Lesson 5A: Data Types and Tables in SQL

In Lesson 5A, we explore the pivotal aspects of SQL that form the backbone of any robust database: Data Types and Table Structures. This lesson is designed to equip you with the knowledge and skills necessary to effectively choose the right data types for your database columns and to create well-structured tables. By understanding these fundamental concepts, you will be able to ensure data integrity, optimize database performance, and lay a solid foundation for advanced data manipulation and analysis within the DVD rental database context or any other relational database you may work with in the future.

Objectives

1. Understand SQL Data Types:
Gain a comprehensive understanding of the variety of data types available in SQL, including numeric, string, date/time, and more. This objective focuses on teaching you how to select the appropriate data type for each column in your database, considering factors like data precision, storage efficiency, and the nature of the data you're storing. Understanding data types is crucial for ensuring that your database can accurately and efficiently store the information it's meant to hold.

2. Learn to Create Tables:
Learn the process of creating tables within a database using SQL. This includes defining table names, specifying columns along with their data types, and understanding how to set primary keys. You'll explore the syntax for the `CREATE TABLE` statement, discuss considerations for table design such as normalization, and learn about table constraints that enforce data integrity (e.g., `UNIQUE`, `NOT NULL`). Creating well-structured tables is fundamental to building a database that is logical, efficient, and easy to query.

3. Master Data Insertion:
Master the techniques for inserting data into your tables. This objective covers the use of the `INSERT INTO` statement to add new records to a table, including inserting a single row, or multiple rows, and using sub-selects for more complex insertions. You'll learn best practices for data insertion, such as batch inserting for efficiency and handling duplicate records or key conflicts. Inserting data accurately is key to maintaining the reliability of your database.

4. Explore Table Alterations and Updates:
Understand how to modify existing table structures and update data within your tables. This part of the lesson will introduce you to the `ALTER TABLE` statement for adding, removing, or modifying columns in a table, and the `UPDATE` statement for changing existing data. We'll discuss scenarios where altering a table's structure or updating its data might be necessary and the implications these actions have on data integrity and database design.

5. Apply Knowledge Through Practical Examples:
Apply the concepts learned through practical examples using the dvdrental database. This objective is designed to solidify your understanding of data types, table creation, and data insertion by putting theory into practice. You'll work through examples that simulate real-world scenarios, reinforcing the lesson's concepts and preparing you for applying these skills in your projects or work.

Topics with Explanations and Examples
In SQL, data types define the kind of value that can be stored in each column of a database table. Choosing the right data type for your data is crucial because it directly impacts the database's performance and storage efficiency. SQL supports a variety of data types, categorized broadly into numeric types, character types, date/time types, and more specialized types like Boolean or binary.

1. SQL Data Types Overview
SQL data types categorize the kind of data that can be stored in a table column. Choosing the correct data type for each column is essential for data integrity and query performance.

Numeric Types: Include `INTEGER` for whole numbers, `DECIMAL` and `NUMERIC` for precise fractional numbers, and `FLOAT` for floating-point numbers.
Character Types: CHAR` (fixed length) and `VARCHAR` (variable length) are used for text. `CHAR` is space-padded to the specified length, while `VARCHAR` adjusts to the data size.
Date/Time Types: `DATE` stores dates, `TIME` stores time, and `TIMESTAMP` stores both date and time values, allowing you to capture moments in time accurately.

Understanding these types helps ensure data is stored in a format that best represents its nature, from simple names and addresses to precise numerical data and timestamps marking specific events.

2. Creating and Structuring Tables

Tables are database structures that hold data in rows and columns. Effective table design is crucial for database scalability and data retrieval efficiency.

Example 1:

Creating a New Table: Creating a table in the dvdrental database to track customer feedback might involve various data types

```
CREATE TABLE customer_feedback (
    feedback_id SERIAL PRIMARY KEY,
    customer_id INT,
    feedback_date DATE,
    feedback_text VARCHAR(255)
);
```

Example 2:

To create a table for tracking special offers in the dvdrental database

```
CREATE TABLE special_offers (
    offer_id SERIAL PRIMARY KEY,
    film_id INT,
    offer_start_date DATE,
    offer_end_date DATE,
    discount_percentage DECIMAL(5,2)
);
```

Example 3:

Scenario: In the dvdrental database, suppose we want to create a new feature for customers to leave feedback on movies they've rented. We would need a new table to store this feedback information.

Step 1: Define the Table and Columns
First, identify what information you need to store. For movie feedback, we might want the customer's ID, the film's ID, the feedback text, and the date the feedback was given.

Step 2: Choose Data Types
Based on what we're storing, we need to select appropriate data types:

- `customer_id` and `film_id` should match the data types used in their respective tables, typically `INTEGER`.
- `feedback_text` could be a `VARCHAR`, allowing variable-length feedback.
- `feedback_date` should be a `DATE` type, storing when the feedback was submitted.

Step 3: Create the Table
Now, we can construct our `CREATE TABLE` statement:

```
CREATE TABLE movie_feedback (
    feedback_id SERIAL PRIMARY KEY,
    customer_id INTEGER NOT NULL,
    film_id INTEGER NOT NULL,
    feedback_text VARCHAR(500),
    feedback_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (film_id) REFERENCES film(film_id)
);
```

Explanation:

- `feedback_id` is an auto-incrementing `SERIAL` type, serving as a unique identifier for each piece of feedback.
- `customer_id` and `film_id` are marked as `INTEGER` and `NOT NULL`, ensuring every piece of feedback is linked to a specific customer and film. They also reference the `customer` and `film` tables to maintain relational integrity.
- `feedback_text` is a `VARCHAR(500)`, giving ample space for feedback but limiting it to 500 characters.
- `feedback_date` records when the feedback was left.

3. Inserting Data into Tables
Inserting data involves adding new records to a table. SQL provides the `INSERT` statement for this purpose.

Example 1:

Inserting a new special offer into the `special_offers` table

```
INSERT INTO special_offers (film_id, offer_start_date, offer_end_date,
discount_percentage)
VALUES (101, '2023-10-01', '2023-10-15', 20.00);
```

Example 2:

Scenario: Let's return to our previous example as seen above. As stated before, in the dvdrental database, suppose we want to create a new feature for customers to leave feedback on movies they've rented. We would need a new table to store this feedback information.

Once you've created your tables, the next step is to populate them with data. This process is fundamental for building up your database's content and preparing it for analysis and queries. Let's continue with our `movie_feedback` table example in the dvdrental database and walk through the process of inserting feedback data.

Step 1: Prepare Your Data
Before inserting data, ensure you have all the necessary information for each column in your table. For the `movie_feedback` table, you need:

1. `customer_id`: The ID of the customer leaving feedback.
2. `film_id`: The ID of the film the feedback is about.
3. `feedback_text`: The text content of the feedback.
4. `feedback_date`: The date the feedback was submitted.

Step 2: Writing the INSERT Statement
The `INSERT INTO` statement is used to add new records to a table. You can insert a single row of data or multiple rows in one go.

Example: Inserting a Single Row
Suppose a customer with ID 101 leaves feedback for a film with ID 202:

```
INSERT INTO movie_feedback (customer_id, film_id,
feedback_text, feedback_date)
VALUES (101, 202, 'Great movie, really enjoyed the special effects!',
'2023-10-05');
```

This statement adds one record to the `movie_feedback` table, specifying values for each column.

Step 3: Inserting Multiple Rows
If you have feedback from multiple customers to insert at once, you can do so in a single `INSERT INTO` statement:

```
INSERT INTO movie_feedback (customer_id, film_id, feedback_text,
feedback_date)
VALUES
(101, 202, 'Great movie, really enjoyed the special effects!',
'2023-10-05'),
(102, 203, 'A bit slow in the middle but a fantastic ending.',
'2023-10-06'),
(103, 204, 'Not what I expected, but worth watching.', '2023-10-07');
```

This inserts three separate pieces of feedback from different customers about different films on different dates.

Step 4: Verifying Data Insertion

After inserting data, it's good practice to verify that the operation was successful and the data was inserted correctly:

```
SELECT * FROM movie_feedback WHERE feedback_date >=
'2023-10-05';
```

This query retrieves all feedback records submitted on or after October 5, 2023, allowing you to check the recently inserted data.

Best Practices

1. Data Consistency: Ensure the data you insert matches the data types and constraints defined in your table schema.
2. Batch Insertion: When possible, insert multiple rows with a single statement to reduce database write operations and improve performance.
3. Error Handling: Be prepared to handle errors that may arise during insertion, such as violations of primary key uniqueness or foreign key constraints. Consider using transactions to maintain data integrity.

Key Terms

- **Data Type:** In SQL, a data type specifies the kind of data that can be stored in a column of a database table. Each column in a table is required to have a data type, which dictates the nature of the data it can store (e.g., numerical, textual, date/time). Data types help ensure data integrity by restricting the kinds of data that can be inserted into a column. For example, trying to insert text into an `INTEGER` data type column will result in an error. Common SQL data types include `VARCHAR` for variable-length strings, `INT` for integers, `DECIMAL` for precise fractional numbers, and `DATE` for dates.

- **Table:** A table in SQL is a structured set of data made up of rows and columns. Each column in a table represents a specific attribute of the data (such as a name or price), and every row represents a single record that contains values for each attribute. Tables are the fundamental building blocks of relational databases, allowing for the organized storage of data in a way that supports efficient data retrieval and manipulation. For example, a `customers` table might store information about customers, with columns for customer ID, name, and contact information.

- **Primary Key:** A primary key is a column (or set of columns) in a database table that uniquely identifies each row in that table. Primary keys ensure that no two records have the same value in the primary key column(s), enforcing uniqueness and enabling efficient data retrieval. In relational databases, primary keys can also be used to establish relationships between tables, with foreign keys in other tables referencing the primary key values. For instance, in an `orders` table, an `order_id` column set as the primary key ensures that each order has a unique identifier.

- **INSERT Statement:** The `INSERT INTO` statement in SQL is used to add one or more new records to a database table. This command specifies the table to insert into, the columns to fill, and the values to insert into those columns. The `INSERT` statement can be used to insert data directly specified by the user or to insert the results of a query into a table. It plays a crucial role in populating tables with data, enabling further data analysis and manipulation. For example, `INSERT INTO customers (name, email) VALUES ('John Doe', 'john.doe@example.com');` adds a new customer record to the `customers` table.

Practice Multiple Choice Questions

1. What does `VARCHAR` data type represent in SQL?

- A) A fixed-length string
- B) A variable-length string
- C) A numeric value
- D) A date value

Answer: B) A variable-length string

2. Which SQL statement is used to add a new record to a table?

- A) UPDATE
- B) CREATE
- C) SELECT
- D) INSERT

Answer: D) INSERT

3. What is the purpose of a primary key in a table?

- A) To sort the table alphabetically
- B) To ensure each row has a unique identifier
- C) To join tables together
- D) To specify the table's data type

Answer: B) To ensure each row has a unique identifier

4. Which of the following is NOT a valid SQL data type?

- A) DATETIME
- B) SERIAL
- C) LINK
- D) INT

Answer: C) LINK

5. When creating a new table, what SQL statement is used?

- A) INSERT INTO
- B) UPDATE TABLE
- C) CREATE TABLE
- D) ALTER TABLE

Answer: C) CREATE TABLE

This lesson provides a foundational understanding of data types and tables in SQL, equipping you with the knowledge to structure and populate databases effectively for your data analysis projects.