Understanding Database Models and SQL

Raghda Al taei

2024

1 Introduction

In this document, we discuss the differences between relational and transactional database models, along with key concepts such as entities, attributes, and relationships. We also explore how to use SQL commands to retrieve and store data, create tables, and understand Entity-Relationship (ER) diagrams.

2 Relational vs. Transactional Database Models

A **relational model** is a database design that emphasizes the relationships between different tables, optimizing data queries and making data access more intuitive. In contrast, a **transactional database model** is more operational, focusing on storing detailed records. For example, in healthcare, a transactional database might store all claims data, which may not be structured for easy analysis. Data often needs to be extracted from the transactional database and transferred into a relational model for analysis.

3 Building Blocks of Relational Models

The three core components of relational models are:

- **Entities**: A person, place, thing, or event that is unique and distinguishable, such as "Customer" or "Product".
- **Attributes**: Characteristics of an entity. For example, a "Customer" entity might have attributes like "Name" and "Email".
- **Relationships**: The associations between entities, such as a "Customer" placing "Orders".

3.1 Types of Relationships

- **One-to-One**: A relationship where one entity is associated with exactly one other entity. Example: A manager managing one store.
- **One-to-Many**: One entity is related to multiple instances of another. Example: A customer can have many orders.
- **Many-to-Many**: Many instances of one entity relate to many instances of another. Example: Students enrolled in multiple classes, and each class having multiple students.

4 Entity-Relationship (ER) Diagrams

ER diagrams visually represent the relationships between entities in a database. They include symbols for entities, attributes, and relationships, helping to understand how data is connected.

4.1 Notations for Relationships in ER Diagrams

- **Chen Notation**: Uses symbols like "1:M" for one-to-many and "M:N" for many-to-many.
- **Crow's Foot Notation**: Uses symbols resembling "train tracks" for one and "crow's foot" for many.
- **UML Class Diagram Notation**: Uses "1..1" for one-to-one and "1..*" for one-to-many.

5 SQL Basics: SELECT Statement

The `SELECT` statement is fundamental for retrieving data in SQL. It specifies what data to select and from which table. The basic syntax is:

```
SELECT column_name FROM table_name;
```

Example:

```
SELECT product_name FROM products;
```

To select multiple columns:

```
SELECT prod_name, prod_id, prod_price FROM products;
```

To select all columns:

```
SELECT * FROM products;
```

5.1 Limiting Results

To limit the number of returned rows, use the `LIMIT` keyword:

```
SELECT * FROM products LIMIT 5;
```

6 Creating Tables with SQL

Data scientists may need to create tables to store model predictions or extracted data. The `CREATE TABLE` statement is used to define new tables:

```
CREATE TABLE shoes (
    shoe_id INT PRIMARY KEY,
    brand VARCHAR(50),
    shoe_type VARCHAR(30),
    color VARCHAR(20),
    price DECIMAL(10, 2),
    description TEXT NOT NULL
);
```

In this example, `shoe_id` is defined as the primary key, ensuring it is unique and non-null. Other columns specify their data types and whether they can contain `NULL` values.

6.1 Understanding NULL Values

A `NULL` value represents the absence of a value, while an empty string (`''`) is considered a value containing no characters. Primary keys cannot have `NULL` values.

7 Lesson 7: Creating Tables in SQL

7.1 Overview

Creating new tables and storing data within them is an essential skill in SQL. This lesson discusses when to create tables, how to define their structure, and how to insert data.

7.2 Key Concepts

- **Purpose of Creating Tables:**
  - Storing predictions for future use in dashboards.
  - Combining web-scraped data with existing datasets.
- **CREATE TABLE Statement:**
  - Syntax:

    ```
    CREATE TABLE table_name (
        column1 datatype [NULL | NOT NULL],
        column2 datatype [NULL | NOT NULL],
        ...
    );
    ```
- **Column Definitions:**
  - Each column must have a name and data type.
  - Primary keys cannot accept null values.
- **Inserting Data:**
  - Use `INSERT INTO` for adding data.
  - Preferred method includes specifying columns:

    ```
    INSERT INTO table_name (column1, column2, ...)
    VALUES (value1, value2, ...);
    ```

8 Lesson 8: Temporary Tables

8.1 Overview

Temporary tables are used to store subsets of data temporarily. They are deleted when the session ends and can improve query performance.

8.2 Key Concepts

- **Creating Temporary Tables:**

  ```
  CREATE TEMPORARY TABLE temp_table_name AS
  SELECT * FROM original_table WHERE condition;
  ```
- **Benefits:**
  - Faster than creating permanent tables.
  - Simplifies complex queries.

9 Lesson 9: Commenting in SQL

9.1 Overview

Adding comments to SQL code improves readability and helps with debugging.

9.2 Key Concepts

- **Importance of Comments:**
  - Aids in understanding code after a break.
  - Useful for troubleshooting by commenting out code sections.
- **Comment Syntax:**
  - Single line comments: `-- comment`
  - Multi-line comments: `/*comment /`