ENTRI elevate

SQL TUTORIAL

SQL is a standard language for storing, manipulating and retrieving data in databases.

Index

Introduction to SQL

What is SQL?

</> SQL stands for Structured Query Language, and it is used to manage data in relational databases.

</> It is a standard language that allows users to create, update, and retrieve data from databases.

Why Learn SQL?

</> SQL is a highly sought-after skill in the IT industry.

</> It is widely used in data analysis, business intelligence, and web development.

</> SQL is easy to learn and has a simple syntax.

Relational Databases

</> A relational database is a collection of tables that are related to each other.

</> Each table represents a specific entity, such as a customer, product, or order.

</> Tables are connected through relationships, such as primary keys and foreign keys.

Introduction to SQL

SQL Syntax

</> SQL uses a specific syntax for querying databases.

</> A SQL statement is composed of keywords, clauses, and expressions.

</> SQL statements are terminated with a semicolon (;)

SQL Basics

Select Statement

The SELECT statement is used to retrieve data from one or more tables in a database. It specifies the columns to retrieve and the table to retrieve them from. The basic syntax of the SELECT statement is:

SELECT column1, column2, ... FROM table_name;

The "SELECT" keyword is followed by the name of the columns to retrieve, separated by commas. The "FROM" keyword is followed by the name of the table to retrieve the data from.

SQL Basics

Where Clause

The WHERE clause is used to filter data based on a specific condition. It is used in conjunction with the SELECT statement to retrieve only the rows that meet the specified condition. The basic syntax of the WHERE clause is:

SELECT column1, column2, ... FROM table_name WHERE condition;

The "WHERE" keyword is followed by the condition that the data must meet. The condition can be a simple comparison, such as "column_name = value", or it can be a complex expression, such as "column_name LIKE '%value%'".

SQL Basics

Order By Clause

The ORDER BY clause is used to sort data in ascending or descending order based on one or more columns. It is used in conjunction with the SELECT statement to retrieve data in a specific order. The basic syntax of the ORDER BY clause is:

SELECT column1, column2, ... FROM table_name ORDER BY column_name ASC/DESC;

The "ORDER BY" keyword is followed by the name of the column to sort the data by. The "ASC" keyword is used to sort the data in ascending order, while the "DESC" keyword is used to sort the data in descending order.

SQL Basics

Group By Clause

The GROUP BY clause is used to group data based on a specific column or columns. It is used in conjunction with the SELECT statement to retrieve data grouped by one or more columns. The basic syntax of the GROUP BY clause is:

```
SELECT column1, COUNT(column2) FROM table_name
GROUP BY column1;
```

The "GROUP BY" keyword is followed by the name of the column or columns to group the data by. The COUNT function is used to count the number of occurrences of each value in the specified column.

SQL Basics

Join Statement

The JOIN statement is used to combine data from two or more tables based on a common column. It is used in conjunction with the SELECT statement to retrieve data from multiple tables. The basic syntax of the JOIN statement is:

SELECT column1, column2, ... FROM table1 JOIN table2 ON table1.column_name = table2.column_name;

The "JOIN" keyword is used to combine data from two or more tables. The "ON" keyword is used to specify the column or columns that the tables are joined on.

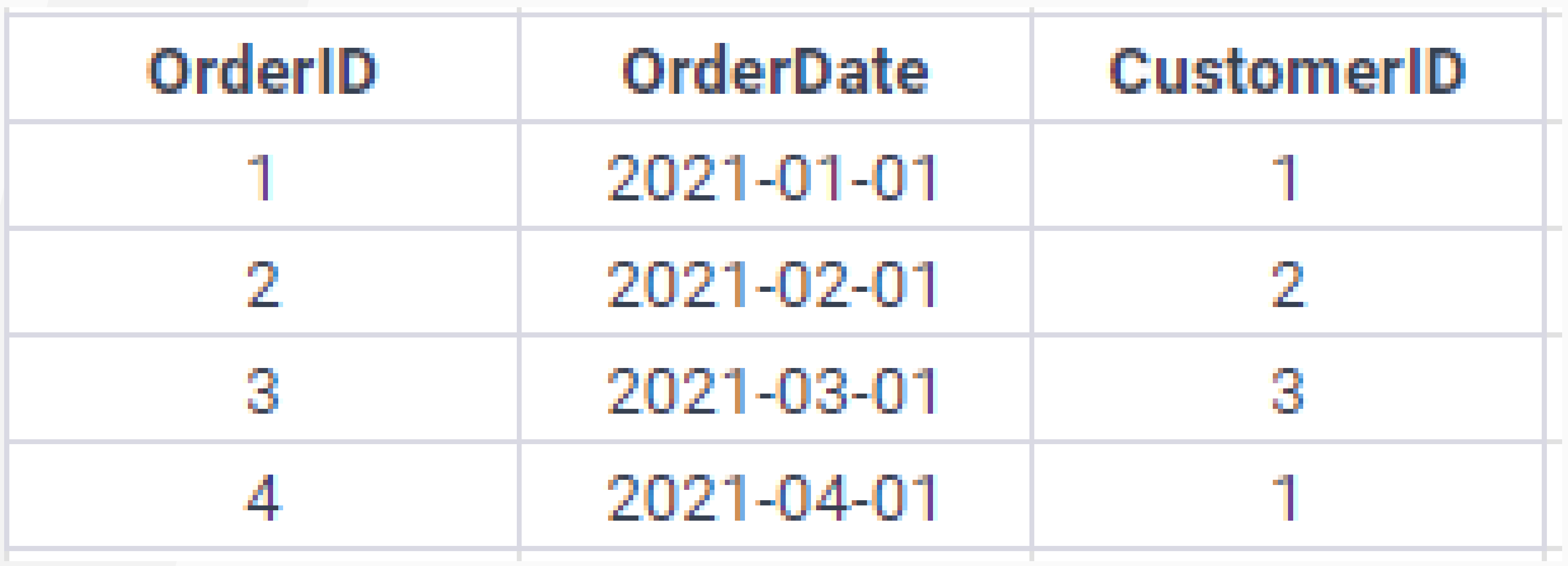
SQL Basics

Inner Join

An INNER JOIN returns only the rows from both tables that match the join condition. In other words, only the rows that have matching values in both tables are included in the result.

Example:
Consider the following two tables:

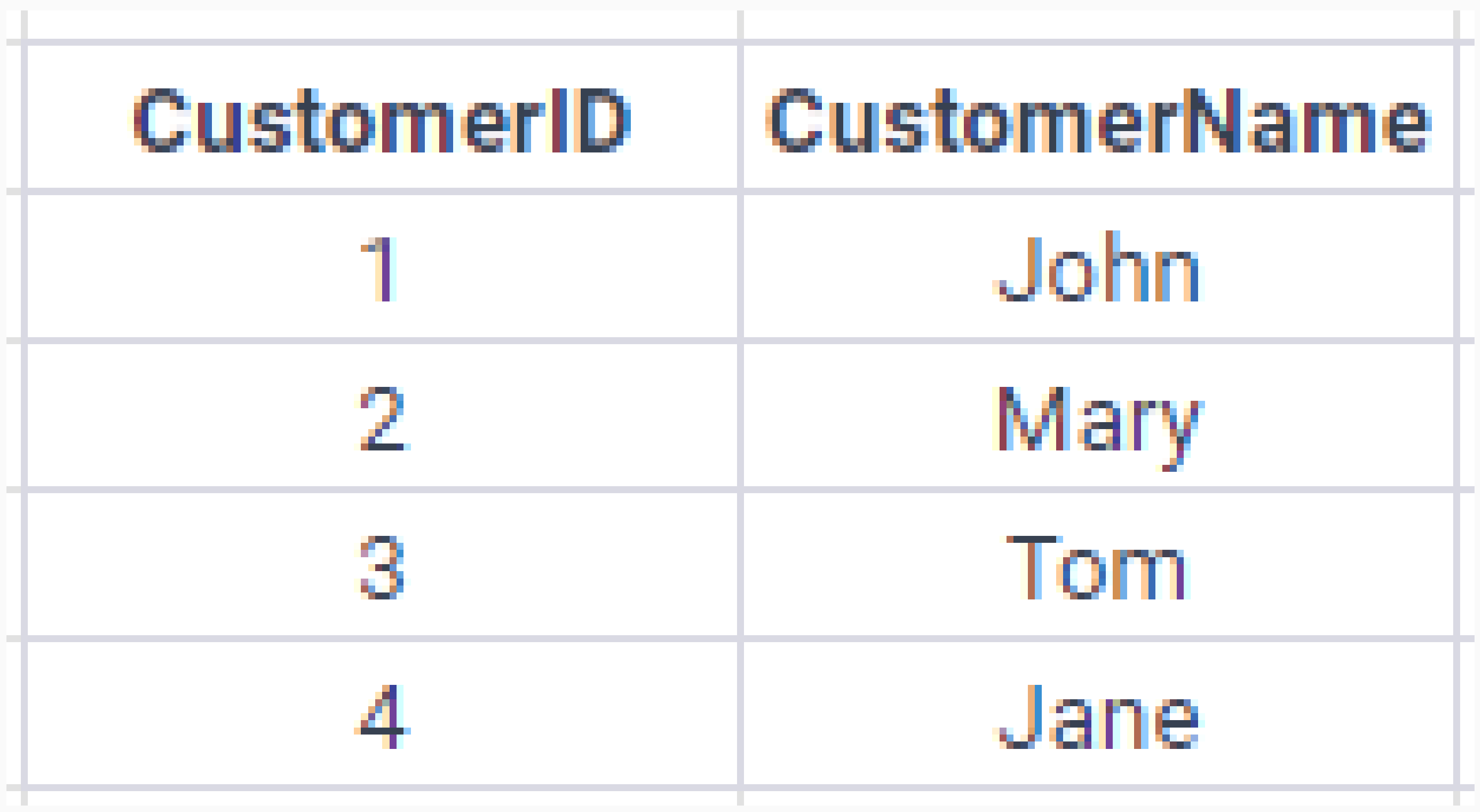
Table1: Orders



OrderID	OrderDate	CustomerID
1	2021-01-01	1
2	2021-02-01	2
3	2021-03-01	3
4	2021-04-01	1

SQL Basics

Table2: Customers



CustomerID	CustomerName
1	John
2	Mary
3	Tom
4	Jane

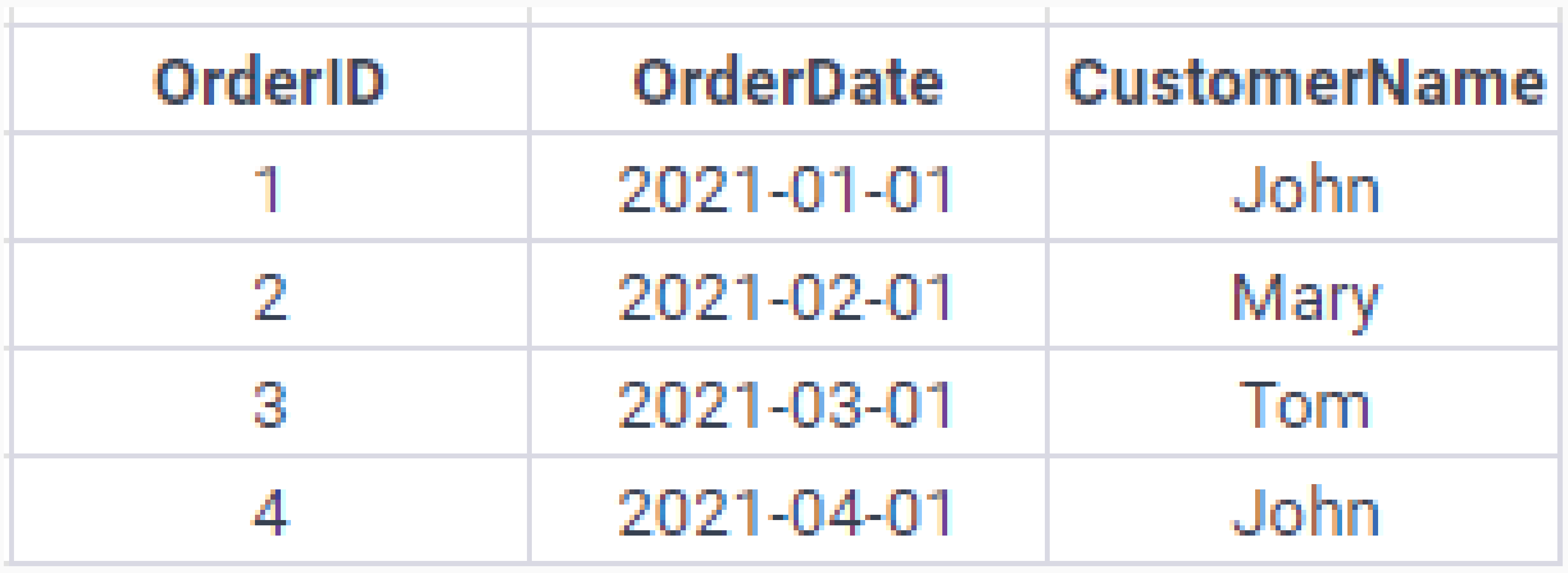
To retrieve a list of all orders and their respective customer names, we can use an INNER JOIN:

```
SELECT Orders.OrderID, Orders.OrderDate,
Customers.CustomerName
FROM Orders
INNER JOIN Customers

ON Orders.CustomerID = Customers.CustomerID;
```

SQL Basics

The result of the above query will be:



OrderID	OrderDate	CustomerName
1	2021-01-01	John
2	2021-02-01	Mary
3	2021-03-01	Tom
4	2021-04-01	John

Note that only the rows that have matching values in both tables are included in the result. The order with OrderID 4 is associated with the customer John because both have the same CustomerID value of 1.

SQL Basics

Outer Join

An OUTER JOIN returns all the rows from one table and the matching rows from the other table. There are three types of OUTER JOINS: LEFT OUTER JOIN, RIGHT OUTER JOIN, and FULL OUTER JOIN.

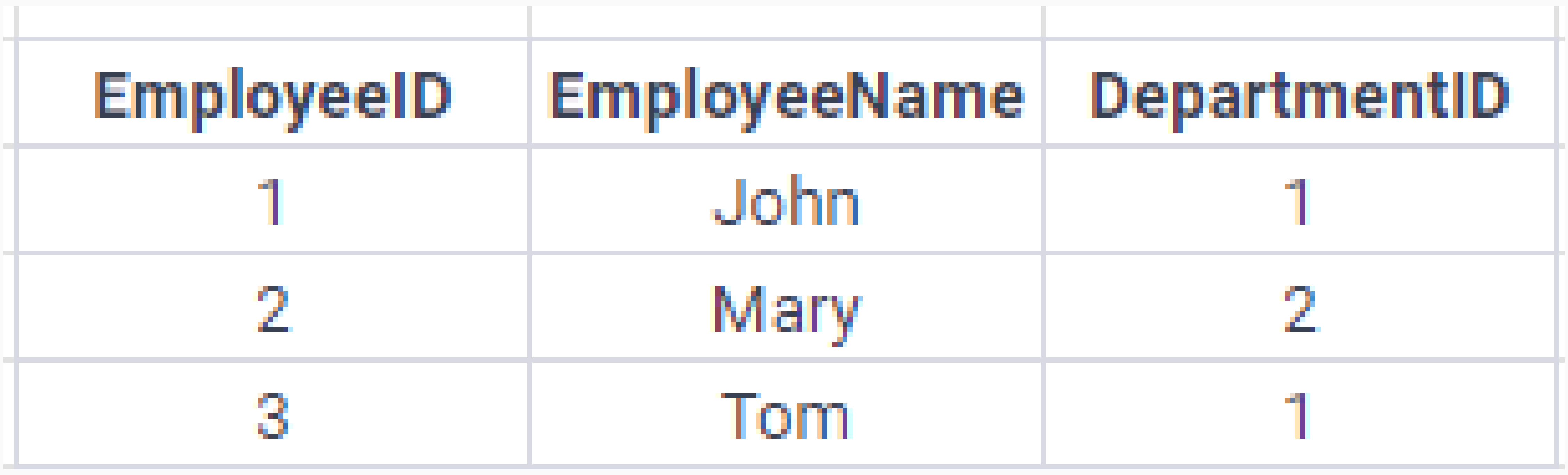
LEFT OUTER JOIN

The LEFT OUTER JOIN returns all the rows from the left table and the matching rows from the right table. If there are no matching rows in the right table, the result will contain NULL values.

SQL Basics

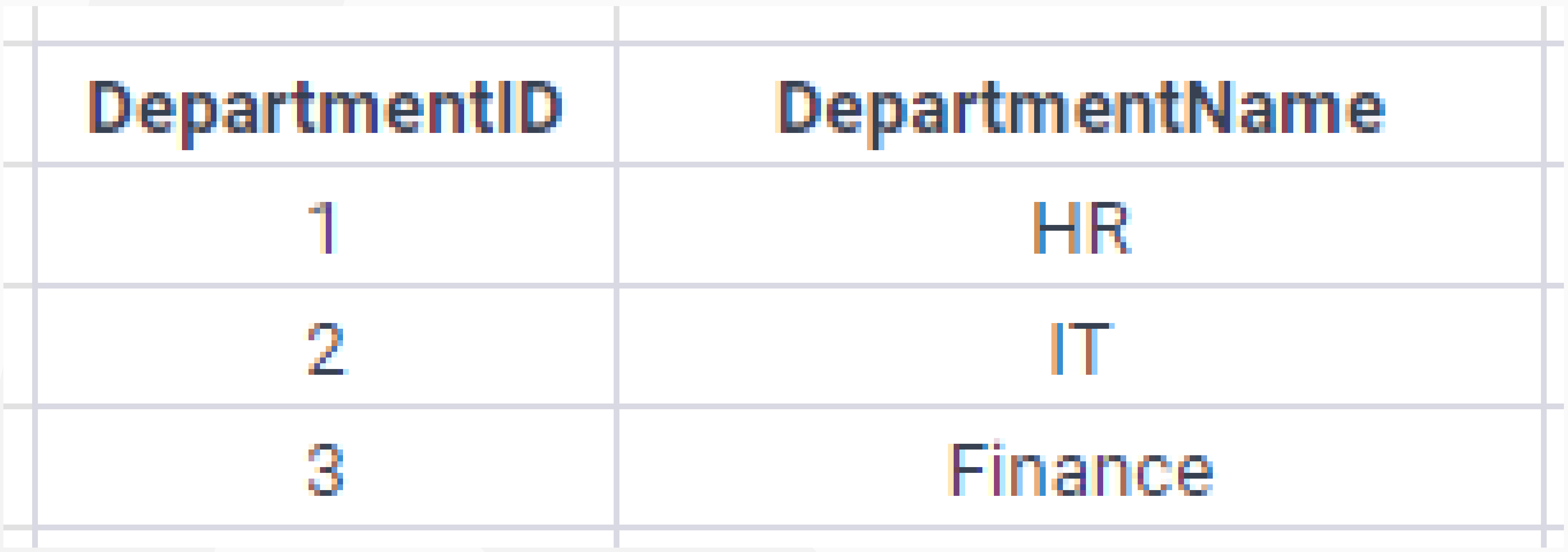
Example:
Consider the following two tables:

Table1: Employees



EmployeeID	EmployeeName	DepartmentID
1	John	1
2	Mary	2
3	Tom	1

Table2: Departments



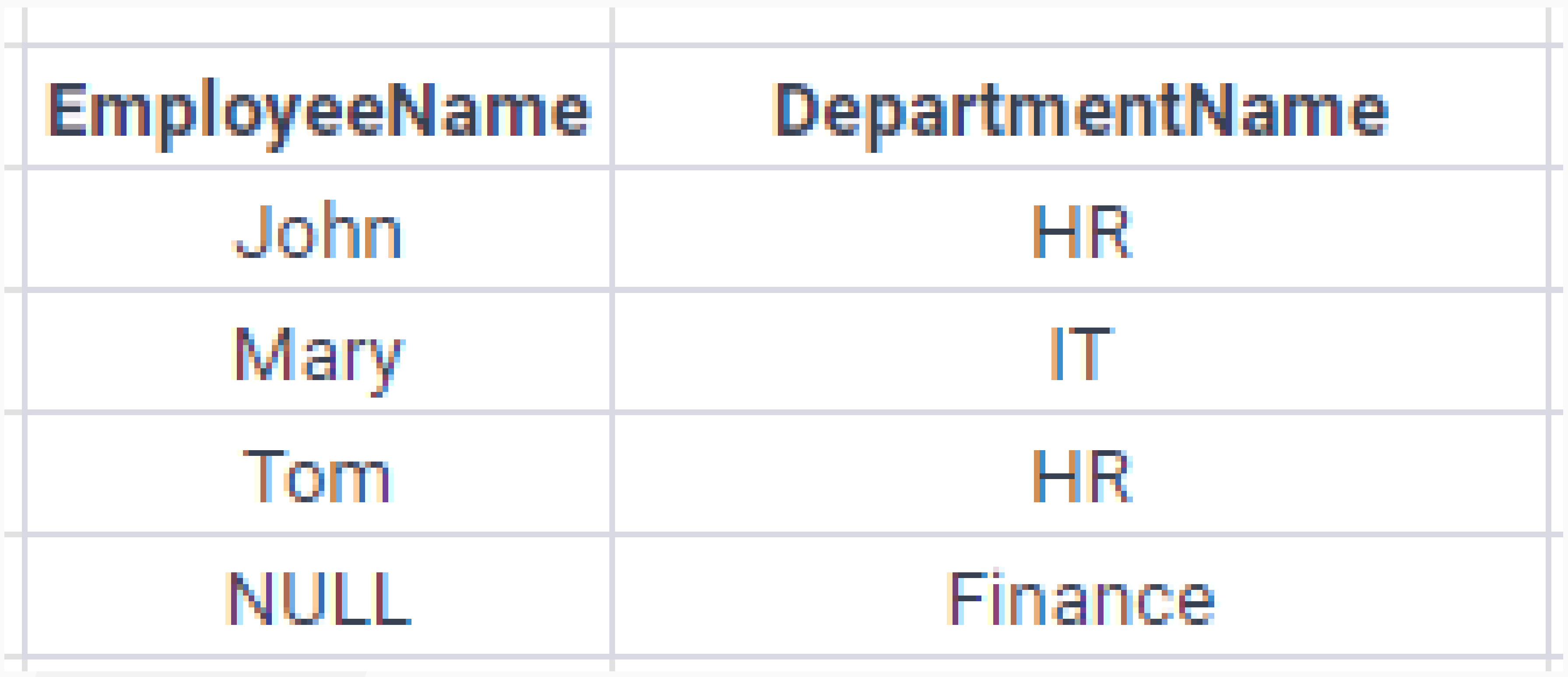
DepartmentID	DepartmentName
1	HR
2	IT
3	Finance

To retrieve all the employees and their respective departments, including those employees who do not belong to any department, we can use a LEFT OUTER JOIN:

SQL Basics

```
SELECT Employees.EmployeeName,
Departments.DepartmentName
FROM Employees
LEFT OUTER JOIN Departments
ON Employees.DepartmentID =
Departments.DepartmentID;
```

The result of the above query will be:



EmployeeName	DepartmentName
John	HR
Mary	IT
Tom	HR
NULL	Finance

Note that the last row contains NULL values for both the EmployeeName and DepartmentName columns, since there is no matching row in the Departments table for the employee with no department.

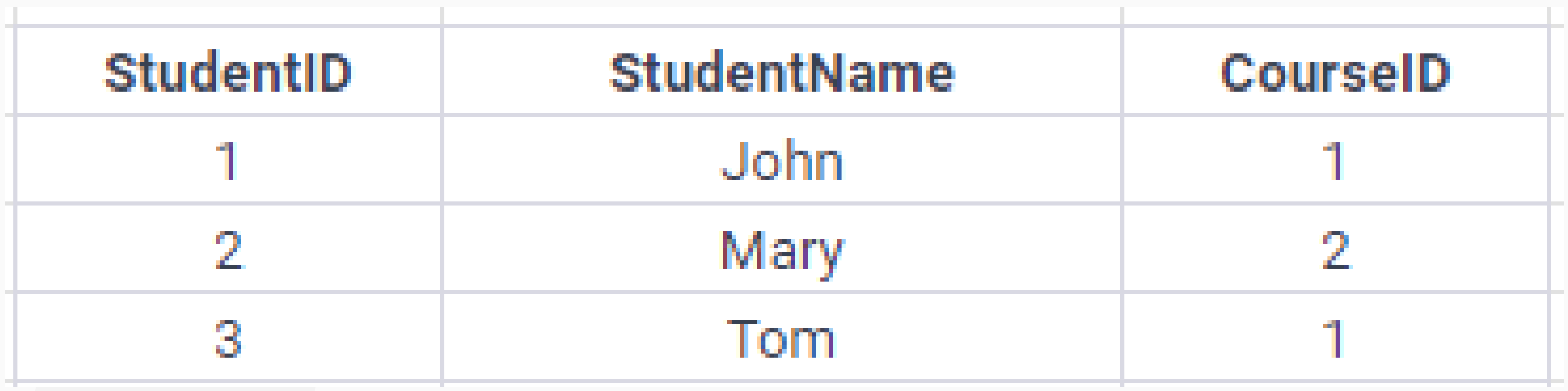
SQL Basics

UNION Statement

The UNION statement is used to combine the results of two or more SELECT statements into a single result set. The columns in each SELECT statement must have the same data type, and the number of columns in each SELECT statement must be the same.

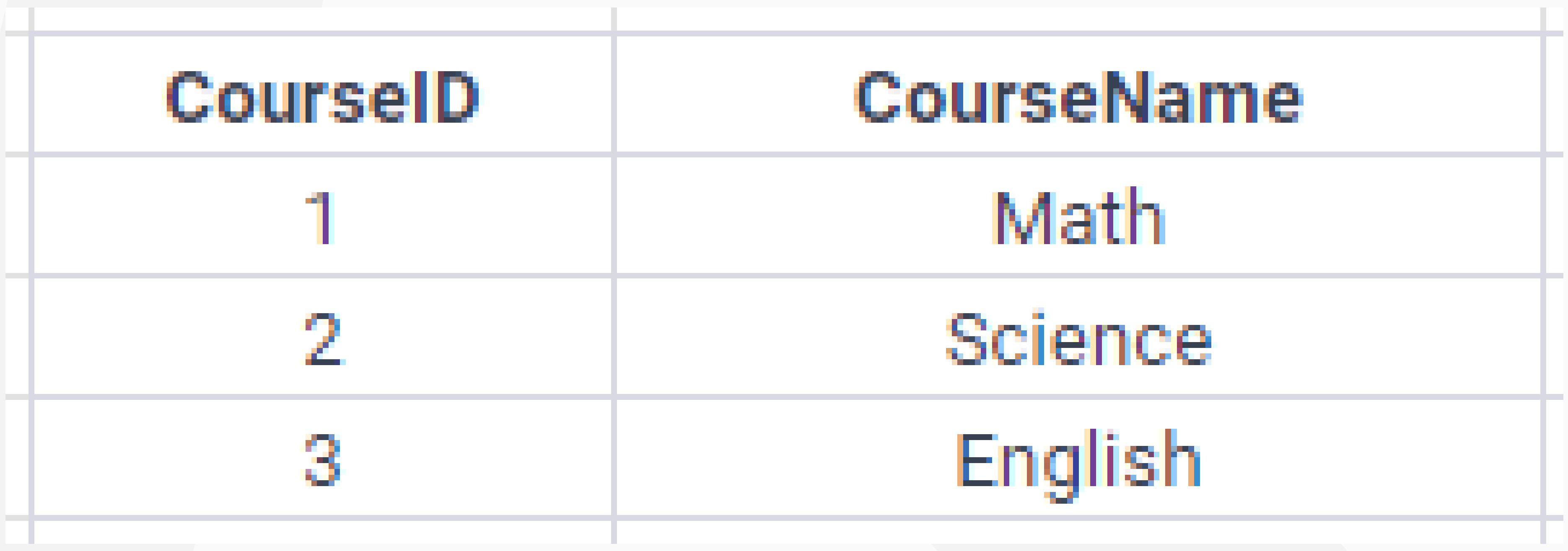
Example:
Consider the following two tables:

Table1: Students



StudentID	StudentName	CourseID
1	John	1
2	Mary	2
3	Tom	1

Table2: Courses



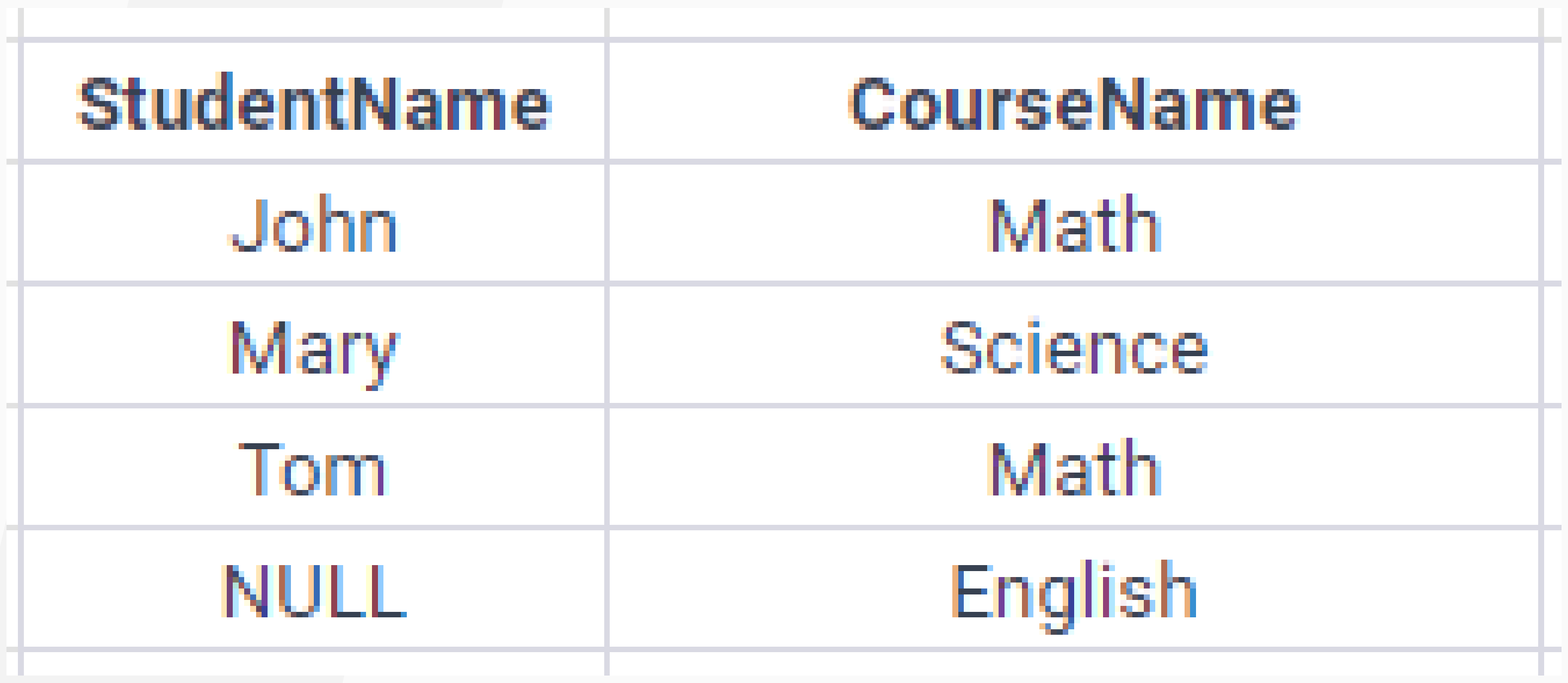
CourseID	CourseName
1	Math
2	Science
3	English

SQL Basics

To retrieve a list of all students and courses, we can use a UNION statement:

```
SELECT StudentName, CourseName FROM Students
INNER JOIN Courses ON Students.CourseID =
Courses.CourseID
UNION
SELECT NULL, CourseName FROM Courses
WHERE CourseID NOT IN (SELECT CourseID FROM
Students);
```

The result of the above query will be:



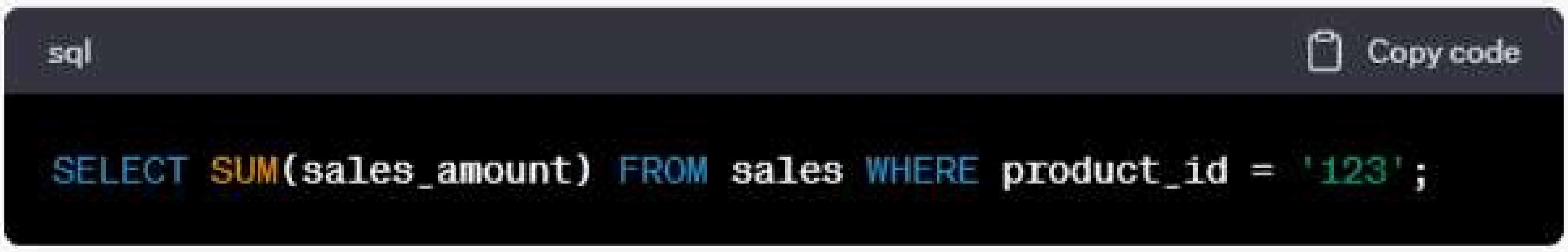
StudentName	CourseName
John	Math
Mary	Science
Tom	Math
NULL	English

Note that the last row contains NULL values for the StudentName column, since there is no matching row in the Students table for the English course.

SQL Functions

Aggregate Functions

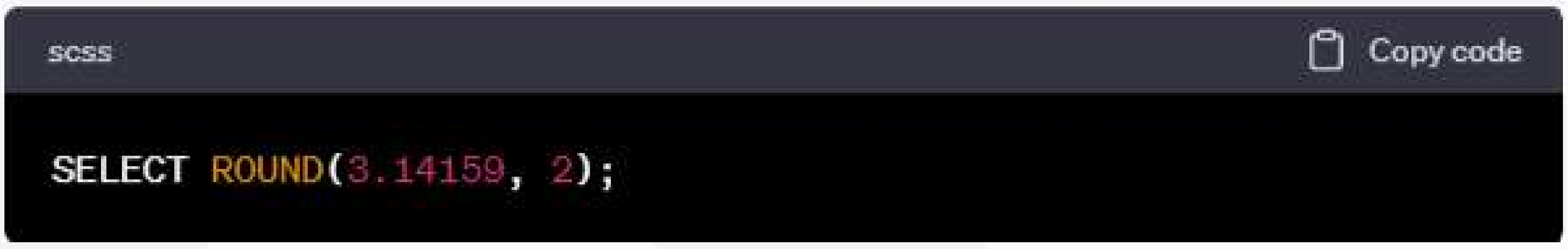
These functions operate on a group of rows and return a single value. The common aggregate functions in SQL are COUNT, SUM, AVG, MAX, and MIN. For example, to calculate the total sales amount for a specific product, you can use the SUM function as follows:



```sql
SELECT SUM(sales_amount) FROM sales WHERE product_id = '123';
```

Mathematical Functions

These functions perform mathematical calculations on the given values. Some examples of mathematical functions in SQL are ABS (absolute value), ROUND (rounding off), and MOD (modulus). Here is an example of using the ROUND function to round off a decimal value:



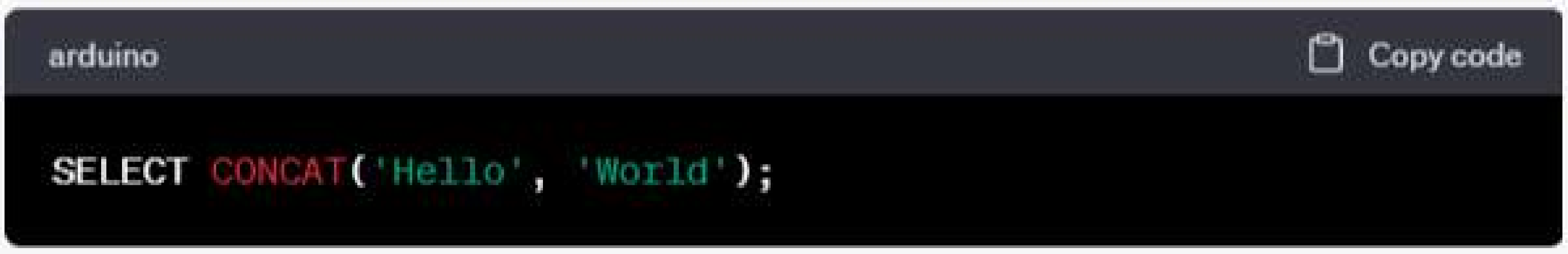
```scss
SELECT ROUND(3.14159, 2);
```

This will return the value 3.14, rounded to 2 decimal places.

SQL Functions

String Functions

These functions manipulate strings or character data. Some commonly used string functions in SQL are CONCAT (concatenation), SUBSTRING (substring extraction), and LENGTH (length of a string). Here's an example of using the CONCAT function to concatenate two strings:

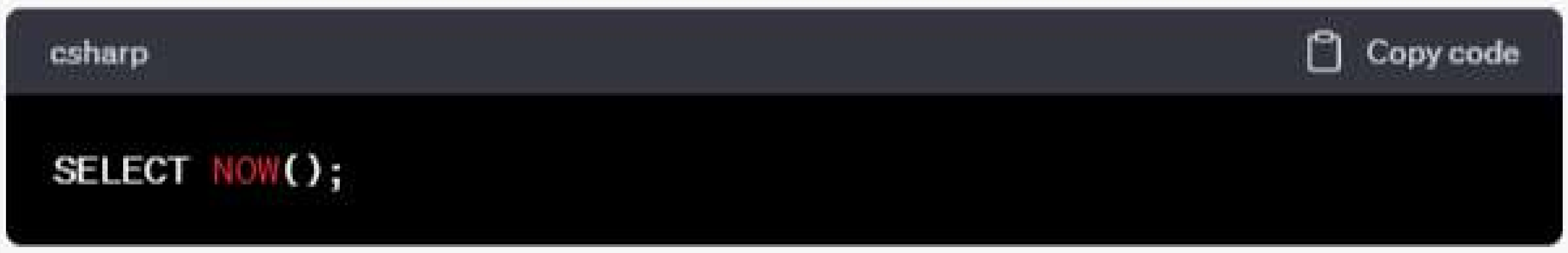


```
SELECT CONCAT('Hello', 'World');
```

This will return the value "HelloWorld".

Date and Time Functions

These functions manipulate date and time values. Examples of date and time functions include DATE (date format), YEAR (extract year from a date), and NOW (current date and time). Here is an example of using the NOW function to get the current date and time:



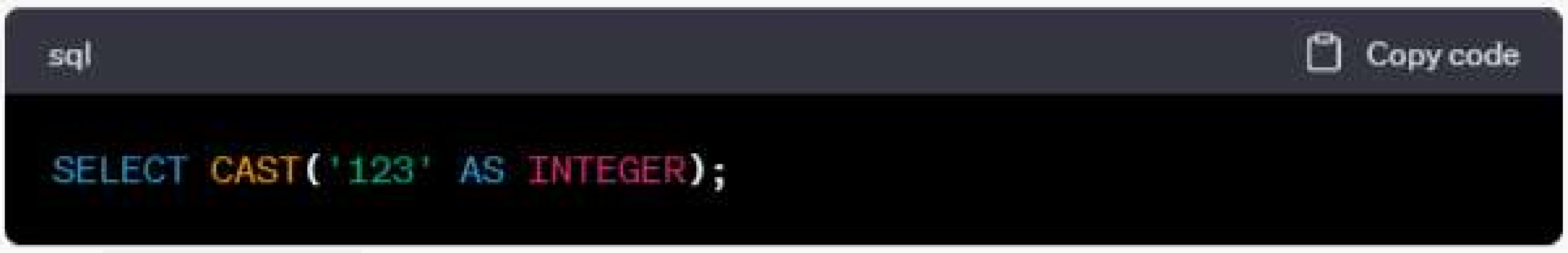
```
SELECT NOW();
```

This will return the current date and time in a standard format.

SQL Functions

Conversion Functions

These functions convert one data type to another. Some common conversion functions in SQL are CAST (explicit conversion), CONVERT (implicit conversion), and TO_CHAR (convert to character). Here's an example of using the CAST function to convert a string value to an integer:



```sql
SELECT CAST('123' AS INTEGER);
```

This will return the value 123 as an integer.

SQL Advanced Concepts

Subqueries

A subquery is a query that is nested within another query. The results of the subquery are used by the outer query to perform further processing. Subqueries are often used to retrieve data based on a condition that is not easily expressed using a single query. Here's an example of a subquery to retrieve the names of all customers who have placed orders:

Example: SELECT customer_name FROM customers WHERE customer_id IN (SELECT customer_id FROM orders);

Views

A view is a virtual table that is created by a query. It can be used like a table in SQL queries, but it does not store any data on its own. Views are often used to simplify complex queries or to provide controlled access to sensitive data. Here's an example of creating a view to retrieve the names and email addresses of all customers:

Example: CREATE VIEW customer_emails AS SELECT customer_name, email FROM customers;

SQL Advanced Concepts

Indexes

An index is a data structure that is used to speed up queries by providing quick access to data based on a specific column or set of columns. Indexes are often used on columns that are frequently used in WHERE clauses or JOIN operations. Here's an example of creating an index on the "product_name" column in the "products" table:

Example: CREATE INDEX product_name_index ON products (product_name);

SQL Advanced Concepts

Transactions

A transaction is a sequence of one or more SQL statements that are executed as a single unit of work. Transactions are used to ensure that all the statements in a sequence are executed successfully, or none of them are executed at all. Transactions are often used in applications that require data consistency, such as banking or e-commerce. Here's an example of a transaction that transfers funds from one account to another:

Example:

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 500 WHERE
account_id = '123';
UPDATE accounts SET balance = balance + 500 WHERE
account_id = '456';
COMMIT;
```

SQL Advanced Concepts

Stored Procedures

A stored procedure is a set of SQL statements that are stored in the database and can be called by other SQL statements or by an application. Stored procedures are often used to simplify complex operations or to enforce business rules. Here's an example of creating a stored procedure to insert a new customer into the "customers" table:

Example:

```
CREATE PROCEDURE add_customer (IN name
VARCHAR(50), IN email VARCHAR(50))
BEGIN
 INSERT INTO customers (customer_name, email)
VALUES (name, email);
END;
```

SQL Advanced Concepts

Triggers

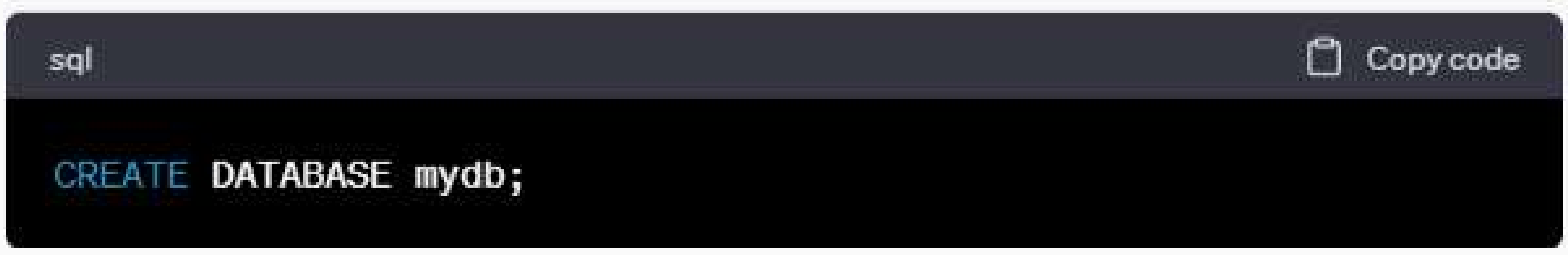
A trigger is a special type of stored procedure that is automatically executed in response to a specific event, such as an INSERT, UPDATE, or DELETE operation on a table. Triggers are often used to enforce business rules or to audit changes to data. Here's an example of creating a trigger to record changes to the "orders" table:

Example:
```
CREATE TRIGGER orders_audit AFTER INSERT,
UPDATE, DELETE ON orders
FOR EACH ROW
BEGIN
  INSERT INTO orders_audit (event_time, event_type,
order_id) VALUES (NOW(), 'INSERT', NEW.order_id);
END;
```

Hands-On Examples

Creating a Database

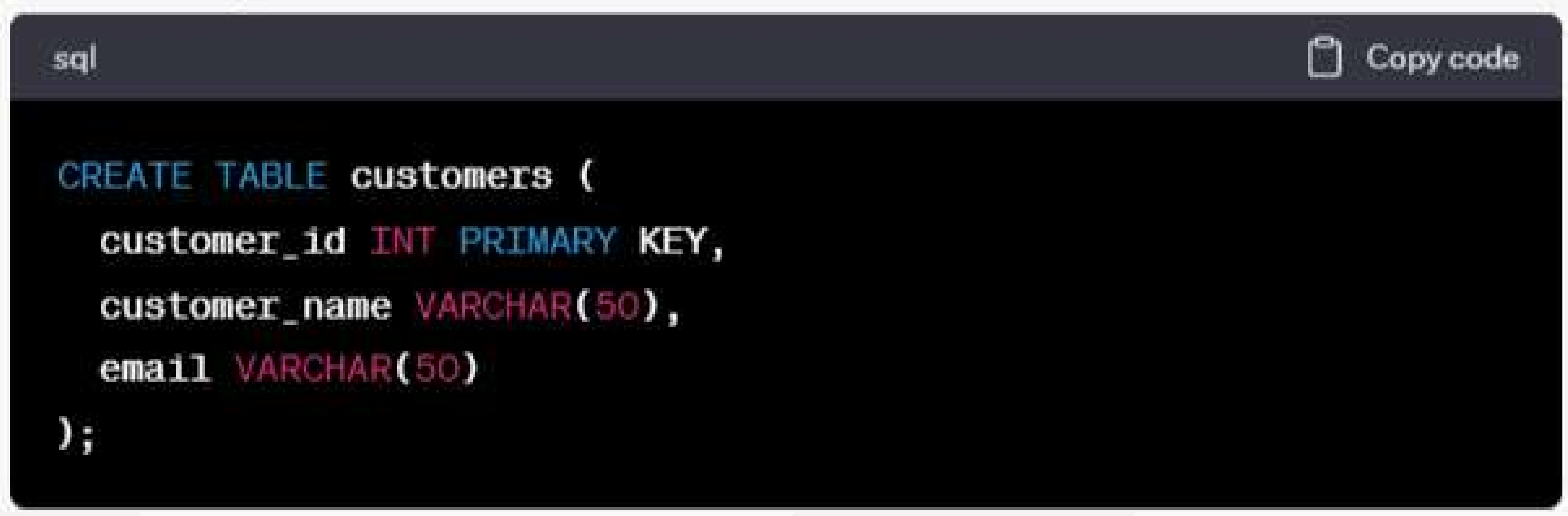
To create a new database in SQL, you can use the CREATE DATABASE statement followed by the database name. For example, to create a database named "mydb", you can use the following SQL command:



```sql
CREATE DATABASE mydb;
```

Creating Tables

To create a new table in a database, you can use the CREATE TABLE statement followed by the table name and column definitions. For example, to create a table named "customers" with columns for customer ID, name, and email, you can use the following SQL command:

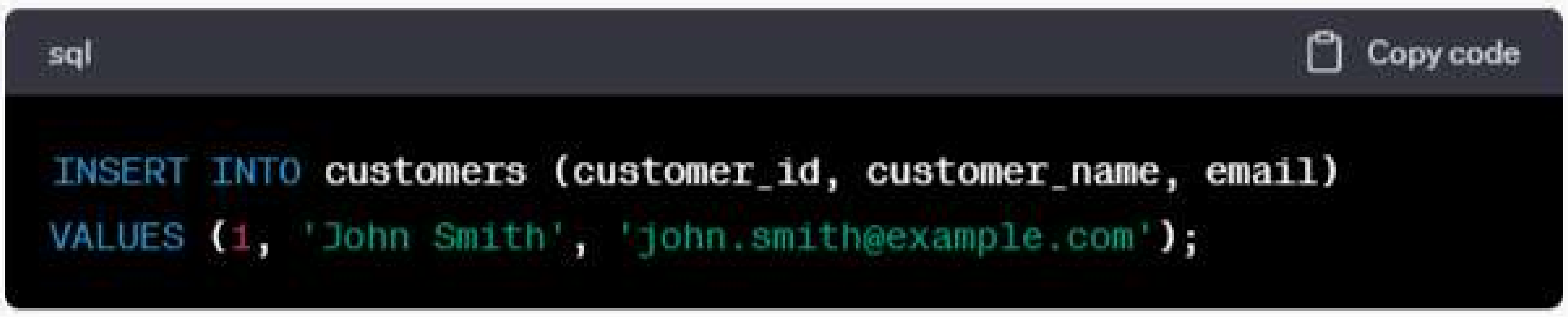


```sql
CREATE TABLE customers (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(50),
  email VARCHAR(50)
);
```

Hands-On Examples

Inserting Data

To insert data into a table, you can use the INSERT INTO statement followed by the table name and column values. For example, to insert a new customer into the "customers" table, you can use the following SQL command:



```sql
INSERT INTO customers (customer_id, customer_name, email)
VALUES (1, 'John Smith', 'john.smith@example.com');
```

Updating Data

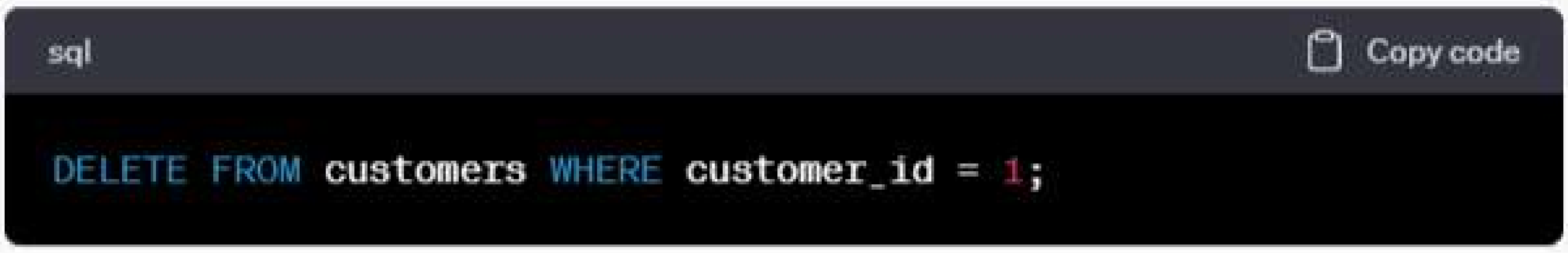
To update existing data in a table, you can use the UPDATE statement followed by the table name, column values, and condition. For example, to update the email address of a customer with ID 1 in the "customers" table, you can use the following SQL command:

Example: UPDATE customers SET email = 'john.smith@newemail.com' WHERE customer_id = 1;

Hands-On Examples

Deleting Data

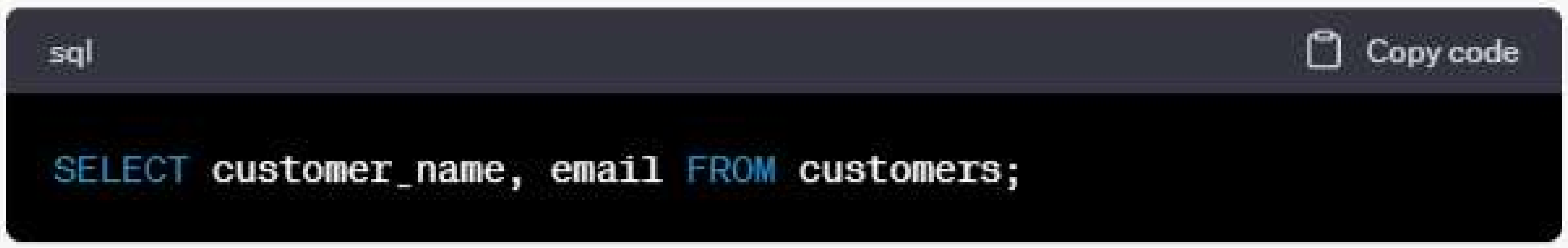
To delete data from a table, you can use the DELETE FROM statement followed by the table name and condition. For example, to delete a customer with ID 1 from the "customers" table, you can use the following SQL command:



```sql
DELETE FROM customers WHERE customer_id = 1;
```

Querying Data

To query data from a table, you can use the SELECT statement followed by the column names and table name. For example, to retrieve the names and email addresses of all customers from the "customers" table, you can use the following SQL command:

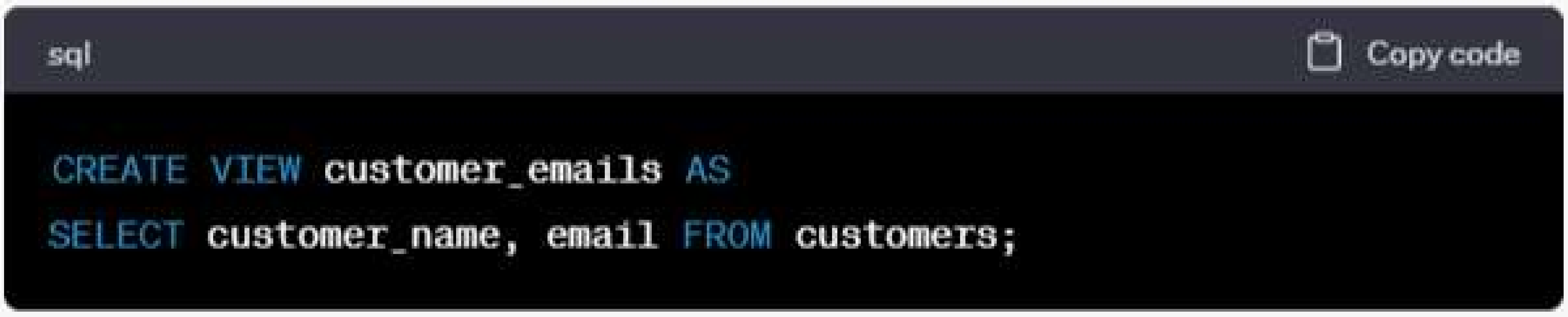


```sql
SELECT customer_name, email FROM customers;
```

Hands-On Examples

Creating Views

To create a view in SQL, you can use the CREATE VIEW statement followed by the view name and SELECT statement. For example, to create a view named "customer_emails" that retrieves the names and email addresses of all customers from the "customers" table, you can use the following SQL command:



```sql
CREATE VIEW customer_emails AS
SELECT customer_name, email FROM customers;
```

Hands-On Examples

Creating Stored Procedures

To create a stored procedure in SQL, you can use the CREATE PROCEDURE statement followed by the procedure name and SQL statements. For example, to create a stored procedure named "add_customer" that inserts a new customer into the "customers" table, you can use the following SQL command:

Example:

```
CREATE PROCEDURE add_customer (IN customer_id
INT, IN customer_name VARCHAR(50), IN email
VARCHAR(50))
BEGIN
  INSERT INTO customers (customer_id,
customer_name, email)
  VALUES (customer_id, customer_name, email);
END;
```

Summary of Key Concepts

SQL (Structured Query Language) is a domain-specific language used for managing relational databases. It consists of a set of commands used for creating, modifying, and querying databases and their contents. Some key concepts of SQL include data types, operators, functions, subqueries, views, indexes, transactions, stored procedures, and triggers.
SQL allows users to perform a wide range of operations on their databases, including creating tables and views, inserting, updating, and deleting data, and querying data using various operators and functions. Advanced concepts in SQL include subqueries, views, indexes, transactions, stored procedures, and triggers, which can provide greater control over data management and processing.

THANK YOU