2024

MySQL Assignment

SECOND ASSIGNMENT

TYLER SLACK

Contents

MySQL Task 1 2
Task 1 2
Task 2 2
Task 3 2
Task 4 3
Task 5 4
Task 6 4
EER Diagram – Task 1 5
MySQL Task 2 6
Task 1 6
Task 2 6
Task 3 7
Task 4 7
Task 5 8
Task 6 8
Task 7 8
Task 8 9
Task 9 9
Optional 9
EER Diagram – Task 2 10
Interview Questions 11

MySQL Task 1

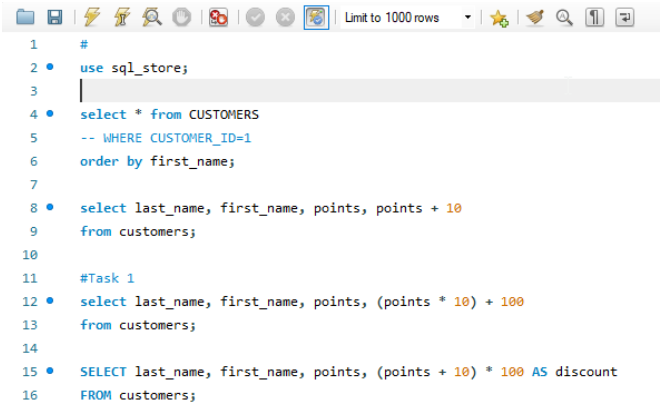
Task 1

For Task 1, I modified the query code to achieve two objectives:

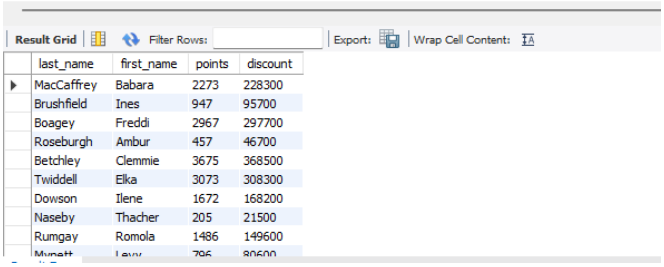
Updated Points Calculation: Changed the points calculation to *(points * 10) + 100*, displaying it alongside customer names.

Added Discount Factor: Created a *discount* column with the formula *(points + 10) * 100*, showing a new value in a header labelled "*discount*."

These adjustments allow us to see both the modified points and an additional discount factor for each customer.

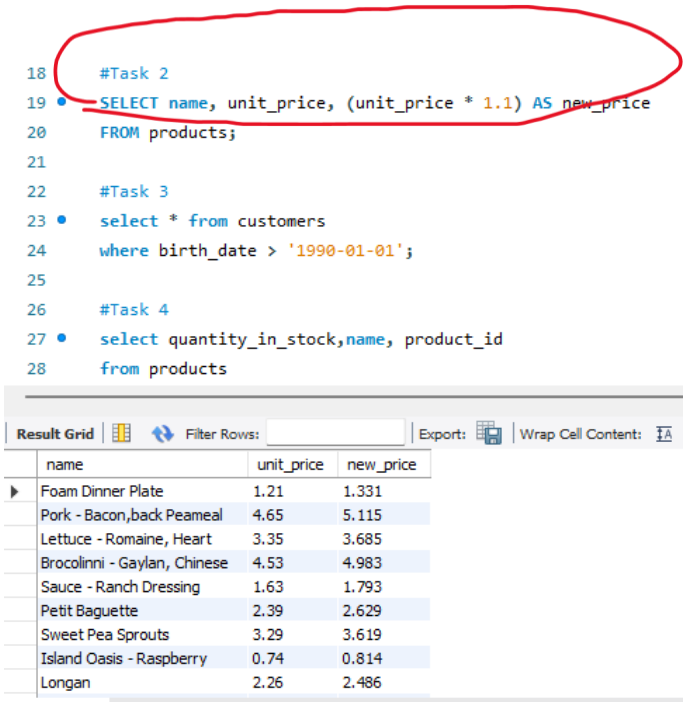


```
1   #
2   use sql_store;
3
4   select * from CUSTOMERS
5   -- WHERE CUSTOMER_ID=1
6   order by first_name;
7
8   select last_name, first_name, points, points + 10
9   from customers;
10
11  #Task 1
12  select last_name, first_name, points, (points * 10) + 100
13  from customers;
14
15  SELECT last_name, first_name, points, (points + 10) * 100 AS discount
16  FROM customers;
```



last_name	first_name	points	discount
MacCaffrey	Babara	2273	228300
Brushfield	Ines	947	95700
Boagey	Freddi	2967	297700
Roseburgh	Ambur	457	46700
Betchley	Clemmie	3675	368500
Twiddell	Elka	3073	308300
Dowson	Ilene	1672	168200
Naseby	Thacher	205	21500
Rumgay	Romola	1486	149600

Task 2



```
18  #Task 2
19  SELECT name, unit_price, (unit_price * 1.1) AS new_price
20  FROM products;
21
22  #Task 3
23  select * from customers
24  where birth_date > '1990-01-01';
25
26  #Task 4
27  select quantity_in_stock,name, product_id
28  from products
```

name	unit_price	new_price
Foam Dinner Plate	1.21	1.331
Pork - Bacon,back Peameal	4.65	5.115
Lettuce - Romaine, Heart	3.35	3.685
Brocolinni - Gaylan, Chinese	4.53	4.983
Sauce - Ranch Dressing	1.63	1.793
Petit Baguette	2.39	2.629
Sweet Pea Sprouts	3.29	3.619
Island Oasis - Raspberry	0.74	0.814
Longan	2.26	2.486

For Task 2, I wrote a SQL query to display each product's original price and an updated price with a 10% increase:

Original Price Column: Retrieved each product's name and *unit_price*.

New Price Calculation: Created a *new_price* column that applies a 10% increase to *unit_price* using the expression (*unit_price * 1.1*).

This query allows us to view each product's original price alongside its updated price after the increase.

Task 3

For or Task 3, I created a SQL query to identify all customers born after January 1, 1990:

Retrieved Customer Data: Selected all columns from the customer's table.

Filtered by Birth Date: Used a *WHERE* clause to include only customers with a *birth_date* greater than *'1990-01-01'*.

This query helps us list all customers who were born after this specific date.

```
22    #Task 3
23    select * from customers
24    where birth_date > '1990-01-01';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	first_name	last_name	birth_date	phone	address	city	state	points
6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	IL	3073
8	Thacher	Naseby	1993-07-17	941-527-3977	538 Mosinee Center	Sarasota	FL	205
9	Romola	Rumgay	1992-05-23	559-181-3744	3520 Ohio Trail	Visalia	CA	1486
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Task 4

For Task 4, I wrote a SQL query to find the product with the highest stock quantity:

```
25
26    #Task 4
27    select quantity_in_stock,name, product_id
28    from products
29    order by quantity_in_stock desc
30    limit 1;
31
```

Result Grid | Filter Rows: | Edit:

quantity_in_stock	name	product_id
98	Sweet Pea Sprouts	7
NULL	NULL	NULL

Selected Relevant Columns: Retrieved *quantity_in_stock*, *name*, and *product_id* from the *products* table using *Select*.

Sorted by Stock Quantity: Used *ORDER BY quantity_in_stock DESC* to sort products in descending order by *quantity_in_stock*.

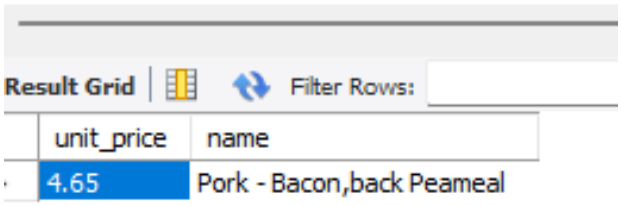
Limited Result to Top Product: Applied *LIMIT 1* to show only the product with the highest stock quantity.

This query provides the product name, ID, and stock quantity for the item with the most stock.

Task 5

For Task 5, I wrote a SQL query to find the most expensive product:

```
32      #Task 5
33 •    use sql_inventory;
34
35 •    select unit_price, name
36      from products
37      order by unit_price desc
38      limit 1;
```



Selected Relevant Columns: Retrieved *unit_price* and name from the products table.

Sorted by Price: Used *ORDER* BY *unit_price DESC* to sort the products in descending order by *unit_price*, so the most expensive product appears at the top.

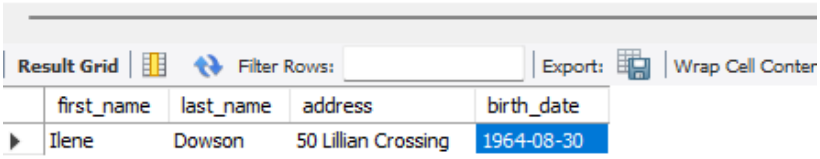
Limited to One Result: Applied *LIMIT 1* to return only the most expensive product.

This query shows the name and price of the product with the highest unit price.

Task 6

For Task 6, I wrote a SQL query to find the oldest customer based on their birthdate:

```
39
40      #Task 6
41 •    use sql_store;
42
43 •    select first_name, last_name, address, birth_date
44      from customers
45      order by birth_date asc
46      limit 1;
```



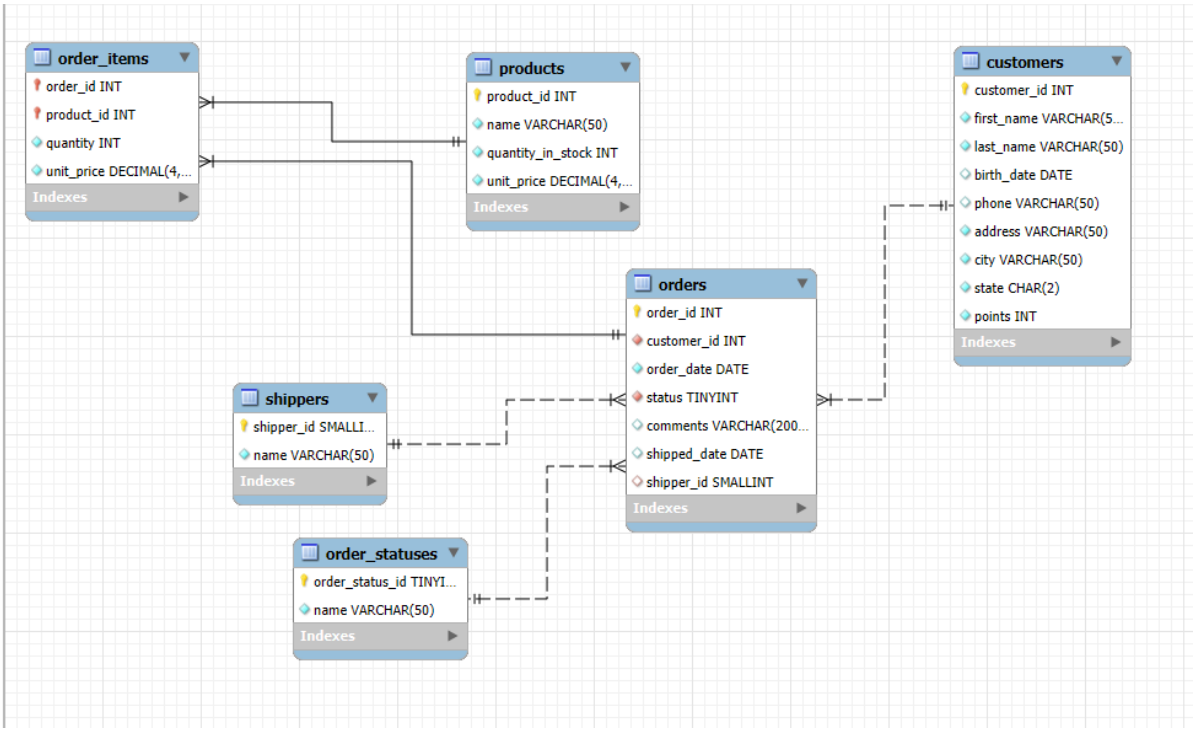
Selected Relevant Columns: Retrieved *first_name*, *last_name*, *address*, and *birth_date* from the *customers* table.

Sorted by Birth Date: Used *ORDER BY birth_date ASC* to sort customers in ascending order by *birth_date*, so the oldest customer (with the earliest birth date) appears first.

Limited to One Result: Applied *LIMIT* 1 to return only the oldest customer.

This query provides the *first_name*, *last_name*, *address*, and *birth_date* of the oldest customer in the database.

EER Diagram – Task 1



In relational databases, tables store structured data, with *primary* and *foreign* keys linking them for organized relationships and efficient querying.

- *Primary Key*: Uniquely identifies each record in a table.
- *Foreign Key*: Links a field in one table to the *primary key* of another, establishing a relationship between the two.

In the given example:

1. **Customers**: Stores customer data, with *customer_id* as the *primary key*.
2. **Products**: Contains product details, with *product_id* as the *primary key*.
3. **Orders**: Records customer orders, with *order_id* as the *primary key* and a *foreign key* linking to *customer_id*.
4. **Order_Items**: Details individual order items, with *foreign keys* to both *order_id* and *product_id*.
5. **Shippers**: Contains shipper details, with *shipper_id* as the *primary key*, linking to orders.
6. **Order_Statuses**: Tracks order statuses, with *order_status_id* as the *primary key*, linking to orders.

These relationships enable easy retrieval and management of order-related data across different tables.

MySQL Task 2

Task 1

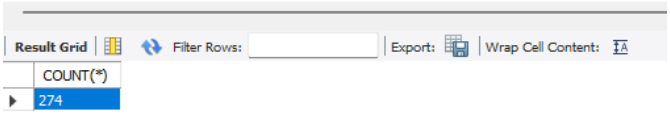
For Task 1, I wrote a SQL query to count the number of cities in the USA:

```
1    #Task 1
2    use world;
3
4    SELECT COUNT(*)
5    FROM city
6    WHERE CountryCode = 'USA';
```

Selected Count of All Rows: Used *COUNT(*)* to count all rows in the *city* table.

Filtered by Country Code: Applied a *WHERE* clause with *CountryCode = 'USA'* to limit the count to cities located in the USA.

This query returned the total number of cities in the USA in the *city* table.



Task 2

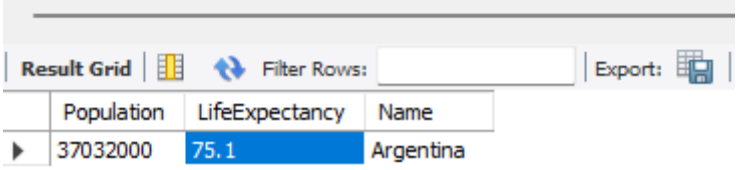
For Task 2, I wrote a SQL query to find the population and average life expectancy for Argentina:

```
8    #Task 2
9    SELECT Population, LifeExpectancy, Name
10   FROM country
11   WHERE Code = 'ARG';
```

Selected Relevant Columns: Retrieved *Population*, *LifeExpectancy*, and *Name* from the *country* table.

Filtered by Country Code: Used a *WHERE* clause with *Code = 'ARG'* to specifically get information for Argentina.

This query displays the population, life expectancy, and country name for Argentina.



Task 3

For Task 3, I wrote a SQL query to find the country with the highest life expectancy:

Selected Relevant Columns: Retrieved *Name* and *LifeExpectancy* from the *country* table.

Sorted by Life Expectancy: Used *ORDER BY LifeExpectancy DESC* to arrange countries in descending order, with the highest life expectancy at the top.

Limited to One Result: Applied *LIMIT 1* to show only the country with the highest life expectancy.

```
13    #Task 3
14    SELECT Name, LifeExpectancy
15    FROM country
16    ORDER BY LifeExpectancy DESC
17    LIMIT 1;
```

Result Grid | Filter Rows: | Export:

Name	LifeExpectancy
Andorra	83.5

This query returns the name and life expectancy of the country with the highest value in the database.

Task 4

For Task 4, I wrote a SQL query to select 25 cities around the world that start with the letter "F":

```
19    #Task 4
20    SELECT Name
21    FROM city
22    WHERE Name LIKE 'F%'
23    LIMIT 25;
24
```

Result Grid | Filter Rows:

Name
Franca
Florianópolis
Foz do Iguaçu
Ferraz de Vasconcelos
Francisco Morato
Franco da Rocha
Fuenlabrada
Faridabad
Firozabad

Selected City Name: Retrieved the *Name* column from the *city* table.

Filtered by Initial Letter: Used *WHERE Name LIKE 'F%'* to find cities whose names begin with "F".

Limited to 25 Results: Applied *LIMIT 25* to return only the first 25 cities that match this condition.

This query displays up to 25 city names starting with "F" from the *city* table.

Task 5

```
25    #Task 5
26 •  select ID, Name, Population
27    from city
28    limit 10;
29
```

Result Grid | Filter Rows:

ID	Name	Population
1	Kabul	1780000
2	Qandahar	237500
3	Herat	186800
4	Mazar-e-Sharif	127800
5	Amsterdam	731200
6	Rotterdam	593321
7	Haag	440900
8	Utrecht	234323
9	Eindhoven	201843
10	Tilburg	193238

city 6

For Task 5, I wrote a SQL query to display the first 10 rows with columns for city ID, name, and population:

Selected Relevant Columns: Retrieved *ID*, *Name*, and *Population* from the *city* table.

Limited to 10 Results: Applied *LIMIT 10* to display only the first 10 rows.

This query provides the ID, name, and population for the first 10 cities in the *city* table.

Task 6

For Task 6, I wrote a SQL query to find cities with populations greater than 2,000,000:

Selected Relevant Columns: Retrieved *Name* and *Population* from the *city* table.

Filtered by Population: Used a *WHERE* clause with *Population > 2000000* to include only cities with a population over 2 million.

This query returns the names and populations of cities with more than 2 million residents.

```
30    #Task 6
31 •  select Name, Population
32    FROM city
33    WHERE Population > 2000000;
```

Result Grid | Filter Rows:

Name	Population
Alger	2168000
Luanda	2022000
Buenos Aires	2982146
Sydney	3276207
Melbourne	2865329
Dhaka	3612850
São Paulo	9968485
Rio de Janeiro	5598953
Salvador	2302832
Belo Horizonte	2139125

city 7

```
35    #Task 7
36 •  select name
37    from city
38    where name like 'BE%'
39
40
```

Result Grid | Filter Rows:

name
Béjaïa
Béchar
Benguela
Berazategui
Belize City
Belmopan
Belo Horizonte
Belém
Belford Roxo
Betim

Task 7

For Task 7, I created a SQL query to list city names that start with "Be":

Selected City Name: Pulled the *name* column from the *city* table.

Filtered by Prefix: Used *WHERE name LIKE 'Be%'* to match cities beginning with "Be" (case-sensitive).

This gives us a list of cities whose names start with "Be" from the *city* table.

Task 8

For Task 8, I wrote a SQL query to find cities with populations between 500,000 and 1,000,000:

```
40    #Task 8
41 •  select population, name
42    from city
43    where population between 500000 and 1000000;
```

Result Grid | Filter Rows: | Export:

population	name
731200	Amsterdam
593321	Rotterdam
609823	Oran
669181	Dubai
907718	Rosario
622013	Lomas de Zamora
559249	Quilmes
538918	Almirante Brown
521936	La Plata
512880	Mar del Plata

Selected Relevant Columns: Retrieved *population* and *name* from the *city* table.

Filtered by Population Range: Used *WHERE population BETWEEN 500000 AND 1000000* to include only cities within this range.

This query gives us a list of cities with populations between 500,000 and 1,000,000.

Task 9

For Task 9, I wrote a SQL query to find the city with the lowest population:

```
45    #Task 9
46 •  select population, name
47    from city
48    order by population asc
49    limit 1;
```

Result Grid | Filter Rows:

population	name
42	Adamstown

Pulled Relevant Data: Grabbed the *population* and *name* columns from the *city* table.

Sorted by Population: Ordered the cities in *ascending* order by population using *ORDER BY population ASC*, so the city with the smallest population comes first.

Limited to One Result: Applied *LIMIT 1* to fetch only that one city.

This query returns the city with the lowest population from the *city* table.

Optional

For the optional task, I wrote a SQL query to find the capital of Spain (ESP):

Joined the Tables: Used a *JOIN* between the *city* table (aliased as c) and the *country* table (aliased as cs) based on the condition that the *ID* from the *city* table matches the *Capital* field in the *country* table.

Filtered by Country Code: Applied *WHERE cs.Code = 'ESP'* to specifically get information for Spain.

Renamed the Column: Used *AS Capital* to label the result as "Capital."

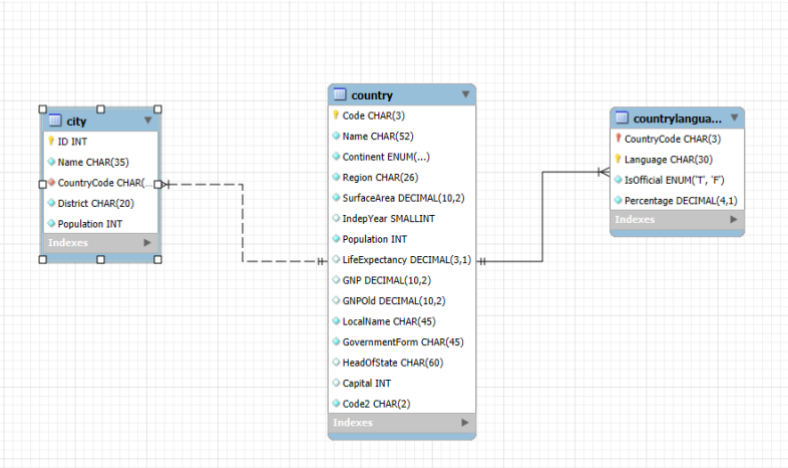
This query gives us the name of the capital city of Spain by joining the relevant data from the two tables.

```
51      #Optional
52  •   SELECT c.Name AS Capital
53      FROM city c
54      JOIN country cs ON c.ID = cs.Capital
55      WHERE cs.Code = 'ESP';
56
57
```

Result Grid | Filter Rows: | Export:

	Capital
▶	Madrid

EER Diagram – Task 2



- **Primary Key in the country table**: Code
- **Primary Key in the city table**: ID
- **Primary Key in the countrylanguage table**: CountryCode, Language (composite primary key)
- **Foreign Key in the city table**: CountryCode (references Code in the country table)
- **Foreign Key in the countrylanguage table**: CountryCode (references Code in the country table)

These relationships link cities and languages to their respective countries.

Interview Questions

What is a Query?

A query in MySQL is basically a request you make to a database to get or manipulate data. It's written in SQL, which stands for Structured Query Language. For example, when you want to retrieve data, you'd use a SELECT query. If you want to add, update, or delete data, you'd use INSERT, UPDATE, or DELETE queries respectively. Essentially, a query helps you communicate with the database to get the results you need or make changes to the data.

What is the SELECT statement?

The SELECT statement in SQL is used to pull data from a database. When you use SELECT, you're telling the database what information you want to see. For example, if you want to get everything from a table, you'd use SELECT *. But if you just want specific columns, like e.g., names or job titles, you'd list those after SELECT. You can also add conditions with WHERE to filter the results or use ORDER BY to sort them. In essence, it's the way you query the database to get the data you're looking for.

What is the WHERE clause?

The WHERE clause is used to almost filter records based on given conditions. For example, it allows you to narrow down the data you retrieve by specifying criteria. In the previous tasks, we had to find customers who were born after 01/01/1990. WHERE was used here to only retrieve those entries that matched the WHERE criteria given, allowing us to target specific rows.

What is the Primary key?

A primary key is a unique identifier for every record in a database table. It makes sure that no two roles can have the same value within a column. Primary keys are also integral in establishing relationships between various tables.

What is a Database?

A database is essentially a structured collection of data that's stored in a way that can be accessed, updated and organised by various software. It allows information to be stored in tables, allowing for relationships to be built between them. Databases are also designed to handle large amounts of data efficiently in order to retrieve or manipulate data whenever necessary.

What is Normalisation?

Normalisation is the process of organising a database in a way that makes it more efficient. Usually, this means a reduction of duplicate entries by breaking up large pieces of data into smaller individual tables with relationships instead, so each set of data is only stored once. This helps to keep a database 'clean'.

Modify query to show the population of Germany.

SELECT population FROM world

WHERE name = 'France'

SELECT population FROM world

WHERE name = 'Germany'

Select the query which gives the name of countries beginning with U.

SELECT name **FROM** world **WHERE** name **LIKE** 'U%'

Select the answer which shows the problem with this SQL code - the intended result should be the continent of France:

SELECT continent **FROM** world **WHERE** 'name' = 'France'

B) 'name' should be name

SELECT continent **FROM** world **WHERE** name = 'France'

Name in quotes causes MySQL to treat it as a string.

Select the code which shows the countries that end in A or L.

SELECT name **FROM** world **WHERE** name **LIKE** '%a' **OR** name **LIKE** '%l'

Given the table on the left, select the query which produces this table below

name	population
Bahrain	1234571
Swaziland	1220000
Timor-Leste	1066409

SELECT name, population **FROM** world **WHERE** population **BETWEEN** 1000000 **AND** 1250000