Data Dynamos

How can we use Python to create a simple relational database that organizes data and avoids duplicates?

Introduction to Data Schemas and Relational Databases

Data schema refers to the model or structure of a relational database. It defines how data is organized in the database, including how tables relate to each other and what data is stored in each table.

Importance of Data Schemas

- Organizing data
- Maintaining Data Integrity
- Data Relationships
- Normalization
- Optimizing Performance
- Scalability

Some Key Components of a Data Schema

Tables

basic unit of a relational database schema. Each represents an entity or object and is made up of columns and rows.

Indexes

Data structures that improve the speed of operations that retrieve data from a table, allowing the database to find records quickly without scanning the entire table.

Constraints

Rules that define limits or restrictions on the data stored in tables. Ensure the accuracy and integrity of the data. eg. NOT NULL, UNIQUE, CHECK, DEFAULT.

Explanation of Primary and Foreign Keys

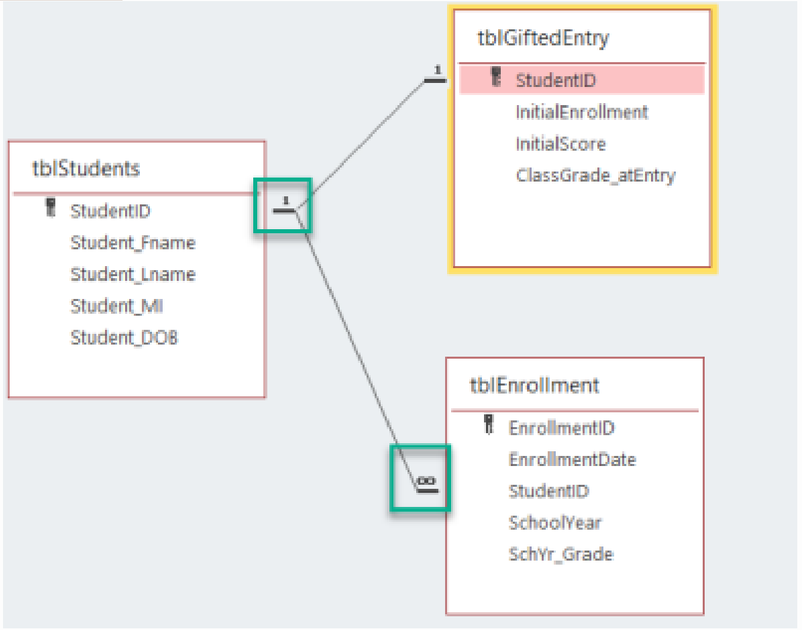
Primary Keys

unique identifier for each record in a database table ensuring that each row is distinct and easily retrievable. It is essential for defining relationships between tables and is typically a single column or a combination of columns that cannot contain duplicate or null values.

Foreign Keys

field (or set of fields) in one table that links to the primary key in another table, establishing a relationship between the two tables. This key enforces referential integrity, ensuring that data across tables remains consistent by requiring that the foreign key in one table corresponds to a valid primary key in the related table.

Example



StudentID is a primary key in tblStudents but a foreign key in tblEnrollment table. That means we have one record for each student in tblStudents, and we have as many enrollment records for that student in as many years as they are enrolled.

What is Normalization, and Why is it Useful?

Normalization

Process of organizing data in a database, and it is used to reduce redundancy from a relation or set of relations.

The main reason for normalizing the relations is removing anomalies. Failure to eliminate anomalies can lead to data redundancy and can cause data integrity and other problems as the database grows.

Purpose

- Ensure accurate data representation.
- Facilitates effective data analysis and decision making.
- Improves the quality and consistency of data.

Forms in relational database

- **First Normal Form (1NF)**
  each table cell should contain only a single value and each column should have a unique name. This form helps to eliminate duplicate data and simplify queries
- **Second Normal Form (2NF)**
  we reduce redundancy by eliminating partial dependencies.
- **Third Normal Form (3NF)**
  we eliminates fields that do not depend on a key.

Python Example of Creating a Simple Database

To establish a connection with SQLite3 database using python you need to import the sqlite3 module using the important statement.

The connect() method accepts the name of the database you need to connect with as a parameter and, returns a connection object

Then create a cursor() object. It is called to send commands to the SQL

Then close the database connection

To execute a query in the database, create an object and write the SQL command in it with being commented. Example: - sql_comm = ”SQL statement”

To execute the command call the cursor method execute() and pass the name of the sql command as a parameter in it. Save a number of commands as the sql_comm and execute them. Save the changes in the file by committing those changes and then lose the connection.

To insert data into the table we write the SQL command as a string and will use the execute() method.

```python
import sqlite3

# Open a connection to the database file
dbase = sqlite3.connect('Our_data.db')
print("Database opened")

# Create a cursor object to interact with the database
cursor = dbase.cursor()

# Create the table if it doesn't exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS employee_records(
        ID INT PRIMARY KEY NOT NULL,
        NAME TEXT NOT NULL,
        DIVISION TEXT NOT NULL,
        STARS INT NOT NULL)
''')
print("Table created")

# Insert a record into the table
cursor.execute('''
    INSERT INTO employee_records(ID, NAME, DIVISION, STARS)
    VALUES(5, 'James', 'Maintenance', 4)
''')
dbase.commit()  # Commit the transaction to save the changes
print("Record inserted")

# Close the cursor and the database connection
cursor.close()  # Close the cursor first
dbase.close()   # Close the database connection
print("Database closed")
```

```
Database opened
Table created
Record inserted
```

Summary of Key Points and Group Reflection

- **Data Schema**
  is the structure and relationships of data in a database.
- **Primary and Foreign Keys**
  Primary key ensure data uniqueness, while Foreign Key establishes relationships between tables.
- **Normalization**
  reduces redundancy and improves data integrity by organizing data across multiple related tables.
- **SQLite**
  is a lightweight, easy-to-use database system that integrates well with Python.

Group Reflection

Primary and foreign keys are essential in relational databases for maintaining data integrity and establishing relationships between tables. Together, they help prevent data anomalies, ensure consistency, and support efficient querying.

Normalization organizes data into related tables, making it easier to update, query, and maintain. This process enhances storage efficiency and ensures data integrity across large-scale systems.

SQLite is ideal for mobile apps that need a lightweight, embedded database without requiring a server connection. .

Relational databases like SQLite offer structured data storage with strong consistency and support for complex queries but may struggle with scalability for large, unstructured data.