Creating Tables:

```
CREATE TABLE customers (
customer_name VARCHAR(100) PRIMARY KEY,
customer_email VARCHAR(150) UNIQUE NOT NULL,
country VARCHAR(50) NOT NULL
);

CREATE TABLE products
(
product_name VARCHAR(200) PRIMARY KEY,
price DECIMAL (10, 2) NOT NULL
);

CREATE TABLE orders(
order_number INT PRIMARY KEY,
customer_email VARCHAR NOT NULL,

FOREIGN KEY (customer_email) REFERENCES customers(customer_email)

);

CREATE TABLE order_items(
order_id SERIAL PRIMARY KEY,
order_number INT NOT NULL,
product_name VARCHAR(200) NOT NULL,
quantity INT NOT NULL CHECK (quantity > 0),
order_price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (order_number) REFERENCES orders(order_number),
FOREIGN KEY (product_name) REFERENCES products(product_name)
);
```

The reason I designed the database this way is that there is a many-to-many relationship between the orders and products. One order can have many products, and one product can be ordered by many people. The reason I had to add an Orders table as well as an Order_items table is that if the primary key order_number can only be used once, then we can't link many products to one order_number. So we have to make an order_items table to be able to include the same order_number many times with different products.

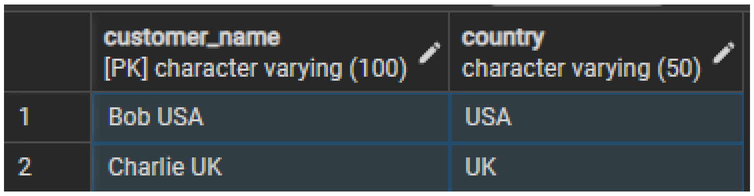
Inserting Data:

```
INSERT INTO customers (customer_name, customer_email, country) VALUES
('Alice Canada', 'alice@test.ca', 'Canada'),
('Bob USA', 'bob@test.us', 'USA'),
('Charlie UK', 'charlie@test.uk', 'UK'),
('Dave NoOrders', 'dave@test.ca', 'Canada');


INSERT INTO products (product_name, price) VALUES
('Laptop', 1000.00),
('Mouse', 50.00),
('Keyboard', 100.00);


INSERT INTO orders (order_number, customer_email) VALUES
(101, 'alice@test.ca'),
(102, 'bob@test.us'),
(103, 'charlie@test.uk');

INSERT INTO order_items (order_number, product_name, quantity,
order_price) VALUES
(101, 'Laptop', 1, 1000.00), -- Alice bought a Laptop
(102, 'Mouse', 2, 50.00),    -- Bob bought 2 Mice
(103, 'Keyboard', 1, 100.00); -- Charlie bought a Keyboard
```

The reasoning for the data I filled into the tables is simply so that I had a case for every query that I had to make.

1. Find all customers from outside of Canada.

```
SELECT customer_name, country
FROM customers
WHERE country != 'Canada';
```

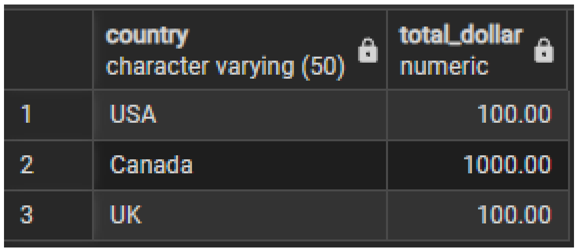


	customer_name [PK] character varying (100)	country character varying (50)
1	Bob USA	USA
2	Charlie UK	UK

The reasoning behind this query is simple. Since I have to find customers outside Canada, I just select the columns customer_name and country from the customers table where the country is NOT Canada.

2. Find the total dollar amount of orders per country.

```
SELECT
               customers.country,
               SUM(order_items.quantity *
           order_items.order_price) AS total_dollar
     FROM customers
     JOIN orders ON customers.customer_email =
     orders.customer_email
     JOIN order_items ON orders.order_number =
     order_items.order_number
     GROUP BY customers.country;
```

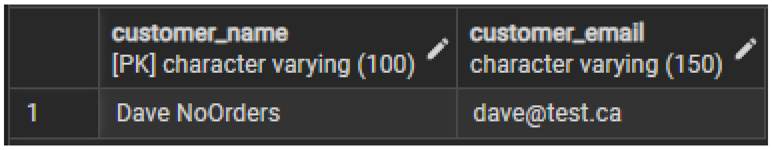


	country character varying (50)	total_dollar numeric
1	USA	100.00
2	Canada	1000.00
3	UK	100.00

The reasoning behind this query is that I have to select the column country from the customers table, and then the next column is the product of the order_items.quantity with the order_items.order_price, which will give me the total cost of the order. To specify that I want the product to be calculated per country, I have to specify FROM customers and then collect the data from orders and customers.customer_email, since customer_email is the primary key. And then I also have to collect the data from the order_items and the order_number values so that the query knows which data to collect. And finally, I group it by Country with the last line of the Query.

3. Find all customers who have no orders.

```
SELECT
              customers.customer_name,
              customers.customer_email
     FROM customers
     LEFT JOIN orders ON customers.customer_email =
     orders.customer_email
     WHERE orders.order_number IS NULL;
```



	customer_name [PK] character varying (100)	customer_email character varying (150)
1	Dave NoOrders	dave@test.ca

For this query, the reasoning is similar to the one before, but instead of getting all the data, since the order number will be NULL if there is no order, we have to use a LEFT JOIN, which will select all the data in the column. Otherwise, there will be no data if we just use the JOIN function, since the data will be empty if there is no order number.

Question:

Additionally, please describe how you would handle a situation where the database becomes too large to manage efficiently.

When the database becomes larger, the biggest issue will become queries. There are a few things we can do to optimize the database, such as indexing. With indexing, instead of the database going through every single data entry one by one to get a certain entry, it can just go straight to the index, similar to a table of contents, which will let the database know where the entry is.

Another thing we can do is create duplicate databases. One of them can be for searching, and then we can have another duplicate database, which would be the main database used for writing new data. This way,y\ if people are searching for an item on the website, it would be a lot faster than if we had one database that would do everything. The reading and writing would be separate.

Another important thing we can do is to start archiving old data, like orders from more than 10 years ago. We would move those to another database, which would not be used for writing new orders and would only be used for storing old data. That way, we can declutter the database that needs to be used for new orders, products and customers.