the combination of student\_id and course\_id serves as the composite primary key, as it uniquely identifies each enrollment.

**3. Foreign Key**

* **Definition**: A foreign key is a column or set of columns in a table that establishes a link between data in two tables. The foreign key in one table refers to the primary key in another, creating a relationship between the tables. It enforces referential integrity, meaning that the values in the foreign key column must match values in the referenced primary key column or be NULL.
* **Example**: In an **Enrollment** table, student\_id could be a foreign key referencing the student\_id column in the **Students** table.

sql

Copy code

Students Table Enrollment Table

+------------+------------+ +------------+-----------+

| student\_id | first\_name | | student\_id | course\_id |

+------------+------------+ +------------+-----------+

| 1 | Alice | | 1 | 101 |

| 2 | Bob | | 1 | 102 |

| 3 | Charlie | | 2 | 101 |

+------------+------------+ +------------+-----------+

Here, student\_id in the **Enrollment** table is a foreign key linking to the student\_id in the **Students** table, establishing a relationship between students and their enrollments.

**Summary Table**

| **Key Type** | **Purpose** | **Example** |
| --- | --- | --- |
| Primary Key | Uniquely identifies each record in a table | student\_id in **Students** table |
| Composite Primary Key | Uniquely identifies records using multiple columns | Combination of student\_id and course\_id in **Enrollment** table |
| Foreign Key | Links records between tables | student\_id in **Enrollment** table linking to **Students** table |

In essence, a primary key uniquely identifies a record within its own table, a composite key does the same using multiple fields, and a foreign key links records between different tables.

**QUESTIONS\_03:**

MySQL supports several numeric data types that allow for storing different kinds of numeric values, each with specific uses and constraints. Here are five common numeric types:

1. **INT** (Integer)
   * Used to store whole numbers without decimal points.
   * Range: -2147483648 to 2147483647 (signed) or 0 to 4294967295 (unsigned).
   * Example: INT, INT(10).
2. **FLOAT** (Floating Point)
   * Used for approximate numerical values with floating-point precision.
   * Typically used to store decimal numbers with moderate precision.
   * Example: FLOAT, FLOAT(7,4) stores numbers with up to 7 total digits, including 4 digits after the decimal.
3. **DOUBLE** (Double Precision Floating Point)
   * Used for storing decimal values with greater precision than FLOAT.
   * It provides more precision and can store larger values than FLOAT.
   * Example: DOUBLE, DOUBLE(10,6).
4. **DECIMAL** (Fixed-Point)
   * Used to store exact numerical values with a specified precision, useful for monetary values where exact precision is essential.
   * Syntax: DECIMAL(M, D), where M is the total number of digits and D is the number of digits after the decimal.
   * Example: DECIMAL(10,2) can store numbers up to 10 digits in total, with 2 decimal places.
5. **TINYINT**
   * Used to store very small integers, conserving storage space.
   * Range: -128 to 127 (signed) or 0 to 255 (unsigned).
   * Example: TINYINT(3).

These data types allow MySQL to handle a range of numeric values with varying levels of precision and storage requirements.

**QUESTIONS\_04:**

Here’s how to create an Employee table in MySQL, insert data for two employees, and then delete one of them:

**Step 1: Create the Employee Table**

sql

Copy code

CREATE TABLE Employee (

EmployeeId INT PRIMARY KEY,

EmployeeName VARCHAR(50) NOT NULL,

EmployeeSalary DECIMAL(10, 2),

JoiningDate DATE

);

* **EmployeeId**: INT data type, used as the primary key.
* **EmployeeName**: VARCHAR(50), string type to store employee names.
* **EmployeeSalary**: DECIMAL(10, 2), used to store exact salary values with two decimal points.
* **JoiningDate**: DATE type to store the joining date of the employee.

**Step 2: Insert Data for Two Employees**

sql

Copy code

INSERT INTO Employee (EmployeeId, EmployeeName, EmployeeSalary, JoiningDate)

VALUES (1, 'John Doe', 55000.00, '2023-01-15');

INSERT INTO Employee (EmployeeId, EmployeeName, EmployeeSalary, JoiningDate)

VALUES (2, 'Jane Smith', 60000.00, '2023-02-20');

This inserts two records with sample data for EmployeeId, EmployeeName, EmployeeSalary, and JoiningDate.

**Step 3: Delete One Employee Data**

sql

Copy code

DELETE FROM Employee

WHERE EmployeeId = 1;

Bottom of Form