UPDATE

1. Basic `UPDATE` Statement:

```
  Update the salary for an employee with ID 101
  UPDATE employees
  SET salary = 55000
  WHERE employee_id = 101;
```

- In this example, we update the "salary" column of the "employees" table to set it to 55,000 for the employee with an "employee_id" of 101.

2. Updating Multiple Columns:

```
  -- Update the department and salary for an employee with ID 102
  UPDATE employees
  SET department = 'HR', salary = 60000
  WHERE employee_id = 102;
```

- Here, we update both the "department" and "salary" columns for the employee with an "employee_id" of 102.

3. Updating All Rows:

```
  -- Increase the salary of all employees by 10%
  UPDATE employees
  SET salary = salary * 1.10;
```

- This query updates all rows in the "employees" table to increase their salaries by 10%.

4. Updating with Expressions:

```
  -- Update the order total by multiplying quantity and unit price
  UPDATE orders
  SET total_amount = quantity * unit_price
  order_id = 123;
```

- In this example, we calculate the new "total_amount" by multiplying "quantity" and "unit_price" and then update the specified order.

5. Updating Using Subqueries:

```
  -- Update product prices based on a subquery

  UPDATE products

  SET price = (SELECT new_price FROM price_updates WHERE products.product_id =
price_updates.product_id);
```

   - This query updates product prices in the "products" table based on a subquery that retrieves new prices from the "price_updates" table.

6. Updating with Joins:

```
  -- Update employee names with department changes using a JOIN

  UPDATE employees AS e

  INNER JOIN department_changes AS d ON e.employee_id = d.employee_id

  SET e.name = d.new_name, e.department = d.new_department;
```

  - This example shows how to update employee names and departments by joining the "employees" table with a "department_changes" table.

The `UPDATE` statement is a powerful tool for modifying data in a SQL database. It allows you to update specific rows and columns based on conditions, perform calculations, and even update data using subqueries and joins. Always use caution when executing `UPDATE` statements to ensure you're modifying the correct data, and consider using transactions for data integrity and rollback capabilities in more complex scenarios.