DELOITTE

POWER BI DEVELOPER INTERVIEW QUESTIONS

CTC: 13-17lpa

YOE : 2-5

---

1. Your manager asks for a dashboard to compare actual sales vs. target sales across regions. How would you design it?

Dashboard for Actual Sales vs Target Sales across Regions

Approach:

- **Understand the Requirement**:
  The manager wants to compare *actual vs. target sales* across *regions* → this is a **variance analysis dashboard**.

Steps:

1. **Data Modeling**
    - Sales Fact Table → Sales[ActualSales], Date, RegionKey, ProductKey
    - Target Table → Targets[TargetSales], Date, RegionKey
    - Region Dimension → Region[RegionName]
    - Relationships:

- Sales[RegionKey] → Region[RegionKey]
- Targets[RegionKey] → Region[RegionKey]

2. **DAX Measures**
3. Actual Sales = SUM(Sales[ActualSales])
4. Target Sales = SUM(Targets[TargetSales])
5. Sales Variance = [Actual Sales] - [Target Sales]
6. Variance % = DIVIDE([Actual Sales] - [Target Sales], [Target Sales], 0)
7. **Visual Design**
    - **KPI Cards** → Overall Actual, Target, Variance %
    - **Bar/Column Chart** → Actual vs. Target by Region
    - **Variance Heatmap** → Conditional formatting to highlight regions underperforming
    - **Trend Line** → Actual vs. Target over time (Monthly/Quarterly)
    - **Filters/Slicers** → Region, Date, Product Category
8. **Best Practices**
    - Use **color coding** (Green = above target, Red = below target)
    - Show **variance %** clearly → helps manager focus on gaps
    - Optimize with **star schema** (Fact-Sales, Fact-Target, Dim-Region, Dim-Date)

👉 **Final Answer**:
I would create a **star schema model**, define **measures for actual, target, and variance**, and design a dashboard with **cards, bar charts, and line charts** to show sales performance across regions. I'll use conditional formatting to highlight underperforming regions and allow drilldowns by date/product.

---

2. You are given a dataset with daily sales data for 3 years. The report takes too long to load, what will you do?

Report Takes Too Long to Load (3 Years Daily Sales Data)

Possible Causes:

- Large dataset (daily level → millions of rows)
- Inefficient data model (snowflake, many-to-many, calculated columns)
- Complex DAX measures (iterators, nested CALCULATEs)
- No aggregations or pre-calculations

Solutions:

1. **Data Model Optimization**
    - Use a **star schema** → Fact (Sales) + Dimensions (Date, Product, Region, Customer)
    - Remove unnecessary columns
    - Avoid bi-directional relationships unless necessary
2. **Performance Optimization Techniques**
    - **Aggregations**: Create summary tables (Monthly/Quarterly sales) for faster visuals.
    - **Incremental Refresh**: Only refresh the latest data (e.g., last 3 months) instead of entire 3 years.
    - **Pre-aggregation in SQL/ETL**: Summarize at source level before loading into Power BI.
    - **Disable Auto Date/Time** in Power BI (creates hidden tables).
3. **DAX Optimization**
    - Replace **Calculated Columns** with **Measures** (calculated at query time).
    - Avoid FILTER() inside CALCULATE when possible → use relationships.
    - Use **SUMMARIZECOLUMNS** for aggregation.
4. **Report Design**
    - Limit visuals per page (max 6–8)
    - Use slicers/filters to reduce data rendered at once
    - Use **Performance Analyzer** to identify slow visuals

👉 **Final Answer**:
I would optimize the **data model to star schema**, use **aggregations/incremental refresh** for historical data, move heavy calculations to the source/Power Query, and optimize DAX with efficient measures. I would also check **Performance Analyzer** to identify bottleneck visuals.

---

3. You have a many-to-many relationship between Products and Customers—how would you handle it in Power BI?

Problem:

- In reality, a **product can belong to multiple customers** (e.g., shared subscriptions), and a **customer can purchase multiple products** → this creates a **many-to-many relationship**.
- Power BI **doesn't handle M2M directly** → leads to double-counting.

Solutions:

1. **Bridge (Mapping) Table Approach**
    - Create a **bridge table** → ProductCustomerMapping with unique ProductID-CustomerID pairs.
    - Model:
        - Products[ProductID] → ProductCustomerMapping[ProductID]
        - Customers[CustomerID] → ProductCustomerMapping[CustomerID]
    - Sales fact table links to **either Product or Customer** (not both).
2. **DAX Measures for M2M**
    - Example: Unique Customers per Product
    - Unique Customers = DISTINCTCOUNT(ProductCustomerMapping[CustomerID])
    - Example: Total Sales by Customer
    - Sales by Customer =
    - CALCULATE(

- SUM(Sales[SalesAmount]),
- TREATAS(VALUES(ProductCustomerMapping[CustomerID]), Customers[CustomerID])
-)

3. **Use Composite Models (If Needed)**
   - DirectQuery + Import
   - If the dataset is huge, use **Aggregations + Bridge tables**

👉 **Final Answer**:
I would handle many-to-many by introducing a **bridge table** that contains unique Product-Customer mappings, and adjust relationships accordingly. This avoids double-counting and allows correct aggregations using DAX with DISTINCTCOUNT or TREATAS.

---

4. Suppose the client requests a dynamic measure (e.g., the user selects whether they want Sales, Profit, or Quantity). How will you implement it?

Dynamic Measure Selection (Sales / Profit / Quantity)

Scenario:

The client doesn't want 3 separate visuals but a **single dynamic chart** where the metric changes based on user selection.

Solution:

1. **Create a Supporting Table (Disconnected Table)**
   - In Power BI, create a new table (manual entry):
   - MetricSelector = DATATABLE(
   - "Metric", STRING,
   - {{"Sales"}, {"Profit"}, {"Quantity"}}
   -)

- This table has no relationships (disconnected).

2. **Add a Slicer**
   - Place MetricSelector[Metric] as a slicer → user picks the metric.
3. **Write a Dynamic Measure**
4. Selected Metric =
5. SWITCH(
6. SELECTEDVALUE(MetricSelector[Metric]),
7. "Sales", SUM(Sales[SalesAmount]),
8. "Profit", SUM(Sales[Profit]),
9. "Quantity", SUM(Sales[Quantity]),
10. BLANK()
11.)
12. **Use the Measure in Visuals**
    - Instead of hardcoding Sales/Profit/Quantity, place [Selected Metric] in visuals.

👉 **Final Answer (Interview Style):**
I would create a **disconnected parameter table** with values "Sales, Profit, Quantity," use it in a slicer, and then write a **SWITCH-based DAX measure** that dynamically picks the right aggregation. This way, the user controls the metric displayed in the visuals without duplicating charts.

---

5. How would you create a Top N customers by revenue report where N is selected dynamically by the user?

Dynamic Top N Customers by Revenue

Scenario:

The client wants to choose a **Top 5 / Top 10 / Top 20 customers** dynamically.

Solution:

1. **Create a Parameter Table for N**
2. TopNSelector = DATATABLE(
3. "N", INTEGER,
4. {{5}, {10}, {20}, {50}}
5.)
    - Add it as a slicer → user picks how many customers to display.
6. **Write a Rank Measure**
7. Customer Rank =
8. RANKX(
9. ALL(Customer[CustomerName]),
10. CALCULATE(SUM(Sales[SalesAmount])),
11. ,
12. DESC
13.)
14. **Write Top N Sales Measure**
15. TopN Sales =
16. VAR SelectedN = SELECTEDVALUE(TopNSelector[N], 10)
17. RETURN
18. IF([Customer Rank] <= SelectedN, SUM(Sales[SalesAmount]))
19. **Apply in Visuals**
    - Place Customer on rows, [TopN Sales] as value.
    - The slicer dynamically controls how many customers appear.

👉 **Final Answer (Interview Style):**
I would create a **parameter table for N**, use it as a slicer, and then write a **RANKX measure** that calculates customer ranking by revenue. Using an IF condition, only customers up to the selected N are displayed. This gives the client full control over how many Top customers they want to see in the report.

6. Explain a case where you chose a specific visualization to solve a business problem. Why did you select it, and what impact did it have?

Choosing the Right Visualization (Business Impact)

Scenario Example:

Let's say I was asked to analyze **churn rate trends for a telecom client**.

- The business team initially used **bar charts** to show churn by month.
- But that only gave them discrete values per month, not **trend insight**.

My Solution:

1. **Understanding the Business Question**:
    - The client wanted to see **how churn evolved over time** and quickly identify peaks/drops.
2. **Choosing Visualization**:
    - Instead of a bar chart, I used a **Line Chart** with **rolling average (3 months)** overlay.
    - Created a DAX measure:
    - Churn Rate = DIVIDE(SUM(Churn[ChurnedCustomers]), SUM(Churn[TotalCustomers]))
    - Rolling Avg (3M) = AVERAGEX(DATESINPERIOD(Date[Date], MAX(Date[Date]), -3, MONTH), [Churn Rate])
3. **Impact**:
    - The line chart clearly showed **seasonal churn spikes** (end of Q4 due to expiring contracts).
    - The rolling average smoothed short-term fluctuations → making it easy for management to plan retention campaigns in advance.
    - The client reported they could reduce **customer churn by 8%** the following quarter because they knew exactly when to launch promotions.

👉 **Final Answer (Interview Style):**
I had a case where the business wanted to analyze **churn trends**. Instead of a bar chart, I used a **line chart with a rolling average** because it better highlighted trends and seasonality. This allowed the business to proactively plan retention offers during high-risk periods, leading to measurable improvement in customer retention.

---

7. How do you optimize a Power BI report for performance? Discuss steps to streamline data, aggregate efficiently, and optimize DAX calculations.

How do you optimize a Power BI report for performance?

When asked this, interviewers expect you to cover **three layers of optimization**:

- **Data layer (modeling & storage)**
- **DAX layer (calculations)**
- **Report layer (visuals & design)**

(A) Streamline Data Modeling

1. **Use Star Schema**
    - Keep 1 fact table (e.g., Sales) and multiple dimension tables (Date, Product, Customer).
    - Avoid snowflake schemas and flatten complex hierarchies in ETL/Power Query.
2. **Remove Unnecessary Columns**
    - Drop unused columns and reduce column cardinality (e.g., split DateTime into Date + Time).
3. **Optimize Relationships**
    - Prefer **single-direction filters** instead of bi-directional relationships.
    - Avoid many-to-many unless business-critical (use bridge tables).
4. **Data Reduction Techniques**

- Apply filters at source (SQL/ETL) instead of bringing raw data.
- Use **incremental refresh** for large fact tables (refresh only latest partitions).
- Disable **Auto Date/Time hierarchy** → it creates hidden tables that slow performance.

(B) Aggregate Efficiently

1. **Aggregation Tables**
   - Pre-calculate monthly/quarterly summaries in SQL or Power Query.
   - Example: Instead of querying 3 years of daily data → use monthly aggregation for trend visuals.
2. **Pre-Compute Measures at Source**
   - If possible, calculate KPIs in the warehouse (SQL/ETL) instead of DAX.
3. **Indexing & Query Optimization**
   - On DirectQuery sources, ensure indexes are present in SQL tables.

(C) Optimize DAX Calculations

1. **Avoid Calculated Columns**
   - Use Measures instead (calculated at query time, not stored).
2. **Efficient DAX Practices**
   - Prefer **SUMX over FILTER + SUM** when possible.
   - Use CALCULATE with relationships instead of iterating over FILTER.
   - Use variables (VAR) in complex measures to avoid repeated evaluation.
   - Replace IF with SWITCH for better readability and performance.
3. **Benchmark DAX**
   - Use **DAX Studio** to analyze query plans, check storage engine (SE) vs. formula engine (FE) usage.

(D) Report Design Optimization

1. Reduce number of visuals per page (6–8 max).
2. Avoid high-cardinality slicers (e.g., customer names for millions of rows). Use search-enabled slicers instead.
3. Turn off unnecessary interactions between visuals.
4. Use **Performance Analyzer** to identify slow visuals.

👉 **Final Answer (Interview Style):**
To optimize Power BI reports, I follow a **three-layer approach**:

- At the **data model level**, I use a star schema, remove unnecessary columns, apply incremental refresh, and keep relationships single-direction.
- At the **aggregation level**, I use pre-aggregated tables, push calculations to the source, and use indexing for DirectQuery.
- At the **DAX level**, I avoid calculated columns, optimize iterators, and use DAX Studio for query tuning.
  Finally, I streamline visuals using Performance Analyzer to detect bottlenecks.

---

8. Explain the difference between DirectQuery and Import mode, and scenarios to use each. explain the above two in detail .

Difference between DirectQuery and Import Mode

Import Mode

- **Data Storage**: Data is imported into Power BI's in-memory VertiPaq engine.
- **Performance**: Fast because data is compressed and cached.
- **Limitations**: Dataset size limited by Power BI capacity (1 GB in Pro, higher in Premium).
- **Refresh**: Requires scheduled refresh to update.

- **Best For**: Small to medium datasets where performance is critical and near-real-time data is not required.

DirectQuery Mode

- **Data Storage**: Data is not stored in Power BI → queries run live against the source (SQL Server, Snowflake, etc.).
- **Performance**: Slower because every interaction triggers a query to the database. Performance depends on network + database.
- **Limitations**:
    - Some DAX functions not supported.
    - Limited transformations in Power Query.
    - Query folding is essential.
- **Refresh**: No scheduled refresh needed since data is live.
- **Best For**: Very large datasets (billions of rows) or when real-time analytics is required.

When to Use Each

1. **Use Import When**:
    - Dataset is manageable in size (< 1–2 GB compressed).
    - Business needs high-performance dashboards with fast slicing/filtering.
    - Data can be refreshed periodically (e.g., hourly, daily).
2. **Use DirectQuery When**:
    - Dataset is huge and cannot fit in memory.
    - Real-time or near real-time reporting is required.
    - Security needs row-level restrictions maintained at source.
3. **Hybrid (Composite Model)**

- Some tables in Import (e.g., small dimensions), some in DirectQuery (e.g., large fact tables).
- Example: A sales fact table with billions of rows in DirectQuery + dimensions in Import.

👉 **Final Answer (Interview Style):**
Import mode loads data into Power BI's in-memory engine, making reports very fast but requiring scheduled refresh. DirectQuery leaves data in the source and queries it live, supporting real-time but often slower with limited DAX functionality.
I typically use **Import** for small/medium datasets needing high speed, **DirectQuery** for very large or real-time scenarios, and sometimes a **composite model** to balance both.

---

9. Key Features of Power Query for Data Cleaning & Shaping

Power Query is the **ETL (Extract, Transform, Load)** engine in Power BI, used before the data model.

Key Features & Usage in Data Cleaning/Shaping

1. **Data Connectivity**
   - Connects to 100+ sources (SQL, Excel, APIs, SharePoint, etc.).
   - Example: Pulling sales data from SQL and customer data from Excel.
2. **Data Profiling**
   - Column statistics, column quality, and column distribution help detect **nulls, errors, duplicates, skewness**.
   - Example: Spot missing customer IDs or invalid dates.
3. **Applied Steps (No-Code Transformation)**
   - Every transformation (remove nulls, change type, merge) is recorded step-by-step → reproducible and editable.
   - Example: Splitting a Full Name column into First Name and Last Name.
4. **Data Shaping Functions**
   - Remove/keep rows (filtering, duplicates).

- Pivot/Unpivot columns (reshaping tables).
- Group By (aggregating).
- Merge/Append queries (joins and unions).
- Example: Unpivot monthly sales columns into rows for a proper fact table structure.

5. **Data Type & Formatting**
   - Assigns correct data types (Date, Currency, Text).
   - Example: Converting text-based dates into Date type for proper time intelligence.
6. **Custom & Conditional Columns**
   - Add calculated fields without writing DAX (M language).
   - Example: If Income > 50000 then "High Income" else "Low Income."
7. **Query Folding**
   - Pushes transformations back to the source (SQL/DB) instead of doing them in Power BI → **performance boost**.
   - Example: Filtering large SQL tables at source rather than after load.
8. **Reusable Parameters**
   - Dynamic queries → pass parameters for server name, date range, etc.
   - Example: Refresh only last 3 months of data using a parameter.

👉 **Final Answer (Interview Style):**
Power Query is the ETL layer in Power BI. I use it to **connect, clean, and reshape data** before loading into the model. Its key features include data profiling to detect errors, applied steps for reproducible transformations, pivot/unpivot for restructuring, merge/append for combining tables, and query folding for performance. It allows me to handle nulls, remove duplicates, change data types, and create calculated/conditional columns efficiently before modeling.

---

10. Creating & Using Power BI Templates in Projects

What is a Power BI Template (.pbit)?

- A template contains **report structure, data model, queries, measures, visuals**, but **no actual data**.
- Used to standardize reports across multiple clients/projects.

How to Create a Template

1. Build a Power BI report → add queries, model, visuals, DAX.
2. Go to **File → Export → Power BI template**.
3. Save as .pbit.

How to Use a Template

1. Open .pbit file → it prompts for data source parameters.
    - Example: Server name, database, date range.
2. Connect to client-specific data → same visuals & measures apply.
3. Refresh → dashboard is ready with new client's data.

Benefits of Templates

1. **Consistency**
    - Same KPIs, same formatting, same logic across multiple reports.
2. **Time Savings**
    - No need to rebuild reports from scratch → just change parameters.
3. **Scalability**
    - Useful in consulting (like Deloitte) → one template can be reused across different clients/regions.
4. **Governance & Standards**

- Ensures reports follow corporate branding and predefined KPIs.

👉 **Final Answer (Interview Style):**
Power BI templates (.pbit) let us save the **data model, queries, and visuals without data**. I create templates when I need reusable dashboards across clients or regions. For example, if Deloitte builds a Sales Performance dashboard, we can export it as a template. When used for another client, the user just updates parameters like server or database, and the same KPIs and visuals work instantly. This ensures **consistency, scalability, and time savings** in BI projects.

11.Describe the types of filters in Power BI such as visual-level, page-level, and report-level. How do you apply and use them effectively?

Types of Filters in Power BI (Visual, Page, Report) & Effective Use

Power BI provides multiple levels of filtering:

(A) Visual-Level Filters

- Apply only to **one visual** (chart/table).
- Example: A bar chart showing only Top 10 customers, while other visuals still show all customers.
- **When to use**: To isolate or focus on a subset of data inside one chart without affecting the page/report.

(B) Page-Level Filters

- Apply to **all visuals on a page**, but not across other pages.
- Example: A regional manager wants a dashboard page filtered only to “Region = North” → every visual on that page follows this filter.
- **When to use**: Useful when each page in the report represents a specific business unit, product category, or region.

(C) Report-Level Filters

- Apply to **all visuals in all pages of the report**.
- Example: Filtering the whole report for Year = 2023.
- **When to use**: Global filters, such as showing only the latest year or excluding test/demo data across the entire report.

Other Important Filters

- **Drillthrough Filters** → Right-click a data point → navigate to a detailed page filtered only for that entity (e.g., see all sales details for one customer).
- **Slicers** → Visuals users can interact with to filter data dynamically.
- **Cross-filtering & Cross-highlighting** → Clicking a bar in one chart highlights related data in another.

👉 **Final Answer (Interview Style):**
Power BI has filters at **visual, page, and report levels**. Visual-level filters affect one chart, page-level filters apply to all visuals on a page, and report-level filters apply across the entire report. I use visual filters for focused insights (e.g., Top 10 customers), page filters for business-unit dashboards, and report filters for global constraints like fiscal year. For interactivity, I also use slicers and drillthrough filters.

---

12. How do you schedule automatic data refresh in the Power BI Service?

Scheduling Automatic Data Refresh in Power BI Service

Steps:

1. **Publish Report to Power BI Service**.
2. Go to **Settings → Datasets → Scheduled Refresh**.
3. Configure:

- **Data Source Credentials** → Provide authentication (Windows, SQL, OAuth).
- **Gateway Setup**:
  - If source is **on-premises** (SQL Server, Oracle, Excel file on local), configure **On-Premises Data Gateway**.
  - If source is **cloud-based** (Azure SQL, Snowflake, Salesforce), no gateway needed.
- **Refresh Frequency**:
  - Up to 8 times/day in Pro.
  - Up to 48 times/day in Premium.
- **Failure Alerts**: Enable email notifications if refresh fails.

Best Practices:

- Use **Incremental Refresh** for large fact tables → refresh only recent partitions.
- Avoid unnecessary columns/rows in Power Query → smaller dataset = faster refresh.
- For critical dashboards, use **Refresh Now** option before key business meetings.

👉 **Final Answer (Interview Style):**
To schedule automatic refresh, I publish the report to Power BI Service, configure data source credentials, and if needed, set up an **On-Premises Data Gateway**. Then I schedule refresh frequency (up to 8 times/day in Pro, 48 in Premium). For large datasets, I use **incremental refresh** so only the latest data refreshes. I also enable failure notifications so the team gets alerts if refresh fails.

---

13. How do you use bookmarks in Power BI? Give a scenario of usage like enabling custom navigation or storytelling.

Using Bookmarks in Power BI (Custom Navigation & Storytelling)

What are Bookmarks?

- Bookmarks capture the **current state of a report page** (filters, slicers, visibility of visuals, drilldown level).

- Can be used for **navigation, custom views, storytelling, and toggling visuals**.

How to Create & Use Bookmarks

1. Configure report view (apply filters, hide/show visuals, set slicer values).
2. Go to **View → Bookmarks Pane → Add Bookmark**.
3. Rename the bookmark (e.g., "Sales Overview").
4. Use **Buttons/Images** linked to bookmarks for navigation.
5. Optionally enable **Bookmark Navigator** (auto navigation between multiple bookmarks).

Scenario 1: Custom Navigation

- Instead of multiple report pages, use bookmarks + buttons to simulate an **app-like navigation**.
- Example: Create bookmarks for “Sales View,” “Profit View,” and “Quantity View.” Add navigation buttons → user clicks button to switch views.

Scenario 2: Storytelling / Presentations

- Use bookmarks to highlight insights step by step.
- Example: While presenting to Deloitte’s client, first show “Overall Sales,” then “Regional Breakdown,” then “Top Customers.” Each click moves to the next bookmark, guiding the story.

Scenario 3: Toggle Between Charts

- Use bookmarks to let users switch between chart types (e.g., Bar chart vs Line chart).
- Example: One button toggles between "Actual vs Target" in Bar Chart and Line Chart.

👉 **Final Answer (Interview Style):**
Bookmarks let me capture a report's state (filters, visuals, slicers) and return to it anytime. I use them for **custom navigation** (like app-style dashboards), **storytelling during client presentations** (progressively showing insights), and **interactive toggles** (switching visuals on the same page). For example, I once created a KPI dashboard where users clicked buttons to switch between Sales, Profit, and Quantity views without leaving the page — this was done entirely with bookmarks.

---

14. What are fact tables and dimension tables? Illustrate with examples from actual project dashboards (e.g., sales data—facts and dimensions).

Fact tables vs Dimension tables (with concrete examples)

Definition (short):

- **Fact table** = transactional / numeric measures you want to aggregate (sales, quantity, cost, profit). Each row is an event/transaction.
- **Dimension table** = descriptive attributes used to slice, filter or group facts (date, product, customer, region, store, sales rep).

Why it matters: star-schema (fact + dims) → smaller, faster model, simpler DAX, better compression.

Sales example (typical project):

Fact table SalesFact (one row per order line):

- SalesID (PK), OrderDateKey (FK → Date[DateKey]), ProductKey (FK → Product[ProductKey]), CustomerKey, StoreKey
- Measures: Quantity, UnitPrice, SalesAmount (Quantity * UnitPrice), Discount, Cost, Profit

Dimension Product:

- ProductKey, ProductName, Category, SubCategory, SKU, Brand

Dimension Customer:

- CustomerKey, CustomerName, Region, Segment, JoinDate

Dimension Date:

- DateKey, Date, Year, Quarter, Month, IsHoliday, FiscalMonth

Typical visuals & how fact/dim are used

- Bar chart: *SalesAmount by Region* → SalesFact[SalesAmount] (measure) on Y, Customer[Region] on X (dimension).
- KPI card: *Total Sales* → measure Total Sales = SUM(SalesFact[SalesAmount]).
- Trend line: *Monthly Sales Trend* → Total Sales broken by Date[Month].

Small worked example (DAX measures):

Total Sales = SUM(SalesFact[SalesAmount])

Total Quantity = SUM(SalesFact[Quantity])

Sales Variance % =

DIVIDE([Total Sales] - [Target Sales], [Target Sales], 0)

(Here Target Sales could be a separate Target fact or a column in a target fact table linked to Date & Region.)

Interview-friendly summary:
"Facts store numeric transactions (e.g., SalesAmount, Quantity); dimensions store descriptive attributes (Product, Customer, Date). I design a star schema so measures live in the fact and slicing comes from dimensions — this improves performance, compression and simplifies DAX."

---

15. How do you handle slow PBIX files? Share performance improvement strategies such as reducing query size, optimizing data models, and efficient DAX.

I like to think in layers: **Extract / Transform**, **Model**, **DAX**, **Report UI**. Below are actionable steps with rationale and short examples.

A. Data extraction & Power Query

1. **Filter at source / Query folding**
    - Push filters to the database instead of loading all rows.
    - Use Table.NestedJoin or native SQL queries so heavy work runs on the DB.
2. **Remove unused columns & reduce cardinality**
    - Drop columns that aren't used for visuals, measures, or joins.
    - Convert textual IDs to integer surrogate keys if possible.
3. **Aggregate where possible**
    - Load monthly/weekly aggregates instead of daily transactional detail for summary pages.
4. **Use incremental refresh** (large fact tables)
    - Keep historical partitions static and refresh only newest partitions.
5. **Avoid Auto date/time**
    - Disable Auto Date/Time and use a proper Date dimension.

Power Query example (merge left outer then expand):

```
let
  Sales = Excel.CurrentWorkbook(){[Name="Sales"]}[Content],
  Products = Excel.CurrentWorkbook(){[Name="Products"]}[Content],
  Merged = Table.NestedJoin(Sales, {"ProductID"}, Products, {"ProductID"}, "Product", JoinKind.LeftOuter),
  Expanded = Table.ExpandTableColumn(Merged, "Product", {"ProductName","Category"})
in
  Expanded
```

B. Data model & relationships

1. **Star schema** — keep single large fact and multiple dims.

2. **Avoid many bi-directional filters** — use single-direction unless RLS or specific behavior requires it.
3. **Hide helper tables/columns** — keep model surface minimal.
4. **Use appropriate data types** (ints for keys, decimals for measures).
5. **Use aggregation tables + composite models** if fact is huge (import summaries, DirectQuery detail).

C. DAX optimization

1. **Prefer measures over calculated columns** (calculated columns increase PBIX size).
2. **Use variables (VAR)** to prevent recomputation.
3. **Replace row-by-row iterators with set-based CALCULATE where possible.**

Bad (slow) example:

```
Avg Order Value (slow) =

AVERAGEX(VALUES(SalesFact[OrderID]), [OrderTotal])
```

Better:

```
Avg Order Value =

DIVIDE([Total Sales], DISTINCTCOUNT(SalesFact[OrderID]))
```

4. **Use SUMMARIZECOLUMNS / ADDCOLUMNS carefully** — prefer SUMMARIZECOLUMNS for grouped tables and CALCULATE for context transition.
5. **Avoid FILTER over large tables inside nested iterators** — filter first using CALCULATE with conditions.
6. **Use DAX Studio / VertiPaq Analyzer** to:
    - See which queries hit formula engine vs storage engine.
    - Find slow measures and optimize.

D. Report & visuals

1. **Limit visuals per page** (6–8 recommended).
2. **Avoid visuals that require full table scans** (e.g., many-to-many ranked tables with heavy measures).

3. **Use slicers wisely** — avoid high-cardinality slicers; use search or hierarchical slicers.
4. **Turn off unnecessary interactions** between visuals.
5. **Use drillthrough/detail pages instead of giant tables.**
6. **Use Performance Analyzer** (in Power BI Desktop) to identify slow visuals and which measure caused the delay.

E. Operational & architectural

1. **Use Premium capacity / larger resources** for very large datasets.
2. **Split very large PBIX into shared datasets** — publish dataset separately and build thin report files connecting to it.
3. **Use gateways & tune DB indices** for DirectQuery.

F. Example measure optimization (using VAR)

Slow version (recomputes):

```
SalesYoY =
CALCULATE([Total Sales], FILTER(ALL(Date), Date[Year] = MAX(Date[Year]) - 1))
```

Optimized:

```
SalesYoY =
VAR _maxYear = MAX(Date[Year])
RETURN
CALCULATE([Total Sales], FILTER(ALL(Date), Date[Year] = _maxYear - 1))
```

(Using VAR once prevents repeated evaluation when referenced multiple times.)

---

16 Types of joins in Power BI and DAX (what exists and how to implement)

Power BI supports joins in **two places**: **Power Query (M)** — best for ETL/merge operations, and **model / DAX** — for runtime joins or lookup behaviors.

A. Power Query (Merge) — Join kinds (recommended for heavy joins)

Power Query Merge provides these join kinds (common ones shown with intent):

- **Left Outer** — keep all rows from left, matching from right. (SQL LEFT JOIN)
- **Right Outer** — keep all rows from right, matching from left. (SQL RIGHT JOIN)
- **Full Outer** — keep rows from both sides (SQL FULL OUTER JOIN)
- **Inner** — keep only matching rows (SQL INNER JOIN)
- **Left Anti** — rows only in left (left minus right)
- **Right Anti** — rows only in right (right minus left)

Power Query M example (Left Outer merge & expand):

```
let
  Sales = Excel.CurrentWorkbook(){[Name="Sales"]}[Content],
  Products = Excel.CurrentWorkbook(){[Name="Products"]}[Content],
  Merged = Table.NestedJoin(Sales, {"ProductID"}, Products, {"ProductID"}, "Product", JoinKind.LeftOuter),
  Expanded = Table.ExpandTableColumn(Merged, "Product", {"ProductName","Category"})
in
  Expanded
```

When to use: Use Power Query merges for schema shaping or when you can push join work to the source (query folding). Efficient and means joined columns are present at model load.

B. DAX / Model-level joins (lookup-like joins or table operations)

DAX doesn't have a single SQL-style JOIN operator for visuals, but provides functions to *combine or relate* tables and do runtime joins:

1. RELATED — Lookup column value using model relationship (equivalent to left join lookup in row context)

- Requires a relationship (many-to-one) from current table to the related table.
  Example (calculated column in SalesFact):

ProductName = RELATED(Product[ProductName])

This pulls ProductName into SalesFact for each sales row (like a left join since every sales row can retrieve a product property).

2. LOOKUPVALUE — lookup a scalar value (like SQL lookup)

ProductName = LOOKUPVALUE(Product[ProductName], Product[ProductID], SalesFact[ProductID])

Use when there's no relationship or for quick lookups. For large datasets, RELATED (with proper relationships) is usually better.

3. NATURALINNERJOIN / NATURALLEFTOUTERJOIN — DAX table functions that perform set-style joins

- NATURALINNERJOIN(TableA, TableB) — inner join on identically-named columns.
- NATURALLEFTOUTERJOIN(TableA, TableB) — left outer join (TableA rows preserved, matching TableB columns appended).

Example:

JoinedTable = NATURALLEFTOUTERJOIN(SalesSummary, ProductList)

Caveat: these require matching column names and return full tables — can blow up memory if used carelessly.

4. CROSSJOIN / GENERATE — Cartesian or row expansion joins (use carefully)

- CROSSJOIN(A,B) returns cartesian product.
- GENERATE can be used to emulate correlated joins (for each row in A, evaluate table B).

5. TREATAS — apply table of values as filter to another table (very useful to emulate join behavior in measures)

Example (measure to sum sales for set of products in a disconnected table PromoProducts):

Promo Sales =

CALCULATE(

[Total Sales],

TREATAS(VALUES(PromoProducts[ProductID]), Product[ProductID])

)

6. Using CALCULATETABLE + FILTER to emulate inner join

```
JoinedRows =
FILTER(
  CROSSJOIN(TableA, TableB),
  TableA[Key] = TableB[Key]
)
```

(Careful — CROSSJOIN can be heavy.)

C. Recommendations — which approach to use and when

- **Prefer Power Query merges** for most join work (ETL): efficient, folds to source, reduces model complexity.
- **Use relationships + RELATED** for simple lookups at model level — they're fast and memory efficient.
- **Use DAX table joins (NATURAL*, TREATAS)** for specialized runtime scenarios or dynamic filtering where you cannot change the query/model. But be cautious of memory and query engine impact.
- **Avoid CROSSJOIN-based solutions** on large tables — they explode cardinality.

---

17. Calculated Columns vs. Measures . Discuss their differences and common usage in dashboards.

Calculated Columns

- **Definition**: New columns added to a table using DAX, computed row by row.
- **Storage**: Stored in the model → increases file size.
- **Evaluation Context**: Works in *row context* (each row gets its own value).
- **Use cases**:

    - When you need the result per row (e.g., “Profit = Sales – Cost” per transaction).
    - Used in relationships or filtering (e.g., Year = YEAR(Date[OrderDate])).
- **Example**:
- Profit = SalesFact[SalesAmount] - SalesFact[Cost]

Measures

- **Definition**: Calculations defined on-the-fly using DAX, aggregated at query time.
- **Storage**: Not stored → lightweight and memory-efficient.
- **Evaluation Context**: Works in *filter context* (depends on slicers/visuals).
- **Use cases**:
    - KPIs, aggregations, dynamic calculations.
    - Example: *Total Sales*, *YOY Growth*, *Top N Customers*.
- **Example**:
- Total Sales = SUM(SalesFact[SalesAmount])

Key Differences

Aspect	Calculated Column	Measure
Storage	Stored in model (increases size)	Not stored (calculated on demand)
Context	Row context	Filter context
Use case	Needed per-row, used in relations	KPIs, aggregated values
Performance	Slower with large data	Faster and efficient

👉 **Interview Answer**:
“Calculated columns are row-level expressions physically stored in the model, used for things like derived keys or categories. Measures are dynamic, context-aware calculations like KPIs or ratios. Best practice: use measures wherever possible and columns only when absolutely required.”

---

18. How do you create and configure row-level security in Power BI? Provide a real-life use case.

Row-Level Security (RLS)

What it is

RLS restricts access to data in Power BI based on roles, so different users see only the data relevant to them.

How to create/configure RLS

1. In Power BI Desktop → **Modeling → Manage Roles**.
2. Create a role with a DAX filter on a table.
   Example: Only allow users to see data from their region.
3. [Region] = "East"
4. Publish the report to Power BI Service.
5. In Service, assign users/groups to roles (via **Security** in dataset settings).
6. Users see filtered data automatically when they open the report.

Dynamic RLS (real-world use case)

Instead of hardcoding values, create a mapping table:

- UserRegionMap: UserEmail → Region.
- Role filter:
- [Region] = LOOKUPVALUE(UserRegionMap[Region], UserRegionMap[UserEmail], USERPRINCIPALNAME())

Now each user sees their own region's data based on login email.

Real-life project example

- **Scenario**: In a retail dashboard, each regional manager should only see sales of their own region, but CEO sees all.
- **Implementation**: Dynamic RLS with a mapping table linking manager email IDs to their region. CEO either excluded from RLS or added to a separate "All Access" role.

👉 **Interview Answer**:
"RLS enforces data security by restricting what each user can see. I usually implement

dynamic RLS using a user-to-attribute mapping table with USERPRINCIPALNAME(). For example, in a sales dashboard, regional managers only see their region's data, while executives see all."

---

19. Compare Power BI dashboards vs. reports. Highlight the differences, best uses, and typical interview scenarios.

Power BI Dashboards vs. Reports

Power BI Report

- Multi-page, interactive, detailed.
- Built in Power BI Desktop, published to Service.
- Allows slicers, drillthrough, bookmarks, multiple visuals per page.
- Can connect to multiple datasets.
- Example: A **Sales Performance Report** with tabs for Sales by Region, Product, Customer, YOY Trends.

Power BI Dashboard

- Single-page (canvas), created in Power BI Service by *pinning visuals* from reports.
- Can include visuals from multiple reports/datasets.
- Used for high-level KPIs and monitoring.
- Less interactive than reports (click takes you back to underlying report).
- Example: **Executive Dashboard** showing KPIs → Total Sales, Profit Margin, Top 5 Customers, YOY Growth.

Key Differences

Feature	Report	Dashboard
Pages	Multi-page	Single-page canvas
Creation	Desktop (then published)	Service (pin visuals from reports)
Data sources	One dataset per report	Multiple datasets possible

Feature	Report	Dashboard
Interactivity	High (slicers, drillthrough, filters)	Limited (click → report)
Audience	Analysts, Managers (detailed)	Executives (high-level summary)

👉 **Interview Answer**:
"Reports are detailed, multi-page analytical tools created in Desktop, while dashboards are one-page high-level KPI views created in Service. For example, a Sales Report may have tabs for product/region trends, while the CEO sees a Dashboard summarizing Total Sales, Profit, and Top Customers."

---

20. How do you use custom visuals and implement drillthrough features for detailed analysis?

Custom visuals & Drillthrough

Custom visuals

- Power BI provides standard visuals (bar, line, pie, etc.), but sometimes business needs advanced visuals.
- **Custom visuals** are imported from **AppSource** or built internally.
- Examples:
  - **Bullet Chart** (vs. targets & benchmarks).
  - **KPI Cards with states** (good for finance/operations dashboards).
  - **Hierarchy Slicer** (multi-level filtering).
  - **Chiclet Slicer** (image-based slicer, good for product dashboards).
  - **MapBox / Synoptic Panel** (for geo or custom shape-based visuals).

How to use:

1. Go to Visualizations → Get more visuals.
2. Import from AppSource or .pbiviz file.
3. Use it like a native visual.

👉 **Interview use case:**
"In a sales dashboard, I used a **Hierarchy Slicer** to let users filter by Product Category → Subcategory → SKU, improving navigation compared to a flat slicer."

Drillthrough

- Lets users right-click on a visual → go to a **detailed page** filtered by that context.
- Setup:
    1. Create a new page in the report.
    2. Add fields to the **Drillthrough filter** section (e.g., CustomerID).
    3. Design visuals on that page for detail analysis.
    4. Users right-click → "Drillthrough → Customer Details" (example).

👉 **Scenario:**

- From a high-level sales report, the user right-clicks on a customer → goes to a *Customer Drillthrough page* with all orders, YOY trend, churn probability.
- Helps maintain a clean summary view while still enabling detail on demand.

Interview Answer (1–2 liner):
"I use custom visuals like Hierarchy Slicers or Bullet Charts when standard visuals don't meet business needs. Drillthrough allows context-specific deep analysis — e.g., from a Top Customers visual, users drillthrough into a Customer Details page showing orders, retention, and profitability."

---

21. What is cardinality, and how do you handle relationships in your Power BI data model?

Cardinality & Handling Relationships

Cardinality in Power BI

- **Definition**: Refers to the uniqueness of data values in a column, affecting relationships between tables.
- Relationship types:

- **One-to-One (1:1)** — rare, e.g., one region has one manager.
- *One-to-Many (1: or :1)* — most common, e.g., Product → Sales (one product, many sales).
- **Many-to-Many (:)** — tricky, e.g., Customers can buy multiple Products, Products can be bought by multiple Customers.

Handling relationships

1. **Star schema modeling** — keep fact table at the center, dimensions around it.
2. **One-to-Many preferred** — dimension (lookup) on one side, fact (transactions) on many side.
3. **For Many-to-Many:**
   - Introduce **bridge table** to break it into 1:* relationships.
   - Use **composite models** or **DAX (TREATAS)** when needed.
   - Example: If Products and Customers both have many-to-many through Purchases, create a **SalesFact** table as the bridge.
4. **Filter direction:**
   - Keep **single direction** for performance.
   - Use **bi-directional** only when business logic demands (e.g., dynamic filtering across dimensions).

👉 **Example:**

- Fact: SalesFact (OrderID, ProductID, CustomerID, SalesAmount)
- Dim: Product, Customer, Date.
- Relationship: Product[ProductID] → SalesFact[ProductID] (1:Many).
- Cardinality is **1:* (one product → many sales)**.

Interview Answer:
"Cardinality defines the uniqueness of values in a relationship. I design star schemas with one-to-many relationships (dimension to fact). For many-to-many, I resolve using a bridge fact table or DAX filters like TREATAS. This ensures a clean, optimized model."

---

22. Time Intelligence Functions (with examples)

What they are

- Pre-built DAX functions to handle common date-based analysis: YOY, MTD, QTD, rolling averages, etc.
- Require a proper **Date table** marked as "Date Table."

Key Functions & Examples

1. DATESBETWEEN

- Returns a table of dates between a start and end date.

```
Sales Last 30 Days =
CALCULATE(
  [Total Sales],
  DATESBETWEEN(
    'Date'[Date],
    TODAY()-30,
    TODAY()
  )
)
```

Use case: Last X days, custom ranges.

---

2. DATESINPERIOD

- Returns a set of dates for a period relative to a given date.

```
Sales Last 3 Months =
CALCULATE(
  [Total Sales],
  DATESINPERIOD(
```

```
    'Date'[Date],
    MAX('Date'[Date]),
    -3,
    MONTH
  )
)
```

Use case: Rolling windows (last N months, weeks, years).

---

3. DATEADD

- Shifts dates by a given interval.

```
Sales LY =
CALCULATE(
  [Total Sales],
  DATEADD('Date'[Date], -1, YEAR)
)
```

Use case: YOY comparisons.

---

4. SAMEPERIODLASTYEAR

- Returns the same period in the previous year.

```
Sales YoY Growth =
DIVIDE([Total Sales] - CALCULATE([Total Sales], SAMEPERIODLASTYEAR('Date'[Date])),
    CALCULATE([Total Sales], SAMEPERIODLASTYEAR('Date'[Date])))
```

Key difference: DATESBETWEEN vs. DATESINPERIOD

- DATESBETWEEN: Explicit start and end dates (fixed/custom window).
- DATESINPERIOD: Relative period from a reference date (rolling window).

👉 **Interview-friendly summary:**
“I use time intelligence functions for MTD, QTD, YOY, and rolling windows. For example, DATESBETWEEN is best for custom ranges like last 30 days, while DATESINPERIOD is better for rolling periods like last 3 months from the current context.”
