COGNIZANT Data Analyst Interview Questions 0-3 YOE CTC : 8-10 lpa

---

Q1:Howwouldyou optimize acomplex PowerBI reportwith multiple data sources and large datasets to improve performance?

Optimizing PowerBI reports iscrucial to ensure they load fastandofferasmoothuser experience, especially when working with large datasets or multiple sources.

Best Practices to Optimize Power BI Reports:

1. Use Import Mode Over DirectQuery (if possible):
    - o Import mode loads data into Power BI memory and performs faster.
    - o DirectQuery should be used only when real-time data is needed, as it can slow performance.
2. Reduce the Data Volume:
    - o Filter unnecessary columns and rows during data load.
    - o Use Power Query to remove nulls, irrelevant columns, and apply filters early.
    - o Aggregate data before loading (e.g., pre-summarized totals).
3. Use Star Schema Modeling:
    - o Create a clear distinction between Fact tables (numerical, transactional) and Dimension tables (categorical).

swipe

- Avoid snowflake or flat file structures as they are less efficient in Power BI.

4. Optimize DAX Calculations:
   - Avoid using CALCULATE unnecessarily.
   - Replace FILTER(ALL(...)) patterns with REMOVEFILTERS() or ALLSELECTED() where appropriate.
   - Use variables (VAR) to store intermediate values.
5. Turn Off Auto Date/Time in Model Settings:
   - This can reduce unnecessary hidden tables and improve memory usage.
6. Use Aggregations:
   - Create summary tables for large datasets and let Power BI use them before querying the detailed table.
7. Reduce the Number of Visuals and Slicers:
   - Each visual generates queries in the background; reduce to only what's needed.
8. Use Performance Analyzer Tool:
   - Helps identify slow visuals and DAX queries, allowing you to fine-tune performance.
9. Disable Unused Interactions Between Visuals:
   - Prevent unnecessary background filtering when it's not needed.
10. Data Source-Level Optimization:
    - Create views or stored procedures in SQL Server instead of using complex logic in Power BI.

Q2: Explain the DAX measures you would use to create a rolling 12-month average in Power BI, considering potential fiscal year differences.

A Rolling 12-Month Average (also known as trailing 12 months or TTM) helps identify trends by averaging values from the last 12 months.

To implement this in Power BI using DAX, you'll typically need:

Assumptions:

- You have a Date table marked as a Date Table.
- There is a Measure (e.g., Total Sales).
- Fiscal year starts in April (adjustable as needed).

Step-by-Step DAX Measure for Rolling 12-Month Average:

```
Rolling 12M Sales =
CALCULATE(
  [Total Sales],
  DATESINPERIOD(
    'Date'[Date],
    MAX('Date'[Date]),
    -12,
    MONTH
  )
)
```

What it does:

- MAX('Date'[Date]): Takes the latest date in context (e.g., in visuals).
- DATESINPERIOD: Generates a 12-month window ending at that date.
- CALCULATE: Recalculates Total Sales over this filtered 12-month window.

To Account for Fiscal Year Differences:

If your fiscal year starts in April (not January), create a Fiscal Year Offset column in your Date Table:

```
FiscalMonthNumber =
```

MOD(MONTH('Date'[Date]) - 4 + 12, 12) + 1

Then, use this logic to build a proper Fiscal Year column and apply a rolling 12-month logic based on fiscal calendar, not just calendar months.

Or alternatively, create a custom fiscal period table and link your model accordingly.

Optional: Rolling 12-Month Average:

Rolling 12M Average Sales =

DIVIDE(

[Rolling 12M Sales],

12

)

This gives a per-month average across the trailing 12 months.

Q3: Describe your approach to implementing row-level security (RLS) in Power BI, both at the dataset and report levels?

Row-Level Security (RLS) allows you to restrict data access for certain users at the row level in Power BI datasets. Users will only see data relevant to them.

At the Dataset Level (Model-Level RLS):

Steps to Implement:

1. Create Roles in Power BI Desktop:
    - o Go to Modeling → Manage Roles.
    - o Define a role (e.g., RegionManager) and apply DAX filters like:

[Region] = "East"

2. Dynamic Security using DAX and UserPrincipalName():
    - o Useful when you want filtering based on the logged-in user.

[Region] = LOOKUPVALUE(EmployeeRegion[Region], EmployeeRegion[Email], USERPRINCIPALNAME())

3. Publish to Power BI Service:

    - After publishing, go to the dataset in the Power BI workspace → Security → Add users to roles.

At the Report Level (Report-Level RLS):

Power BI does not directly support report-level RLS (meaning RLS defined only inside the report layer). However, you can achieve this via:

- Multiple Reports with different filters
- Shared Datasets across reports with RLS at the dataset level

Best Practice: Always enforce RLS at the dataset level, not report level, for better security and maintenance.

Example:

Scenario: Sales managers should only see their region's sales.

- Dataset has columns: Sales, Region, Salesperson
- EmployeeRegion table maps emails to regions

Implement a role:

Sales[Region] = LOOKUPVALUE(EmployeeRegion[Region], EmployeeRegion[Email], USERPRINCIPALNAME())

Add managers' email IDs to roles in Power BI Service.

Q4: How would you handle slowly changing dimensions (SCDs) in SQL when preparing data for Power BI consumption?

Slowly Changing Dimensions (SCD) are attributes in a dimension table that change slowly over time rather than frequently.

SCD Types You Need to Know:

- Type 1: Overwrites old data with new data (no history)
- Type 2: Keeps history by adding a new row with versioning or dates
- Type 3: Stores only limited historical data in extra columns

---

SCD Type 1 – SQL Example:

```
MERGE INTO DimCustomer AS Target
USING StagingCustomer AS Source
ON Target.CustomerID = Source.CustomerID
WHEN MATCHED THEN
  UPDATE SET Target.Address = Source.Address
WHEN NOT MATCHED THEN
  INSERT (CustomerID, Name, Address)
  VALUES (Source.CustomerID, Source.Name, Source.Address);
```

Old data is replaced — no historical record is maintained.

SCD Type 2 – SQL Example with Historical Tracking:

```
-- Expire current record
UPDATE DimCustomer
SET EndDate = GETDATE()
WHERE CustomerID = @CustomerID AND EndDate IS NULL;

-- Insert new record with current changes
INSERT INTO DimCustomer (CustomerID, Name, Address, StartDate, EndDate)
VALUES (@CustomerID, @NewName, @NewAddress, GETDATE(), NULL);
```

Keeps full historical changes by closing the old row and inserting a new one.

In Power BI Context:

- For Type 2, your fact table should be joined to the active row using EndDate IS NULL or to a time-aware version of the dimension using BETWEEN FactDate AND Dim.StartDate/EndDate.
- Use Power Query or SQL views to present the latest version or full history depending on need.

Real-Life Use Case:

Scenario: A customer changes their address, and you want to track previous addresses in your sales reports.

- Use SCD Type 2 to create historical snapshots.
- In Power BI, use a relationship with filters like:

DimCustomer[StartDate] <= FactSales[OrderDate] &&

(DimCustomer[EndDate] IS NULL OR FactSales[OrderDate] <= DimCustomer[EndDate])

Q5: Write a SQL query to perform a window function that calculates the running total of sales by date, partitioned by product category.

Explanation:

A running total (aka cumulative sum) calculates the sum of a column, incrementally, over a defined order—here it's by sale_date, partitioned by product_category.

Sample Table: SalesData

sale_id	product_category	sale_date	sale_amount
1	Electronics	2023-09-01	200

sale_id	product_category	sale_date	sale_amount
2	Electronics	2023-09-02	300
3	Furniture	2023-09-01	150
4	Electronics	2023-09-03	100
5	Furniture	2023-09-02	200

Query with Window Function:

```
SELECT
  sale_id,
  product_category,
  sale_date,
  sale_amount,
  SUM(sale_amount) OVER (
    PARTITION BY product_category
    ORDER BY sale_date
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ) AS running_total
FROM SalesData;
```

Output:

sale_id	product_category	sale_date	sale_amount	running_total
1	Electronics	2023-09-01	200	200
2	Electronics	2023-09-02	300	500
4	Electronics	2023-09-03	100	600
3	Furniture	2023-09-01	150	150
5	Furniture	2023-09-02	200	350

Use this in dashboards to show cumulative revenue, growth trends, or progress toward a target.

Q6: Explain the differences between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN in SQL, and provide scenarios where each would be most appropriate.

INNER JOIN

- Returns only matching rows from both tables.
- If no match, row is excluded.

Use Case: Get customers who placed at least one order.

```
SELECT *
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id;
```

Best when you're only interested in common data between tables.

LEFT JOIN (LEFT OUTER JOIN)

- Returns all rows from the left table, and matched rows from the right.
- If no match, NULLs appear from the right.

Use Case: Show all products, including those that haven't been sold.

```
SELECT *
FROM Products p
LEFT JOIN Sales s ON p.product_id = s.product_id;
```

Use when your focus is on the left table, and you want to include unmatched records.

RIGHT JOIN (RIGHT OUTER JOIN)

- Like LEFT JOIN, but returns all rows from the right table and matched from the left.

Use Case: List all employees in a department, even if they have no task assigned.

```
SELECT *
FROM Tasks t
RIGHT JOIN Employees e ON t.employee_id = e.employee_id;
```

Use when you want to retain all data from the right table.

FULL OUTER JOIN

- Returns all rows from both tables, matched and unmatched.
- NULLs appear where there's no match.

Use Case: Combine customer and supplier addresses into one list, even if some customers haven't bought or some suppliers haven't supplied.

```
SELECT *
FROM Customers c
FULL OUTER JOIN Suppliers s ON c.city = s.city;
```

Best when you want to combine two datasets and retain everything.

Summary Table:

Join Type	Returns Matching rows only All	NULLs for Non-Matches
INNER JOIN	from left + matches from right All	
LEFT JOIN	from right + matches from left	on right
RIGHT JOIN		on left
FULL OUTER JOIN	All rows from both with matches & gaps	on both

---

Q7: How would you implement incremental refresh in Power BI to efficiently update large datasets?

What is IncrementalRefresh?

Incremental refresh is a technique in Power BI that refreshes only new or changed data, instead of reloading the entire dataset, significantly improving performance and reducing load time—ideal for large datasets.

Steps to Implement Incremental Refresh:

1. Add Parameters in Power BI Desktop:

- Create two parameters:
    - o RangeStart (Date/Time)
    - o RangeEnd (Date/Time)

2. Use Parameters in Power Query:

- Filter the table using a date column based on RangeStart and RangeEnd.

powerquery

= Table.SelectRows(Source, each [SaleDate] >= RangeStart and [SaleDate] < RangeEnd)

3. Define Incremental Policy:

- In Power BI Service, right-click on the table → Manage incremental refresh.
- Choose:
    - o Store rows in the past → e.g., 5 years.
    - o Refresh rows in the last → e.g., 1 day or week.

4. Publish to Power BI Service:

- Power BI automatically replaces only the new or changed partitions.

Use Case:

For a dataset containing sales transactions since 2010, you don’t need to refresh all 10 years daily—just the latest day's data.

Q8: Describe a situation where you'd use a CTE (Common Table Expression) in SQL instead of a subquery or temporary table?

What is a CTE?

A CTE (Common Table Expression) is a temporary named result set defined using WITH, which you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

When to Use CTE Instead of Subquery or Temp Table:

1. To Improve Readability in Multi-Step Logic

When breaking down logic into modular steps, especially when nesting subqueries makes it unreadable.

```
WITH FilteredSales AS (
  SELECT product_id, SUM(sale_amount) AS total_sales
  FROM Sales
  GROUP BY product_id
)
SELECT product_id
FROM FilteredSales
WHERE total_sales > 1000;
```

---

2. For Recursive Queries

Example: Get hierarchy or organization structure.

```
WITH EmployeeCTE AS (
```

```
    SELECT EmployeeID, ManagerID, 1 AS Level
    FROM Employees
    WHERE ManagerID IS NULL
    UNION ALL
    SELECT e.EmployeeID, e.ManagerID, Level + 1
    FROM Employees e
    INNER JOIN EmployeeCTE ec ON e.ManagerID = ec.EmployeeID
)
SELECT * FROM EmployeeCTE;
```

---

3. Avoid Repeating the Same Logic

Rather than repeating the same subquery multiple times, define it once using a CTE and reuse it.

---

4. When Temp Tables Are Overkill

Temp tables persist until dropped, whereas CTEs exist only during query execution—ideal for one-time transformations.

CTE vs Subquery vs Temp Table

Feature	CTE	Subquery	Temp Table
Scope	Only within that query	Nested inside query	Exists until dropped
Readability	High	Can get messy when nested	Medium
Recursive Use	Yes	No	No
Reuse in Query	Yes	No	Yes

Feature	CTE	Subquery	Temp Table
Suitable for	Modular logic, hierarchy	Simple filtering/calculations	Complex multi-step logic

---

Q9: How would you create a custom visual in Power BI using R or Python? What considerations would you keep in mind?

Purpose:

Creating custom visuals using R or Python helps when native Power BI visuals aren't enough — for example, heatmaps, advanced forecasting, or machine learning charts.

Steps to Create a Custom Visual in Power BI using R or Python:

Using Python in Power BI:

1. Enable Python scripting under Options → Python scripting.
2. In Power BI, go to Visualizations Pane → choose Python visual.
3. Drag required fields into the values section.
4. Enter Python code in the script editor.

```
import matplotlib.pyplot as plt
import seaborn as sns


dataset = dataset.dropna()
sns.boxplot(x='Category', y='Sales', data=dataset)
plt.show()
```

Using R in Power BI:

1. Enable R script support under Options.
2. Use the R visual from the visual pane.

3. Enter code in the script editor:

```
library(ggplot2)
ggplot(dataset, aes(x=Category, y=Sales)) +
 geom_boxplot()  +
 theme_minimal()
```

Considerations:

Consideration	Why It's Important
Data size limit (150K rows)	R/Python visuals in Power BI are limited to 150,000 rows.
Performance	R/Python visuals render slower than native visuals.
Package dependencies	Ensure required packages (e.g., ggplot2, matplotlib) are installed.
Security	Scripts run locally — verify code is safe.
No interaction	R/Python visuals are non-interactive with slicers/filters.

Use Case Example:

You want to create a correlation heatmap between sales metrics across regions — which isn't supported by native visuals. You can use Seaborn in Python to build it.

Q10: Write a SQL query to pivot data from rows to columns without using the PIVOT function?

Goal:

Transform row-based data into a column-based format — manually using CASE WHEN or aggregate functions like MAX(), SUM() etc.

Sample Table: Sales

Region	Quarter	Revenue
East	Q1	1000
East	Q2	1200
West	Q1	900
West	Q2	1100

Expected Output (Pivoted):

Region	Q1_Revenue	Q2_Revenue
East	1000	1200
West	900	1100

SQL Query Without PIVOT:

```
SELECT
  Region,
  MAX(CASE WHEN Quarter = 'Q1' THEN Revenue END) AS Q1_Revenue,
  MAX(CASE WHEN Quarter = 'Q2' THEN Revenue END) AS Q2_Revenue
FROM Sales
GROUP BY Region;
```

Explanation:

- CASE WHEN Quarter = 'Q1' THEN Revenue END → isolates Q1 revenue.
- MAX() used to ensure single value is returned per Region for each quarter.
- GROUP BY Region keeps one row per Region.

Why not use PIVOT?

- Some databases like MySQL or SQLite don't support PIVOT.
- Using CASE gives flexibility and works universally.