NIQ Interview Questions and Answers

**1. Write a SQL query to print only the names of students who passed all subjects.**

**Table: student\_marks**

**Columns: Name, Subject, Marks**

**Example Data:**

**anand, math, 89**

**anand, science, 78**

**anand, english, 30**

**prasad, math, 89**

**prasad, science, 67**

**prasad, english, 56**

**student is considered pass if he/she scores >= 40 in all subjects**

**find names only of students who passed in all subjects**

**Answer:**

1. **SQL Query :**

SELECT Name

FROM student\_marks

GROUP BY Name

HAVING MIN(Marks) >= 40;

1. **PySpark Code:**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col,min

spark = SparkSession.builder.appName("StudentsPassAllSubjects").getOrCreate()

data = [

("anand", "math", 89),

("anand", "science", 78),

("anand", "english", 30),

("prasad", "math", 89),

("prasad", "science", 67),

("prasad", "english", 56)

]

columns = ["Name","Subject","marks"]

df = spark.createDataFrame(data,columns)

passed\_students =df.groupBy("Name").agg(min("marks").alias("min\_marks")).filter(col("min\_marks")>=40)

passed\_students.select("Name").show()

1. **Python Code:**

student\_marks = [

('anand', 'math', 89),

('anand', 'science', 78),

('anand', 'english', 30),

('prasad', 'math', 89),

('prasad', 'science', 67),

('prasad', 'english', 56),

]

from collections import defaultdict

students = defaultdict(list)

for name, subject, marks in student\_marks:

students[name].append((subject, marks))

def find\_passed\_students(students):

passed\_students = []

for student, subjects in students.items():

# Check if all subjects have marks >= 40

if all(marks >= 40 for subject, marks in subjects):

passed\_students.append(student)

return passed\_students

passed\_students = find\_passed\_students(students)

print("Students who passed in all subjects:", passed\_students)

**Q2. Write a query to find the second highest salary from an Emp table.**

**Table: Emp**

**Columns: EmployeeID,Name, Gender, JobTitle, Salary, City**

**Example Data:**

**1 Vinay Male Sales 25,000 Delhi**

**2 Swathi Female Sales 46,000 Hyderabad**

**3 Suman Male Sales 62,000 Hyderabad**

**4 Hemanth Male Sales 43,000 Delhi**

**5 Prasad Male Sales 50,000 Kerala**

**Answer:**

1. **SQL Query:**

SELECT salary

FROM (

SELECT salary, ROW\_NUMBER() OVER (ORDER BY salary DESC) AS row\_num

FROM Emp) WHERE row\_num = 2;

1. **Pyspark Code:**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, row\_number

from pyspark.sql.window import Window

spark = SparkSession.builder.appName("SecondHighestSalary").getOrCreate()

data = [

(1, "Vinay", "Male", "Sales", 25000, "Delhi"),

(2, "Swathi", "Female", "Sales", 46000, "Hyderabad"),

(3, "Suman", "Male", "Sales", 62000, "Hyderabad"),

(4, "Hemanth", "Male", "Sales", 43000, "Delhi"),

(5, "Prasad", "Male", "Sales", 50000, "Kerala")

]

columns = ["EmployeeID", "Name", "Gender", "JobTitle", "Salary", "City"]

emp\_df = spark.createDataFrame(data, columns)

window\_spec = Window.orderBy(col("Salary").desc())

emp\_with\_row\_number = emp\_df.withColumn("row\_num", row\_number().over(window\_spec))

second\_highest\_salary\_df = emp\_with\_row\_number.filter(col("row\_num") == 2)

second\_highest\_salary\_df.select("Salary").show()

1. **Python Code:**

import pandas as pd

data = {

'EmployeeID': [1, 2, 3, 4, 5],

'Name': ['Vinay', 'Swathi', 'Suman', 'Hemanth', 'Prasad'],

'Gender': ['Male', 'Female', 'Male', 'Male', 'Male'],

'JobTitle': ['Sales', 'Sales', 'Sales', 'Sales', 'Sales'],

'Salary': [25000, 46000, 62000, 43000, 50000],

'City': ['Delhi', 'Hyderabad', 'Hyderabad', 'Delhi', 'Kerala']

}

df = pd.DataFrame(data)

df\_sorted = df.sort\_values(by='Salary', ascending=False).reset\_index(drop=True)

df\_sorted['row\_num'] = df\_sorted.index + 1 # Adding 1 because index starts from 0

second\_highest\_salary = df\_sorted[df\_sorted['row\_num'] == 2]

print(second\_highest\_salary[['Salary']])

**Q3) Table: customer**

**col**

**1**

**1**

**1**

**1**

**null**

**Table : product**

**col**

**1**

**1**

**1**

**1**

**null**

**Write a query and no of rows if we apply below joins:**

**1. left join**

**2. right join**

**3. inner join**

**Answer:**

1. **SQL Query:**

1. Left Join:

SELECT customer.col, product.col

FROM customer

LEFT JOIN product ON customer.col = product.col;

Number of rows in result: 5 (all customer rows are included).

2. Right Join:

SELECT customer.col, product.col

FROM customer

RIGHT JOIN product ON customer.col = product.col;

Number of rows in result: 5 (all product rows are included).

3. Inner Join:

SELECT customer.col, product.col

FROM customer

INNER JOIN product ON customer.col = product.col;

Number of rows in result: 4 (only matching rows are included)

1. **Pyspark Code:**

from pyspark.sql import SparkSession

from pyspark.sql import Row

spark = SparkSession.builder.appName("JoinExample").getOrCreate()

customer\_data = [Row(col=1), Row(col=1), Row(col=1), Row(col=1), Row(col=None)]

product\_data = [Row(col=1), Row(col=1), Row(col=1), Row(col=1), Row(col=None)]

customer\_df = spark.createDataFrame(customer\_data)

product\_df = spark.createDataFrame(product\_data)

# Left Join

left\_join\_df = customer\_df.join(product\_df, customer\_df.col == product\_df.col, "left")

print("Left Join Result:")

left\_join\_df.show()

print(f"Number of rows in left join: {left\_join\_df.count()}")

# Right Join

right\_join\_df = customer\_df.join(product\_df, customer\_df.col == product\_df.col, "right")

print("Right Join Result:")

right\_join\_df.show()

print(f"Number of rows in right join: {right\_join\_df.count()}")

# Inner Join

inner\_join\_df = customer\_df.join(product\_df, customer\_df.col == product\_df.col, "inner")

print("Inner Join Result:")

inner\_join\_df.show()

print(f"Number of rows in inner join: {inner\_join\_df.count()}")

1. **Python Code:**

import pandas as pd

customer\_data = {'col': [1, 1, 1, 1, None]}

product\_data = {'col': [1, 1, 1, 1, None]}

customer\_df = pd.DataFrame(customer\_data)

product\_df = pd.DataFrame(product\_data)

# Left Join

left\_join\_df = customer\_df.merge(product\_df, on='col', how='left')

print("Left Join Result:")

print(left\_join\_df)

print(f"Number of rows in left join: {len(left\_join\_df)}")

# Right Join

right\_join\_df = customer\_df.merge(product\_df, on='col', how='right')

print("\nRight Join Result:")

print(right\_join\_df)

print(f"Number of rows in right join: {len(right\_join\_df)}")

# Inner Join

inner\_join\_df = customer\_df.merge(product\_df, on='col', how='inner')

print("\nInner Join Result:")

print(inner\_join\_df)

print(f"Number of rows in inner join: {len(inner\_join\_df)}")

**Q4) Below is the transaction table of users in a particular food ordering app.**

**a) Write a query to obtain the third transaction of every user. Output the user id, spend and transaction date.**

**b) Write a query to fine the latest transaction of all user id**

**who have no of transaction below 3 and 3rd transaction of all other who have 3 or more transactions.**

**Example Data:**

**user\_id spend transaction\_date**

**111 100.5 1/8/2022 12:00**

**111 55 1/10/2022 12:00**

**121 36 1/18/2022 12:00**

**145 24.99 1/26/2022 12:00**

**111 89.5 2/5/2022 12:00**

**Answer:**

**A. SQL Query:**

**a) Query to obtain the third transaction of every user.**

WITH RankedTransactions AS (

SELECT

user\_id,

spend,

transaction\_date,

ROW\_NUMBER() OVER (PARTITION BY user\_id ORDER BY transaction\_date) AS transaction\_rank

FROM transactions

)

SELECT

user\_id,

spend,

transaction\_date

FROM RankedTransactions

WHERE transaction\_rank = 3;

**b) Query to find the latest transaction of users with fewer than 3 transactions and the third transaction of users with 3 or more transactions.**

WITH RankedTransactions AS (

SELECT

user\_id,

spend,

transaction\_date,

ROW\_NUMBER() OVER (PARTITION BY user\_id ORDER BY transaction\_date) AS transaction\_rank,

COUNT(\*) OVER (PARTITION BY user\_id) AS total\_transactions

FROM transactions

)

SELECT

user\_id,

spend,

transaction\_date

FROM RankedTransactions

WHERE

(total\_transactions < 3 AND transaction\_rank = total\_transactions)

OR (total\_transactions >= 3 AND transaction\_rank = 3);

**B. Pyspark Code:**

**a) PySpark DataFrame Code to Obtain the Third Transaction of Every User**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, row\_number

from pyspark.sql.window import Window

spark = SparkSession.builder.appName("TransactionApp").getOrCreate()

data = [

(111, 100.5, "1/8/2022 12:00"),

(111, 55, "1/10/2022 12:00"),

(121, 36, "1/18/2022 12:00"),

(145, 24.99, "1/26/2022 12:00"),

(111, 89.5, "2/5/2022 12:00")

]

df = spark.createDataFrame(data, ["user\_id", "spend", "transaction\_date"])

df = df.withColumn("transaction\_date", col("transaction\_date").cast("timestamp"))

window\_spec = Window.partitionBy("user\_id").orderBy("transaction\_date")

df\_ranked = df.withColumn("rank", row\_number().over(window\_spec))

third\_transactions = df\_ranked.filter(col("rank") == 3).select("user\_id", "spend", "transaction\_date")

third\_transactions.show()

**b) PySpark DataFrame Code to Find the Latest Transaction of Users with Fewer Than 3 Transactions, and the Third Transaction of Users with 3 or More Transactions**

from pyspark.sql.functions import count

window\_spec = Window.partitionBy("user\_id").orderBy("transaction\_date")

df\_ranked = df.withColumn("rank", row\_number().over(window\_spec))

df\_counts = df.groupBy("user\_id").agg(count("\*").alias("total\_transactions"))

df\_with\_counts = df\_ranked.join(df\_counts, on="user\_id", how="left")

result = df\_with\_counts.filter(

(col("total\_transactions") < 3 & col("rank") == col("total\_transactions")) |

(col("total\_transactions") >= 3 & col("rank") == 3)

).select("user\_id", "spend", "transaction\_date")

result.show()

**C) Python Code:**

import pandas as pd

data = [

(111, 100.5, "1/8/2022 12:00"),

(111, 55, "1/10/2022 12:00"),

(121, 36, "1/18/2022 12:00"),

(145, 24.99, "1/26/2022 12:00"),

(111, 89.5, "2/5/2022 12:00")

]

df = pd.DataFrame(data, columns=["user\_id", "spend", "transaction\_date"])

df["transaction\_date"] = pd.to\_datetime(df["transaction\_date"])

# a) Find the third transaction of every user

df\_sorted = df.sort\_values(by=["user\_id", "transaction\_date"])

df\_sorted["rank"] = df\_sorted.groupby("user\_id").cumcount() + 1

third\_transactions = df\_sorted[df\_sorted["rank"] == 3][["user\_id", "spend", "transaction\_date"]]

print("Third Transaction of Each User:")

print(third\_transactions)

print("\n")

# b) Find the latest transaction for users with fewer than 3 transactions and the third transaction for users with 3 or more transactions

# Get the count of transactions per user

user\_counts = df.groupby("user\_id").size().reset\_index(name="total\_transactions")

df\_with\_counts = df.merge(user\_counts, on="user\_id", how="left")

df\_with\_counts = df\_with\_counts.sort\_values(by=["user\_id", "transaction\_date"])

df\_with\_counts["rank"] = df\_with\_counts.groupby("user\_id").cumcount() + 1

# Filter the users with fewer than 3 transactions to get their latest transaction

latest\_transactions = df\_with\_counts[df\_with\_counts["total\_transactions"] < 3 & (df\_with\_counts["rank"] == df\_with\_counts["total\_transactions"])]

# Filter the users with 3 or more transactions to get their third transaction

third\_transactions\_more\_than\_three = df\_with\_counts[df\_with\_counts["total\_transactions"] >= 3 & (df\_with\_counts["rank"] == 3)]

final\_result = pd.concat([latest\_transactions[["user\_id", "spend", "transaction\_date"]],

third\_transactions\_more\_than\_three[["user\_id", "spend", "transaction\_date"]]])

print("Latest or Third Transaction of Users:")

print(final\_result)

**Q5) Write a query to get the below output from given table.**

**Table:**

**student\_id, Student\_name, Subject, Mark**

**1, ram, maths, 25**

**2, ram, science, 80**

**3, sunil, maths, 40**

**4, sunil, science, 35**

**5, abi, maths, 79**

**6, abi, science, 39**

**7, abi, che, 90**

**output needed:**

**subject,count**

**maths,1**

**science,2**

**che,0**

**count is student failed in each subject.**

**Answer:**

**A. SQL Query:**

WITH ranked\_students AS (

SELECT student\_id,

Student\_name,

Subject,

Mark,

ROW\_NUMBER() OVER (PARTITION BY Subject ORDER BY Mark) AS row\_num

FROM Std\_dtls

)

SELECT Subject,

COUNT(CASE WHEN Mark < 40 THEN 1 END) AS count

FROM ranked\_students

GROUP BY Subject

ORDER BY Subject;

**Output:**

Subject count

che 0

maths 1

science 2

**B. Pyspark Code:**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, when, count

spark = SparkSession.builder.master("local").appName("Failed Students Count").getOrCreate()

data = [

(1, "ram", "maths", 25),

(2, "ram", "science", 80),

(3, "sunil", "maths", 40),

(4, "sunil", "science", 35),

(5, "abi", "maths", 79),

(6, "abi", "science", 39),

(7, "abi", "che", 90)

]

columns = ["student\_id", "Student\_name", "Subject", "Mark"]

df = spark.createDataFrame(data, columns)

df\_with\_failure = df.withColumn("Failed", when(col("Mark") < 40, 1).otherwise(0))

result = df\_with\_failure.groupBy("Subject").agg(

count(when(col("Failed") == 1, 1)).alias("count")

)

result.show()

+--------+-----+

| Subject|count|

+--------+-----+

| maths| 1|

| science| 2|

| che| 0|

+--------+-----+

**C. Python Code:**

import pandas as pd

data = {

"student\_id": [1, 2, 3, 4, 5, 6, 7],

"Student\_name": ["ram", "ram", "sunil", "sunil", "abi", "abi", "abi"],

"Subject": ["maths", "science", "maths", "science", "maths", "science", "che"],

"Mark": [25, 80, 40, 35, 79, 39, 90]

}

df = pd.DataFrame(data)

df['Failed'] = df['Mark'].apply(lambda x: 1 if x < 40 else 0)

result = df.groupby('Subject')['Failed'].sum().reset\_index()

result.rename(columns={'Failed': 'count'}, inplace=True)

all\_subjects = ['maths', 'science', 'che']

result = result.set\_index('Subject').reindex(all\_subjects, fill\_value=0).reset\_index()

print(result)

Subject count

0 maths 1

1 science 2

2 che 0

**Q6. Write a SQL query to remove duplicate rows from the table.**

**Answer:**

**A. SQL Query:**

WITH CTE AS (

SELECT \*, ROW\_NUMBER() OVER (PARTITION BY EmployeeID ORDER BY EmployeeID) AS RowNum

FROM Employee

)

DELETE FROM Employee

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

**B. Pyspark Code:**

from pyspark.sql import SparkSession

from pyspark.sql.functions import row\_number

from pyspark.sql.window import Window

spark = SparkSession.builder.appName("RemoveDuplicates").getOrCreate()

data = [

(1, "John", "HR"),

(1, "John", "HR"),

(2, "Jane", "Finance"),

(3, "Doe", "IT"),

(2, "Jane", "Finance"),

]

columns = ["EmployeeID", "EmployeeName", "Department"]

df = spark.createDataFrame(data, columns)

windowSpec = Window.partitionBy("EmployeeID").orderBy("EmployeeID")

df\_with\_row\_number = df.withColumn("RowNum", row\_number().over(windowSpec))

df\_no\_duplicates = df\_with\_row\_number.filter(df\_with\_row\_number.RowNum == 1).drop("RowNum")

df\_no\_duplicates.show()

**C. Python Code:**

employee\_data = [

(1, "John", "HR"),

(1, "John", "HR"),

(2, "Jane", "Finance"),

(3, "Doe", "IT"),

(2, "Jane", "Finance"),

]

seen = set()

unique\_employee\_data = []

for record in employee\_data:

employee\_id = record[0]

if employee\_id not in seen:

seen.add(employee\_id)

unique\_employee\_data.append(record)

for record in unique\_employee\_data:

print(record)

**Q7: Write a SQL query to list the employees who have the same job title,**

**ordered by the number of employees in each title (i.e., more common titles should appear first).**

**Answer :**

**A. SQL Query:**

WITH JobTitleCounts AS (

SELECT job\_title, COUNT(\*) AS title\_count

FROM employees

GROUP BY job\_title

),

EmployeeRanked AS (

SELECT e.employee\_id, e.employee\_name, e.job\_title, j.title\_count,

ROW\_NUMBER() OVER (PARTITION BY e.job\_title ORDER BY j.title\_count DESC) AS row\_num

FROM employees e

JOIN JobTitleCounts j ON e.job\_title = j.job\_title

)

SELECT employee\_id, employee\_name, job\_title, title\_count

FROM EmployeeRanked

ORDER BY title\_count DESC, row\_num;

**B. Pyspark Code:**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, count, row\_number

from pyspark.sql.window import Window

spark = SparkSession.builder.appName("EmployeeJobTitle").getOrCreate()

data = [

(1, 'Alice', 'Software Engineer'),

(2, 'Bob', 'Software Engineer'),

(3, 'Charlie', 'Data Scientist'),

(4, 'David', 'Software Engineer'),

(5, 'Eva', 'Data Scientist'),

(6, 'Frank', 'Manager')

]

columns = ['employee\_id', 'employee\_name', 'job\_title']

df = spark.createDataFrame(data, columns)

job\_title\_counts = df.groupBy('job\_title').agg(count('\*').alias('title\_count'))

df\_with\_count = df.join(job\_title\_counts, on='job\_title', how='inner')

window\_spec = Window.partitionBy('job\_title').orderBy(col('title\_count').desc())

df\_with\_row\_number = df\_with\_count.withColumn('row\_num', row\_number().over(window\_spec))

result = df\_with\_row\_number.select('employee\_id', 'employee\_name', 'job\_title', 'title\_count') \

.orderBy(col('title\_count').desc(), 'job\_title', 'row\_num')

result.show(truncate=False)

**C. Python Code:**

import pandas as pd

data = [

(1, 'Alice', 'Software Engineer'),

(2, 'Bob', 'Software Engineer'),

(3, 'Charlie', 'Data Scientist'),

(4, 'David', 'Software Engineer'),

(5, 'Eva', 'Data Scientist'),

(6, 'Frank', 'Manager')

]

df = pd.DataFrame(data, columns=['employee\_id', 'employee\_name', 'job\_title'])

title\_counts = df['job\_title'].value\_counts().reset\_index()

title\_counts.columns = ['job\_title', 'title\_count']

df\_with\_count = pd.merge(df, title\_counts, on='job\_title', how='inner')

df\_sorted = df\_with\_count.sort\_values(by=['title\_count', 'job\_title', 'employee\_id'], ascending=[False, True, True])

df\_sorted['row\_num'] = df\_sorted.groupby('job\_title').cumcount() + 1

df\_final = df\_sorted[['employee\_id', 'employee\_name', 'job\_title', 'title\_count', 'row\_num']]

print(df\_final)