1. 🔑 **What is a Primary Key in Power BI?**
   - In Power BI, a **primary key** means a column (or set of columns) that uniquely identifies each row in a table.
   - Unlike SQL databases, Power BI does not **enforce** primary key constraints.
   - However, primary keys are essential when you build **relationships** between tables.

📊 **Example:**

- **Customers table**
  - CustomerID → Primary Key (each customer has a unique ID).
- **Orders table**
  - CustID → Foreign Key (indicates which customer placed the order).

In Power BI's **Model view**:
👉 You create a relationship between Customers[CustomerID] (Primary Key) and Orders[CustID] (Foreign Key).

⚖️ **Key Points:**

1. Power BI doesn't require you to define a primary key like SQL.
2. But **choosing a unique identifier** for each table is critical.
3. Relationships in Power BI usually follow **one-to-many** (1 customer → many orders) or sometimes **many-to-many**.

2. In **Power BI**, there are two main types of table relationships:

1. **One-to-Many (1:*):**
   - The most common relationship.
   - Example: One *Customer* can have many *Orders*.
   - Usually, the **primary key** is on the "one" side, and the **foreign key** is on the "many" side.
2. **Many-to-Many (:):**
   - Both tables can have multiple matching rows.
   - Example: A *Student* can enroll in many *Courses*, and each *Course* can have many *Students*.

- In Power BI, this is handled using a **bridge table** or by enabling a direct many-to-many relationship.

✨ There is also **One-to-One (1:1)**, but it's less common and usually just means two tables share the same unique key (can often be combined into one table).

3. 🔹 **Method 1: Using the Model View (most common)**
   1. Go to **Model View** (left-hand panel → the third icon with relationships).
   2. Drag a field (usually a **primary key**) from one table onto the corresponding field (usually a **foreign key**) in the other table.
   3. The **Manage Relationships** dialog will open automatically.
   4. Choose the **Cardinality** (One-to-Many, Many-to-Many, or One-to-One).
   5. Choose **Cross filter direction** (Single or Both).
   6. Click **OK**.

🔹 **Method 2: Using the Manage Relationships dialog**

1. On the **Home** tab → click **Manage Relationships**.
2. Click **New**.
3. Select the two tables and the matching columns.
4. Configure **Cardinality** and **Cross filter direction**.
5. Click **OK** to create the relationship.

✅ Example:

- **Customers[CustomerID]** (Primary Key) → **Orders[CustomerID]** (Foreign Key).
- Relationship type: **One-to-Many (1:*).**

4. A star schema is a type of database schema (data model) commonly used in Power BI, data warehousing, and analytics. It organizes data in a way that makes reporting and querying faster and easier to understand.

🔹 Structure of a Star Schema

- At the center, there is a Fact Table:
  - Stores quantitative data (numbers you want to measure, e.g., Sales Amount, Quantity, Revenue).
  - Contains foreign keys that link to dimension tables.
- Around it, there are multiple Dimension Tables (the "points of the star"):
  - Contain descriptive attributes (e.g., Customer Name, Product Category, Order Date).
  - Provide context for the facts.

🔹 Example (Sales Data Model in Power BI)

- Fact Table: Sales
  - Columns: OrderID, CustomerID, ProductID, DateKey, SalesAmount, Quantity.
- Dimension Tables:
  - Customers → CustomerID, Name, Country
  - Products → ProductID, Name, Category
  - Dates → DateKey, Year, Month, Day
  - Stores → StoreID, City, Region

👉 Visually, the fact table is in the middle, and dimensions branch out like a star.

🔹 Benefits of Star Schema in Power BI

- Simple and intuitive model for users.
- Efficient performance (optimized for queries).
- Works well with Power BI's DAX and relationships.
- Reduces data redundancy compared to flat tables.

5. In a **sales dataset**, the **Fact Table** is typically the **Sales (or Orders) table**.

✅ **Why?**

- It stores the **measurable, numeric values** you want to analyze.
- Examples of columns in the **Fact Table (Sales)**:
    - OrderID
    - CustomerID (foreign key → Customers table)
    - ProductID (foreign key → Products table)
    - DateKey (foreign key → Date table)
    - Quantity
    - SalesAmount
    - Discount
    - Revenue

✅ **Dimension tables in the same model:**

- **Customers** (CustomerID, Name, Country)
- **Products** (ProductID, Category, Brand)
- **Dates** (DateKey, Year, Month, Day)
- **Stores** (StoreID, Location, Region)

👉 So, in short: **The Sales (or Orders) table is the Fact Table in a sales dataset.**

6. In a **Sales.csv** table (fact table), ProductID is considered a **foreign key** because:

🔑 **Reason:**

- A **foreign key** is a column that links to a **primary key** in another table (usually a **dimension table**).
- In **Sales.csv**, ProductID does not describe the product itself (name, category, price).
- Instead, it just stores an **ID** that points to the **Products table**, where the full product information lives.

📊 **Example:**

Products.csv (Dimension table)

ProductID	ProductName	Category	Price
101	Laptop	Electronics	800
102	Mouse	Accessories	20

Sales.csv (Fact table)

SaleID	CustomerID	ProductID	Quantity	SalesAmount
1	201	101	2	1600
2	202	102	3	60

Here:

- ProductID in **Sales.csv** → **Foreign Key**
- ProductID in **Products.csv** → **Primary Key**

This allows **Sales** to link with **Products**, so you can answer questions like:
👉 "How many Laptops did we sell last month?"

⚡ In **Power BI**, this becomes a **One-to-Many relationship**:

- **Products[ProductID]** (One side, primary key)
- **Sales[ProductID]** (Many side, foreign key)

7. 🔎 **Why the error happens:**
   - In **Sales.csv**, ProductID might be **Whole Number (Int64)**.
   - In **Products.csv**, ProductID might be loaded as **Text** (e.g., "101", "102").
   - Power BI only allows relationships if **both columns are the same data type**.

✅ **How to Fix It**

Option 1: In Power Query

1. Go to **Power Query Editor** (Transform Data).
2. Open **Sales** table → Right-click on ProductID column → **Change Type** → **Whole Number**.
3. Open **Products** table → Do the same, set ProductID to **Whole Number**.
4. Click **Close & Apply**.

5. Recreate the relationship:
    - **Products[ProductID] (Primary Key) → Sales[ProductID] (Foreign Key)**.

Option 2: In Model View

- You can't directly change types here, but you'll see a warning (data type mismatch).
- Always fix it in **Power Query**.

8. A **star schema** is the recommended modeling approach in **Power BI** (and most analytics tools) because it makes queries simpler and **improves performance**. Here's why:

🔹 **Why Star Schema Improves Performance**

1. Clear Separation of Fact and Dimension Tables

- **Fact table** → numeric, transactional data (Sales, Orders, Revenue).
- **Dimension tables** → descriptive attributes (Products, Customers, Dates).
- This separation reduces duplication and keeps the fact table lean.

👉 Example: Instead of storing ProductName, Category, Price in every Sales row, you just store ProductID.

---

2. Smaller Fact Tables = Faster Queries

- Fact tables contain **millions of rows**, so keeping them only with **numeric values and foreign keys** makes them much smaller.
- Dimension tables are smaller and only store descriptive data once.
- Power BI's **VertiPaq engine** compresses these better, making aggregations very fast.

---

3. Efficient Relationships

- In a star schema, relationships are **simple One-to-Many** (Dimension → Fact).
- Power BI handles *1: joins** very efficiently.

- If you use a flat table (all data in one), or snowflake schema (too many joins), queries become slower.

---

4. Optimized for DAX Calculations

- Measures like SUM(Sales[Amount]), AVERAGE(Sales[Quantity]), or CALCULATE(..., Products[Category]) run faster,
  because Power BI only scans the **fact table** and uses dimension filters.

---

5. Better Compression (Storage Engine Benefit)

- Repeating text values (like "Laptop", "Mouse", "Laptop") in a flat table takes more memory.
- In a star schema, text lives once in the **dimension table**, and the fact table only stores small integer keys.
- This makes the **in-memory model much smaller and faster**.

📊 **Simple Example:**

Flat table (slow):
| OrderID | CustomerName | ProductName | Category | Amount |

Star schema (fast):

- **Sales (Fact)** → OrderID, CustomerID, ProductID, Amount
- **Customers (Dimension)** → CustomerID, Name, Country
- **Products (Dimension)** → ProductID, Name, Category

✅ In short: **Star schema improves performance because it reduces data duplication, keeps fact tables small, enables efficient compression, and makes relationships simpler (1:*).**

11. A **circular relationship** (relationship loop) happens when your tables are connected in such a way that Power BI cannot decide how to filter them (it creates ambiguity).

🔎 **Example of Circular Relationship**

Suppose you have:

- **Sales → Customers → Regions**
- **Sales → Products → Regions**

Now Regions is linked twice, and Power BI forms a **loop**.

✅ **How to Resolve Circular Relationships**

1. Remove redundant relationships

- Keep only **one clear path** between fact and dimensions.
- If two paths exist, decide which one makes more business sense.
- For example, link **Customers → Regions**, but not **Products → Regions**.

---

2. Use a Bridge Table

- Create an intermediate table to break the loop.
- Example: instead of linking Regions directly to both Customers and Products,
  create a Region Bridge with only unique RegionID, and link Customers and Products to it.

---

3. Deactivate extra relationships

- In Model view, set some relationships to **inactive**.
- Then activate them only when needed using **DAX USERELATIONSHIP()**.
- Example:
- Sales by Ship Date =
- CALCULATE(
- SUM(Sales[Amount]),
- USERELATIONSHIP(Sales[ShipDate], Dates[Date])
-)

---

4. Normalize dimension tables (Star Schema)

- Always aim for a **star schema** (Fact in center, Dimensions around).
- Avoid snowflake-like chains of dimensions if possible.

---

5. Use Power Query merges instead of relationships

- If unavoidable, flatten some lookups in Power Query (merge Customers with Regions).
- This reduces relationship complexity and eliminates loops.

⚡ **Best Practice**

- In Power BI **never leave a loop in place**.
- **Star Schema** with **One-to-Many** relationships → fastest, cleanest, easiest to maintain.

14. 🔹 **When NOT to use bidirectional filtering**

- In a **clean star schema** (Fact in the middle, Dimensions around it) → you almost never need bidirectional filters.
- Dimensions should filter facts (One → Many), not the other way around.
- If you turn on "Both" everywhere, you risk **circular relationships** and **unexpected totals**.

✅ **When it *is* appropriate**

1. Many-to-Many Relationships

If you truly need to filter **both directions** (e.g., Customers ↔ Sales Territories where multiple customers can belong to multiple territories), sometimes **Both** is required.
But usually better to resolve with a **bridge table**.

---

2. Helper (Bridge) Tables

When you build a **bridge table** between two dimensions and need filters to flow both ways.
Example:

- Sales fact table
- Customers dimension
- Products dimension
- A custom bridge (e.g., MarketingCampaigns linked to both Customers and Products).

Here, bidirectional filtering allows campaigns to filter both Customers and Products.

---

3. Role-Playing Dimensions (sometimes)

If you have multiple Date relationships and want **one slicer to control multiple date roles**, you may use bidirectional filters.
(Although often better solved with USERELATIONSHIP or duplicate Date tables).

---

4. Specialized Analysis Needs

- You want to see **which customers bought which products** directly (Customer ↔ Product through Sales).
- Normally you'd query via the Sales fact, but with Both filtering you can sometimes shortcut.
- ⚠️ Should still prefer explicit bridge tables.

🔑 Best Practice

- Use **Single direction (One → Many)** by default.
- Turn on **Both** only for *specific* analytical needs.
- If you enable Both, **document why**, and check for **ambiguity loops** in Model View.

15. This is a smart way to make your Power BI model more robust when referential integrity (foreign key ↔ primary key) is broken — e.g., a CustomerID in **Sales** does not exist anymore in **Customers**.

Since Power BI doesn't enforce referential integrity automatically (unlike SQL), you handle it with **DAX**.

✅ Option 1: Flag invalid Sales rows

Create a calculated column in **Sales**:

```
CustomerValid =
IF (
    ISBLANK ( RELATED ( Customers[CustomerID] ) ),
    "Orphan",    -- Customer does not exist
```

"Valid"

)

- RELATED() → tries to pull CustomerID from **Customers**.
- If it's blank → that means the Customer was deleted (no match).

Now you can filter out "Orphan" rows in reports.

✅ Option 2: Default replacement for missing Customers

Sometimes you want to show "Unknown Customer" instead of blanks.
Create a new column in **Sales**:

```
CustomerNameSafe =
IF (
    ISBLANK ( RELATED ( Customers[CustomerName] ) ),
    "Unknown Customer",
    RELATED ( Customers[CustomerName] )
)
```

This way, reports won't break when a Customer is deleted.

✅ Option 3: Measure-level enforcement

If you only care about measures, you can wrap aggregations with TREATAS or COALESCE:

```
Total Sales (Valid Customers) =
CALCULATE (
    SUM ( Sales[SalesAmount] ),
    TREATAS ( VALUES ( Customers[CustomerID] ), Sales[CustomerID] )
)
```

This ensures only Sales linked to real Customers are included.

⚡ Best Practice

- **If possible**: fix it at the data source (SQL constraints or Power Query cleaning).
- **In DAX**: use ISBLANK(RELATED()) to catch broken foreign keys.

- Show "Unknown Customer" instead of dropping rows → more user-friendly.

👉 Do you want me to also show you how to **prevent these orphan rows in Power Query** (before loading the data), or should we stick to DAX handling only?