Section 1: Create the Sales Table

-- Step 1: Create the database if it doesn't already exist

CREATE DATABASE IF NOT EXISTS store;

-- Step 2: Select the store database to use

USE store;

-- Step 3: Create the sales table with the specified columns

CREATE TABLE IF NOT EXISTS sales (

id INT PRIMARY KEY,

product\_name VARCHAR(100),

quantity INT,

price\_per\_unit DECIMAL(10, 2)

);

Section 2: Insert Sample Data, INSERT INTO sales (id, product\_name, quantity, price\_per\_unit)

VALUES

(1, 'Laptop', 5, 899.99),

(2, 'Smartphone', 10, 499.50),

(3, 'Desk Chair', 7, 129.99),

(4, 'Wireless Mouse', 15, 24.99),

(5, 'Monitor', 4, 199.95);

Section 3: Aggregation with COUNT and SUM

4. Write a SQL query to count the total number of sales records in the table.

--SELECT COUNT(\*) AS products\_with\_quantity\_gt\_5

--FROM sales

--WHERE quantity > 5;

5. Write a SQL query to count how many products have a quantity greater

than 5.

SELECT SUM(quantity) AS total\_quantity\_sold

FROM sales;

--6. Write a SQL query to find the total quantity of products sold.

SELECT SUM(quantity \* price\_per\_unit) AS total\_sales\_amount

FROM sales;

--7. Write a SQL query to calculate the total sales amount across all products.

(Hint: Use quantity \* price\_per\_unit inside the SUM() function.)

SELECT SUM(quantity \* price\_per\_unit) AS total\_sales\_amount

FROM sales;

----8. Write a SQL query to calculate the total sales amount for products where the

price\_per\_unit is greater than 1,000.

SELECT SUM(quantity \* price\_per\_unit) AS total\_sales\_amount

FROM sales

WHERE price\_per\_unit > 1000;

**-----Instructions**

Your supervisor wants you to preview the data types for the orders table to verify that they are correctly assigned.

1. Write a query that displays the data types of the orders table.

PRAGMA table\_info(orders);

-----A typical task for data analysts is to derive new columns from existing columns by using arithmetic operations. For example, what if we want to find out how much sales tax there is on each sale in the orders table?

We can write a query to pull the order\_id, sales amount and multiply sales by 7% to estimate sales tax:

SELECT order\_id,

sales,

sales \* .07

FROM orders;

---Notice that the query uses **.07** instead of **7%**. This is because we can't calculate with percentages directly and must convert them to a decimal number first.

Looking at our column names, we can see that the calculated field is currently called **sales \* .07**, which isn't very readable. Let's rewrite our query with an alias:

SELECT order\_id,

      sales,

      sales \* .07 AS sales\_tax

 FROM orders;

---**Instructions**

Your manager has requested you find the profit margin on all orders from the orders table.

1. Write a query that includes order\_id, sales, profit, and profit margin (profit divided by sales).
2. Alias the calculated field as profit\_margin.
3. Only display 8 rows.

--**Instructions**

Your manager has requested you find the profit margin on all orders from the orders table.

Write a query that includes order\_id, sales, profit, and profit margin (profit divided by sales).

Alias the calculated field as profit\_margin.

Only display 8 rows.

**Answer**

SELECT order\_id, sales, profit, profit/sales AS profit\_margin

 FROM orders

LIMIT 8;

PAASWORD ID

912560

---**Instructions**

You have been asked to write a query using the orders table that helps identify whether customers are more likely to purchase an *even* number of an item or an *odd* number to help with ordering inventory. You can do this by dividing the quantity by 2 to help separate results that are whole numbers vs. decimal numbers.

1. Write a query that includes:
   * product\_id
   * quantity
   * a calculated field that divides quantity by 2 (remember to CAST your integer to a real number)
2. Alias the calculated field as even\_or\_odd
3. Limit your results to 5 rows

SELECT product\_id,

quantity, CAST(quantity AS REAL)/2 AS even\_or\_odd

FROM orders

LIMIT 5

--In addition to arithmetic operators, SQL makes available a number of functions to use on fields.

On the previous screen we saw that the values for sales and our calculated field, sales\_tax, had too many digits after the decimal point to represent dollars. Here's a reminder of what that query and output looked like:

SELECT order\_id,

      sales,

      sales \* .07 AS sales\_tax

 FROM orders;

--Now that we know about the ROUND() function, let's see it in action. We'll use separate ROUND() functions on sales and our calculated field, sales\_tax, and give each an alias to make the names more readable:

SELECT order\_id,

      ROUND(sales, 2) AS rounded\_sales,

      ROUND(sales \* .07, 2) AS rounded\_sales\_tax

 FROM orders;

--**Instructions**

Many customers buy multiples of the same item, which makes it difficult to see how much each item costs.

Write a query from the orders table that includes order\_id, sales, and quantity.

Create a field price\_per\_unit that divides sales by quantity.

Round price\_per\_unit to two decimal places.

Only display 10 rows.

**Answer**

SELECT order\_id,

      sales,

      quantity,

      ROUND(sales/quantity,2) AS price\_per\_unit

 FROM orders

LIMIT 10;

--**Instructions**

Customers enter their own names when they purchase items from the superstore online, which means some customer names have inconsistent capitalization formatting.

Write a query that converts customer\_name from the orders table to all lowercase letters.

Alias the newly formatted names as customer\_name\_lower.

**Answer**

SELECT LOWER(customer\_name) AS customer\_name\_lower

 FROM orders;

--We've seen that it's possible to change the output of text fields with UPPER() and LOWER(). It's also possible to *concatenate*, or combine, multiple pieces of text together.

Perhaps we'd like to put the city and state columns together like this: **Henderson, Kentucky**.

SQLite uses the concatenate operator (||) to join two strings into one.

SELECT city || ", " || state AS "location"

 FROM orders;

--**Instructions**

Every superstore location is named after the city where it's located. For example, the store in Los Angeles, California is called "Superstore Los Angeles."

Write a query that includes order\_id, region, and state.

Create a new column called local\_store that concatenates the word "Superstore" with city. There should be one space between the word "Superstore" and the city name (i.e. "Superstore Dallas")

Limit your results to 10 rows.

**Answer**

SELECT order\_id,

      region,

      state,

      "Superstore " || city AS local\_store

 FROM orders

LIMIT 10;

--Sometimes we'll want to add a column to our query that shows a constant number for every record. We can do this by specifying the number in the SELECT clause and adding an alias for readability (note that the alias isn't required):

SELECT sales, 2 AS promotional\_discount

 FROM orders;

--**Instructions**

You've been asked to create a table that shows the salary for all the regional managers.

1. Write a query from managers that includes all fields.
2. Create a new column with the manager salary amount: $51,000. Alias it as salary.

**Answer**

1

SELECT \*, 51000 AS salary

2

 FROM managers;

--**Instructions**

1. Create a field called address that combines city, state, and postal\_code in the following format:
   * city, state 99999
2. Calculate a total\_cost field cost as sales, shipping, and tax. (Shipping costs 4.99 and tax is calculated at 7%). Round the field to two decimal places.
3. Create a field called tax that finds 7% of sales, rounded to two decimal places.
4. Create a field called shipping\_cost with a value of 4.99.
5. For readability, your final query should include the following fields:
   * address
   * sales
   * tax
   * shipping\_cost
   * total\_cost

Limit your results to 10 rows.

We've made this screen experimental so you can test out different queries, but we encourage you to try this challenge. If you're curious to see the suggested answer to this challenge question, click "Get Help" to see the way we wrote the query.

**Answer**

SELECT city || ", " || state || " " || postal\_code AS address,

  sales,

      ROUND(sales + 4.99 + sales\*0.07, 2) AS total\_cost,

  ROUND(sales \* 0.07, 2) AS tax,

  4.99 AS shipping\_cost

 FROM orders

LIMIT 10;

--**Filter the data with the WHERE clause**

It's time to introduce the WHERE clause to compare two values in SQL. The clauses we've learned thus far will always be written in queries in this order:

1. **SELECT:** Specify what fields we want information from.
2. **FROM:** Specify what tables those fields are coming from.
3. **WHERE:** Specify any criteria that records in those fields should meet.
4. **LIMIT:** Specify how many records to return in results.

Let's see the power of WHERE in action. For example, we can check to see if there have been any sales over $10,000 in the orders table by writing the following query with WHERE:

SELECT order\_id, category, product\_name,

      quantity, sales

 FROM orders

WHERE sales > 10000;

--**Instructions**

Your supervisor is curious about sales that have lost the company money. They've asked you to look at orders that have lost more than $1,000 in profit.

1. Write a query that includes order\_id, product\_name, sales, discount, and profit.
2. Filter your results so that your query only shows records that have lost more than $1000.

***Tip***  
To show a $1000 **loss**, you'll need to write it as a negative number (-1000).

**Answer**

SELECT order\_id, product\_name, sales, discount, profit

 FROM orders

WHERE profit < -1000;