Lesson - 5 Homework:

Question 1: What is a primary key in a table?

You got it. Let's break it down like I'm explaining it to a friend over coffee.

Basically, a primary key is like the ultimate unique ID for everything in a table. Its one job is to make sure every single row can be told apart from every other row.

Think of it like this:

In a spreadsheet of your friends, you might use their phone number as the primary key. Even if two people have the same name, you know they're different people because their numbers are unique.

For a list of your favorite songs, the primary key could be a combo of Song Title + Artist Name. Together, that combo is unique for each track.

There are two ironclad rules it has to follow:

It has to be unique. No duplicates allowed. Ever.

It can't be empty. Every single row has to have a value for it.

Why does this matter so much? It's all about relationships. This unique ID is what you use to connect this table to other tables without getting your data mixed up. It's the anchor that keeps everything organized and linked together properly.

So yeah, it's just the designated unique identifier for a table. Nothing more, nothing less.

Question 2: Name the two types of table relationships in Power BI.

1. One-to-Many (The Most Common)
This is by far the most frequent relationship you'll set up. It means:

One row in Table A can match many rows in Table B.

The "one" side table is usually a dimension or lookup table (like a list of Products, Customers, or Dates).

The "many" side table is usually a fact table (like Sales, Orders, or Transactions).

Real-World Example:

Products table (One side): Has one row for each unique product (ProductID 101: "Coffee Mug").

Sales table (Many side): Has many rows of sales transactions, and many of them can be for ProductID 101.

Power BI automatically creates a 1 on the "one" side and an * on the "many" side to show this relationship.

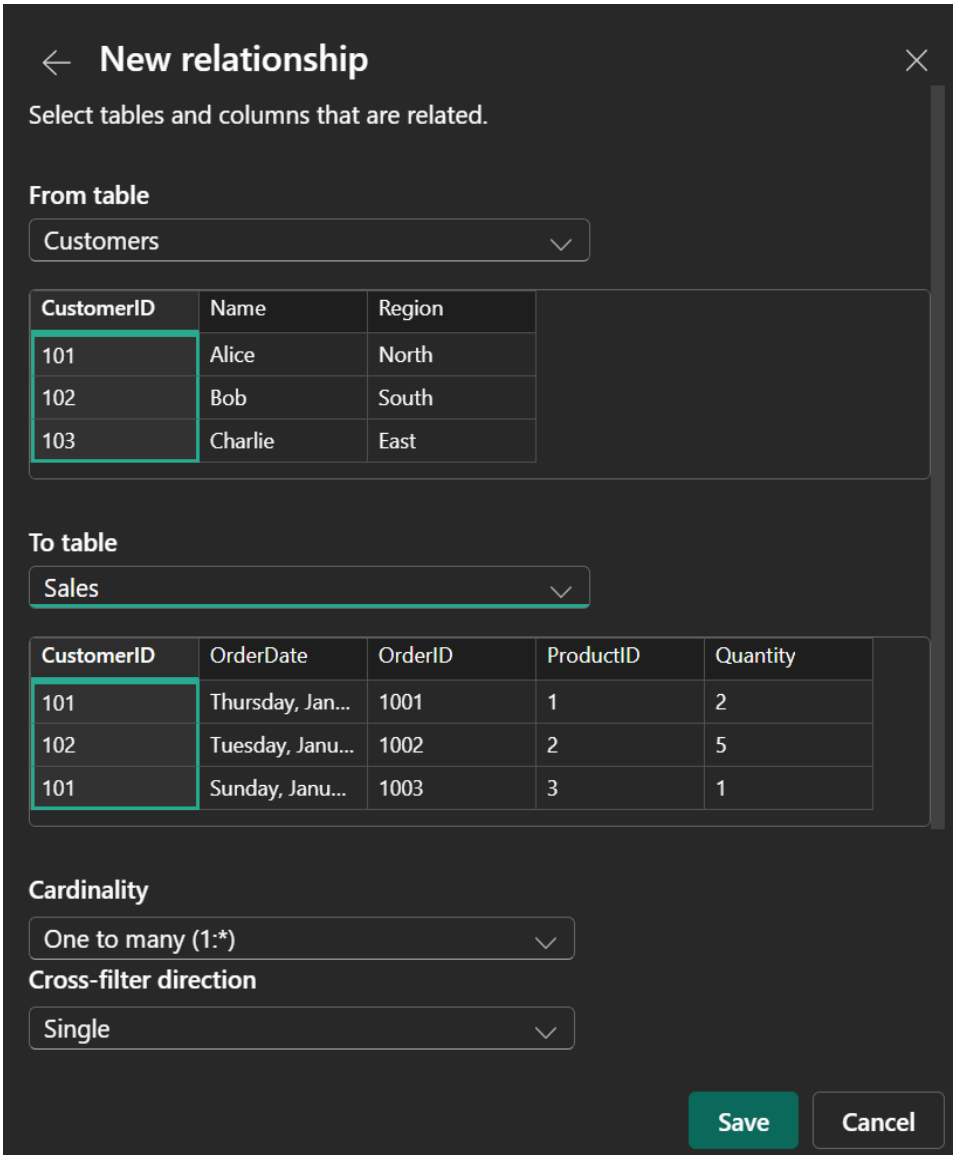
2. Many-to-Many
This is less common and requires more careful thinking. It means:

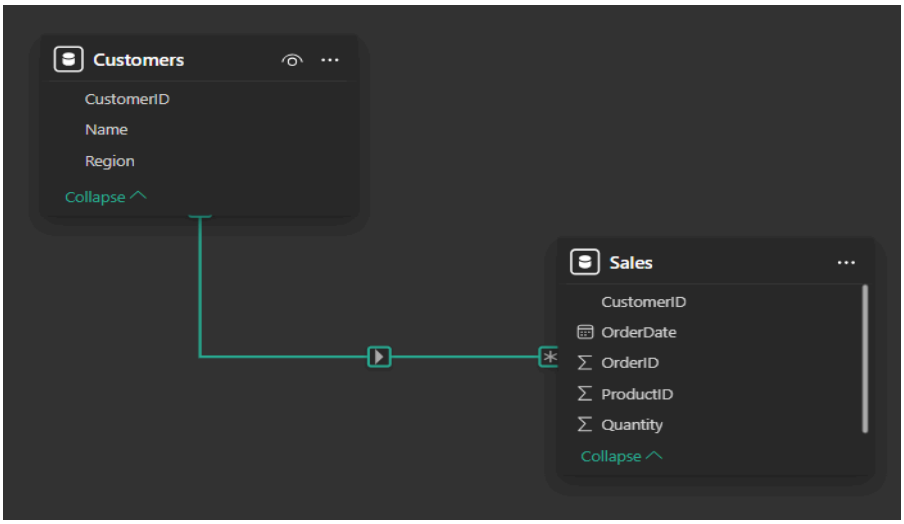
Many rows in Table A can match many rows in Table B.
Real-World Example:

Students table: A single student (e.g., Alice) can be enrolled in many classes.
Classes table: A single class (e.g., Biology 101) can have many students enrolled.

Question 3: How do you create a relationship between two tables in Power BI?

1. Go to the Modeling tab.
2. Click Manage Relationships > New.
3. Select the two tables.
4. Choose the matching columns for each.
5. Click OK.
6. Or, just drag the field from one table and drop it onto the matching field in the other table in the Model view.





Question 4: What is a "star schema"?

A star schema is a way to organize data for reporting.

It has two types of tables:

One Fact Table: The main table. It holds the numbers you want to analyze (like sales, quantity).
Many Dimension Tables: Surround the fact table like stars. They describe the facts (like product, date, customer).

Simple Example:
Fact Table: Sales (with Sales Amount, Quantity)
Dimension Tables: Products, Dates, Stores

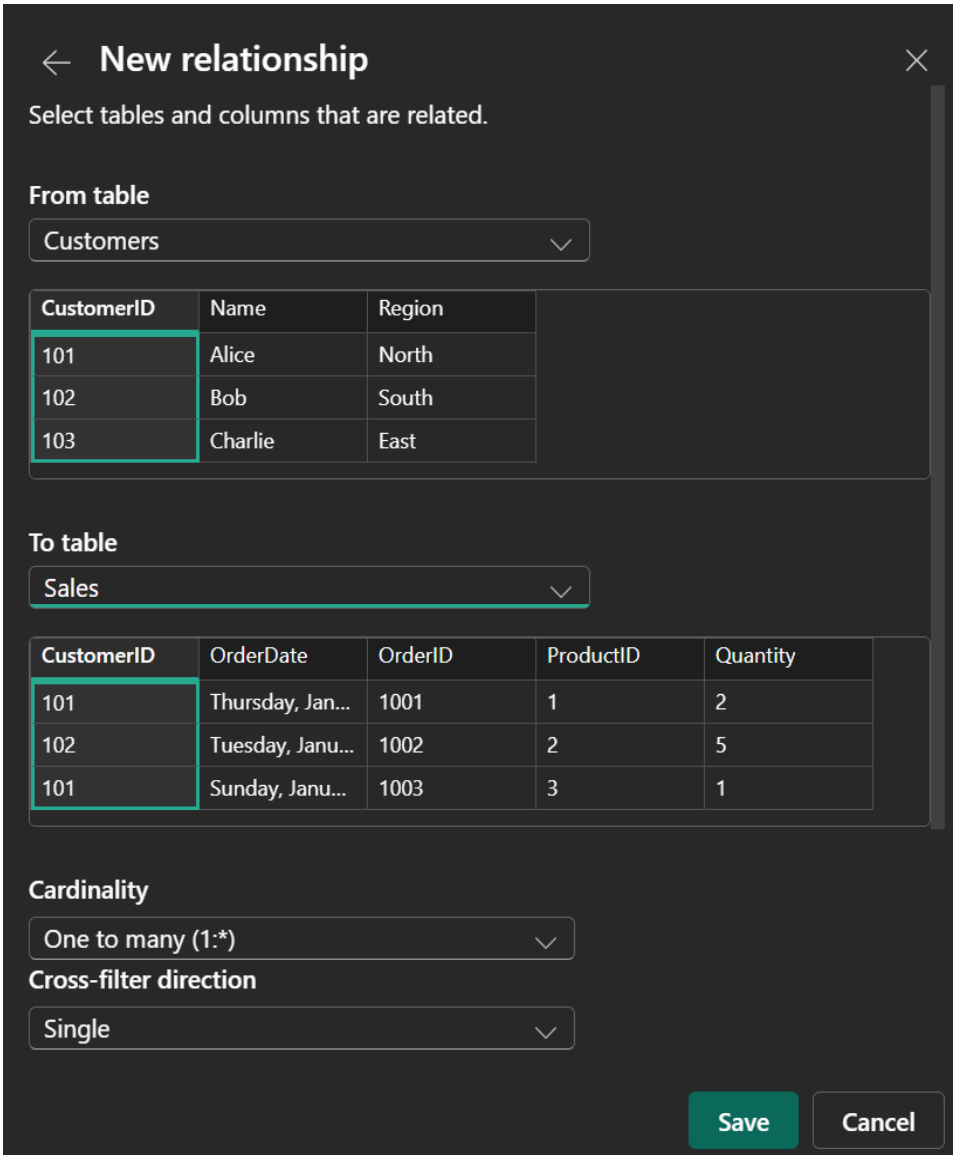
Question 5: Which table is typically the fact table in a sales dataset?

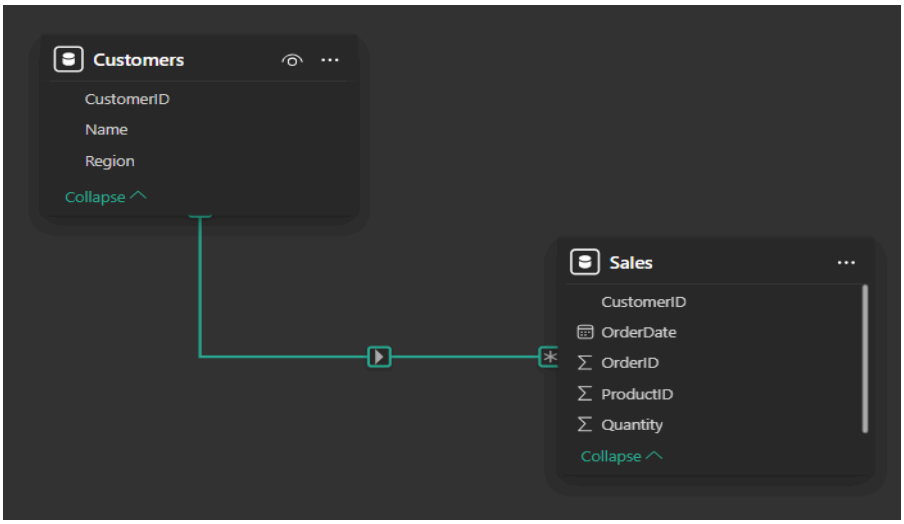
The Sales table.

It's the fact table because it contains the measurable numbers (facts) like:
Sales Amount
Quantity Sold
Profit
Cost

It connects to dimension tables like Product, Date, Customer, and Store which describe the who, what, and when of each sale.

Question 6: Link Sales.csv to Customers.csv using CustomerID (one-to-many).





Question 7: Why is ProductID in Sales.csv a foreign key?

Because it belongs to a different table.

Primary Key: It lives in the Products table, where it uniquely identifies each product.
Foreign Key: It lives in the Sales table to point back to which product was sold in each transaction.

It creates the link between the Sales fact table and the Products dimension table.

Question 8: Fix a relationship error where ProductID has mismatched data types.

Find the Tables: Go to the Data or Model view.

Check Data Types:

Find the ProductID column in the Sales table.
Find the ProductID column in the Products table.

Make Them Match:

Right-click the column with the wrong type (usually Whole Number or Text).
Select Change Type and choose the correct type to match the other table.

Refresh: The relationship should now work automatically.

Question 9: Explain why a star schema improves performance.

Why it's faster:

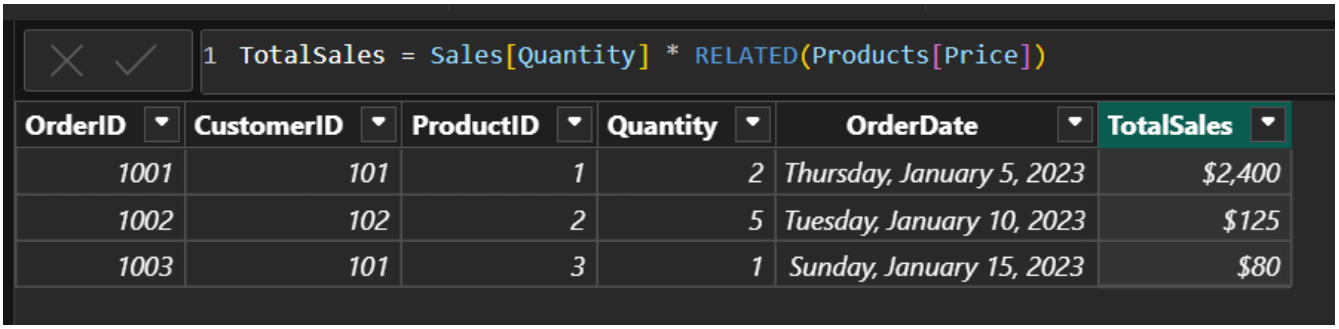
Simpler Queries: The database can quickly connect the fact table to any dimension with a single, simple link.

Less Data to Scan: Filters (like Year = 2023) are applied to small dimension tables first, drastically reducing the number of rows scanned in the large fact table.

Optimized for Aggregation: The design is perfect for tools like Power BI to sum, count, and average the numbers in the fact table, which is their main job.

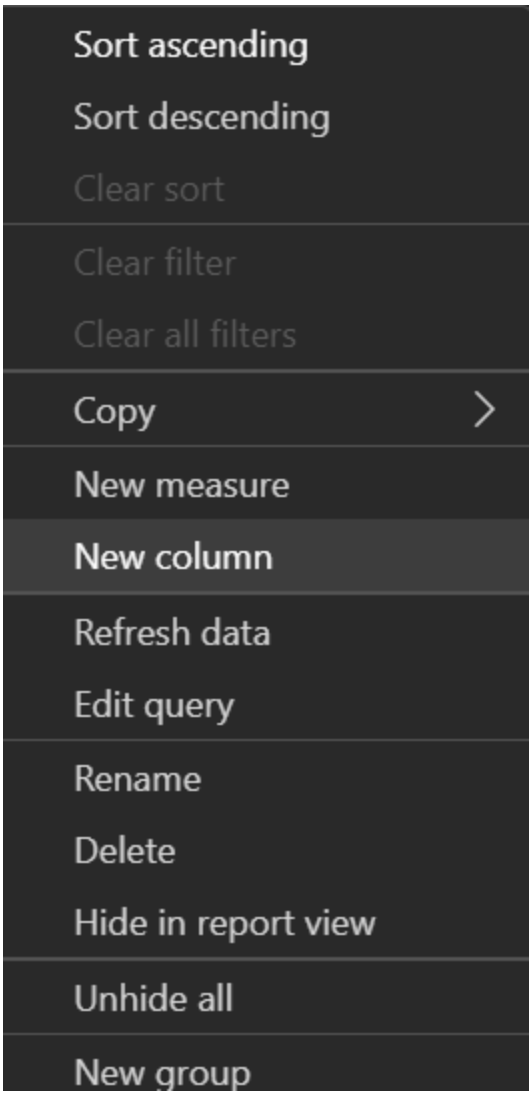
In short, it helps the engine find and calculate your data much faster.

Question 10: Add a new column TotalSales in Sales (Quantity * Price from Products).



```
1 TotalSales = Sales[Quantity] * RELATED(Products[Price])
```

OrderID	CustomerID	ProductID	Quantity	OrderDate	TotalSales
1001	101	1	2	Thursday, January 5, 2023	$2,400
1002	102	2	5	Tuesday, January 10, 2023	$125
1003	101	3	1	Sunday, January 15, 2023	$80



Question 11: Optimize a model with circular relationships—how would you resolve it?

Resolve circular relationships by breaking the loop. Here's how:

1. Identify the Loop
Check the Model view — a loop exists if there's a closed path of relationships between tables (e.g., Table A → B → C → A).

2. Remove or Adjust Redundant Relationships
Delete unnecessary relationships: Remove any redundant or unused relationships that contribute to the loop.
Use one-directional filters: Ensure relationships flow in a single direction (from dimension → fact), not both ways.

3. Leverage Role-Playing Dimensions
If a table (e.g., Date) is connected multiple times to the same fact table, create separate dimension tables for each role (e.g., OrderDate, ShipDate) instead of reusing one table.

4. Use DAX to Handle Complex Paths
If you can't delete a relationship, use USERELATIONSHIP() in measures to specify which relationship to use dynamically.

Example Fix:
Before:
Sales → Date (OrderDate)
Sales → Date (ShipDate)
(This creates a loop if other tables connect back to Date.)

After:
Create two copies of the Date table:
OrderDate and ShipDate, then connect each to Sales separately.
This breaks the loop while preserving functionality.

Question 12: Optimize a model with circular relationships—how would you resolve it?

Duplicate the Date Table
Right-click your Date table in the Fields pane.

Select Duplicate.

Rename the new tables:

OrderDate
ShipDate

2. Create Relationships
Go to the Model view.

Drag and drop:
OrderDate[Date] → Sales[OrderDate]
ShipDate[Date] → Sales[ShipDate]

3. Hide the Original Date Table
Right-click the original Date table and select Hide in report view (to avoid confusion).

4. Use in Reports
Use OrderDate fields for order-based analysis (e.g., "Sales by Order Date").

Use ShipDate fields for shipping-based analysis (e.g., "Shipping Delay").

Question 13: Handle a many-to-many relationship between Customers and Products.

Create a Junction Table
Build a bridge table that contains all unique combinations of CustomerID and ProductID.

This table is often called a junction table or bridge table.

Example: Name it CustomerProducts, with columns:
CustomerID (links to Customers table)
ProductID (links to Products table)
(Optional: Add metrics like TotalQuantityPurchased)

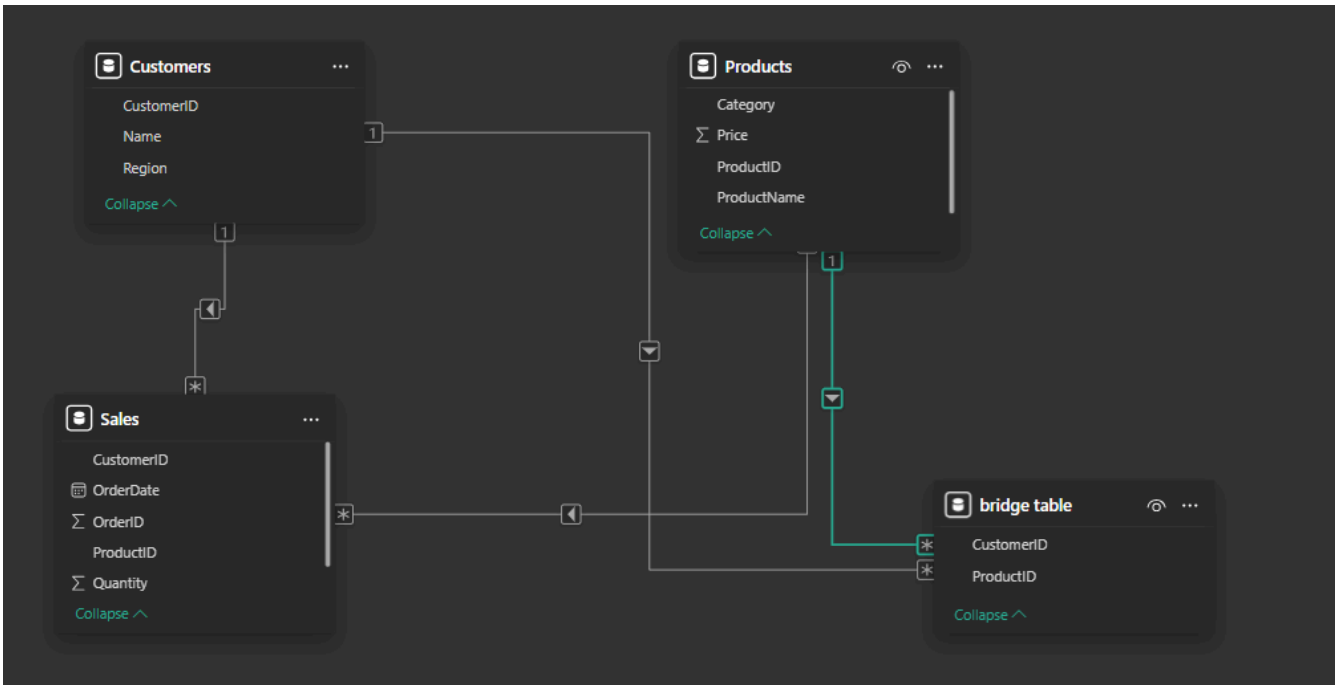
2. Set Up Relationships
Create two one-to-many relationships:

Customers (1) → CustomerProducts (*)

Products (1) → CustomerProducts (*)

DAX

```
TotalProductsPerCustomer =
CALCULATE(
   DISTINCTCOUNT(CustomerProducts[ProductID]),
   ALL(Products)
)
```



Name	Keyboard	Laptop	Mouse	**Total**
Alice	1	2		**3**
Bob			5	**5**
Total	**1**	**2**	**5**	**8**

Question 14: Use bidirectional filtering sparingly—when is it appropriate?

When is it Appropriate?
Use bidirectional filtering sparingly and only in very specific cases. It is appropriate only when you have a specific, complex filtering need that cannot be solved by a better data model design.

The Two Main Appropriate Use Cases:
For a Fact-to-Fact Table Relationship (Very Rare):

When: You have two fact tables (e.g., Sales and Budget) that are connected through a common dimension (e.g., Date), and you need to filter one fact table based on the other.

Example: You want a report that shows sales and budget for the same dates. A bidirectional filter on the Date table allows selecting a date from the Sales table to also filter the Budget table, and vice-versa.

Warning: This can easily hurt performance and create confusion. It's often better to combine these facts into one table.

For a "Double Hop" Filter (More Common, but Still Careful):

When: You have a chain of tables, and you need to filter the first table based on the last table.

Example: You have Products <- Sales -> Customers. You want to see a list of Products purchased by a specific Customer. A bidirectional filter on the Sales table allows filtering Products based on Customers.

Better Solution: Usually, a better-designed model with a star schema avoids this need. This is often a sign that your model needs a bridge table instead.

The Golden Rule: When NOT to Use It
Never use bidirectional filtering on a regular dimension table (like Products, Customers, Date) in a standard star schema.

Why?

Performance: It can severely slow down your report because Power BI has to check and filter many more tables.

Confusion: It can lead to unexpected and confusing results, like filters affecting tables you didn't intend to filter.

Circular Logic: It can easily create circular relationships, which break your model.

What to Do Instead :
99% of the time, you should use single-direction filtering (the default). If you think you need bidirectional filtering, first try to redesign your data model (e.g., using a bridge table for many-to-many relationships).

Question 15: Write DAX to enforce referential integrity if a CustomerID is deleted.

Since you can't do it in DAX, here is the correct way to think about and answer the question:

"DAX is a calculation and analysis language within Power BI, which is a reporting tool. Power BI does not handle data deletion or the enforcement of referential integrity rules; this is managed at the database level.

To enforce referential integrity for a CustomerID in a relational database (like SQL Server), you would define a FOREIGN KEY constraint on the Sales.CustomerID column that references the Customers.CustomerID column, and set the delete rule to CASCADE or RESTRICT.

ON DELETE CASCADE: Automatically deletes all sales records for a customer when that customer is deleted from the Customers table.

ON DELETE RESTRICT: Prevents a customer from being deleted if any sales records for them exist.