Introduction to Relational Databases and SQL

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-11-01

Table des matières

1 Overview 2
2 What is a Database? 2
3 What is a Relational Database? 3
4 Key Concepts: Tables, Rows, and Columns 3
4.1 Example Table: customers 4
4.2 Example Table: orders 4
4.3 Example Table: Customers 4
5 Advantages of Relational Databases 4
6 Introduction to SQL 5
7 Various RDBMS Following the SQL Standard 5
8 Relational Algebra and SQL 6
9 SELECT 6
10 INSERT 7
10.1 UPDATE 7
10.2 DELETE 7
11 SQL Data Types in PostgreSQL 7
11.1 Example Table with Various Data Types 8
12 Primary Key 8
12.1 Declaring a Table with a Primary Key 9
12.2 Declaring a table with an Auto-generated Primary Key 9
12.3 Integrity Constraints 9
12.4 ALTER and DROP 10
12.5 Foreign Keys 10
13 Cascading Actions 11
14 Joins 12
14.1 Example Tables 12
14.2 Example ER Diagram 14
14.3 Example of an INNER JOIN 14

14.4 Example of a LEFT JOIN 15
14.5 Example of a RIGHT JOIN 15
14.6 Example of a FULL JOIN 16
15 Using GROUP BY in SQL 16
15.1 Group By Syntax 16
15.2 Example: Grouping and Counting Orders by Customer 16
15.3 Example: Calculating Total Sales by Product 17
16 Practical Examples in E-commerce 17
16.1 Example Tables 17
16.1.a Customers Table 17
16.1.b Products Table 18
16.1.c Orders Table 18
16.1.d Categories Table 18
16.1.e ProductCategories Table 18
16.2 Diagram 18
16.3 DDL Queries to Create and Populate the Database 19
16.4 Useful Queries 21
17 Summary 22
18 Final Thoughts 22

1 Overview

- **Topics**:
  - What is a Database?
  - Introduction to Relational Databases
  - Key Concepts: Tables, Rows, and Columns
  - Advantages of Relational Databases
  - Introduction to SQL
  - Various RDBMS Following the SQL Standard
  - Relational Algebra and SQL
  - Basic SQL Queries: SELECT, INSERT, UPDATE, DELETE
  - SQL Data Types in PostgreSQL
  - Primary Key and Integrity Constraints
  - Foreign Keys and Cascading Actions
  - Joins: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN
  - Using GROUP BY in SQL
  - Practical Examples in E-commerce

2 What is a Database?

- **Definition**:

- A database is an organized collection of data, generally stored and accessed electronically from a computer system.
- **Purpose**:
  - Databases are used to store, manage, and retrieve data efficiently.
- **Key Features**:
  - **Data Integrity**: Ensures accuracy and consistency of data.
  - **Data Security**: Protects data from unauthorized access.
  - **Scalability**: Handles increasing amounts of data and users.
  - **Backup and Recovery**: Provides mechanisms to restore data in case of failure.
- **Examples**:
  - **Customer Information**: Storing customer names, addresses, and contact details.
  - **Product Details**: Managing product names, descriptions, prices, and stock levels.
  - **Order History**: Tracking customer orders, statuses, and transaction details.

3 What is a Relational Database?

- **Relational Model**:
  - **Concept**: Proposed by E.F. Codd in 1970, the relational model organizes data into one or more tables (or « relations ») of columns and rows, with a unique key identifying each row.
  - **Relations vs. Tables**:
    - **Relation**: A theoretical concept representing an entity type, defined by a set of attributes.
    - **Table**: The physical implementation of a relation in a database, consisting of rows (tuples) and columns.
  - **Attributes vs. Columns**:
    - **Attribute**: A property or characteristic of an entity type in the relational model.
    - **Column**: The physical representation of an attribute in a table, defining the data type and constraints for the values in that column.
- **Relational Database**:
  - A type of database that stores data in tables.
  - tuples (rows) are identified by a primary key.
  - Tables can be related to each other through keys.

4 Key Concepts: Tables, Rows, and Columns

- **Table**:
  - A table is a collection of related data entries.
  - Example: Customers, Products, Orders
- **Row**:

- A row represents a single record in a table.
- Example: A single customer's information

- **Column**:
  - A column represents a specific attribute of a record.
  - Example: Customer Name, Email, Phone Number

4.1 Example Table: customers

customer_id	name	email	phone_number	country
1	John Doe	john@example.com	123-456-7890	USA
2	Jane Smith	jane@example.com	987-654-3210	USA
3	Pierre Dupont	pierre@example.fr	555-987-6543	France

4.2 Example Table: orders

order_id	order_date	customer_id
101	2023-01-15	1
102	2023-01-16	2
103	2023-01-17	1

4.3 Example Table: Customers

Custome-rID	Name	Email	Phone Number	Country
1	John Doe	john@example.com	123-456-7890	USA
2	Jane Smith	jane@example.com	987-654-3210	USA
3	Pierre Dupont	pierre@example.fr	555-987-6543	France

5 Advantages of Relational Databases

- **Organized Structure**:
  - Data is stored in a structured format using rows and columns.
- **Ease of Querying**:
  - Simple to query and retrieve data using SQL.
- **Support for Complex Queries**:
  - Capable of handling complex queries and transactions.
- **Data Integrity**:
  - Ensures accuracy and consistency of data through constraints and normalization.
- **Scalability**:

- Can handle large amounts of data and concurrent users efficiently.
- **Security**:
  - Provides robust security features to protect data, including access controls and encryption.

6 Introduction to SQL

- **SQL (Structured Query Language)**:
  - A standard language for accessing and manipulating relational databases.
  - Developed in the 1970s by IBM and standardized by ANSI and ISO.
  - The latest version is SQL:2016, which includes enhancements for JSON support, polymorphic table functions, and more.
- **Purpose**:
  - Used to perform tasks such as querying data, updating records, and managing database structures.
  - Supports Data Definition Language (DDL) for defining database schemas.
  - Supports Data Manipulation Language (DML) for inserting, updating, deleting, and retrieving data.
  - Supports Data Control Language (DCL) for controlling access to data.
  - Supports Transaction Control Language (TCL) for managing transactions.
- **Key Features**:
  - **Declarative Language**: Allows users to specify what data to retrieve rather than how to retrieve it.
  - **Portability**: SQL can be used across different database systems with minimal changes.
  - **Extensibility**: Supports extensions and procedural languages like PL/SQL and T-SQL.

7 Various RDBMS Following the SQL Standard

- **Relational Database Management Systems (RDBMS)**:
  - Systems used to manage relational databases.
  - They follow the SQL (Structured Query Language) standard for database operations, but there can be differences or extensions in how each RDBMS implements the standard.

RDBMS	Open Source	Key Features	Ideal Use Cases
PostgreSQL	Yes	Robustness, extensibility, advanced data types	Complex applications, data warehousing
MySQL	Yes	Speed, reliability, ease of use	Web applications, e-commerce

RDBMS	Open Source	Key Features	Ideal Use Cases
SQLite	Yes	Lightweight, serverless, self-contained	Embedded applications, small projects
Microsoft SQL Server	No	Integration with Microsoft products, enterprise features	Enterprise applications, business intelligence
Oracle Database	No	Scalability, advanced features	Large enterprises, mission-critical applications

8 Relational Algebra and SQL

- **Relational Algebra**:
  - A procedural query language that works on the relational model.
  - Provides a set of operations to manipulate relations (tables).
  - Fundamental operations include:
    - **Selection (σ)**: Selects rows that satisfy a given predicate.
    - **Projection (π)**: Selects specific columns from a table.
    - **Union (∪)**: Combines the results of two queries.
    - **Set Difference (−)**: Returns rows from one query that are not in another.
    - **Cartesian Product (×)**: Combines all rows from two tables.
    - **Join (⋈)**: Combines rows from two tables based on a related column.
- **SQL (Structured Query Language)**:
  - A declarative query language used to interact with relational databases.
  - SQL operations are based on relational algebra but are more user-friendly.
  - Key SQL operations include:
    - **SELECT**: Retrieves data from one or more tables.
    - **INSERT**: Adds new records to a table.
    - **UPDATE**: Modifies existing records in a table.
    - **DELETE**: Removes records from a table.
    - **JOIN**: Combines rows from two or more tables based on a related column.

9 SELECT

- Used to retrieve data from a database.
- Based on relational algebra operations such as selection (σ) and projection (π).

```sql
SELECT customer_id, name, email FROM customers
WHERE country = 'USA';
```

- **Relational Algebra Equivalent**:
  - **Projection (π)**: Selecting specific columns (CustomerID, Name, Email).

- **Selection (σ)**: Filtering rows based on conditions (Country = “USA”).

- Result:

customer_id	name	email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com

- **Note**: The `WHERE` clause is optional and is used to filter the rows based on specified conditions.

10 INSERT

- Used to add new records to a table.
- Example:

```
INSERT INTO customers (customer_id, name, email, phone_number, country)
VALUES (4, 'Alice Johnson', 'alice@example.com', '555-123-4567', 'Canada');
```

- Result:
  - A new record is added to the `customers` table with the specified values.

10.1 UPDATE

- Used to modify existing records in a table.
- Example:

```
UPDATE customers
SET email = 'john.doe@example.com'
WHERE customer_id = 1;
```

10.2 DELETE

- Used to remove records from a table.
- Example:

```
DELETE FROM customers
WHERE customer_id = 2;
```

11 SQL Data Types in PostgreSQL

- **SQL Data Types**:
  - Data types define the kind of data that can be stored in a table column.

    - Choosing the correct data type is crucial for data integrity and performance.
- Numeric Types
    - **INTEGER**: Whole numbers.
    - **DECIMAL**: Fixed-point numbers.
- Character Types
    - **VARCHAR**: Variable-length character strings.
    - **TEXT**: Variable-length character strings with no length limit.
- Date/Time Types
    - **DATE**: Calendar date (year, month, day).
    - **TIMESTAMP**: Date and time (without time zone).
- Boolean Type
    - **BOOLEAN**: Logical Boolean (true/false).
- Array Type
    - **ARRAY**: Array of values.

11.1 Example Table with Various Data Types

```
CREATE TABLE example_table (
  product_id SERIAL PRIMARY KEY,     -- Numeric Type: INTEGER (auto-incremented)
  price DECIMAL(10, 2),               -- Numeric Type: DECIMAL
  name VARCHAR(100),                  -- Character Type: VARCHAR
  description TEXT,                   -- Character Type: TEXT
  order_date DATE,                    -- Date/Time Type: DATE
  event_time TIMESTAMP,               -- Date/Time Type: TIMESTAMP
  is_active BOOLEAN,                  -- Boolean Type: BOOLEAN
  product_ids INTEGER[]               -- Array Type: INTEGER[]
);
```

12 Primary Key

In relational algebra, a primary key is a unique identifier for a tuple (row) in a relation (table). It ensures that no two tuples have the same value for the primary key attribute(s), thereby maintaining the integrity and uniqueness of the data within the relation.

Key characteristics of a primary key:

- **Uniqueness**: Each value of the primary key must be unique across the relation.
- **Non-nullability**: Primary key attributes cannot contain NULL values.
- **Minimality**: The primary key must be minimal, meaning no subset of the primary key attributes can uniquely identify a tuple.
- **Composite Key**: A primary key can be composed of multiple attributes, known as a composite key.

12.1 Declaring a Table with a Primary Key

```
CREATE TABLE orders (
  order_id INT,
  product_id INT,
  quantity INT,
  PRIMARY KEY (order_id)
);
```

12.2 Declaring a table with an Auto-generated Primary Key

- A primary key can be auto-generated by the database system.
- In PostgreSQL, you can use the `SERIAL` data type to auto-generate unique IDs for the primary key.
- The `SERIAL` type automatically increments the ID for each new record.

```
CREATE TABLE customers (
  customer_id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100),
  phone_number VARCHAR(15),
  country VARCHAR(50)
);
```

12.3 Integrity Constraints

- Integrity constraints are rules that ensure the accuracy and consistency of data in a database.
- Types of integrity constraints:
  - **Primary Key Constraint**: Ensures that each record has a unique identifier.
  - **Foreign Key Constraint**: Ensures that a value in one table matches a value in another table, creating a relationship between the tables.
  - **Unique Constraint**: Ensures that all values in a column are unique.
  - **Not Null Constraint**: Ensures that a column cannot have NULL values.
  - **Check Constraint**: Ensures that all values in a column satisfy a specific condition.

```
CREATE TABLE Employees (
  EmployeeID SERIAL PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  Email VARCHAR(100) UNIQUE,
```

```
  Salary DECIMAL CHECK (Salary > 0)
);
```

12.4 ALTER and DROP

- **ALTER**:
  - Used to modify an existing database object, such as a table.
  - Example:

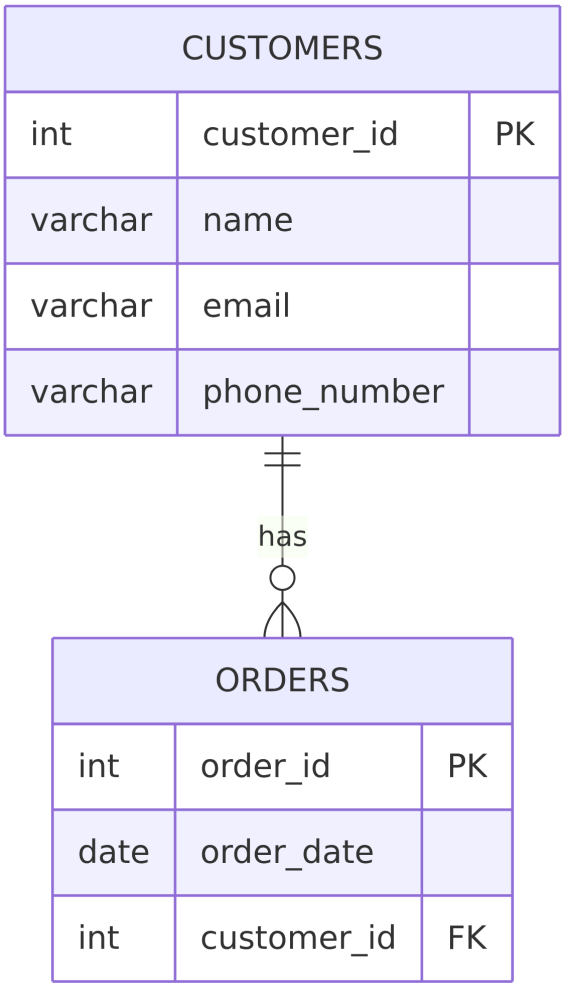
```
ALTER TABLE products
ADD COLUMN stock INT;
```

- **DROP**:
  - Used to delete an existing database object, such as a table.
  - Example:

```
DROP TABLE products;
```

12.5 Foreign Keys

- A foreign key is a field (or collection of fields) in one table that uniquely identifies a row in another table.
- It creates a relationship between two tables, ensuring referential integrity.
- Key characteristics:
  - **Referential Integrity**: Ensures that a value in the foreign key column(s) must match a value in the primary key column(s) of the referenced table.




```
CREATE TABLE orders (
  order_id SERIAL PRIMARY KEY,
  order_date DATE,
  customer_id INT,
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

13 Cascading Actions

- Cascading actions can be defined to automatically update or delete related rows when a referenced row is updated or deleted.
- Common cascading actions:
  - **ON DELETE CASCADE**: Automatically deletes related rows in the child table when a row in the parent table is deleted.

- **ON UPDATE CASCADE**: Automatically updates related rows in the child table when a row in the parent table is updated.

```
CREATE TABLE Orders (
  OrderID SERIAL PRIMARY KEY,
  OrderDate DATE,
  CustomerID INT,
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

14 Joins

- **JOIN**:
  - Used to combine rows from two or more tables based on a related column.
  - Types of Joins:
    - **INNER JOIN**: Returns records that have matching values in both tables.
    - **LEFT JOIN**: Returns all records from the left table, and the matched records from the right table.
    - **RIGHT JOIN**: Returns all records from the right table, and the matched records from the left table.
    - **FULL JOIN**: Returns all records when there is a match in either left or right table.

14.1 Example Tables

Custome-rID	Name	Email	Phone Number	Country
1	John Doe	john@example.com	123-456-7890	USA
2	Jane Smith	jane@example.com	987-654-3210	USA
3	Pierre Dupont	pierre@example.fr	555-987-6543	France

order_id	order_date	customer_id
101	2023-01-15	1
102	2023-01-16	2
103	2023-01-17	1

```
-- Create Customers Table
CREATE TABLE customers (
```

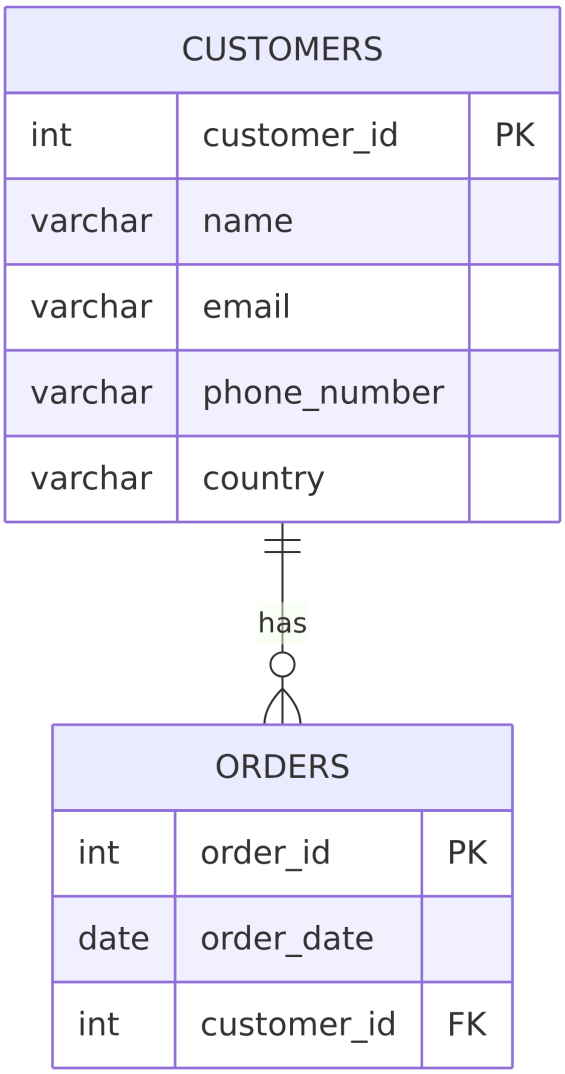
```sql
  customer_id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100),
  phone_number VARCHAR(20),
  country VARCHAR(50)
);

-- Insert Data into Customers Table
INSERT INTO customers (name, email, phone_number, country) VALUES
('John Doe', 'john@example.com', '123-456-7890', 'USA'),
('Jane Smith', 'jane@example.com', '987-654-3210', 'USA'),
('Pierre Dupont', 'pierre@example.fr', '555-987-6543', 'France');

-- Create Orders Table
CREATE TABLE orders (
  order_id SERIAL PRIMARY KEY,
  order_date DATE,
  customer_id INT,
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- Insert Data into Orders Table
INSERT INTO orders (order_date, customer_id) VALUES
('2023-01-15', 1),
('2023-01-16', 2),
('2023-01-17', 1);
```

14.2 Example ER Diagram



14.3 Example of an INNER JOIN

- **INNER JOIN**:
  - Returns records that have matching values in both tables.
- SQL Query:

```
SELECT customers.customer_id, customers.name, orders.order_date
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id;
```

- Result:

customer_id	name	order_date
1	John Doe	2023-01-15
2	Jane Smith	2023-01-16
1	John Doe	2023-01-17

14.4 Example of a LEFT JOIN

- **LEFT JOIN**:
  - Returns all records from the left table, and the matched records from the right table.
- SQL Query:

```
SELECT customers.customer_id, customers.name, orders.order_date
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

- Result:

customer_id	name	order_date
1	John Doe	2023-01-15
2	Jane Smith	2023-01-16
1	John Doe	2023-01-17
3	Pierre Dupont	NULL

14.5 Example of a RIGHT JOIN

- **RIGHT JOIN**:
  - Returns all records from the right table, and the matched records from the left table.
- SQL Query:

```
SELECT customers.customer_id, customers.name, orders.order_date
FROM customers
RIGHT JOIN orders ON customers.customer_id = orders.customer_id;
```

- Result:

customer_id	name	order_date
1	John Doe	2023-01-15
2	Jane Smith	2023-01-16
1	John Doe	2023-01-17

14.6 Example of a FULL JOIN

- **FULL JOIN**:
  - Returns all records when there is a match in either left or right table.
- SQL Query:

```
SELECT customers.customer_id, customers.name, orders.order_date
FROM customers
FULL JOIN orders ON customers.customer_id = orders.customer_id;
```

- Result:

customer_id	name	order_date
1	John Doe	2023-01-15
2	Jane Smith	2023-01-16
1	John Doe	2023-01-17
3	Pierre Dupont	NULL

15 Using GROUP BY in SQL

- **GROUP BY Clause**:
  - The `GROUP BY` clause is used to group rows that have the same values in specified columns into summary rows.
  - It is often used with aggregate functions (COUNT, SUM, AVG, MAX, MIN) to perform calculations on each group of rows.

15.1 Group By Syntax

```
SELECT column1, aggregate_function(column2)
FROM table_name
WHERE condition
GROUP BY column1;
```

15.2 Example: Grouping and Counting Orders by Customer

- **Scenario**:

- You want to find out how many orders each customer has placed.

- **SQL Query**:

```sql
SELECT customer_id, COUNT(order_id) AS order_count
FROM orders
GROUP BY customer_id;
```

- **Result**:

customer_id	order_count
2	1
1	2

15.3 Example: Calculating Total Sales by Product

- **Scenario**:
  - You want to calculate the total sales amount for each product.
- **SQL Query**:

```sql
SELECT product_id, SUM(quantity * price) AS total_sales
FROM order_details
GROUP BY product_id;
```

- **Result**:

product_id	total_sales
101	4999.95
102	2999.94
103	1999.96

16 Practical Examples in E-commerce

16.1 Example Tables

16.1.a Customers Table

customer_id	name	email	phone_number	country
1	John Doe	john@example.com	123-456-7890	USA
2	Jane Smith	jane@example.com	987-654-3210	USA

customer_id	name	email	phone_number	country
3	Pierre Dupont	pierre@example.fr	555-987-6543	France

16.1.b Products Table

product_id	product_name	price
101	Laptop	999.99
102	Smartphone	499.99

16.1.c Orders Table

order_id	customer_id	product_id	quantity	status
1001	1	101	2	Pending
1002	2	102	1	Shipped

16.1.d Categories Table

category_id	category_name
1	Electronics
2	Furniture

16.1.e ProductCategories Table

product_id	category_id
101	1
102	1

16.2 Diagram

```
erDiagram
    CUSTOMERS {
        int customer_id PK
        varchar name
        varchar email
        varchar phone_number
        varchar country
    }
    PRODUCTS {
        int product_id PK
        varchar product_name
```

```
        decimal price
    }
    ORDERS {
        int order_id PK
        int customer_id FK
        int product_id FK
        int quantity
        varchar status
    }
    CATEGORIES {
        int category_id PK
        varchar category_name
    }
    PRODUCTCATEGORIES {
        int product_id FK
        int category_id FK
    }
    CUSTOMERS ||--o{ ORDERS : places
    PRODUCTS ||--o{ ORDERS : contains
    PRODUCTS ||--o{ PRODUCTCATEGORIES : belongs_to
    CATEGORIES ||--o{ PRODUCTCATEGORIES : includes
```

16.3 DDL Queries to Create and Populate the Database

```sql
-- Create Customers Table
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    phone_number VARCHAR(15),
    country VARCHAR(50)
);

-- Create Products Table
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    price DECIMAL(10, 2)
);

-- Create Orders Table
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INT REFERENCES customers(customer_id),
```

```
    product_id INT REFERENCES products(product_id),
    quantity INT,
    status VARCHAR(50)
);

-- Create Categories Table
CREATE TABLE categories (
    category_id SERIAL PRIMARY KEY,
    category_name VARCHAR(100)
);

-- Create ProductCategories Table
CREATE TABLE product_categories (
    product_id INT REFERENCES products(product_id),
    category_id INT REFERENCES categories(category_id),
    PRIMARY KEY (product_id, category_id)
);

-- Insert Data into Customers Table
INSERT INTO customers (customer_id,name, email, phone_number, country) VALUES
(1,'John Doe', 'john@example.com', '123-456-7890', 'USA'),
(2,'Jane Smith', 'jane@example.com', '987-654-3210', 'USA'),
(3,'Pierre Dupont', 'pierre@example.fr', '555-987-6543', 'France');

-- Insert Data into Products Table
INSERT INTO products (product_id,product_name, price) VALUES
(101,'Laptop', 999.99),
(102,'Smartphone', 499.99);

-- Insert Data into Orders Table
INSERT INTO orders (customer_id, product_id, quantity, status) VALUES
(1, 101, 2, 'Pending'),
(2, 102, 1, 'Shipped');

-- Insert Data into Categories Table
INSERT INTO categories (category_name) VALUES
('Electronics'),
('Furniture');

-- Insert Data into ProductCategories Table
INSERT INTO product_categories (product_id, category_id) VALUES
(101, 1),
(102, 1);
```

16.4 Useful Queries

- **Querying Product Information**:
  - Example:

```
SELECT product_id, product_name, price FROM products;
```

  - Result:

product_id	product_name	price
101	Laptop	999.99
102	Smartphone	499.99

- **Adding a New Order**:
  - Example:

```
INSERT INTO orders (customer_id, product_id, quantity, status)
VALUES (1, 101, 2, 'Pending');
```

- **Updating Order Status**:
  - Example:

```
UPDATE orders
SET status = 'Shipped'
WHERE order_id = 1;
```

- **Deleting a Customer Record**:
  - Example:

```
DELETE FROM customers
WHERE customer_id = 3;
```

- **Querying Products by Category**:
  - Example:

```
SELECT p.product_id, p.product_name, c.category_name
FROM products p
JOIN product_categories pc ON p.product_id = pc.product_id
JOIN categories c ON pc.category_id = c.category_id;
```

  - Result:

product_id	product_name	category_name
101	Laptop	Electronics
102	Smartphone	Electronics

17 Summary

- **Key Points**:
  - Databases are essential for storing and managing data.
  - Relational databases organize data into tables, rows, and columns.
  - SQL is the standard language for interacting with relational databases.
  - Basic SQL queries include SELECT, INSERT, UPDATE, and DELETE.
  - IDs, joins, and foreign keys are crucial for establishing relationships between tables.
  - DDL commands are used to define and manage database structures.
  - PostgreSQL and MySQL are popular relational database management systems.

18 Final Thoughts

- **Quick Bootstrap**:
  - This lecture is a quick bootstrap to get you started with relational databases and SQL.
  - It is not a full course, so further study and practice are recommended to gain a deeper understanding.
- **Importance of Databases**:
  - Understanding databases and SQL is crucial for managing data in e-commerce.
  - Databases ensure data integrity, security, and efficient retrieval, which are essential for any data-driven application.
  - Keep practicing and exploring to become proficient in SQL and database management.