Relational Database

Definitions

Primary key: unique identifier for each record in a database table.

Foreign key: column(s) in a database table that provides a **link/relationship** between data in two tables by referencing the primary key of the source table.

1NF (First Normal Form): no repeating fields/groups.

2NF (Second Normal Form): no partial key dependency.

3NF (Third Normal Form): no non-key dependency.

Attributes of a Database

Table: A collection of related data entries and it consists of columns and rows.

Record: A single, implicitly structured data item in a table.

Field: Also known as a column, a field represents a data type for a single piece of data in a record.

Key Types in Relational Databases

Primary Key: A field (or combination of fields) that uniquely identifies a record in a table.

Secondary Key: A field that is not a primary key but can be used for searching, sorting, and indexing.

Composite Key: A combination of two or more columns used to uniquely identify a record where a single column is not enough.

Foreign Key: A field in a table that links to the primary key of another table to maintain the relational aspect of the database.

Data Redundancy and Data Dependency

Data Redundancy: Occurs when the same piece of data exists in multiple places. This can lead to inconsistencies.

Data Dependency: When data structure changes affect the data retrieval, this shows a dependency that can complicate updates and maintenance.

Example: Suppose we have two tables, 'Orders' and 'Customers'. If customer address data is duplicated across both tables, any change in the address must be updated in multiple places, leading to potential inconsistencies.

Understanding Normal Forms: 1NF and 2NF

First Normal Form (1NF)

A table is in First Normal Form (1NF) if it contains no repeating groups or arrays. This means:

- Each column must hold only atomic (indivisible) values.
- Each column in a table must be unique.
- The order in which data is stored does not affect the database's integrity.

Example: Suppose we have a student table where the 'Courses' field contains multiple courses separated by commas, this would violate 1NF. To conform to 1NF, each course should have its own record.

Here’s the non-1NF and 1NF version of a 'Student' table:

Non-1NF and 1NF Student Tables

Non-1NF Student Table

ID	StudentName	Courses
1	John Doe	Math, Science
2	Jane Smith	Literature, Art

1NF Student Table

ID	StudentName	Course
1	John Doe	Math
1	John Doe	Science
2	Jane Smith	Literature
2	Jane Smith	Art

Second Normal Form (2NF)

A table is in Second Normal Form (2NF) if:

- It is in 1NF.
- All non-key attributes are fully functional dependent on the primary key.

This means that there should be no partial dependency of any column on the primary key.

Example: If a composite key exists, such as (ID, DepartmentID), then each attribute that is not part of the key must depend on the whole key for its existence, not just part of the key.

Ensuring 2NF in our student table:

Table: Student

ID	DepartmentID	StudentName	Course
1	5	John Doe	Math
1	5	John Doe	Science
2	9	Jane Smith	Literature
2	9	Jane Smith	Art

Table: Department

DepartmentID	Name
5	Science
9	Humanities

Reducing Data Redundancy to 3NF

Third Normal Form (3NF): A table design is in 3NF if it is in Second Normal Form (2NF) and no non-prime attribute is transitively dependent on the primary key.

Example: In a 'Student' table, to be in 3NF, there must not be attributes that are indirectly related to the primary key through another attribute.

Structured 'Student' table for 3NF:

Table: Student

ID	StudentName	DepartmentID
1	John Doe	5
2	Jane Smith	5

Table: Department

DepartmentID	Name
5	Science

Introduction to Basic SQL Syntax with SQLite 3

SQL (Structured Query Language) is a standardized programming language used for managing relational databases and performing various operations on the data within them. SQLite 3 is a lightweight, disk-based database that doesn't require a separate server process and allows access to the database using a nonstandard variant of the SQL query language. Here are some basic SQL commands:

Creating a Database and Tables

To start using SQL in SQLite 3, you first need to create a database and then create tables within that database to hold the data.

Create Table Syntax:

```
CREATE TABLE tablename (
    column1 datatype PRIMARY KEY,
    column2 datatype NOT NULL,
    column3 datatype DEFAULT 'value',
    ...
);
```

Example:

```
CREATE TABLE Students (
    StudentID int PRIMARY KEY,
    StudentName text NOT NULL,
    Age int,
    Major text
);
```

Inserting Data

Once your table is created, you can start inserting data into the table.

Insert Syntax:

```
INSERT INTO tablename (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Example:

```
INSERT INTO Students (StudentID, StudentName, Age, Major)
VALUES (1, 'John Doe', 20, 'Computer Science');
```

Querying Data

To retrieve data from a database, you use the SELECT statement.

Select Syntax:

```
SELECT column1, column2, ...
FROM tablename
WHERE condition
ORDER BY column
LIMIT number;
```

Example:

```
SELECT StudentName, Major FROM Students
WHERE Age >= 20
ORDER BY StudentName;
```

Updating Data

To modify data in the database, you use the UPDATE statement.

Update Syntax:

```
UPDATE tablename
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example:

```
UPDATE Students
SET Major = 'Information Technology'
WHERE StudentID = 1;
```

Deleting Data

To delete data from a table, you use the DELETE statement.

Delete Syntax:

```
DELETE FROM tablename
WHERE condition;
```

Example:

```
DELETE FROM Students
WHERE StudentID = 1;
```

SQLITE 3 with Python

Importing the Module

First, you need to import the sqlite3 module into your Python script.

Example:

```
import sqlite3
```

Creating a Connection

To perform tasks on the database, you first need to create a connection to it. This is done using the `connect()` method, which returns a connection object.

Example:

```
conn = sqlite3.connect('example.db')
```

Creating a Cursor Object

Once a connection is established, you can create a cursor object using the cursor method of the connection object. The cursor is used to execute SQL commands.

Example:

```
cursor = conn.cursor()
```

Executing SQL Commands

You can execute SQL commands through the cursor object using the `execute()` method.

Create a Table Example:

```
cursor.execute("CREATE TABLE students (id INTEGER PRIMARY KEY, name
TEXT, grade INTEGER)")
```

Insert Data Example:

```
cursor.execute("INSERT INTO students (name, grade) VALUES ('John Doe',
90)")
```

Select Data Example:

```
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

Committing Transactions

To save changes to the database, you need to commit the transactions. This is done using the `commit()` method of the connection object.

Example:

```
conn.commit()
```

Closing the Connection

Finally, it's important to close the connection when you are done with it to free up system resources.

Example:

```
conn.close()
```

Using Parameterized Queries to Insert Data

To securely insert data into a SQLite database using Python's `sqlite3` module, you can use parameterized queries with placeholders (`?`). This method allows you to pass data values in a tuple or list at the time of executing the command, which helps prevent SQL injection attacks.

Example:

```
# Example of inserting data using parameterized queries
cursor.execute("INSERT INTO students (name, grade) VALUES (?, ?)",
('Jane Doe', 85))
```

In this example, the `?` placeholders in the SQL command are replaced by the values in the tuple `('Jane Doe', 85)` at the time of execution. This approach ensures that data is handled safely and efficiently.