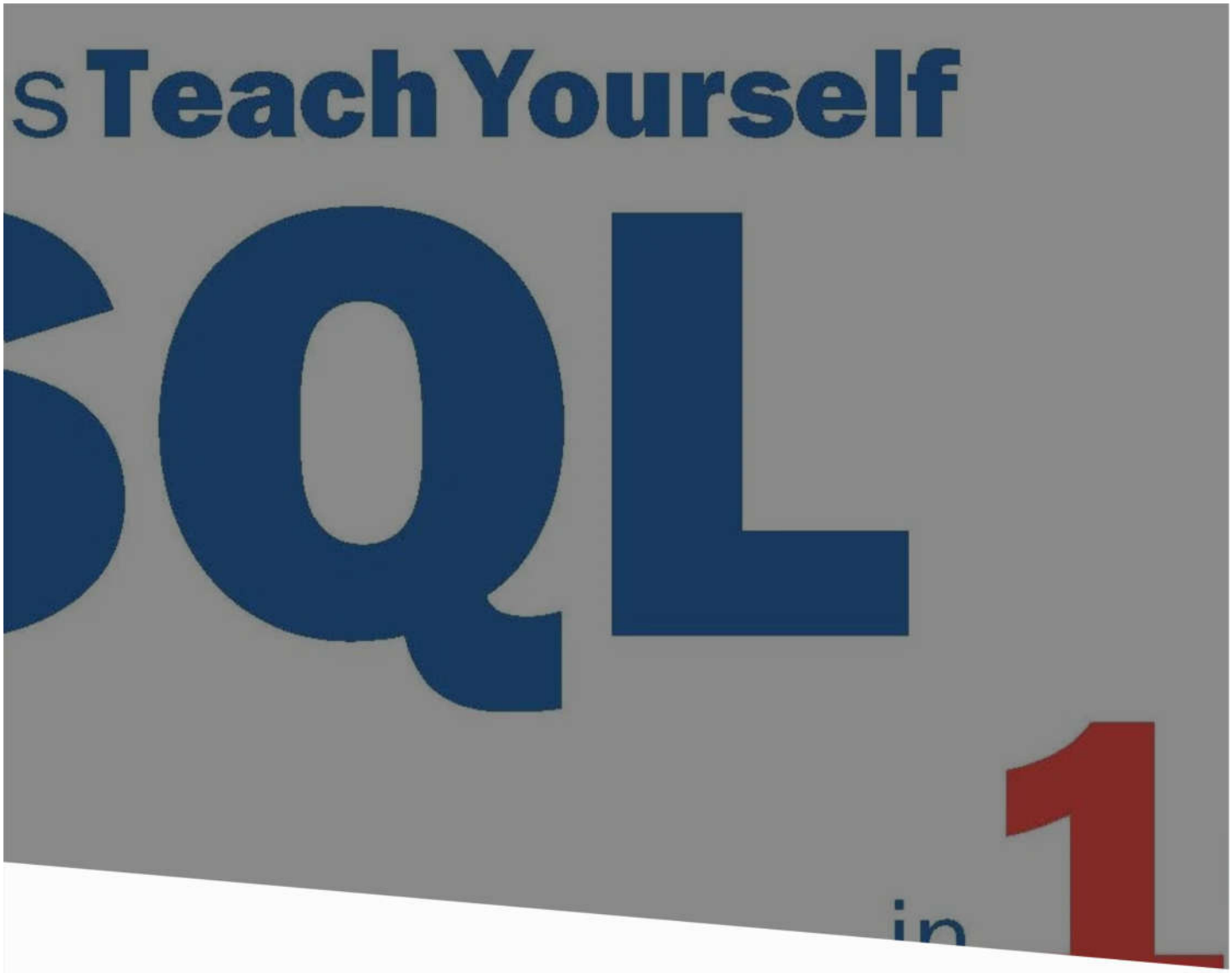


Sams Teach Yourself SQL in 10 Minutes (Fifth Edition) includes challenges at the end of some lessons. Solutions to the challenges are presented here. Just keep in mind that there is rarely one solution to a SQL challenge, so if your solutions look different but produce the desired result, that's ok.

Lesson 2

Challenge 1

Write a SQL statement to retrieve all customer ids (cust_id) from the Customers table.

```
SELECT cust_id
FROM Customers;
```

Challenge 2

The OrderItems table contains every item ordered (and some were ordered multiple times). Write a SQL statement to retrieve a list of the products (prod_id) ordered (not every order, just a unique list of products). Here's a hint, you should end up with seven unique rows displayed.

```
SELECT DISTINCT prod_id
FROM OrderItems;
```

Challenge 3

Write a SQL statement that retrieves all columns from the Customers table, and an alternate SELECT that retrieves just the customer id. Use comments to "comment out" one SELECT so as to be able to run the other. (And of course, test both statements).

```
SELECT *
# SELECT cust_id
FROM Customers;
```

Lesson 3

Challenge 1

Write a SQL statement to retrieve all customer names (cust_name) from the Customers table, and display the results sorted from Z to A.

```
SELECT cust_name
FROM Customers
```

```
ORDER BY cust_name DESC;
```


Challenge 2

Write a SQL statement to retrieve customer id (cust_id) and order number (order_num) from the Orders table, and sort the results first by customer id, and then by order date in reverse chronological order.

```
SELECT cust_id, order_num
FROM Orders
ORDER BY cust_id, order_date DESC;
```

Challenge 3

Our fictitious store obviously prefers to sell more expensive items, and lots of them. Write a SQL statement to display the quantity and price (item_price) from the OrderItems table, sorted with the highest quantity and highest price first.

```
SELECT quantity, item_price
FROM OrderItems
ORDER BY quantity DESC, item_price DESC;
```

Challenge 4

What is wrong with the following SQL statement? (Try to figure it out without running it):

```
SELECT vend_name,
FROM Vendors
ORDER vend_name DESC;
```

There should not be a comma after vend_name (a comma is only used to separate multiple columns), and BY is missing after ORDER.

Lesson 4

Challenge 1

Write a SQL statement to retrieve the product id (prod_id) and name (prod_name) from the Products table, returning only products with a price of 9.49.

```
SELECT prod_id, prod_name
FROM Products
WHERE prod_price = 9.49;
```

Challenge 2

Write a SQL statement to retrieve the product id (prod_id) and name (prod_name) from the Products table, returning only products with a price of 9 or more.

```
SELECT prod_id, prod_name
FROM Products
WHERE prod_price >= 9;
```

Challenge 3

Now let's combine Lessons 3 and 4. Write a SQL statement that retrieves the unique list of order numbers (order_num) from the OrderItems table which contain 100 or more of any item.

```
SELECT DISTINCT order_num
FROM OrderItems
WHERE quantity >=100;
```

Challenge 4

One more. Write a SQL statement which returns the product name (prod_name) and price (prod_price) from Products for all products priced between 3 and 6. Oh, and sort the results by price. (There are multiple solutions to this one and we'll revisit it in the next lesson, but you can solve it using what you've learned thus far).

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price BETWEEN 3 AND 6
ORDER BY prod_price;
```

Challenge 1

Write a SQL statement to retrieve the vendor name (vend_name) from the Vendors table, returning only vendors in California (this requires filtering by both country (USA) and state (CA), after all, there could be a California outside of the USA). Here's a hint, the filter requires matching strings.

```
SELECT vend_name
FROM Vendors
WHERE vend_country = 'USA' AND vend_state = 'CA';
```

Challenge 2

Write a SQL statement to find all orders where at least 100 of items BR01, BR02, or BR03 were ordered. You'll want to return order number (order_num), product id (prod_id), and quantity for the OrderItems table, filtering by both the product id and quantity. Here's a hint, depending on how you write your filter, you may need to pay special attention to order of evaluation.

```
-- Solution 1
SELECT order_num, prod_id, quantity
FROM OrderItems
WHERE (prod_id='BR01' OR prod_id='BR02' OR prod_id='BR03')
 AND quantity >=100;

-- Solution 2
SELECT order_num, prod_id, quantity
FROM OrderItems
WHERE prod_id IN ('BR01','BR02','BR03')
 AND quantity >=100;
```

Challenge 3

Now let's revisit a challenge from the previous lesson. Write a SQL statement which returns the product name (prod_name) and price (prod_price) from Products for all products priced between 3 and 6. Use an AND, and sort the results by price.

```
SELECT prod_name, prod_price
```

Challenge 4

What is wrong with the following SQL statement? (Try to figure it out without running it):

```
SELECT vend_name
FROM Vendors
ORDER BY vend_name
WHERE vend_country = 'USA' AND vend_state = 'CA';
```

ORDER BY must came after any WHERE clause.

Lesson 6

Challenge 1

Write a SQL statement to retrieve the product name (prod_name) and description (prod_desc) from the Products table, returning only products where the word toy is in the description.

```
SELECT prod_name, prod_desc
FROM Products
WHERE prod_desc LIKE '%toy%';
```

Challenge 2

Now let's flip things around. Write a SQL statement to retrieve the product name (prod_name) and description (prod_desc) from the Products table, returning only products where the word toy doesn't appear in the description. And this time, sort the results by product name.

```
SELECT prod_name, prod_desc
FROM Products
WHERE NOT prod_desc LIKE '%toy%'
ORDER BY prod_name;
```

Write a SQL statement to retrieve the product name (prod_name) and description (prod_desc) from the Products table, returning only products where both the words toy and carrots appear in the description. There are a couple of ways to do this, but for this challenge use AND and two LIKE comparisons.

```
SELECT prod_name, prod_desc
FROM Products
WHERE prod_desc LIKE '%toy%' AND prod_desc LIKE '%carrots%';
```

Challenge 4

This next one is a little trickier. I didn't show you this syntax specifically, but see if you can figure it out anyway based on what you have learned thus far. Write a SQL statement to retrieve the product name (prod_name) and description (prod_desc) from the Products table, returning only products where both the words toy and carrots appear in the description in that order (the word toy before the word carrots). Here's a hint, you'll only need one LIKE with 3 % symbols to do this.

```
SELECT prod_name, prod_desc
FROM Products
WHERE prod_desc LIKE '%toy%carrots%';
```

Lesson 7

Challenge 1

A common use for aliases is to rename table column fields in retrieved results (perhaps to match specific reporting or client needs). Write a SQL statement that retrieves vend_id, vend_name, vend_address, and vend_city from Vendors, renaming vend_name to vname, vend_city to vcity, and vend_address to vaddress. Sort the results by vendor name (you can use the original name or the renamed name).

```
SELECT vend_id,
       vend_name as vname,
       vend_address AS vaddress,
       vend_city AS vcity
FROM Vendors
ORDER BY vname;
```

Challenge 2

Our example store is running a sale and all products are 10% off. Write a SQL statement that returns prod_id, prod_price, and sale_price from the Products table. sale_price is a calculated field that contains, well, the sale price. Here's a hint, you can multiply by 0.9 to get 90% of the original value (and thus the 10% off price).

```
SELECT prod_id, prod_price,
       prod_price * 0.9 AS sale_price
FROM Products;
```

Lesson 8

Challenge 1

Our store is now online and customer accounts are being created. Each user needs a login, and the default login will be a combination of their name and city. Write a SQL statement that returns customer id (cust_id), customer name (cust_name) and user_login which is all upper case and comprised of the first two characters of the customer contact (cust_contact) and the first three characters of the customer city (cust_city). So, for example, my login (Ben Forta living in Oak Park) would be BEOAK. Hint, for this one you'll use functions, concatenation, and an alias.

```
-- DB2, PostgreSQL
SELECT cust_id, cust_name,
       UPPER(LEFT(cust_contact, 2)) || UPPER(LEFT(cust_city, 3)) AS user_login
FROM customers;
-- Oracle, SQLite
SELECT cust_id, cust_name,
       UPPER(SUBSTR(cust_contact, 1, 2)) || UPPER(SUBSTR(cust_city, 1, 3)) AS
user_login
  FROM customers;
-- MySQL
```

```
SELECT cust_id, cust_name,
       UPPER(LEFT(cust_contact, 2)) + UPPER(LEFT(cust_city, 3)) AS user_login
FROM customers;
```

Challenge 2

Write a SQL statement to return the order number (order_num) and order date (order_date) for all orders placed in January 2020, sorted by order date. You should be able to figure this out based on what you have learned thus far, but feel free to consult your DBMS documentation as needed.

```
-- DB2, MariaDB, MySQL
SELECT order_num, order_date
FROM Orders
WHERE YEAR(order_date) = 2020 AND MONTH(order_date) = 1
ORDER BY order_date;
-- Oracle, PostgreSQL
SELECT order_num, order_date
FROM Orders
WHERE EXTRACT(year FROM order_date) = 2020 AND EXTRACT(month FROM order_date) = 1
ORDER BY order_date;
-- PostgreSQL
SELECT order_num, order_date
FROM Orders
WHERE DATE_PART('year', order_date) = 2020
 AND DATE_PART('month', order_date) = 1
ORDER BY order_num;
-- SQL Server
SELECT order_num, order_date
FROM Orders
WHERE DATEPART(yy, order_date) = 2020 AND DATEPART(mm, order_date) = 1
ORDER BY order_date;
-- SQLite
SELECT order_num
FROM Orders
WHERE strftime('%Y', order_date) = '2020'
 AND strftime('%m', order_date) = '01';
```

Lesson 9

Challenge 1

Write a SQL statement to determine the total number of items sold (using the quantity column in OrderItems).

Challenge 2

Modify the statement you just created to determine the total number of product item (prod_item) BR01 sold.

```
SELECT SUM(quantity) AS items_ordered
FROM OrderItems
WHERE prod_id = 'BR01';
```

Challenge 3

Write a SQL statement to determine the price (prod_price) of the most expensive item in the Products table which costs no more than 10, name the calculated field max_price.

```
SELECT MAX(prod_price) AS max_price
FROM Products
WHERE prod_price <= 10;
```

Lesson 10

Challenge 1

The OrderItems table contains the individual items for each order. Write a SQL statement that returns the number of lines (as order_lines) for each order number (order_num) and sort the results buy order_lines.

```
SELECT order_num, COUNT(*) as order_lines
FROM OrderItems
GROUP BY order_num
ORDER BY order_lines;
```

Challenge 2

Write a SQL statement that returns a field named cheapest_item which contains the lowest cost item for each vendor (using prod_price in the Products table), and sort the results from lowest to highest cost.

```
SELECT vend_id, MIN(prod_price) AS cheapest_item
FROM Products
GROUP BY vend_id
ORDER BY cheapest_item;
```

Challenge 3

It's important to identify the best customers, so write a SQL statement to return the order number (order_num in OrderItems table) for all orders of at least 100 items.

```
SELECT order_num
FROM OrderItems
GROUP BY order_num
HAVING SUM(quantity) >= 100
ORDER BY order_num;
```

Challenge 4

Another way to determine the best customers is by how much they have spent. Write a SQL statement to return the order number (order_num in OrderItems table) for all orders with a total price of at least 1000. Hint, for this one you'll need to calculate and sum the total (item_price multiplied by quantity). Sort the results by order number.

```
SELECT order_num, SUM(item_price*quantity) AS total_price
FROM OrderItems
GROUP BY order_num
HAVING SUM(item_price*quantity) >= 1000
ORDER BY order_num;
```

Challenge 5

What is wrong with the following SQL statement? (Try to figure it out without running it):

```
SELECT order_num, COUNT() AS items
FROM OrderItems
GROUP BY items
HAVING COUNT() >= 3
ORDER BY items, order_num;
```

GROUP BY items is incorrect. GROUP BY must be an actual column, not the one being used to perform the aggregate calculations. GROUP BY order_num would be allowed.

Lesson 11

Challenge 1

Using a subquery, return a list of customers who bought items priced 10 or more. You'll want to use the OrderItems table to find the matching order numbers (order_num), and then the Orders table to retrieve the customer id (cust_id) for those matched orders.

```
SELECT cust_id
FROM Orders
WHERE order_num IN (SELECT order_num
                    FROM OrderItems
                    WHERE item_price >= 10);
```

Challenge 2

You need to know the dates when product BR01 was ordered. Write a SQL statement that uses a subquery to determine which orders (in OrderItems) purchased prod_id BR01, and then returns customer id (cust_id) and order date (order_date) for each from the Orders table. Sort the results by order date.

```
SELECT cust_id, order_date
FROM orders
WHERE order_num IN (SELECT order_num
                    FROM OrderItems
                    WHERE prod_id = 'BR01')
ORDER BY order_date;
```

Challenge 3

Now let's make it a bit more challenging. Update the previous challenge to return the customer email (cust_email in the Customers table) for any customers who purchased item with a prod_id of BR01. Hint, this involves the SELECT statement, the innermost one returning order_num from OrderItems, and the middle one returning cust_id from Orders.

```
SELECT cust_email FROM Customers
WHERE cust_id IN (SELECT cust_id
                  FROM Orders
                  WHERE order_num IN (SELECT order_num
                                      FROM OrderItems
                                      WHERE prod_id = 'BR01'));
```

Challenge 4

We need a list of customer ids with the total amount they have ordered. Write a SQL statement to return customer id (cust_id in Orders table) and total_ordered using a subquery to return the total of orders for each customer. Sort the results by amount spent from greatest to the least. Hint, you've used the SUM() to calculate order totals previously.

```
SELECT cust_id,
       (SELECT SUM(item_price*quantity)
        FROM OrderItems
        WHERE Orders.order_num = OrderItems.order_num) AS total_ordered
FROM Orders
ORDER BY total_ordered DESC;
```

Challenge 5

One more. Write a SQL statement that retrieves all product names (prod_name) from the Products table, along with a calculated named quant_sold containing the total number of this item sold (retrieved using a subquery and a SUM(quantity) on the OrderItems table).

```
SELECT prod_name,
       (SELECT Sum(quantity)
        FROM OrderItems
        WHERE Products.prod_id=OrderItems.prod_id) AS quant_sold
FROM Products;
```

Challenge 1

Write a SQL statement to return customer name (cust_name) from the Customers table and related order numbers (order_num) from the Orders table, sorting the result by customer name and then by order number. Actually, try this one twice, once using simple equijoin syntax and once using an INNER JOIN.

```
-- Equijoin syntax
SELECT cust_name, order_num
FROM Customers, Orders
WHERE Customers.cust_id = Orders.cust_id
ORDER BY cust_name, order_num;

-- ANSI INNER JOIN syntax
SELECT cust_name, order_num
FROM Customers INNER JOIN Orders
 ON Customers.cust_id = Orders.cust_id
ORDER BY cust_name, order_num;
```

Challenge 2

Let's make the previous challenge more useful. In addition to returning the customer name and order number, add a third column named OrderTotal containing the total price of each order. There are two ways to do this, you can create the OrderTotal column using a subquery on the OrderItems table, or you can join the OrderItems table to the existing tables and use an aggregate function. Here's a hint, watch out for where you need to use fully qualified column names.

```
-- Solution using subqueries
SELECT cust_name,
       order_num,
       (SELECT Sum(item_price*quantity)
       FROM OrderItems
       WHERE Orders.order_num=OrderItems.order_num) AS OrderTotal
FROM Customers, Orders
WHERE Customers.cust_id = Orders.cust_id
ORDER BY cust_name, order_num;

-- Solution using joins
SELECT cust_name,
       Orders.order_num,
       Sum(item_price*quantity) AS OrderTotal
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Orders.cust_id
```

Challenge 3

Let's revisit Challenge 1 from Lesson 11. Write a SQL statement that retrieves the dates when product BR01 was ordered, but this time use a join and simple equijoin syntax. The output should be identical to the one from Lesson 12.

```
SELECT cust_id, order_date
FROM Orders, OrderItems
WHERE Orders.order_num = OrderItems.order_num
 AND prod_id = 'BR01'
ORDER BY order_date;
```

Challenge 4

That was fun, let's try it again. Recreate the SQL you wrote for Lesson 11 Challenge 3, this time using ANSI INNER JOIN syntax. The code you wrote there employed two nested subqueries, and to recreate it you'll need two INNER JOIN statements, each formatted like the INNER JOIN example earlier in this lesson. And don't forget the WHERE clause to filter by prod_id.

```
SELECT cust_email
FROM Customers
 INNER JOIN Orders ON Customers.cust_id = Orders.cust_id
 INNER JOIN OrderItems ON Orders.order_num = OrderItems.order_num
WHERE prod_id = 'BR01';
```

Challenge 5

One more, and to make things more fun we'll mix joins, aggregates functions, and grouping, too. Ready? Back in Lesson 10 I issued you a Challenge to find all order numbers with a value of 1000 or more. Those results are useful, but what would be even more useful is the name of the customers who placed orders of at least that amount. So, write a SQL statement that uses joins to return customer name (cust_name) from the Customers table, and the total price of all orders from the OrderItems table. Here's a hint, to join those tables you'll also need to include the Orders table (as Customers is not related directly to OrderItems, Customers is related to Orders and Orders is related to OrderItems). Don't forget

the GROUP BY and HAVING, and sort the results by customer name. You can use simple equijoin or ANSI INNER JOIN syntax for this one. Or, if you are feeling brave, try writing it both ways.

```
-- Equijoin syntax
SELECT cust_name, SUM(item_price*quantity) AS total_price
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Orders.cust_id
 AND Orders.order_num = OrderItems.order_num
GROUP BY cust_name HAVING SUM(item_price*quantity) >= 1000
ORDER BY cust_name;

-- ANSI INNER JOIN syntax
SELECT cust_name, SUM(item_price*quantity) AS total_price
FROM Customers
 INNER JOIN Orders ON Customers.cust_id = Orders.cust_id
 INNER JOIN OrderItems ON Orders.order_num = OrderItems.order_num
GROUP BY cust_name
HAVING SUM(item_price*quantity) >= 1000
ORDER BY cust_name;
```

Lesson 13

Challenge 1

Write a SQL statement using an INNER JOIN to retrieve customer name (cust_name in Customers) and all order numbers (order_num in Orders) for each.

```
SELECT cust_name, order_num
FROM Customers
 JOIN Orders ON Customers.cust_id = Orders.cust_id
ORDER BY cust_name;
```

Challenge 2

Modify the SQL statement you just created to list all customers, even those with no orders.

```
SELECT cust_name, order_num
FROM Customers
 LEFT OUTER JOIN Orders ON Customers.cust_id = Orders.cust_id
ORDER BY cust_name;
```

Use an OUTER JOIN to join the Products and OrderItems tables, returning a sorted list of product names (prod_name) and the order numbers (order_num) associated with each.

```
SELECT prod_name, order_num
FROM Products LEFT OUTER JOIN OrderItems
 ON Products.prod_id = OrderItems.prod_id
ORDER BY prod_name;
```

Challenge 4

Modify the SQL statement created in the previous challenge so that it returns a total of number of orders for each item (as opposed to the order numbers).

```
SELECT prod_name, COUNT(order_num) AS orders
FROM Products LEFT OUTER JOIN OrderItems
 ON Products.prod_id = OrderItems.prod_id
GROUP BY prod_name
ORDER BY prod_name;
```

Challenge 5

Write a SQL statement to list vendors (vend_id in Vendors) and the number of products they have available, including vendors with no products. You'll want to use an OUTER JOIN and the COUNT() aggregate function to count the number of products for each in the Products table. Pay attention, the vend_id column appears in multiple tables so any time you refer to it you'll need to fully qualify it.

```
SELECT Vendors.vend_id, COUNT(prod_id)
FROM Vendors
 LEFT OUTER JOIN Products ON Vendors.vend_id = Products.vend_id
GROUP BY Vendors.vend_id;
```

Lesson 14

Challenge 1

Write a SQL statement that combines two SELECT statements that retrieve product id (prod_id) and quantity from the OrderItems table, one filtering for rows with a quantity of exactly 100, and the other filtering for products with an ID that begins with BNBG. Sort the results by product id.

```
SELECT prod_id, quantity FROM OrderItems
WHERE quantity = 100
UNION
SELECT prod_id, quantity FROM OrderItems
WHERE prod_id LIKE 'BNBG%'
ORDER BY prod_id;
```

Challenge 2

Rewrite the SQL statement you just created to use a single SELECT statement.

```
SELECT prod_id, quantity FROM OrderItems
WHERE quantity = 100 OR prod_id LIKE 'BNBG%'
ORDER BY prod_id;
```

Challenge 3

This one is a little nonsensical, I know, but it does reinforce a note earlier in this lesson. Write a SQL statement which returns and combines product name (prod_name) from Products and customer name (cust_name) from Customers, and sort the result by product name.

```
SELECT prod_name
FROM Products
UNION
SELECT cust_name
FROM Customers
ORDER BY prod_name;
```

Challenge 4

What is wrong with the following SQL statement? (Try to figure it out without running it):

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
```

```
WHERE cust_state  = 'MI'
ORDER BY cust_name;
UNION
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state = 'IL'
ORDER BY cust_name;
```

The ; after the first SELECT statement shouldn't be there, it is terminating the statement. Also, if sorting SELECT statements combined with UNION only one ORDER BY may be used and it must come after the last SELECT.

Lesson 15

Challenge 1

Using INSERT and columns specified, add yourself to the Customers table.

```
-- Obviously replace the details with your own
INSERT INTO Customers(cust_id,
                      cust_name,
                      cust_address,
                      cust_city,
                      cust_state,
                      cust_zip,
                      cust_country,
                      cust_email)
VALUES(1000000042,
       'Ben''s Toys',
       '123 Main Street',
       'Oak Park',
       'MI',
       '48237',
       'USA',
       'ben@forta.com');
```

Challenge 2

```
-- MySQL, MariaDB, Oracle, PostgreSQL, SQLite
CREATE TABLE OrdersBackup AS SELECT * FROM Orders;
CREATE TABLE OrderItemsBackup AS SELECT * FROM OrderItems;

-- SQL Server
SELECT * INTO OrdersBackup FROM Orders;
SELECT * INTO OrderItemsBackup FROM OrderItems;
```

Lesson 16

Challenge 1

USA State abbreviations should always be in upper case. Write a SQL statement to update all USA addresses, both vendor states (vend_state in Vendors) and customer states (cust_state in Customers) so that they are upper case.

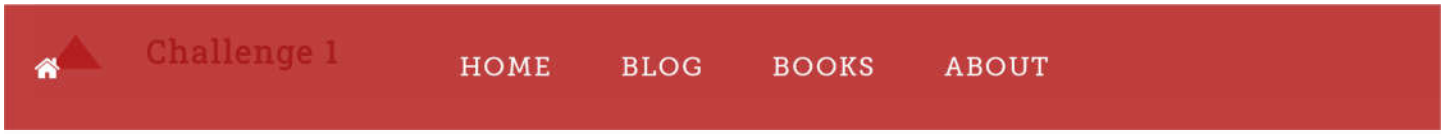
```
UPDATE Vendors
SET vend_state = UPPER(vend_state)
WHERE vend_country = 'USA';
UPDATE Customers
SET cust_state = UPPER(cust_state)
WHERE cust_country = 'USA';
```

Challenge 2

In Lesson 15 Challenge 1 I asked you to add yourself to the Customers table. Now delete yourself. Make sure to use a WHERE clause (and test it with a SELECT before using it in DELETE) or you'll delete all customers!

```
-- First test the WHERE to make sure it selects only what you want to delete
SELECT * FROM Customers
WHERE cust_id = 1000000042;
-- Then do it!
DELETE Customers
WHERE cust_id = 1000000042;
```

Lesson 17



Add a web site column (vend_web) to the Vendors table. You'll want a text field big enough to accommodate a URL.

```
ALTER TABLE Vendors
ADD vend_web CHAR(100);
```

Challenge 2

Use UPDATE statements to update Customer records with phone numbers.

```
UPDATE Vendors
SET vend_web = 'https://google.com/'
WHERE vend_id = 'DLL01';
```

Lesson 18

Challenge 1

Create a view called CustomersWithOrders that contains all of the columns in Customers, but only includes those who have placed orders. Hint, you can JOIN the Orders table to filter just the customers you want. Then use a SELECT to make sure you have the right data.

```
CREATE VIEW CustomersWithOrders AS
SELECT Customers.cust_id,
       Customers.cust_name,
       Customers.cust_address,
       Customers.cust_city,
       Customers.cust_state,
       Customers.cust_zip,
       Customers.cust_country,
       Customers.cust_contact,
       Customers.cust_email
FROM Customers
JOIN Orders ON Customers.cust_id = Orders.cust_id;

SELECT * FROM CustomersWithOrders;
```

What is wrong with the following SQL statement? (Try to figure it out without running it):

```
CREATE VIEW OrderItemsExpanded AS
SELECT order_num,
       prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM OrderItems
ORDER BY order_num;
```

ORDER BY is not allowed in views. Views are used like tables, if you need sorted data use ORDER BY in the SELECT that retrieves data from the view.

Share this:

FOLLOW BEN

© 2019 Ben Forta