Question 1

Based on your own experience of using Apache Pig and Apache Spark in this subject, briefly explain (i) the similarity of data processing between Apache Pig and Apache Spark, and (ii) the advantages of Apache Spark over Apache Pig.

Ans:

1. Both Apache Pig and Apache Spark are popular frameworks used for data processing in big data environments. They share some similarities in terms of their approach to data processing:
2. Data Flow Language: Apache Pig and Apache Spark both provide a high-level data flow language that allows users to express data transformations and analysis tasks. In Pig, the language is called Pig Latin, while Spark provides APIs in multiple languages like Scala, Java, Python, and R.
3. Distributed Processing: Both frameworks are designed for distributed processing, allowing users to scale their computations across a cluster of machines. They handle the complexities of parallelism and fault tolerance, enabling efficient processing of large datasets.
4. Data Abstraction: Both Pig and Spark provide abstractions for working with structured and semi-structured data. Pig operates on the concept of DataFlow operators, while Spark introduces the concept of RDDs (Resilient Distributed Datasets) and later evolved into DataFrames and Datasets.
5. Support for Various Data Sources: Both frameworks support a wide range of data sources, including Hadoop Distributed File System (HDFS), Apache HBase, Apache Cassandra, and more. They also provide connectors to various data formats such as CSV, JSON, Avro, and Parquet.

(ii) Advantages of Apache Spark over Apache Pig:

While Apache Pig is a powerful data processing tool, Apache Spark offers several advantages over Pig:

1. Speed: Apache Spark is known for its in-memory processing capabilities, which can significantly speed up data processing tasks. It keeps the intermediate data in memory, reducing the need for disk I/O and resulting in faster execution times compared to Pig.
2. Rich APIs: Spark provides a rich set of APIs in multiple programming languages, allowing developers to write complex data processing logic in a familiar language. It supports APIs for batch processing (RDD), structured processing (DataFrames and Datasets), and streaming processing (Spark Streaming).
3. Advanced Analytics: Spark offers a wide range of built-in libraries for advanced analytics and machine learning, such as Spark MLlib and Spark GraphX. These libraries provide efficient implementations of algorithms and tools for data scientists to perform tasks like classification, regression, clustering, and graph processing.
4. Integration with Streaming Data: Apache Spark has native support for stream processing through its Spark Streaming module. It allows real-time data ingestion and processing from sources like Apache Kafka, enabling near-real-time analytics and event-driven applications.
5. Spark SQL: Spark provides a SQL interface called Spark SQL, which allows users to write SQL queries directly on structured data. This simplifies the data processing workflow for users familiar with SQL and facilitates integration with existing SQL-based systems and tools.
6. Ecosystem and Community: Spark has a vibrant and active community with extensive documentation, tutorials, and resources. It has a larger ecosystem of tools and frameworks built around it, making it easier to integrate Spark into existing data processing pipelines and leverage additional functionalities.

Overall, Apache Spark's speed, rich APIs, advanced analytics capabilities, streaming support, SQL interface, and ecosystem make it a compelling choice for data processing in big data environments, providing advantages over Apache Pig in many scenarios.

Question 2

A table named X contains a column of keys and a column of values:

|  |  |
| --- | --- |
| Key | Value |
| K1 | 1 |
| K1 | 2 |
| K2 | 3 |
| K2 | 4 |

Note that the first row of X contains the column names. Explain how to implement the following SQL-like query in the MapReduce model:

SELECT key, SUM(value) FROM X GROUP BY key

You need to specify the key-value data in the input and output of the Map and Reduce stages.

Ans:

- The table is split into blocks of fixed size. Each blocks then sent to the Map function, and the Map function produces the following key value pairs:

(k1, 1), (k1, 2), (k2,3), (k2, 4)

- The partitioner creates partitions and forms the groups. The partitioner ensures that each key-value pair is passed to one reducer. After the shuffle-and-sort, the input to the Reduce function are as follows:

(k1,[1,2]), (k2,[3,4])

- The Reduce functions compute the sum for each key, producing the grouped key value pair aggregated by sum. Hence, the final output is:

(k1,3), (k2, 7)

Question 3

Read and analyse a specification of data warehouse domain described below.

A university administration would like to record time spent by the students in the lecture classes. To achieve that, the administration installed at the entries to the lecture halls the devices that can read and recognize student cards. Now, each time a student enters and leaves a lecture hall she/he swaps a card against a device, and the total time spent by a Student in a lecture hall is computed and recorded.

The administration has the operational databases that contain information about students, subjects, timetables and enrolments. The administration would like to use these databases and information obtained from scanning of student cards to create a data warehouse. It should be possible to get from the warehouse information about the total time spent by each student in lecture classes per given period of time, such as per day, per week, per month, per session, and per year. It should be possible to get information about the total number of times each student attended the lectures per day, per week, per month etc. It should be possible to get information about time spent by each student in the lecture classes per subject, per degree enrolled, and per school that offers the subjects. It should be also possible to get information about the total number of students who attended lectures per subject, per degree enrolled, and per school that offers the subjects.

In your solution, describe each level of dimension with at least 1 and at most 3 attributes. The names of attributes that you choose must relate to the domain described above. Use either ID tags or underscores to denote the identifiers. The notations that you use must conform with the one taught in ISIT312 or ISIT912.

(3.1) Create a conceptual schema for the above specification of a sample data warehouse domain. The graphical notation that you use must conform with the one taught in ISIT312 or ISIT912.

Fact table:

Lecture Attendance

Entity:

STUDENT(stdId, name) PK:stdId

SUBJECT(subCode, school) PK: subCode

TIMETABLE(subCode, Date, time) PK:(subCode, Date, time)

FK: subCode reference SUBJECT(subCode)

Enrolment(stdId, subCode, enrolDate) PK:(stdId, subCode,, enrolDate)

FK1:stdId reference STUDENT(stdId)

FK2:subCode reference SUBJECT(stdCode)

Measurement:

Total time spent

Total Num of Attendance

Time spent in class

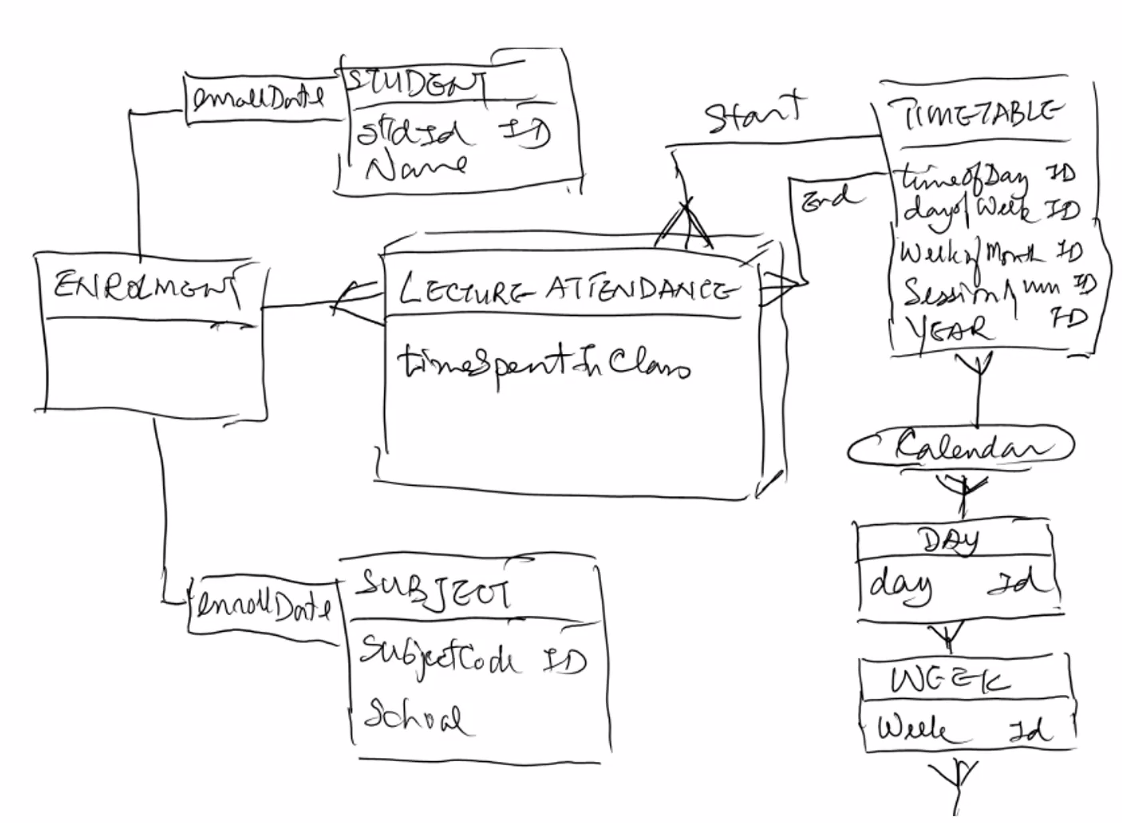
Total Num of students

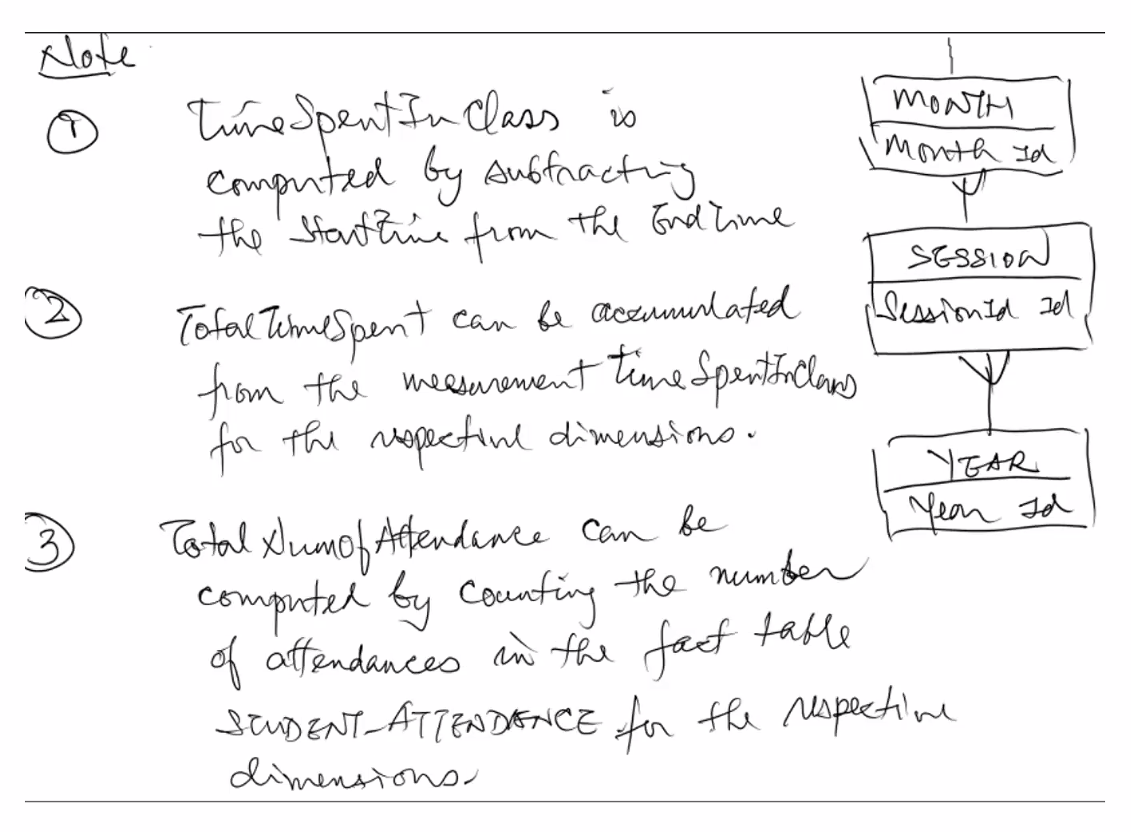
Dimension

Time, Subject, degree enrolled, school

Hierarchy:

Time: time of day, day, week, month, session, year





Note:

1. Time spent in class is computed by subtracting the startTime from the endTime
2. Total time spent can be accumulated from the measurement time spent in class for the respective dimensions
3. Total num of attendance can be computed by counting the number of attendance in the fact table STUDENT-ATTENDANCE for the respective dimension
4. Total num of student can be computed by counting the number of students in the enrollment in the fact table(using group by clause)

(3.2) Use the relational-algebraic notation to specific the following OLAP query: -“Find the total time spent by each student in lecture classes per session and lecture”

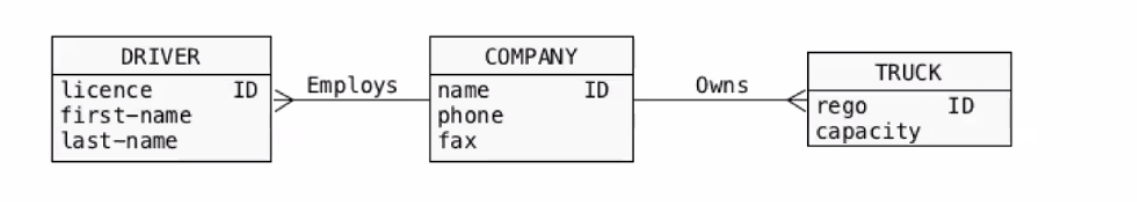
Select stdId, subCode, EnrollDate, sessionNum

From LECTURE-ATTENDANCE

Group by stdId, subCode, EnrollDate, sessionNum;

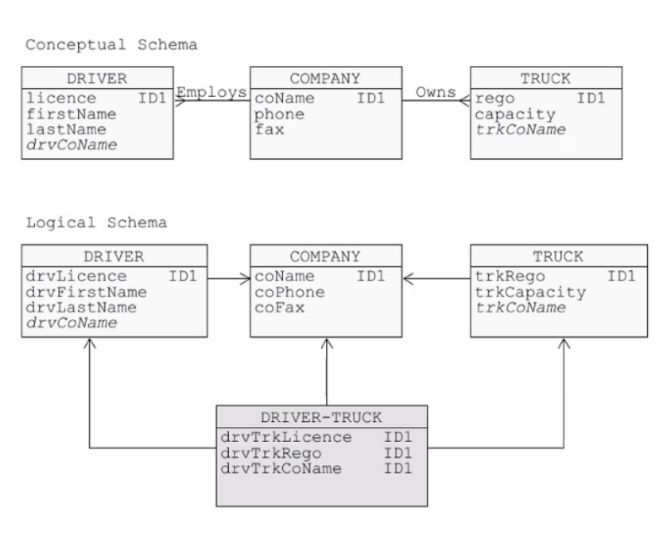
Question 4

Consider the following conceptual schema of some data cube:



(4.1) Present a drawing of a logical schema based on the above conceptual schema. The notation that you use must conform with the one taught in ISIT312

Ans:



(4.2) Write the HQL statements that implement the logical schema resulted in the previous step as internal tables in Hive. (You should make reasonable assumption about the row, column and file for mats of the physical data.)

Ans:

create table DRIVER(

drvLicence varchar(30),

drvFirstName varchar(30),

drvLastname varchar(30),

drvCoName varchar(30))

Row format delimited fields terminated by ‘,’

stored as textfile;

create table COMPANY(

coName varchar(30),

coPhone varchar(20),

CoFax varchar(20))

row format delimited fields terminated by ‘,’

stored as textfile;

create table DRIVER-TRUCK(

drvTrkLicence varchar(30)

drvTrkRego varchar(30),

drvTrkCo varchar(30))

row format delimited fields terminated by ‘,’

stored as textfile;

(4.3) Write an HQL query for finding "the average number of drivers which are associated with a truck”

SELECT drvTrkRego, avg(\*)

FROM DRIVER-TRUCK

GROUP BY drvTrkRego;

Question 5

Implement as a single HBase table a database that contains information described by the following conceptual schema.

(5.1) Write HBase shell commands that create the HBase table and load some sample data into the table. The sample data includes information about at least two accidents and two cars and one person involved in both accidents. All other information is up to you.

Ans:

# Create a table ‘QUESTION5' with four columns

# families - PERSON,ACCIDENT, CAR,LOCATION, and DATE.

create ‘QUESTION’, ‘PERSON’, ‘ACCIDENT’, ‘CAR’, ‘LOCATION’, ‘DATE’

# Insert at least two accidents, two cars, and one person involved in both accidents

# two cars

put ‘QUESTION5’, ‘car:SLF1234L’, ‘CAR:regno’, ‘ SLF1234L’

put ‘QUESTION5’, ‘car:SLF1234L’, ‘CAR:make’, ‘ Honda’

put ‘QUESTION5’, ‘car:SLF1234L’, ‘CAR:model’, ‘ Civic’

put ‘QUESTION5’, ‘car:SLQ4321M’, ‘CAR:regno’, ‘SLQ4321M ’

put ‘QUESTION5’, ‘car:SLQ4321M’, ‘CAR:make’, ‘ Toyota’

put ‘QUESTION5’, ‘car:SLQ4321M’, ‘CAR:model’, ‘ Camry’

# Insert at least two accidents, two cars, and one person involved in both accidents

# one person

Put ‘QUESTION5', person:S2598828C’, ‘PERSON:licenceNum'，S2598828C’

put 'QUESTION5’, ‘person:S2598828C’, ‘PERSON:fName’, ’James’

put 'QUESTION5’, ‘person:S2598828C’, ‘PERSON:IName’, ‘Bond’

# accident 1

put 'QUESTION5’, ‘accident:S2598828C|SLD1234L|2023|Singapore’,

‘ACCIDENT:severity', ‘serious’

Put’QUESTION5’, ‘accident:S2598828C|SLD1234L|2023| Singapore', ACCIDENT:damagelnDollar', 5000

Put’QUESTION5’, ‘accident:S2598828C|SLD1234L|2023| Singapore',

‘PERSON:licenceNum', ‘S2598828C’

Put’QUESTION5’, ‘accident:S2598828C|SLD1234L|2023| Singapore',

‘CAR:regno', ‘SLF1234L’

Put’QUESTION5’, ‘accident:S2598828C|SLD1234L|2023| Singapore',

‘DATE:year’, ‘2023’

Put’QUESTION5’, ‘accident:S2598828C|SLD1234L|2023| Singapore',

‘LOCATION:city', ‘Singapore’

# accident 2

put 'QUESTION5’, ‘accident:S2598828C|SLD4321M|2023|Singapore’,

‘ACCIDENT:severity', ‘light’

Put’QUESTION5’, ‘accident:S2598828C|SLQ4321M||2023| Singapore', ACCIDENT:damagelnDollar', 300

Put’QUESTION5’, ‘accident:S2598828C||SLQ4321M||2023| Singapore',

‘PERSON:licenceNum', ‘S2598828C’

Put’QUESTION5’, ‘accident:S2598828C||SLQ4321M|L|2023| Singapore',

‘CAR:regno', ‘SLQ4321M’

Put’QUESTION5’, ‘accident:S2598828C||SLQ4321M||2023| Singapore',

‘DATE:year’, ‘2023’

Put’QUESTION5’, ‘accident:S2598828C||SLQ4321M||2023| Singapore',

‘LOCATION:city', ‘Singapore’

# one location

put 'QUESTION5’, ‘location:Bukit Timah|Singapore’,

‘LOCATION:street', ‘Bukit Timah’

put 'QUESTION5’, ‘location:Bukit Timah|Singapore’,

‘LOCATION:city', ‘Singapore’

put 'QUESTION5’, ‘location:Bukit Batok|Singapore’,

‘LOCATION:street', ‘Bukit Batok’

put 'QUESTION5’, ‘location:Bukit Batok|Singapore’,

‘LOCATION:city', ‘Singapore’

# two date

put 'QUESTION5’, ‘date:30|May|2023’, ‘DATE:day’, 30

put 'QUESTION5’, ‘date:30|May|2023’, ‘DATE:month’, ‘May’

put 'QUESTION5’, ‘date:30|May|2023’, ‘DATE:year’, 2023

put 'QUESTION5’, ‘date:30|June|2023’, ‘DATE:day’, 3

put 'QUESTION5’, ‘date:30|June|2023’, ‘DATE:month’, ’June’

put 'QUESTION5’, ‘date:30|June|2023’, ‘DATE:year’, 2023

(5.2) Assume that this HBase table has been in use and a lot of data has been populated in it. Write the HBase shell commands that implement the following queries.

1. Find all information about the accidents having damages higher than 1000.

Ans:

Scan ‘QUESTION5’, {COLUMNS => [‘ACCIDENT’],

FILLTER => (“SingleColumnValueFilter(‘ACCIDENT’,

‘damage’, > , ‘binary:1000’ ”) }

(b) List the first and last names of people involved in accidents in Sydney in 2019.

Ans:

Scan ‘QUESTION5’, {COLUMNS => [‘PERSON:fName’,

‘PERSON:lName’],

FILLTER => (“SingleColumnValueFilter(‘ACCIDENT’,

‘city’, = , ‘binary:Sydney’) AND

(SingleColumnValueFillter(‘DATE’, ‘year’, = ,

‘binary:2019,)”) }

Question6

Assume that the files students.txt, majors.txt and GPAs.txt are stored in the HDFS directory hdfs://localhost:8080/records/.

--students.txt

--schema: studentid, name, majorid

1,Henry,100

2,Karen,100

3,Paul,101

4,Jimmy,102

5,Janice,102

--majors.txt

--schema: majorid, majorname

100,Software Engineering

101,Information Management

102,Big Data

103,Cyber Security

-- GPAs.txt

--schema: studentid, marjorid, GPA

1,100,3.0

2,100,4.0

3,101,2.5

4,102,4.5

5,100,4.0

Write Pig-Latin statements that perform the following operations. For (6.2) and (6.3), also present the

correct output data.

(6.1) Load datasets by using the provided relation names and field names. The fields of each relation

must have the suitable data types.

students = LOAD 'hdfs://localhost:8080/records/students.txt' USING PigStorage(',') AS (studentid:int, name:chararray, majorid:int);

majors = LOAD 'hdfs://localhost:8080/records/majors.txt' USING PigStorage(',') AS

(majorid:int, majorname:chararray);

gpas = LOAD 'hdfs://localhost:8080/records/GPAs.txt' USING PigStorage(',') AS

(studentid:int, majorid:int, GPA:float);

(6.2) Define a relation named studentMajor which is the left outer join of students and majors. Then list the data in this relation.

studentMajor = JOIN students BY majorid LEFT OUTER, majors BY majorid;

-- List data in relation studentMajor

DUMP studentMajor;

(6.3) Define a relation named SEMajor which contains two fields: the highest GPA value of the Software Engineering major, and the name of the student who has the highest GPA. Then list the data in this relation.

SEMajor = FOREACH (GROUP gpas BY majorid) {

ordered = ORDER gpas BY GPA DESC;

top = LIMIT ordered 1;

GENERATE FLATTEN(top.GPA), FLATTEN(top.studentid);

}

-- List data in relation SEMajor

DUMP SEMajor;

-- Filter students and GPAs for Software Engineering major

se\_students = FILTER students BY majorid == 100;

se\_gpas = FILTER gpas BY majorid == 100;

-- Join se\_students and se\_gpas on studentid

se\_data = JOIN se\_students BY studentid, se\_gpas BY studentid;

-- Find the student with the highest GPA

se\_max\_gpa = FOREACH (GROUP se\_data BY se\_students::studentid) {

ordered\_data = ORDER se\_data BY se\_gpas::GPA DESC;

top\_student = LIMIT ordered\_data 1;

GENERATE group AS studentid, FLATTEN(top\_student.(se\_gpas::GPA, se\_students::name)) AS (max\_gpa:float, student\_name:chararray);

}

-- Define SEMajor relation

SEMajor = FOREACH (GROUP se\_max\_gpa ALL) {

max\_gpa = MAX se\_max\_gpa.max\_gpa;

highest\_gpa\_student = FILTER se\_max\_gpa BY max\_gpa == se\_max\_gpa.max\_gpa;

GENERATE FLATTEN(highest\_gpa\_student.(max\_gpa, student\_name));

}

-- List the data in SEMajor

DUMP SEMajor;

(7.1) Assume that the file people .txt is in the HDFS directory

hdfs://localhost:8080/people/. The contents in people.txt are as follows:

Michael, 29, software engineer

Andy, 30, data scientist

Justin,19, business analyst

...

Explain how to load people.txt into a Spark dataframe named peopleDF, where the first and third columns should have a string field and the second column should have an integer field. Also present the Scala source code of your operations.

Ans:

case class Person(name: String, age: Long, job: String)

val personDF = spark.sparkContext.textFile(" people/people.txt ")

.map(\_.split(","))

.map(attributes => attributes(0)

attributes(1).trim.tolnt,

attributes(2))

.toDF

(7.2) Assume that a dataframe named FlightsDF of flight statistics is defined in Spark, with the following code processed

FlightsDF.printSchema()

Out:

root

|-- DEST\_CITY: string (nullable = true)

|-- DEST\_COUNTRY\_NAME: string (nullable = true)

|-- ORIGIN\_CITY: string (nullable = true)

|-- ORIGIN\_COUNTRY\_NAME: string (nullable = true)

DF.show(2)

Out:

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

|DEST\_CITY|DEST\_COUNTRY|ORIGIN\_CITY|ORIGIN\_COUNTRY|

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

|Sydney |Australia |Melbourne |Australia |

|Auckland |New Zealand |Singapore |Singapore |

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

only showing top 2 rows

Based on FlightsDF, write down the Scala code in Spark to implement the operation

- Find the countries with most international flights

Note. An international flight has different original and destination flights

Ans:

flightDF.createOrReplaceTempView("flightsView")

Select originCountry, count(\*) as totalIntFlight

From flightsView

Where destCountry != originCountry

Group by originCountry

Order by 2

Limit 1;