FLIPKART Business Analyst Interview Experience (1-3 years) CTC - 14 LPA

SQL

1. What are window functions, and how do they differ from aggregate functions? Can you give a use case?

Explanation:

- **Window Functions** perform calculations across a set of table rows that are related to the current row, without collapsing the rows into a single summary result.
- **Aggregate Functions**, on the other hand, return a single result for a group of rows, reducing the number of rows in the result set.

Use Case:

If you want to calculate a running total or rank without losing the row-level granularity, window functions are useful.

Example Query:

Use Case: Calculate the running total of sales for each salesperson.

Schema:

```
CREATE TABLE Sales (
    SalesID INT,
    SalesPerson VARCHAR(50),
    SaleAmount INT,
    SaleDate DATE
```

);

INSERT INTO Sales VALUES

(1, 'Alice', 200, '2025-01-01'),

(2, 'Alice', 150, '2025-01-02'),

(3, 'Bob', 300, '2025-01-01'),

(4, 'Bob', 100, '2025-01-03'),

(5, 'Alice', 250, '2025-01-03');

Query:

SELECT

SalesPerson,

SaleAmount,

SaleDate,

SUM(SaleAmount) OVER (PARTITION BY SalesPerson ORDER BY SaleDate) AS RunningTotal

FROM Sales;

Result:

SalesPerson	SaleAmount	SaleDate	RunningTotal
Alice	200	2025-01-01	200
Alice	150	2025-01-02	350
Alice	250	2025-01-03	600
Bob	300	2025-01-01	300
Bob	100	2025-01-03	400

2. Explain indexing. When would an index potentially reduce performance, and how would you approach indexing strategy for a large dataset?

Explanation:

- An **index** improves the speed of data retrieval operations by creating a structure (like a B-tree) for faster lookups.
- **Downside of Indexing:**
    - Indexes increase storage requirements.
    - They slow down write operations (INSERT, UPDATE, DELETE) because the index needs to be updated.

When Indexing May Reduce Performance:

- **Small Tables**: Full table scans are often faster than index lookups.
- **Frequent Updates**: When the table has frequent write operations, maintaining indexes adds overhead.
- **Unused Indexes**: An index on columns rarely queried wastes resources.

Indexing Strategy for Large Datasets:

1. **Index Frequently Queried Columns**: Use indexes on columns often used in WHERE, JOIN, GROUP BY, and ORDER BY clauses.
2. **Avoid Over-Indexing**: Index only what is necessary.
3. **Composite Indexes**: Create multi-column indexes for queries involving multiple columns.
4. **Regular Monitoring**: Use tools like EXPLAIN or Query Analyzer to ensure indexes are effective.

Example:

```
CREATE INDEX idx_customer_id ON Orders (CustomerID);
```

---

3. Write a query to retrieve customers who have made purchases in the last 30 days but did not purchase anything in the previous 30 days.

Example Query:

Schema:

```
CREATE TABLE Purchases (
  PurchaseID INT,
  CustomerID INT,
  PurchaseDate DATE
);

INSERT INTO Purchases VALUES
(1, 101, '2024-12-15'),
(2, 102, '2024-12-25'),
(3, 101, '2024-11-10'),
(4, 103, '2024-12-01'),
(5, 102, '2024-11-01');
```

Query:

```
WITH RecentPurchases AS (
  SELECT CustomerID
  FROM Purchases
  WHERE PurchaseDate >= CURDATE() - INTERVAL 30 DAY
),
PreviousPurchases AS (
  SELECT CustomerID
  FROM Purchases
  WHERE PurchaseDate BETWEEN CURDATE() - INTERVAL 60 DAY AND CURDATE() -
INTERVAL 30 DAY
)
```

```
SELECT DISTINCT CustomerID
FROM RecentPurchases
WHERE CustomerID NOT IN (SELECT CustomerID FROM PreviousPurchases);
```

---

4. Given a table of transactions, find the top 3 most purchased products for each category.

Example Query:

Schema:

```
CREATE TABLE Transactions (
  TransactionID INT,
  ProductName VARCHAR(50),
  Category VARCHAR(50),
  Quantity INT
);

INSERT INTO Transactions VALUES
(1, 'ProductA', 'Category1', 10),
(2, 'ProductB', 'Category1', 5),
(3, 'ProductC', 'Category1', 20),
(4, 'ProductD', 'Category2', 15),
(5, 'ProductE', 'Category2', 10),
(6, 'ProductF', 'Category2', 25);
```

Query:

```
WITH RankedProducts AS (
  SELECT
    Category,
```

```
    ProductName,
    Quantity,
    RANK() OVER (PARTITION BY Category ORDER BY Quantity DESC) AS Rank
  FROM Transactions
)
SELECT
  Category,
  ProductName,
  Quantity
FROM RankedProducts
WHERE Rank <= 3;
```

Result:

Category	ProductName	Quantity
Category1	ProductC	20
Category1	ProductA	10
Category1	ProductB	5
Category2	ProductF	25
Category2	ProductD	15
Category2	ProductE	10

---

5. How would you identify duplicate records in a large dataset, and how would you remove only the duplicates, retaining the first occurrence?

Example Query:

Schema:

```
CREATE TABLE Employees (
```

```
  EmployeeID INT,
  Name VARCHAR(50),
  Department VARCHAR(50)
);

INSERT INTO Employees VALUES
(1, 'Alice', 'HR'),
(2, 'Bob', 'Finance'),
(3, 'Alice', 'HR'),
(4, 'Charlie', 'IT'),
(5, 'Bob', 'Finance');
```

Query to Identify Duplicates:

```
SELECT Name, Department, COUNT(*)
FROM Employees
GROUP BY Name, Department
HAVING COUNT(*) > 1;
```

Query to Remove Duplicates (Retain First Occurrence):

```
WITH CTE AS (
  SELECT
    EmployeeID,
    Name,
    Department,
    ROW_NUMBER() OVER (PARTITION BY Name, Department ORDER BY EmployeeID) AS RowNum
  FROM Employees
)
```

```
DELETE FROM Employees
WHERE EmployeeID IN (
  SELECT EmployeeID
  FROM CTE
  WHERE RowNum > 1
);
```

PYTHON

1. Write a Python function to find the longest consecutive sequence of unique numbers in a list.

Explanation:

The problem is to find the longest subarray where all the elements are unique and consecutive. This can be solved using a sliding window technique:

1. Use a set to track unique elements in the current window.
2. Use two pointers (start and end) to expand and contract the window as needed.
3. Update the maximum length of the subarray when the condition is met.

Code:

```
def longest_consecutive_sequence(nums):
  if not nums:
    return 0, []

  unique_set = set()
  start = 0
  max_length = 0
  longest_seq = []
```

```
    for end in range(len(nums)):
        while nums[end] in unique_set:
            unique_set.remove(nums[start])
            start += 1

        unique_set.add(nums[end])
        current_length = end - start + 1

        if current_length > max_length:
            max_length = current_length
            longest_seq = nums[start:end + 1]

    return max_length, longest_seq

# Example usage
nums = [1, 2, 3, 1, 4, 5, 6, 2, 7, 8]
length, sequence = longest_consecutive_sequence(nums)
print("Longest Length:", length)
print("Longest Sequence:", sequence)
```

Example Output:

For the input [1, 2, 3, 1, 4, 5, 6, 2, 7, 8]:

mathematica

CopyEdit

Longest Length: 6

Longest Sequence: [1, 4, 5, 6, 2, 7]

---

2. If you're working with a large dataset with missing values, what Python libraries would you use to handle missing data, and why?

Libraries to Use:

1. **pandas**:
    - Provides powerful tools like fillna(), dropna(), and isnull() to handle missing data effectively.
    - Suitable for structured datasets like dataframes.
2. **numpy**:
    - Provides numpy.nan for identifying missing values in arrays and operations like np.isnan() to detect and manipulate them.
    - Useful for numerical computations with arrays.
3. **scikit-learn**:
    - Offers the SimpleImputer and IterativeImputer classes for statistical imputation.
    - Best for preprocessing data before machine learning tasks.
4. **pyjanitor** (optional):
    - Built on top of pandas, it simplifies cleaning operations, including handling missing data.

Examples:

Example Dataset:

```
import pandas as pd

import numpy as np


data = {

  "Name": ["Alice", "Bob", "Charlie", None],

  "Age": [25, np.nan, 30, 22],

  "Salary": [50000, 60000, None, 45000]
```

```
}

df = pd.DataFrame(data)
print("Original Dataset:\n", df)
```

Example 1: Using pandas

```
# Drop rows with missing values
df_dropped = df.dropna()
print("\nAfter Dropping Missing Values:\n", df_dropped)

# Fill missing values with a default value
df_filled = df.fillna({
    "Name": "Unknown",
    "Age": df["Age"].mean(),
    "Salary": df["Salary"].median()
})
print("\nAfter Filling Missing Values:\n", df_filled)
```

Example 2: Using scikit-learn

```
from sklearn.impute import SimpleImputer

# Impute missing values in the "Age" and "Salary" columns
imputer = SimpleImputer(strategy="mean")
df[["Age", "Salary"]] = imputer.fit_transform(df[["Age", "Salary"]])
print("\nAfter Imputation Using Scikit-learn:\n", df)
```

Example Output:

```
Original Dataset:
      Name   Age   Salary
```

0 Alice 25.0 50000.0

1 Bob NaN 60000.0

2 Charlie 30.0 NaN

3 None 22.0 45000.0

After Dropping Missing Values:

Name Age Salary

0 Alice 25.0 50000.0

After Filling Missing Values:

Name Age Salary

0 Alice 25.000000 50000.0

1 Bob 25.666667 60000.0

2 Charlie 30.000000 47500.0

3 Unknown 22.000000 45000.0

After Imputation Using Scikit-learn:

Name Age Salary

0 Alice 25.0 50000.0

1 Bob 25.7 60000.0

2 Charlie 30.0 47500.0

3 None 22.0 45000.0

Guesstimates

1. Estimate the number of online food delivery orders in a large metropolitan city over a month:

- **Assume population:**

    - Large metropolitan city population ≈ 10 million.
    - Online food delivery users ≈ 40% (4 million).
- **Assume order frequency per user:**
    - Regular users (20%): 15 orders/month.
    - Occasional users (80%): 4 orders/month.
- **Estimate orders:**
    - Regular users: 0.2 × 4M × 15 = 12M orders.
    - Occasional users: 0.8 × 4M × 4 = 12.8M orders.
- **Total monthly orders:**
    - **12M + 12.8M = 24.8M orders/month.**

2. How many customer service calls would a telecom company receive daily for a customer base of 1 million?

- **Assume complaint rate:**
    - Active users (95% of 1M): 950,000.
    - Daily issue rate: 2%.
- **Breakdown of issues:**
    - Billing (30%): 5,700 calls/day.
    - Network (40%): 7,600 calls/day.
    - Other (30%): 5,700 calls/day.
- **Total daily calls:**
    - **950,000 × 0.02 = 19,000 calls/day.**

Case Studies

1. A sudden decrease in conversion rate is observed in a popular product category. How would you investigate the cause and propose solutions?

- **Data Analysis:**
    - Analyze traffic sources for sudden drops.
    - Compare conversion rates across platforms and devices.
    - Study user demographics and behavior trends.
- **Operational Factors:**
    - Check inventory and pricing issues.
    - Identify recent policy changes (return policies, delivery fees).
    - Monitor seasonal trends or competitor campaigns.
- **Technical Investigation:**
    - Audit website or app performance (loading time, errors).
    - Identify bugs in the checkout process.
- **Solutions:**
    - Optimize product pages and pricing.
    - Run A/B testing for checkout improvements.
    - Launch targeted marketing campaigns.

2.Imagine the company is considering adding a new subscription model. How would you evaluate its potential impact on customer lifetime value and revenue?

- **Market Research:**
    - Survey customer willingness to pay for subscriptions.
    - Study competitor subscription offerings.
- **Revenue Impact:**
    - Calculate incremental revenue from expected subscribers.
    - Factor in cannibalization of existing one-time purchases.
- **CLV Analysis:**
    - Assess changes in average customer tenure.

    - Include upsell opportunities and churn rate reduction.
- **Implementation Feasibility:**
    - Evaluate operational costs for managing subscriptions.
    - Plan loyalty benefits and exclusivity perks.

Managerial Questions

1. Describe a time when you faced conflicting priorities on a project. How did you manage your workload to meet deadlines?

Managing conflicting priorities on a project:

- **Identify and prioritize:**
    - Break tasks into urgent vs. important.
    - Align priorities with organizational goals.
- **Delegate and negotiate:**
    - Reassign tasks to team members based on expertise.
    - Negotiate extended deadlines or resource allocation.
- **Time management:**
    - Use tools like Gantt charts or Kanban boards.
    - Allocate focused time slots for high-priority tasks.
- **Communicate effectively:**
    - Update stakeholders on progress and challenges.
    - Seek guidance from managers to resolve bottlenecks.

2. How would you handle a disagreement within the team on an analytical approach?

Handling a disagreement within the team on an analytical approach:

- **Encourage open dialogue:**
    - Facilitate a brainstorming session to hear all perspectives.

    - Promote a culture of constructive feedback.
- **Use data as the arbitrator:**
    - Test multiple approaches with sample data.
    - Choose the method that delivers the best outcome.
- **Promote collaboration:**
    - Merge ideas into a hybrid approach if feasible.
    - Assign team members to analyze pros and cons objectively.
- **Escalate if needed:**
    - Seek guidance from a senior manager if the disagreement persists.
    - Ensure the final decision aligns with organizational goals.