Lesson-5

1.What is a primary key in a table?

A primary key in a table is a column (or a set of columns) that uniquely identifies each row in that table.

Key Properties of a Primary Key:

Uniqueness – No two rows can have the same primary key value.

Non-Null – A primary key column cannot contain NULL values.

Immutable (Ideally) – Should not change over time to maintain data integrity.

Single or Composite – Can consist of one column (simple key) or multiple columns (composite key)

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

In Power BI, the two main types of table relationships are:

1. One-to-Many (1:N)

The most common relationship type.

One row in the primary table (dimension) can relate to multiple rows in the related table (fact).

Example:

Products (one) → Sales (many)

A single product can appear in many sales records.

2. Many-to-Many (N:N)

Allows multiple rows in one table to relate to multiple rows in another.

Requires a bridge/junction table (unless using Power BI's "dual" storage mode).

Example:

Students (many) ↔ Courses (many)

A student can enroll in multiple courses, and a course can have multiple students.

Additional Notes:

One-to-One (1:1) relationships exist but are rare (e.g., Employee ↔ EmployeeDetails).

Relationships are defined in Power BI's "Model" view using drag-and-drop.

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

How to Create a Relationship Between Two Tables in Power BI

To establish a relationship between two tables in Power BI, follow these steps:

Method 1: Using the "Model" View (Drag-and-Drop)

Open Power BI Desktop and go to the "Model" view (bottom-left pane).

Drag a column from one table (e.g., CustomerID in Customers) and drop it onto the matching column in another table (e.g., CustomerID in Sales).

Configure the Relationship (if needed):

Cardinality: Choose One-to-Many (1:N) (most common), Many-to-Many (N:N), or One-to-One (1:1).

Cross-filter direction: Typically Single (from "one" side to "many" side) or Both (bidirectional filtering).

Make this relationship active: Ensures it’s used by default in calculations.

Method 2: Using "Manage Relationships"

Go to "Home" → "Manage Relationships".

Click "New" → Select the two tables and their matching columns.

Set the relationship type and cross-filter direction → Click "OK".

4.What is a "star schema"?

A star schema is a database modeling technique used in data warehousing and Power BI to organize data for efficient querying and analysis. It consists of:

One Central Fact Table

Contains measurable data (e.g., sales revenue, quantity sold).

Includes foreign keys linking to dimension tables.

Example: Sales (with columns like SaleID, ProductID, CustomerID, DateID, Amount).

Multiple Dimension Tables (Surrounding the Fact Table, Like a Star)

Store descriptive attributes (e.g., product details, customer info, dates).

Each dimension table connects to the fact table via a primary key → foreign key relationship.

Examples:

Products (ProductID, ProductName, Category)

Customers (CustomerID, Name, Region)

Dates (DateID, Day, Month, Year)

Key Features of a Star Schema:

☑ Simplified Queries: Optimized for fast aggregations (e.g., sum of sales by product category).

☑ Denormalized Structure: Dimension tables may contain redundant data (e.g., Category repeated in Products) to avoid complex joins.

☑ Clear Relationships: One-to-many (1:N) links between dimensions and facts.

☑ Power BI-Friendly: Ideal for DAX measures and efficient visuals.

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

In a sales dataset, the fact table is typically the Sales table (or similarly named, like FactSales, Transactions, or Orders).

Why is Sales the Fact Table?

Contains Measurable Metrics (Facts):

Stores numerical data like:

Revenue

QuantitySold

Profit

Discount

Holds Foreign Keys to Dimensions:

Links to dimension tables via IDs, such as:

ProductID → Products table

CustomerID → Customers table

DateID → Dates table

StoreID → Stores table

Each row often represents a single sale/order line item.

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

Follow these steps to create a 1:N (One-to-Many) relationship between the Sales fact table and the Customers dimension table:

Step 1: Load the Data

Open Power BI Desktop.

Go to Home → Get Data → Text/CSV.

Import both files:

Sales.csv (Fact Table)

Customers.csv (Dimension Table)

Step 2: Verify the Columns

Ensure both tables have a CustomerID column (same data type, e.g., Whole Number or Text).

Example structure:

Customers (Dimension - "One" Side)

CustomerID (PK)	CustomerName	Region
101	John Doe	East
102	Jane Smith	West

Sales (Fact - "Many" Side)

OrderID	CustomerID (FK)	Product	Revenue
5001	101	Laptop	$999
5002	102	Mouse	$25

Step 3: Create the Relationship

Method 1: Drag-and-Drop in Model View

Go to the "Model" view (bottom-left icon).

Drag CustomerID from Customers (dimension) and drop it onto CustomerID in Sales (fact).

Power BI will auto-detect the One-to-Many (1:N) relationship.

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

In a relational database or Power BI data model, ProductID in Sales.csv is a foreign key because:

1. It Links to a Primary Key in Another Table

The Products table (dimension) has ProductID as its primary key (PK) (unique identifier).

The Sales table (fact) references this PK to establish a relationship, making ProductID in Sales a foreign key (FK).

2. It Enforces a One-to-Many Relationship

One product (in Products) can appear in many sales transactions (in Sales).

Example:

Products: ProductID = "P100" (e.g., "Laptop")

Sales: Multiple rows with ProductID = "P100" (each representing a sale of that laptop).

3. It Maintains Data Integrity

Prevents invalid entries: A ProductID in Sales must exist in the Products table (unless NULLs are allowed).

Ensures consistency: Product details (name, price) are stored once in Products, avoiding redundancy

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

Step 1: Identify the Mismatch

Go to Data View → Check the data type of ProductID in both tables:

Products table: ProductID might be Text (e.g., "P100").

Sales table: ProductID might be Whole Number (e.g., 100).

Error Example:

"Cannot create relationship because data types do not match."

Step 2: Change Data Types to Match

Option 1: Convert Both to Text (Recommended for Alphanumeric IDs)

In Data View, select the ProductID column in both tables.

Go to Column Tools → Data Type → Select Text.

Option 2: Convert Both to Whole Number (If IDs are Numeric)

Only works if ProductID values are numbers only (e.g., 100 instead of "P100").

Option 3: Transform Data in Power Query

If the format is inconsistent (e.g., "P100" vs. 100):

Open Power Query Editor (Home → Transform Data).

For the Sales table:

Select ProductID → Transform → Format → Add Prefix (e.g., "P").

Or use Replace Values to standardize formats.

Click Close & Apply.

Step 3: Recreate the Relationship

Go to Model View.

Delete the old (broken) relationship (right-click → Delete).

Drag ProductID from Products to Sales to create a new One-to-Many relationship.

Step 4: Verify the Fix

Check the Model View for a valid connection (line between tables).

Test with a visual (e.g., a table showing ProductName from Products and Revenue from Sales).

Prevention Tips

- ✔ Standardize Data Types Early: Ensure PK/FK columns match during data import.
- ✔ Use Power Query for Cleaning: Fix inconsistencies before modeling.
- ✔ Error Checks: Use View → Relationships to spot warnings.

9.Explain why a star schema improves performance.

A star schema enhances performance in analytical queries and Power BI reports due to its denormalized, simplified structure, optimized for fast aggregations and filtering. Here's why:

1. Fewer Joins = Faster Queries

Normalized schemas (e.g., snowflake) require complex joins across multiple tables, slowing down queries.

Star schema minimizes joins:

One central fact table connects directly to denormalized dimension tables.

Example: To analyze sales by product category, Power BI only needs to join Sales → Products (no extra hops).

2. Optimized for Aggregations

Fact tables store numeric measures (e.g., Revenue, Quantity) in a columnar format, enabling:

Faster SUM/AVG/COUNT calculations (DAX measures execute efficiently).

Compression benefits: Columnar storage (VertiPaq engine) compresses repetitive values (e.g., ProductID).

3. Efficient Filtering

Dimension tables act as filter slicers for the fact table:

Filtering by Category (in Products) directly impacts Sales without intermediate tables.

Enables predicate pushdown (filters apply early in query execution).

4. Simplified DAX Measures

Relationships are clear (1:N), avoiding ambiguity in calculations.

Example:

dax

Total Revenue = SUM(Sales[Revenue]) // Works seamlessly with filters from `Products` or `Customers`.

5. Better Compression & Storage

Dimension tables are small and reusable (e.g., Dates table used across multiple facts).

Fact tables are optimized for large-scale analytics (millions of rows).

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

To calculate TotalSales in the Sales table (using Quantity from Sales and Price from Products), follow these steps:

Method 1: Using Power Query (Recommended for ETL)

Open Power Query Editor:

Go to Home → Transform Data → Transform Data.

Merge Sales with Products:

Select the Sales table → Home → Merge Queries.

Choose Products as the second table.

Select ProductID in both tables → Join Kind: Left Outer → OK.

Expand the Products Column:

Click the ↗ icon on the new Products column.

Check only Price → OK.

Add Custom Column:

Go to Add Column → Custom Column.

Name it TotalSales.

Formula:

powerquery

[Quantity] * [Price]

Set data type to Decimal Number → OK.

Remove the Price Column (Optional):

Right-click Price → Remove.

Apply Changes:

Click Close & Apply.

Method 2: Using DAX (Calculated Column)

If you prefer DAX:

Go to Data View → Select the Sales table.

Click New Column and enter:

dax

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

RELATED fetches the Price from the Products table using the ProductID relationship.

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

1. Identify the Circular Path

Example Scenario:

Sales → Customers → Regions → Sales

(Sales filter flows through Customers and Regions back to Sales, creating a loop).

Check Power BI's "Relationship View" for dotted lines (inactive relationships) or warnings.

2. Solutions to Break the Loop

Option 1: Remove Redundant Relationships

Delete the least critical relationship (e.g., remove Regions → Sales if Customers → Sales already exists).

When to use: If one relationship is redundant for your reporting needs.

Option 2: Use Inactive Relationships + USERELATIONSHIP

Keep all relationships but mark one as inactive (right-click → Properties → uncheck "Active").

Use USERELATIONSHIP in DAX to activate it only when needed:

dax

```
Revenue by Region =
CALCULATE(
   SUM(Sales[Revenue]),
   USERELATIONSHIP(Sales[RegionID], Regions[RegionID])  // Override active relationship
)
```

When to use: When both paths are needed contextually.

Option 3: Flatten Dimensions (Denormalize)

Merge columns from Regions into Customers (e.g., add RegionName directly to Customers).

When to use: If dimensions are small and rarely updated.

Option 4: Bridge Tables for Many-to-Many

If the loop involves many-to-many (e.g., Products ↔ Categories), introduce a bridge table.

3. Validate the Fix

Test with a measure like:

dax

Total Sales = SUM(Sales[Revenue]) // Should return correct values without ambiguity.

Check for performance improvements in DAX Studio or Performance Analyzer.

Example: Before vs. After

Before (Circular)

Sales → Customers → Regions → Sales

After (Fixed)

Sales → Customers → Regions

Sales → Customers // Direct relationship remains active

Key Takeaways

☑ Eliminate ambiguity: Ensure one clear filter path.

☑ Use USERELATIONSHIP for dynamic scenarios.

☑ Denormalize sparingly: Balance simplicity with maintainability.

12.Create a role-playing dimension for OrderDate and ShipDate.

Implementation Steps

Method 1: Duplicate Date Table (Recommended)

Duplicate the Dates table twice:

OrderDates: Linked to Sales[OrderDate].

ShipDates: Linked to Sales[ShipDate].

Create relationships:

OrderDates[Date] → Sales[OrderDate] (Active).

ShipDates[Date] → Sales[ShipDate] (Active).

Method 2: Single Date Table with Inactive Relationships

Keep one Dates table.

Create relationships

Active: Dates[Date] → Sales[OrderDate].

Inactive: Dates[Date] → Sales[ShipDate].

Use DAX for ship-date analysis:

dax

```
Shipped Revenue =
CALCULATE(
   SUM(Sales[Revenue]),
   USERELATIONSHIP(Sales[ShipDate], Dates[Date])
)
```

3. Expected Output

Report Visuals

Order Date Analysis:

"Total Revenue by Order Month" (uses OrderDates).

Ship Date Analysis:

"Average Shipping Delay" (uses ShipDates).

Data Model Diagram

OrderDates[Date] ——→ Sales[OrderDate]

ShipDates[Date] ———→ Sales[ShipDate]

4. Key Notes

Data Types: Ensure OrderDate/ShipDate in Sales and Date in dimension tables are the same (e.g., Date/Time).

Time Intelligence: Mark OrderDates as the official date table for functions like TOTALYTD().

Performance: Method 1 (duplicate tables) is faster for large datasets.

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

In a scenario where:

One customer buys multiple products, and

One product is purchased by multiple customers,

you need to resolve the many-to-many relationship to avoid ambiguity in calculations.

Solution: Use a Bridge/Junction Table

Create a bridge table that records all unique combinations of CustomerID and ProductID (typically your Sales or Orders table acts as this bridge).

Step-by-Step Implementation

1. Sample Data Structure

Tables:

Customers

CustomerID CustomerName

101 John Doe

102 Jane Smith

Products

ProductID ProductName Price

P100 Laptop 999

P200 Mouse 20

Sales (Bridge Table)

OrderID CustomerID ProductID Quantity

5001 101 P100 1

5002 101 P200 2

5003 102 P100 1

2. Create Relationships

Link Sales to both dimensions:

Customers[CustomerID] → Sales[CustomerID] (1:N)

Products[ProductID] → Sales[ProductID] (1:N)

Model View:

Customers (1) → Sales (N) ← Products (1)

3. DAX Measures for Accurate Aggregations

Use SUMX or iterators to respect the bridge table:

dax

```
Total Revenue =
SUMX(
   Sales,
   Sales[Quantity] * RELATED(Products[Price])  // Correctly handles N:N
)
```

For customer-specific metrics:

```
dax
Revenue per Customer =
CALCULATE(
   [Total Revenue],
   ALL(Products)  // Remove product filter if needed
)
```

Key Considerations

Avoid Direct N:N Relationships:

Power BI allows direct many-to-many relationships, but they can cause:

Performance issues.

Incorrect aggregations (double-counting).

Use a Bridge Table Instead:

Ensures accurate filtering (e.g., "Show customers who bought laptops").

Works with all DAX functions.

Handle Sparse Data:

If not all customer-product combinations exist, use CROSSJOIN in DAX:

```
dax
All Combinations =
GENERATE(
   CROSSJOIN(Customers, Products),
   VAR Rev = [Total Revenue]
   RETURN ROW("Revenue", Rev)
)
```

Example Report Output

CustomerName	ProductName	Revenue
John Doe	Laptop	999
John Doe	Mouse	40
Jane Smith	Laptop	999

Advanced: Aggregations Without a Physical Bridge

If no bridge table exists, create a virtual bridge using DAX:

dax

```
CustomersToProducts =
SUMMARIZE(
   Sales,
   Sales[CustomerID],
   Sales[ProductID]
)
```

Then use this table in relationships (less performant than a physical table).

Why This Works

The bridge table resolves ambiguity by storing valid combinations.

Relationships flow through the bridge, ensuring correct filtering.

14.Use bidirectional filtering sparingly—when is it appropriate?

Bidirectional cross-filtering (setting relationships to "Both") can be powerful but risky. Here's when it's appropriate—and when to avoid it:

☑ Appropriate Use Cases

1. Star Schema with Dimensions Sharing Filters

Scenario: Two dimensions (e.g., Customers and Products) both need to filter a fact table (Sales) and each other.

Example:

Filter Sales by Region (from Customers) and Category (from Products).

Bidirectional filtering between Sales and both dimensions ensures the filters apply correctly.

2. Bridge Tables in Many-to-Many (N:N) Relationships

Scenario: A bridge table (e.g., Sales) connects two dimensions (e.g., Customers and Products).

Why: Allows filters to flow from Customers → Sales → Products and vice versa.

Example:

Show products bought by customers in a specific region.

3. Small, Highly Related Dimensions

Scenario: Two small tables (e.g., Employees and Departments) where mutual filtering is critical.

Example:

Filter Employees by Department while also showing department stats based on selected employees.

❌ When to Avoid Bidirectional Filtering

1. Large Datasets or Complex Models

Risk: Performance degradation due to excessive filter propagation.

Fix: Use single-directional filtering (from "one" side to "many" side).

2. Multiple Relationships Between Tables

Risk: Ambiguity in filter context (Power BI may throw errors).

Example:

Dates linked to both OrderDate and ShipDate in Sales.

Use inactive relationships + USERELATIONSHIP() instead.

3. Circular Relationships

Risk: Infinite loops (e.g., TableA → TableB → TableC → TableA).

Fix: Flatten dimensions or use DAX measures with CROSSFILTER().

⚡ Best Practices

Default to Single-Directional: Start with "Single" (1→N) and only switch to "Both" if absolutely needed.

Test Performance: Check report speed with bidirectional filtering enabled.

Use DAX as Backup: For edge cases, override filtering with:

dax

```
CALCULATE(
    [Measure],
    CROSSFILTER(TableA[ID], TableB[ID], BOTH)  -- Enable bidirectionally in DAX
)
```

Example: Safe Bidirectional Setup

Model:

Customers (1) → Sales (N) ← Products (1)

Relationship Settings:

Customers → Sales: Single (Customers filter Sales).

Products → Sales: Single (Products filter Sales).

Only enable "Both" if you need Customers to filter Products via Sales.

Key Takeaway

Bidirectional filtering is a tool, not a default. Use it sparingly for:

Small, related dimensions.

Bridge tables in N:N scenarios.

Star schemas where dimensions must interact.

For all other cases, stick to single-directional relationships and handle filtering in DAX.

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

Power BI doesn't natively enforce referential integrity like relational databases, but you can implement safeguards using DAX measures and calculated columns. Here's how to handle cases when a CustomerID might be deleted:

1. Basic Validation Measure

dax

```
Valid Customer Check =
IF(
  ISFILTERED(Customers[CustomerID]) && NOT(ISEMPTY(Sales)),
  IF(
    COUNTROWS(
      INTERSECT(
        VALUES(Customers[CustomerID]),
        VALUES(Sales[CustomerID])
      )
    ) = COUNTROWS(VALUES(Customers[CustomerID])),
    "All customers valid",
    "Warning: Orphaned sales records exist"
  ),
```

```
    BLANK()
)
```

2. Orphaned Records Detection

dax

```
Orphaned Sales Count =
COUNTROWS(
    FILTER(
        Sales,
        NOT(CONTAINS(
            Customers,
            Customers[CustomerID], Sales[CustomerID]
        ))
    )
)
```

3. Protected Sales Calculation

dax

```
Protected Sales Amount =
SUMX(
    FILTER(
        Sales,
        CONTAINS(
            Customers,
            Customers[CustomerID], Sales[CustomerID]
        )
    ),
    Sales[Amount]
)
```

4. Alternative: Safe Customer Filter

dax

```
Safe Customers =
```

```
FILTER(
   Customers,
   COUNTROWS(
      FILTER(
         Sales,
         Sales[CustomerID] = Customers[CustomerID]
      )
   ) > 0
)
```

Implementation Notes:

For Reports:

Add the validation measure to report visuals

Use conditional formatting to highlight issues

For Data Loading:

These checks should ideally happen in Power Query during ETL

Consider using SQL source queries with proper joins

Limitations:

DAX can only detect issues, not prevent them

True referential integrity requires database-level constraints

Best Practice:

Create a separate "Invalid Records" table to capture orphans

Schedule regular data quality checks

Remember that Power BI is primarily a reporting tool - for true referential integrity enforcement, you should implement these rules in your source database system.