**DATA BASE CONNECTIVITY USING PYTHON**

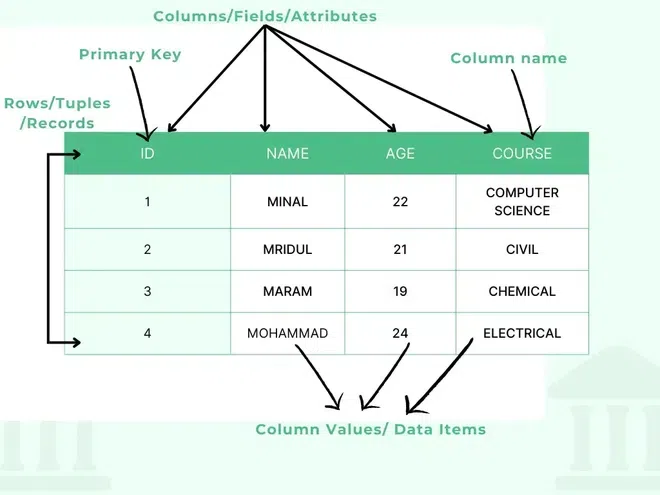
## What is SQL?

* MySQL is free and open-source.MySQL is ideal for both small and large applications.
* SQL is the standard language for dealing with Relational Databases.
* MySQL is a widely used relational database management system (RDBMS).
* SQL is used to insert, search, update, and delete database records.

## What is RDBMS?

* RDBMS stands for Relational Database Management Systems.
* It is a program that allows us to create, delete, and update a relational database.
* A Relational Database is a database system that stores and retrieves data in a tabular format organized in the form of **rows and columns**.

## What is a Database Table?

* A table is a **collection of related data** in an organized manner in the form of rows and columns.
* It is an organized arrangement of data and information in **tabular form**containing rows and columns, making it easier to understand and compare data.
* Here is the pictorial representation of the table and its different components containing the data about different students that is ID, name, Age, and course.
* 

Key features of an RDBMS include:

1. **Data Integrity and Consistency**: It ensures that data remains accurate and consistent through constraints (like primary keys, foreign keys, and unique constraints).
2. **ACID Properties**: RDBMS follow ACID (Atomicity, Consistency, Isolation, Durability) principles to ensure reliable transactions.
3. **Data Security**: They offer security mechanisms, including user authentication and permissions.
4. **Normalization**: RDBMS support normalization, which reduces redundancy by organizing data into related tables.
5. **Data Relationships**: Relationships can be defined between tables using foreign keys, allowing for complex data interconnections and more efficient querying.

**ACID Properties**

The concept of ACID properties in DBMS that are necessary for maintaining data consistency, integrity, and reliability while performing transactions in the database.

### 1. Atomicity

**Definition**: Ensures that all operations within a transaction are completed; otherwise, none of them are applied.

**Example**: Imagine transferring $100 from Alice's account to Bob's account. The transaction involves:

* 1. Deducting $100 from Alice’s account.
  2. Adding $100 to Bob’s account.

If any part of this transaction fails (e.g., deducting from Alice’s account works, but adding to Bob’s account fails), Atomicity ensures that neither action is completed. This way, the accounts remain as they were before the transaction started.

### 2. Consistency

**Definition**: Ensures that a transaction takes the database from one valid state to another, maintaining all predefined rules.

**Example**: If a business rule states that an account balance cannot go below $0, then any transaction that would result in a negative balance (like withdrawing $100 from an account with $50) will fail. Consistency ensures that the rule is upheld.)

### 3. Isolation

**Definition**: Ensures that transactions occur independently, without interference. Intermediate states of a transaction are invisible to other transactions.

**Example**: Imagine two people simultaneously transferring money to Bob’s account:

* 1. Alice transfers $100.
  2. Charlie transfers $50.

Isolation guarantees that these transactions do not interfere with each other. The final balance will reflect both transactions correctly, regardless of their order.

### 4. Durability

**Definition**: Ensures that once a transaction is committed, it remains in the system, even if the system crashes afterward.

**Example**: After transferring $100 from Alice’s account to Bob’s and committing the transaction, the changes will be permanently saved. Even if the database crashes right after, Bob’s balance will still reflect the $100 increase when the system recovers.

**Normalization**

**Normalization** is a process in database design that organizes data to reduce redundancy and dependency by **dividing larger tables into smaller, related tables**.

This process ensures the database is efficient and reduces issues like **data anomalies during insertions, updates, and deletions.**

## Steps of Normalization

* First Normal Form (1NF): Ensure that each column contains only atomic values (no repeating groups or arrays).
* Second Normal Form (2NF): Achieve 1NF and ensure that all non-key columns are fully dependent on the primary key (remove partial dependencies).
* Third Normal Form (3NF): Achieve 2NF and remove transitive dependencies, where non-key columns depend on other non-key columns rather than directly on the primary key.

## Example of Normalization

* Unnormalized Table (Not in 1NF):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID | CustomerName | CustomerAddress | ItemPurchased | Quantity |
| 1 | Alice | 123 Street A | Pen, Notebook | 2, 1 |
| 2 | Bob | 456 Street B | Pencil, Eraser | 3, 2 |

### Step 1: Convert to 1NF

Separate each item into its own row to ensure atomicity.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID | CustomerName | CustomerAddress | ItemPurchased | Quantity |
| 1 | Alice | 123 Street A | Pen | 2 |
| 1 | Alice | 123 Street A | Notebook | 1 |
| 2 | Bob | 456 Street B | Pencil | 3 |
| 2 | Bob | 456 Street B | Eraser | 2 |

### Step 2: Convert to 2NF

Separate the customer information to avoid partial dependencies on the OrderID.

#### **Customer Table**

|  |  |  |
| --- | --- | --- |
| CustomerID | CustomerName | CustomerAddress |
| 1 | Alice | 123 Street A |
| 2 | Bob | 456 Street B |

#### Order Table

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID | CustomerID | ItemPurchased | Quantity |
| 1 | 1 | Pen | 2 |
| 1 | 1 | Notebook | 1 |
| 2 | 2 | Pencil | 3 |
| 2 | 2 | Eraser | 2 |

### Step 3: Convert to 3NF

Remove any transitive dependencies, if present. Since there are no transitive dependencies here, the table is already in 3NF.

## Boyce-Codd Normal Form (BCNF)

BCNF is an extension of the third normal form (3NF). A table is in BCNF if it is in 3NF, and every determinant (a column that determines another column) is a candidate key.

### Example:

Imagine a table 'CourseInstructor' that records which instructor can teach which course:

|  |  |  |
| --- | --- | --- |
| **Course** | **Instructor** | **Department** |
| Math | Prof. A | Science |
| Math | Prof. B | Science |
| Physics | Prof. A | Science |
| English | Prof. C | Arts |

To convert this table to BCNF, split it into two tables:  
1. CourseDepartment Table:

|  |  |
| --- | --- |
| **Course** | **Department** |
| Math | Science |
| Physics | Science |
| English | Arts |

2. InstructorCourse Table:

|  |  |
| --- | --- |
| **Instructor** | **Course** |
| Prof. A | Math |
| Prof. B | Math |
| Prof. A | Physics |
| Prof. C | English |

## Fourth Normal Form (4NF)

Definition: A table is in 4NF if it is in BCNF and has no multi-valued dependencies, which means no column should have multiple values associated with another column.

### Example:

Consider a table with EmployeeID, Skill, and Language:

## EmployeeSkill Table

|  |  |  |
| --- | --- | --- |
| **EmployeeID** | **Skill** | **Language** |
| E001 | Java | English |
| E001 | Java | Spanish |
| E001 | Python | English |
| E002 | C++ | French |

To achieve 4NF, split this table into:

## **1.EmployeeSkill Table**

|  |  |
| --- | --- |
| **EmployeeID** | **Skill** |
| E001 | Java |
| E001 | Python |
| E002 | C++ |

**2. EmployeeLanguage Table:**

|  |  |
| --- | --- |
| **mployeeID** | **Language** |
| E001 | English |
| E001 | Spanish |
| E002 | French |

## Fifth Normal Form (5NF)

Definition: A table is in 5NF if it is in 4NF and cannot be decomposed further to eliminate redundancy.

### Example:

Consider a table with Project, Task, and Employee:

|  |  |  |
| --- | --- | --- |
| **Project** | **Task** | **Employee** |
| Project1 | Design | E001 |
| Project1 | Design | E002 |
| Project1 | Coding | E001 |
| Project2 | Testing | E003 |

To achieve 5NF, split this table into:  
1. ProjectTask Table:

|  |  |
| --- | --- |
| **Project** | **Task** |
| Project1 | Design |
| Project1 | Coding |
| Project2 | Testing |

2. TaskEmployee Table:

|  |  |
| --- | --- |
| **Task** | **Employee** |
| Design | E001 |
| Design | E002 |
| Coding | E001 |
| Testing | E003 |

3. ProjectEmployee Table:

|  |  |
| --- | --- |
| **Project** | **Employee** |
| Project1 | E001 |
| Project1 | E002 |
| Project2 | E003 |