SQL v/s PySpark

Interview Questions & Solutions

Abhinav Singh

1. Find the Top 3 Products with the Highest Average Rating in Each Category

Table Schema - Products,Reviews

Products Table

ProductID	Name	CategoryID
1	Product_1	102
2	Product_2	102
3	Product_3	103
4	Product_4	101
5	Product_5	102
6	Product_6	102
7	Product_7	101
8	Product_8	103
9	Product_9	102
10	Product_10	101
11	Product_11	103
12	Product_12	102
13	Product_13	103
14	Product_14	103
15	Product_15	101
16	Product_16	101
17	Product_17	103
18	Product_18	103
19	Product_19	102
20	Product_20	101

Reviews Table

ReviewID	ProductID	Rating
1	6	4.3
2	10	4.2
3	12	3.1
4	18	2.6
5	12	1.0
6	18	1.3
7	7	3.9
8	9	4.8
9	6	3.3
10	17	1.2
11	8	2.5
12	8	4.4
13	13	1.8
14	9	3.5
15	15	1.6
16	2	3.7
17	4	2.7
18	13	1.5
19	4	4.5
20	4	3.4

Abhinav Singh

Output Table

ProductID	Name	CategoryID	AvgRating
9	Product_9	102	4.8
6	Product_6	102	4.3
2	Product_2	102	3.7
4	Product_4	101	4.5
10	Product_10	101	4.2
7	Product_7	101	3.9
8	Product_8	103	4.4
3	Product_3	103	4.2
18	Product_18	103	2.6

Abhinav Singh

PySpark Solution

```
joined_df = products.join(reviews, "ProductID")
avg_ratings = joined_df.groupBy("ProductID", "Name",
"CategoryID").agg(avg("Rating").alias("AvgRating"))

window_spec =
Window.partitionBy("CategoryID").orderBy(col("AvgRating").desc())
ranked_ratings = avg_ratings.withColumn("RatingRank",
rank().over(window_spec))

top_products = ranked_ratings.filter(col("RatingRank") <= 3)
top_products.select("ProductID", "Name", "CategoryID", "AvgRating").show()
```

SQL Solution

```
WITH RankedRatings AS (
    SELECT p.ProductID,
           p.Name,
           p.CategoryID,
           AVG(r.Rating) AS AvgRating,
           RANK() OVER (PARTITION BY p.CategoryID ORDER BY AVG(r.Rating)
DESC) AS RatingRank
    FROM Products p
    JOIN Reviews r ON p.ProductID = r.ProductID
    GROUP BY p.ProductID, p.Name, p.CategoryID
)
SELECT ProductID, Name, CategoryID, AvgRating
FROM RankedRatings
WHERE RatingRank <= 3;
```

Abhinav Singh

2. Identify Customers with Orders Increasing for Three Consecutive Weeks

Table Schema - Orders

Orders Table

OrderID	CustomerID	OrderDate	Quantity
1	101	2024-01-01	50
2	102	2024-01-08	60
3	103	2024-01-15	70
4	101	2024-01-22	80
5	102	2024-01-29	90
6	103	2024-02-05	100
7	101	2024-02-12	110
8	102	2024-02-19	120
9	103	2024-02-26	130
10	101	2024-03-04	140
11	102	2024-03-11	150
12	103	2024-03-18	160
13	101	2024-03-25	170
14	102	2024-04-01	180
15	103	2024-04-08	190

Abhinav Singh

Output Table

CustomerID	Week	WeeklyQuantity
101	10	140
101	11	150
101	12	160
102	10	150
102	11	160
102	12	170

Abhinav Singh

PySpark Solution

```
 weekly_orders = orders.groupBy("CustomerID",
weekofyear("OrderDate").alias("Week")).agg(_sum("Quantity").alias("WeeklyQuan
tity"))

 window_spec = Window.partitionBy("CustomerID").orderBy("Week")
 order_growth = weekly_orders.withColumn("PrevWeek1", lag("WeeklyQuantity",
1).over(window_spec)) \
                             .withColumn("PrevWeek2", lag("WeeklyQuantity",
2).over(window_spec))

 increasing_orders = order_growth.filter((col("WeeklyQuantity") >
col("PrevWeek1")) & (col("PrevWeek1") > col("PrevWeek2")))
 increasing_orders.select("CustomerID", "Week", "WeeklyQuantity").show()
```

SQL Solution

```
WITH WeeklyOrders AS (
    SELECT CustomerID,
           DATE_TRUNC('week', OrderDate) AS Week,
           SUM(Quantity) AS WeeklyQuantity
    FROM Orders
    GROUP BY CustomerID, DATE_TRUNC('week', OrderDate)
),
OrderGrowth AS (
    SELECT CustomerID,
           Week,
           WeeklyQuantity,
           LAG(WeeklyQuantity, 1) OVER (PARTITION BY CustomerID ORDER BY
Week) AS PrevWeek1,
           LAG(WeeklyQuantity, 2) OVER (PARTITION BY CustomerID ORDER BY
Week) AS PrevWeek2
    FROM WeeklyOrders
)
SELECT CustomerID, Week, WeeklyQuantity
FROM OrderGrowth
WHERE WeeklyQuantity > PrevWeek1 AND PrevWeek1 > PrevWeek2;
```

Abhinav Singh

3. List Employees Who Received a Bonus in Consecutive Years

Table Schema - Bonuses

Bonuses Table

BonusID	EmployeeID	BonusDate
1	201	2022-01-15
2	202	2022-03-20
3	203	2022-05-25
4	201	2023-01-10
5	202	2023-03-15
6	203	2023-05-20
7	201	2024-01-05
8	202	2024-03-10
9	203	2024-05-15
10	204	2022-07-30
11	205	2023-08-05
12	206	2024-09-10
13	204	2023-07-25
14	205	2024-08-30
15	206	2023-09-05

Abhinav Singh

Output Table

EmployeeID
201
202
203

Abhinav Singh

PySpark Solution

```
 yearly_bonuses = bonuses.groupBy("EmployeeID",
year("BonusDate").alias("BonusYear")).count()

 window_spec = Window.partitionBy("EmployeeID").orderBy("BonusYear")
 consecutive_years = yearly_bonuses.withColumn("PrevYear", lag("BonusYear",
1).over(window_spec))

 consecutive_employees = consecutive_years.filter(col("BonusYear") ==
col("PrevYear") + 1).select("EmployeeID").distinct()
 consecutive_employees.show()
```

SQL Solution

```
WITH YearlyBonuses AS (
    SELECT EmployeeID,
           EXTRACT(YEAR FROM BonusDate) AS BonusYear
    FROM Bonuses
    GROUP BY EmployeeID, EXTRACT(YEAR FROM BonusDate)
),
ConsecutiveYears AS (
    SELECT EmployeeID,
           BonusYear,
           LAG(BonusYear, 1) OVER (PARTITION BY EmployeeID ORDER BY
BonusYear) AS PrevYear
    FROM YearlyBonuses
)
SELECT DISTINCT EmployeeID
FROM ConsecutiveYears
WHERE BonusYear = PrevYear + 1;
```

Abhinav Singh

4. Find the Third Lowest Price for Each Product Category

Table Schema - Products

Products Table

ProductID	CategoryID	Price
1	301	10.99
2	302	15.49
3	301	12.99
4	302	18.99
5	301	9.99
6	302	14.99
7	301	11.99
8	302	17.49
9	301	13.99
10	302	19.99
11	301	8.99
12	302	16.99
13	301	7.99
14	302	20.49
15	301	14.99

Abhinav Singh

Output Table

CategoryID	Price
301	11.99
302	16.99

Abhinav Singh

PySpark Solution

```
window_spec = Window.partitionBy("CategoryID").orderBy("Price")
ranked_prices = products.withColumn("PriceRank",
dense_rank().over(window_spec))

third_lowest_prices = ranked_prices.filter(col("PriceRank") == 3)
third_lowest_prices.select("CategoryID", "Price").show()
```

SQL Solution

```
WITH RankedPrices AS (
    SELECT CategoryID,
           Price,
           DENSE_RANK() OVER (PARTITION BY CategoryID ORDER BY Price ASC)
PriceRank
    FROM Products
)
SELECT CategoryID, Price
FROM RankedPrices
WHERE PriceRank = 3;
```

Abhinav Singh

5. Rolling Sum of Sales for Each Product Over the Last 4 Weeks

Table Schema - Sales

Sales Table

SaleID	ProductID	SaleDate	Amount
1	401	2024-01-01	100.00
2	401	2024-01-08	150.00
3	401	2024-01-15	200.00
4	401	2024-01-22	250.00
5	401	2024-01-29	300.00
6	401	2024-02-05	350.00
7	401	2024-02-12	400.00
8	401	2024-02-19	450.00
9	401	2024-02-26	500.00
10	401	2024-03-04	550.00
11	401	2024-03-11	600.00
12	401	2024-03-18	650.00
13	401	2024-03-25	700.00
14	401	2024-04-01	750.00
15	401	2024-04-08	800.00

Abhinav Singh

Output Table

ProductID	SaleDate	Amount	RollingSum
401	2024-01-22	250.00	700.00
401	2024-01-29	300.00	900.00
401	2024-02-05	350.00	1100.00
401	2024-02-12	400.00	1300.00
401	2024-02-19	450.00	1500.00
401	2024-02-26	500.00	1700.00
401	2024-03-04	550.00	1900.00

Abhinav Singh

PySpark Solution

```
 window_spec =
Window.partitionBy("ProductID").orderBy("SaleDate").rowsBetween(-3,
Window.currentRow)
 rolling_sum_sales = sales.withColumn("RollingSum",
sum("Amount").over(window_spec))

 rolling_sum_sales.select("ProductID", "SaleDate", "Amount",
"RollingSum").show()
```

SQL Solution

```
  SELECT ProductID,
         SaleDate,
         Amount,
         SUM(Amount) OVER (PARTITION BY ProductID ORDER BY SaleDate ROWS
 BETWEEN 3 PRECEDING AND CURRENT ROW) AS RollingSum
  FROM Sales;
```

Abhinav Singh