ITC 6107: Data Storage and Retrieval Techniques Project Winter Term 2020

Compare the performance of a standard RDBMS and SPARK-SQL on medium-size data

Instructor: Dr Ioannis T. Christou

Theodoros Kalmanidis ID:218580

Contents

Table of Figures

Abstract

The main goal of this project is to compare the performance of a standard RDBMS and SPARK-SQL Framework on medium-size data. This could be achieved by storing and retrieving data from our RDBMS, as well as from our local disk using Spark SQL. For this demonstration we are going to run complex queries both on the RDBMS of our choice and on our local disk using Spark Framework.. Especially for our Spark Framework processing we are going to use the PySpark library with purpose to query our dataset using only one core in the beginning and at a later stage utilizing all the available cores. For our RDBMS processing we will use Microsoft's MSSQL Server 2012, Oracle 11 G R2, MySql 8 and Postgre-sql 10.12 and we will query our data using the classical PL/T-SQL using the appropriate Python Connector for each database.Calculating the execution time for those two cases will give us a crystal clear picture of the performance and speed for our two scenarios (RDBMS / SPARK-SQL) with final goal to run a statistical analysis of the extracted results.

The dataset

The dataset that we are going to use for this project is called Churn Modelling and it is about Banking Data from an Individual Bank. It is freely available on Kaggle (https://www.kaggle.com/kmalit/bank-customer-churn-prediction/data) and it contains about 10,000 rows and 14 feature variables. Each row corresponds to a customer infomation, and includes the following variables:

1. RowNumber: simply the number of the dataset's row
2. CustomerId: The unique Id for each specific customer
3. Surname: The surname of the customer
4. CreditScore: The calculated Credit Score of the customer
5. Geography: The country of the customer
6. Gender: The gender of the customer
7. Age: The age of the customer
8. Tenure: Number of Tenure payment plan
9. Balance: The current balance of customers' deposit
10. NumOfProducts: Number of Bank product services
11. HasCrCard: The possession of Credit Card
12. IsActiveMember: The activity status of the customer
13. EstimatedSalary: The estimated salary of the customer
14. Exited: The current churn status of the customer

We can very easily notice that this dataset full fills our requirements

- >10000 rows
- >10 attributes
- 3 columns with less than 10 distinct values (Exited, Gender, Geography for instance)

First things first let's see our dataset's description:

```
There are 10000 rows and 14 columns
There are no missing values
[    1     2     3 ...  9998  9999 10000]
[15634602 15647311 15619304 ... 15584532 15682355 15628319]
['Hargrave' 'Hill' 'Onio' ... 'Kashiwagi' 'Aldridge' 'Burbidge']
[619 608 502 699 850 645 822 376 501 684 528 497 476 549 635 616 653 587
 726 732 636 510 669 846 577 756 571 574 411 591 533 553 520 722 475 490
 804 582 472 465 556 834 660 776 829 637 550 698 585 788 655 601 656 725
 511 614 742 687 555 603 751 581 735 661 675 738 813 657 604 519 664 678
 757 416 665 777 543 506 493 652 750 729 646 647 808 524 769 730 515 773
 814 710 413 623 670 622 785 605 479 685 538 562 721 628 668 828 674 625
 432 770 758 795 686 789 589 461 584 579 663 682 793 691 485 650 754 535
 716 539 706 586 631 717 800 683 704 615 667 484 480 578 512 606 597 778
 514 525 715 580 807 521 759 516 711 618 643 671 689 620 676 572 695 592
 567 694 547 594 673 610 767 763 712 703 662 659 523 772 545 634 739 771
 681 544 696 766 727 693 557 531 498 651 791 733 811 707 714 782 775 799
 602 744 588 747 583 627 731 629 438 642 806 474 559 429 680 749 734 644
 626 649 805 718 840 630 654 762 568 613 522 737 648 443 640 540 460 593
 801 611 802 745 483 690 492 709 705 560 752 701 537 487 596 702 486 724
 548 464 790 534 748 494 590 468 509 818 816 536 753 774 621 569 658 798
 641 542 692 639 765 570 638 599 632 779 527 564 833 504 842 508 417 598
 741 607 761 848 546 439 755 760 526 713 700 666 566 495 688 612 477 427
 839 819 720 459 503 624 529 563 482 796 445 746 786 554 672 787 499 844
 450 815 838 803 736 633 600 679 517 792 743 488 421 841 708 507 505 456
 435 561 518 565 728 784 552 609 764 697 723 551 444 719 496 541 830 812
 677 420 595 617 809 500 826 434 513 478 797 363 399 463 780 452 575 837
 794 824 428 823 781 849 489 431 457 768 831 359 820 573 576 558 817 449
 440 415 821 530 350 446 425 740 481 783 358 845 451 458 469 423 404 836
 473 835 466 491 351 827 843 365 532 414 453 471 401 810 832 470 447 422
 825 430 436 426 408 847 418 437 410 454 407 455 462 386 405 383 395 467
 433 442 424 448 441 367 412 382 373 419]
```

Image 1: Dataset Description

```
['France' 'Spain' 'Germany']
['Female' 'Male']
[42 41 39 43 44 50 29 27 31 24 34 25 35 45 58 32 38 46 36 33 40 51 61 49
 37 19 66 56 26 21 55 75 22 30 28 65 48 52 57 73 47 54 72 20 67 79 62 53
 80 59 68 23 60 70 63 64 18 82 69 74 71 76 77 88 85 84 78 81 92 83]
[ 2  1  8  7  4  6  3 10  5  9  0]
[     0.    83807.86 159660.8  ...  57369.61  75075.31 130142.79]
[1 3 2 4]
[1 0]
[1 0]
[101348.88 112542.58 113931.57 ...  42085.58  92888.52  38190.78]
[1 0]
```

Image 2: Dataset Description-2

We can confirm that our dataset consists of 10000 rows and 14 columns and we can also see all the distinct values for each variable contained in our dataframe. As we can see we have a mix of numerical and categorical and values ; and we can notice that for some columns like Gender (Male/Female), Geography(France/Spain/Germany) and Exited which is a binary attribute (0/1) we do not have more than 3 different distinct values. On the other hand

some other columns like Estimated Salary and Surname contain multiple values (more than eight).

Before we proceed to our Data Storage and Retrieve step, let's visualize our dataset with purpose to get more familiar with it. This will help us to get a better understanding of our result when we will write our queries.

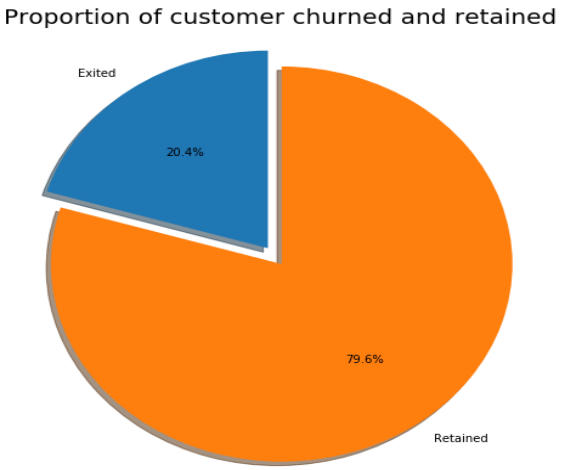

Image 3 : Churned Customer Piechart

In the above piechart we can see that the majority of the customers in our dataset are not churners. So what we are going to use for our queries are attributes which are responsible for the percentages we see on our pie.

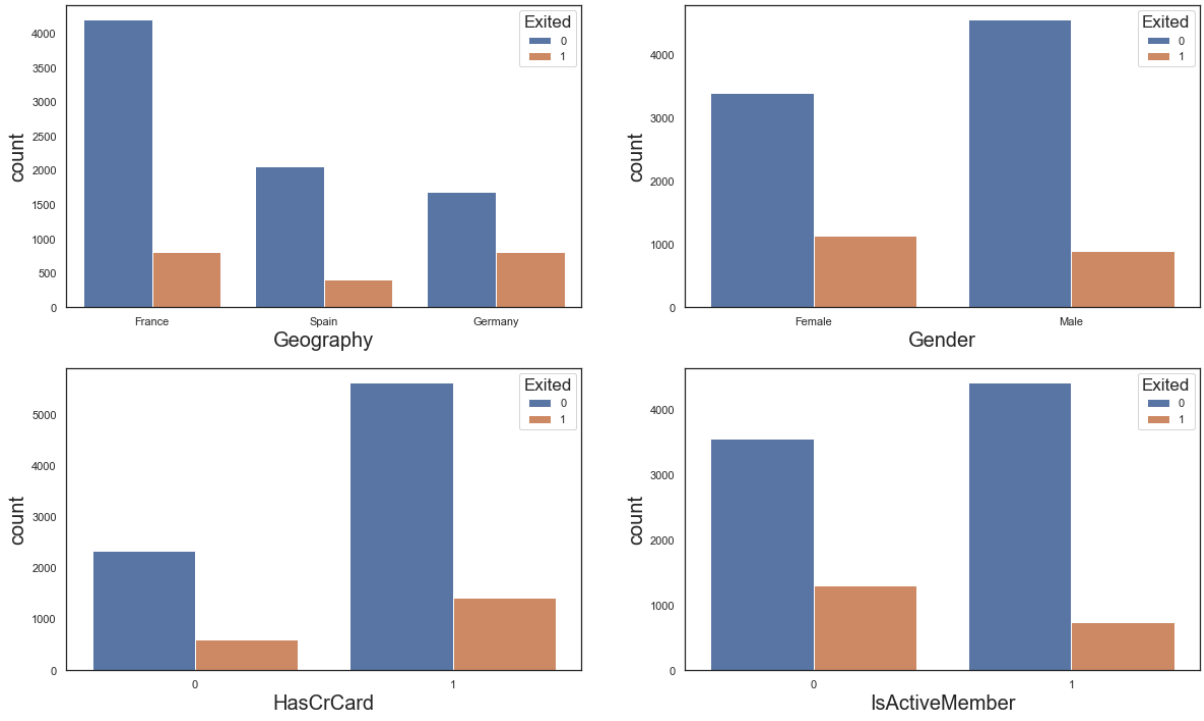

Image 4: Categorical Data Barplots

The above barplots can help us to understand what are the main characteristics of our churners. The majority of our customers and inevitably our churners are located in France and are Males. It is also interesting to notice that the majority of churners have a credit card. Unsurprisingly the inactive members column is higher than the active members, so the less time a customer spends with the bank the more likely he is to become a churner.

For the numerical attributes we created the following box-plots.

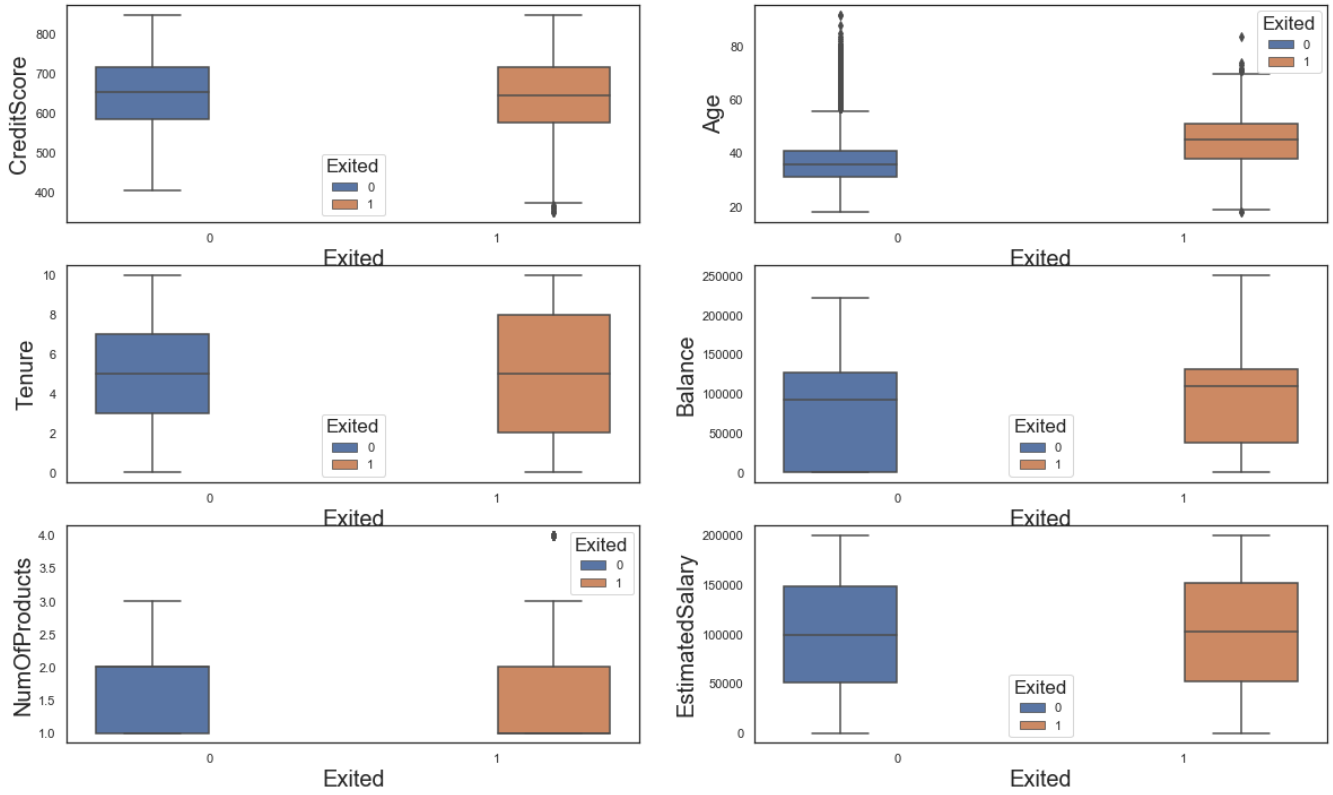

Image 5: Box plots for Numerical Data

If we observe each box-plot carefully we can see that there is no significant difference between the y axis values and that all the boxes (churner and not churner) are pretty much at the same height. The only two variables that are slight different is the Age variable and the Estimated Salary variable, which simply means that older Customers and people with high salary are more likely to leave the bank.

Importing and Storing Data

In this part we are going to see what technological stack did we use to store our data and what are the main differences between each database. Ideally we want our data to be stored exactly as appeared in our Excel File:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
RowNumb	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.8	1	1	1	79084.1	0
6	15574012	Chu	645	Spain	Male	44	8	113755.8	2	1	0	149756.71	1

Image 6: Dataset in Excel form

The first database we used for our storage is an SQL Server 2012 database called "ITC6107A1" and it contains one table with the same name as our Excel file "Churn_Modelling" as the following image depicts:

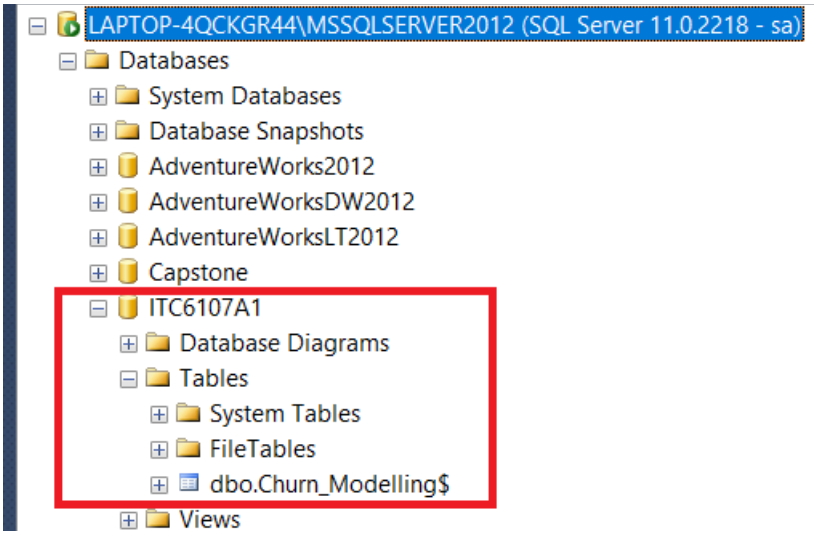

Image 7 : SQL Server Menu

Using the Microsoft's Importing Excel Wizard was very easy and it resulted in storing the data exactly as we wanted them to be:

LAPTOP-4QCKGR44\....Churn_Modelling$

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfPro...	HasCrCard	IsActiveMe...	EstimatedS...	Exited
	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1	0
	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
	9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.5	0
	10	15592389	H?	684	France	Male	27	2	134603.88	1	1	1	71725.73	0

Image 8: SQL Server Churn_Modelling Table

We can query our data stored in our SQL Server database through Python just by installing the “pyodbc” library. Then all we have to do is to create the correct connection string and start querying our data. An Indicative Example is the following:

```
from datetime import datetime
import pyodbc
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=LAPTOP-4QCKGR44\MSSQLSERVER2012;'
                      'Database=ITC6107a1;'
                      'Trusted_Connection=yes;')

cursor = conn.cursor()









#Query No#1
start=datetime.now()
cursor.execute('SELECT COUNT(CustomerId) AS [Number of Exited Customers]  FROM ITC6107A1.dbo.Churn_Modelling$ WHERE Exited=1 ')

for row in cursor:
    print(row)


print("-----------------------------------")
print ('The time for Query No#1 is %s' % (datetime.now()-start))
print("-----------------------------------")
```

Image 9: SQL Server connector code

We can see that for the SQL Server queries we have put the correct database name and we have entered also the correct server name as we showed earlier. Then we used the datime library to count the execution and plotting time of each specific query. As we can see the result is the data and on the bottom we can see the execution time.

```
In [123]: runfile('C:/Users/30694/SQLServer.py', wdir='C:/
Users/30694')
(2037, )
-----------------------------------
The time for Query No#1 is 0:00:00.002138
-----------------------------------
```

Image 10: SQL Server code results

The very first query we used is “SELECT COUNT(CustomerID) AS Number_of_Exited_Customers FROM churn_modelling WHERE Exited=1” we are going to explain all our queries at a later stage but this simply aggregate query with only one aggregate function will be our basic example for the rest databases in order to have a common line and be sure that the same query produces the exact same results in all our databases.

Similarly we did exactly the same steps for our MySql 8.0.19 Database in the Workbench studio, we used the same database and table names as we used in SQL Server and again we can see that the table's schema is the one we are looking for:

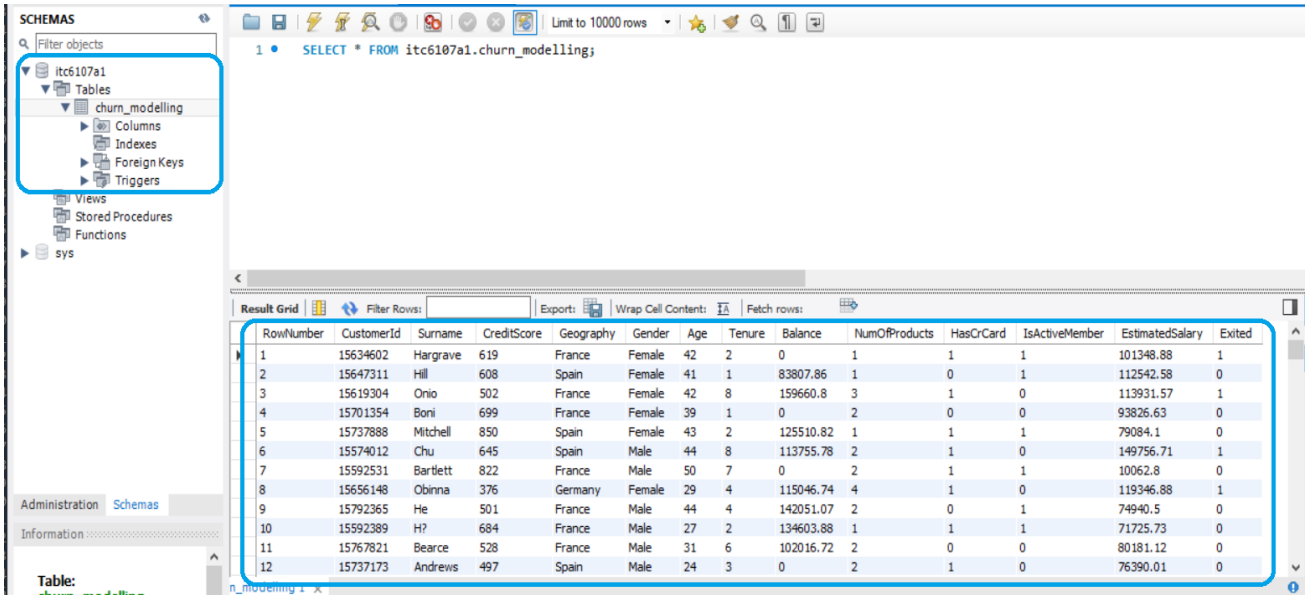

Image 11: MySql Menu

The querying concept via Python is exactly the same with the SQL Server but this time we are going to use the "mysql-connector" library in order to be able to communicate with the MySql Server:

```
from datetime import datetime
import mysql.connector

mydb = mysql.connector.connect(user='root', password='Bnft!123',
                               host='127.0.0.1', database='itc6107a1',
                               auth_plugin='mysql_native_password')


mycursor = mydb.cursor()

# Query No# 1
start=datetime.now()
mycursor.execute("SELECT COUNT(CustomerID) AS Number_of_Exited_Customers  FROM churn_modelling WHERE Exited=1 ")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)

print("------------------------------------")
print ('The time for Query No#1 is %s' % (datetime.now()-start))
print("------------------------------------")
```

Image 12: MySql Connector code

We can see that we are following the same "clean code" manner in our Python script. This helps us to spot very easily the differences between the each's databases' Python script. The datetime library and the Printing format is exactly the same, the only difference is the definition of the connection string which impacts the declaration of the database's cursor variable. Again we can see that the result is exactly the same '2037' for our counter which means that both databases are at the same state.

The third database that we chose to use for this demonstration is an Oracle 11 G R2 database. Due to the fact that setting up and a defining a new Oracle table is quite complex we are going to use the default “HR” database that comes with the Oracle’s Installation but the table’s name is going to be exactly the same churn_modelling:

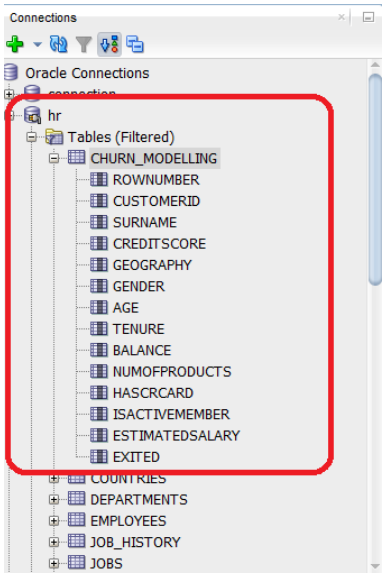

Image 13: Oracle Churn_Modelling Table Schema

The data stored in this table follow exactly the table structure and the order we wanted them to be:

ROWNUMBER	CUSTOMERID	SURNAME	CREDITSCORE	GEOGRAPHY	GENDER	AGE	TENURE	BALANCE	NUMOFPRODUCTS	HASCRCARD	ISACTIVEMEMBER	ESTIMATEDSALARY	EXITED
222	15787155	Yang	514	Spain	Male	30	7	0	1	0	1	125010.24	0
223	15727829	McIntyre	567	France	Male	42	2	0	2	1	1	167984.61	0
224	15733247	Stevenson	850	France	Male	33	10	0	1	1	0	4861.72	1
225	15568748	Poole	671	Germany	Male	45	6	99564.22	1	1	1	108872.45	1
226	15699029	Bagley	670	France	Male	37	4	170557.91	2	1	0	198252.88	0

Image 14: Oracle Churn_Modelling Table Data

The library that we installed and used this time is the cx_Oracle library. Notice that this time we did not define any new database but we used the “HR” database name because the Oracle’s files and databases live on the generic ‘Orcl’ node database with the credentials we are using. Although Oracle uses PL-SQL instead of T-SQL for our queries we are not going to see many differences.

```
# importing module
import cx_Oracle
from datetime import datetime

con = cx_Oracle.connect('hr/hr@localhost/orcl')



#Query No#1
start=datetime.now()

c = con.cursor()
c.execute('SELECT COUNT(CustomerID) AS Number_of_Exited_Customers  FROM churn_modelling WHERE Exited=1' ) #
for result in c:
    print (result) # this only shows the first two columns. To add an additional column you'll need to add


print("-----------------------------------")
print ('The time for Query No#1 is %s' % (datetime.now()-start))
print("-----------------------------------")

c.close()
```

Image 15: Oracle connector code

If we observe the Python code for the cx_Oracle library we can see that we are opening and closing each connection before and after each query execution respectively and this is because the Oracle consumes a lot of memory not only for just storing the data and living as a service but also during execution time. For the other databases we did not chose to do that because the impact on the final time is minor.

```
In [126]: runfile('C:/Users/30694/Oracle_sc
Users/30694')
(2037,)
-----------------------------------
The time for Query No#1 is 0:00:00.003989
-----------------------------------
```

Image 16: Oracle code results

The last database that we are going to use is the PostgreSQL database 10.12 and similarly to what we did for our first two databases we name the database as “ITC6107A1” and the table “Churn_Modeling”. The database’s management program is the pgAdmin 4 which stands for Postgre-Sql Admin and it is a web based dashboard for creating and managing the database. Unfortunately pgAdmin does not provide any automatic Excel File Import wizard so we had to manually created the table’s schema and import the data using a bulk-insert wizard.

The result of the insert operation is the following:

rownumber bigint	customerid [PK] bigint	surname text	creditscore bigint	geography text	gender text	age bigint	tenure bigint	balance double precision	numofproducts bigint	hascrcard bigint	isactivem bigint
1	15634602	Hargrave	619	France	Female	42	2	0	1	1	
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	
4	15701354	Boni	699	France	Female	39	1	0	2	0	
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	
7	15592531	Bartlett	822	France	Male	50	7	0	2	1	
8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	
9	15792365	He	501	France	Male	44	4	142051.07	2	0	
10	15592389	H?	684	France	Male	27	2	134603.88	1	1	
11	15767821	Bearce	528	France	Male	31	6	102016.72	2	0	

Image 17: PostgreSQL Churn_Modeling table

All the column data types had to be chosen by the database administrator so the bigint and text types were chose very carefully. Of course we can confirm with the following image that we chose the same names for convenience as we did before.

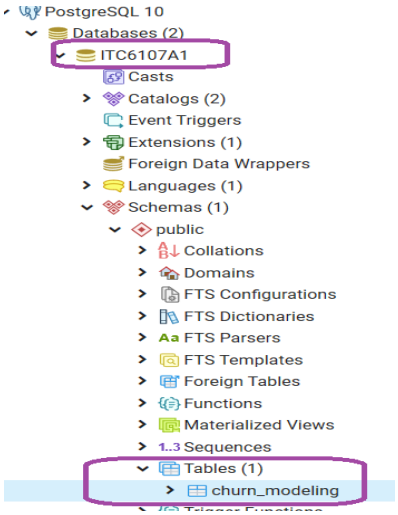

Image 18: PostgreSQL 10 Menu

The library that we used to communicate via Python was psycopg2 and the code format mirrors the previous script we saw. This time are targeting the database "ITC6107A1" and we are defining the username, the password, the post and the port that the connector we need to use in order to exchange information with the database.

```
import psycopg2
from datetime import datetime
connection = psycopg2.connect(database="ITC6107A1", user="sysadmin", password="Bnft!123", host="127.0.0.1", port=5432)



#Query No#1
start=datetime.now()

c = connection.cursor()
c.execute('SELECT COUNT(customerid) AS Number_of_exited_Customers  FROM churn_modeling WHERE exited=1 ') # use triple qu
for result in c:
    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2]


print("-------------------------------------")
print ('The time for Query No#1 is %s' % (datetime.now()-start))
print("-------------------------------------")

c.close()
```

Image 19: Postgre SQL Connector Code
The query results are are always the same.

```
In [129]: runfile('C:/Users/30694/Postgre_sql.py',
(2037,)
-------------------------------------
The time for Query No#1 is 0:00:00.014179
```

Image 20: PostgreSQL Connector Code results

Last but not least we are going to see the setup and the configuration of the Spark Framework. Since our machine consists of a Windows 10 machine with an i7 core 10th generation we need to simply download Spark from the appropriate link (https://spark.apache.org/downloads.html) and extract the contents of the installation folder:

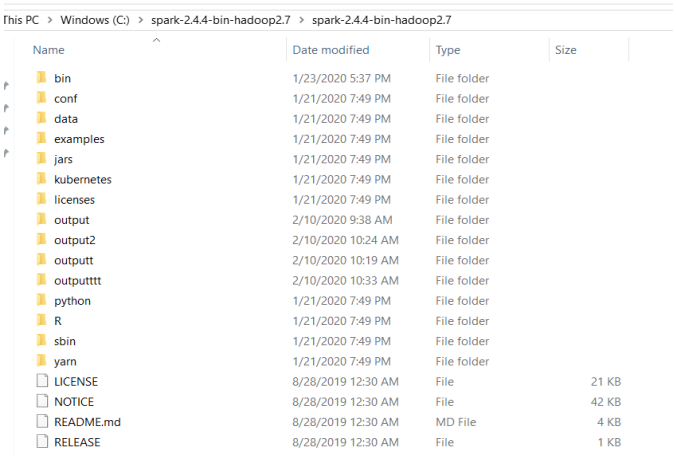

Image 21: Apache Spark Installation

Then we simply need to Edit our Enviroment variables so that Spark Framework can be ready for usage. First we make sure that the %SPARK_HOME% installation is located in our system variables along with the appropriate JDK Java installation.

And then we define as

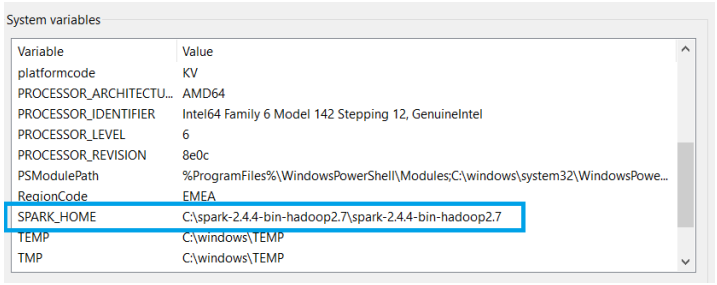

Image 22: SPARK_HOME System Variable

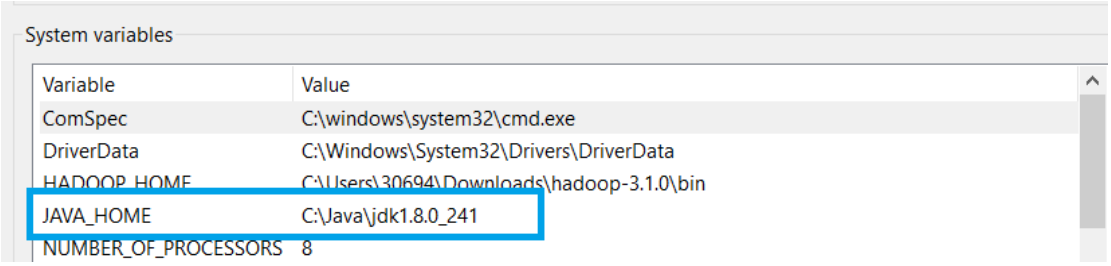

Image 23: JAVA_HOME System Variable

And then we add only the bin folder of our %SPARK_HOME% variable to our system path:

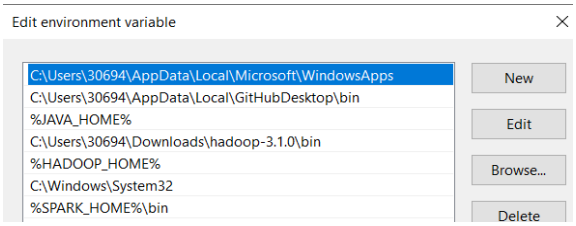

Image 24: System Path Variables

Then for our Query Execution we will install and use the Pyspark Library so that we can write Spark-Sql from our Python Intepreter. There are many possible variations but for this project we are going to use only two. In the first variation we are going to utilize one core and in the second variation we will utilize all the available cores which are eight (8).

The code that we will use to run SQL queries through Spark framework is the following:

```
import sys
from pyspark import SparkContext,SparkConf
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import pandas as pd


df = pd.read_csv("C:\\Users\\30694\\Downloads\\Churn_Modelling.csv")

def kill_current_spark_context():
    SparkContext.getOrCreate().stop()

kill_current_spark_context()
sc = SparkContext('local[*]', 'FinalProject')

spark=SparkSession.builder.appName("spark sql test2").getOrCreate()

spark_df = spark.createDataFrame(df)

spark_df.registerTempTable("Table")


# Query No# 1
start=datetime.now()
query_result=spark.sql("SELECT COUNT(CustomerID) AS  ExitedCustomers FROM Table WHERE Exited=1 ").show()

# query_result.show()

print ('The time for Query No#1 is %s' % (datetime.now()-start))
```

Image 25: Apache Spark Code

The main idea is the same with the previous scripts when we were using databases but in this scenario we simply define a Spark Context instead of a connection string so that we can start and have a reference variable for the Spark Framework. Once we have done that we define function which simply stops all the previously opened Spark Session, we run it and we create a new session. Then we create a Spark Dataframe from our Python Dataframe and we load that data in an temporary table which lives on our machine's memory. That temporary table will store temporary the data which will be queried at a later stage.

```
In [133]: runfile('C:/Users/30694/ApacheSparkSQL.py',
+---------------+
|ExitedCustomers|
+---------------+
|           2037|
+---------------+
```

Image 26: Apache Spark Code Results

Queries

Since we have already discussed about the Spark Framework and the different databases that we will query, let's see the SQL Code of those 20 queries. We need to use complex group-by queries along with aggregate functions. Some of the queries use only one aggregate functions and some other use some more complex code like multiple aggregate functions with multiple sql conditions.

As we will see below not all the queries are complex in the same way nor they a contain the group by statement or the same name of aggregate functions. The variety of using queries of medium complexity in the beginning and then increasing the difficulty, will give us a better understanding of how each database and framework responds to each query(easy,medium,hard).

-- Query 1 count how many customers left from the bank

SELECT COUNT(CustomerID) AS Number_of_Exited_Customers FROM churn_modelling WHERE Exited=1

In the first query we are using simply the Count aggregate function, of course we could have added a Group By function but this would have resulted in a very big list of ashes and this is not something aesthetic for the first query which was used as a reference example in the previous chapter.

-- Query 2 Get Minimum,Maximum Average Customer Salary

SELECT AVG(EstimatedSalary) As Average_Estimated_Customer_Salary,MIN(EstimatedSalary) As Minimum_Estimated_Customer_Salary ,MAX(EstimatedSalary) As Maximum_Estimated_Customer_Salary FROM churn_modelling

For the Second Query we want to find some basic statistics for the Estimated Salary because all this aggregate functions together will "slow" the databases' engine response.

-- Query 3 Count Active Members who exited

SELECT COUNT(CustomerID) FROM churn_modelling WHERE Exited=1 AND IsActiveMember=1

In the third query we use the count aggregate function along with two conditions with purpose to make the query a little bit more complex than the first one.

--Query 4 Male/Female Percentage

SELECT (COUNT(CustomerID)*100/(SELECT COUNT(*) FROM churn_modelling)) AS Total_Male_Percentage FROM churn_modelling WHERE Gender='Male'

For the fourth query we are using a combination of a division of an aggregate function with a select statement. The result will produce a 'heavier' query not only because of the complex calculations but also because we added a where condition

--Query 5 Count people between 20 and 40

SELECT COUNT (CustomerId) AS COUNTERSUM FROM churn_modelling GROUP BY Age,CustomerId HAVING Age BETWEEN 20 AND 40 ;

For the fifth query we count the number of people between a specific age. We combine a group by query with a Having statement with purpose to filter the age range we are looking for.

-- Query 6 Count Customers from Spain and France

SELECT COUNT(CustomerId), Geography FROM churn_modelling WHERE Geography='Spain' or Geography='France' GROUP BY Geography

For the sixth query we use another group by query but in this case we have a double where query because we are using the word 'or', note that we still use the count aggregate function like the previous query.

-- Query 7 Select Last NAmes of Customers who are young, active member and also have credit card

SELECT CustomerId, Surname from churn_modelling WHERE NOT (HasCrCard=0 AND IsActiveMember=0) AND (Age<40) GROUP BY Surname,CustomerId ORDER BY Surname DESC

For the query number 7 we are using a combination of a Group-Order by statement, generally the Order By increases the response time of the query.

-- Query 8 Get All Customers whos name begins with A and are located in France/Germany

SELECT CustomerId,Surname, Geography FROM churn_modelling WHERE Surname LIKE 'A%' AND (Geography IN('Germany','France')) GROUP BY CustomerId,Surname,Geography ORDER BY CustomerId, Surname DESC,Geography DESC

For the query number 8 we added more columns on the Order By statement so that will increase even more the response time. Not that used also for the first time the LIKE SQL command which filters the data which are followed after the IN statement.

-- Query 9 Get All Customer with Tenure either 2,4,6,8 located in Spain/Germany and there age is only 20 or 30 or 40

SELECT CustomerId,Surname FROM churn_modelling WHERE Tenure IN ('2' ,'4' ,'6' ,'8') AND Geography IN('Spain' ,'Germany') AND AGE IN ('20', '30','40') GROUP BY CustomerId,Surname,Tenure,Geography,Age ORDER BY CustomerId,Surname,Tenure,Geography,Age

The query number 9 is one of the slow ones because as we can see it contains many values for the Where clause and also for the Group-Order By statements.

-- Query 10 Count Spanish Male with Credit Score over 500

SELECT COUNT(CustomerId) FROM churn_modelling WHERE (Gender='Male' AND Geography='Spain') OR(CreditScore>500)

For the Query number 10 we can see again the Count aggregate function but this time with a more complicated where clause. To be more specific we use 2 logical operator the 'AND' the 'OR' word so that the query can search even more within the data.

-- Query 11 Get Customers Over 30 with High Balance and Numb of Products 3 or 4

SELECT CustomerId,NumOfProducts FROM churn_modelling WHERE (Balance>150000 AND Age>30) OR (NumOfProducts IN('3' ,'4')) GROUP BY CustomerId,NumOfProducts

The eleventh query contains a three conditional where clause along with two columns in the group by sections, this also creates a mixed query that will take also a lot of time to be executed because we have the 'OR' logical operator which will fetch even more data.

-- Query 12 Count Customers with Credit Score over 500

SELECT Age,CreditScore FROM churn_modelling WHERE CreditScore>500 GROUP BY Age, CreditScore HAVING Age>(Select AVG(Age) FROM churn_modelling)

For the query number 12 we have added a having statement that compares the age attribute with a whole select statement.

-- Query 13 Min,Max,Avg for grouped Balance,Age,CreditScore,Tenure

SELECT MIN (Balance) AS Minimum_Balance,MAX (Balance) AS Maximum_Balance, AVG (Balance) AS Average_Balance, MIN (Age) AS Minimum_Age,MAX (Age) AS Maximum_Age, AVG (Age) AS Average_Age , MIN (CreditScore) AS Minimum_CreditScore,MAX (CreditScore) AS Maximum_CreditScore, AVG (CreditScore) AS Average_CreditScore , MIN (Tenure) AS Minimum_Tenure,MAX (Tenure) AS Maximum_Tenure, AVG (Tenure) AS Average_Tenure FROM churn_modelling GROUP BY Balance,Age,CreditScore,Tenure

The query number 13 is the longest one because we use many aggregate functions for many columns in our table, so it is the query that takes the longest time to be executed and it contains the most words.

-- Query 14 Get Geography Balance Per Customer

SELECT Geography,Balance,sum(Balance) OVER(PARTITION BY Geography) AS TotalBalPerRegion FROM churn_modelling GROUP BY Geography, Balance

The query number 14 contains a group by clause with the Over Partition By sql command and this help us to create a better grouping over a particular set of rows. The results will be to split the table into geographical regions and then fetch that data to our group by clause.

-- Query 15 Get Customers whose Salary is Greater than Average

SELECT CustomerId,EstimatedSalary FROM churn_modelling GROUP BY CustomerId,EstimatedSalary HAVING EstimatedSalary >(SELECT AVG(EstimatedSalary) FROM churn_modelling)

Similarly to query 12 we have the Having statement in our query but this time we compare our attribute with a whole sql query that contains also an aggregate function.

-- Query 16 Get Customer whose Balance is Greater than Average

SELECT CustomerId, Surname FROM churn_modelling GROUP BY CustomerId,Surname,Balance,CreditScore HAVING Balance >(SELECT AVG(Balance)FROM churn_modelling) AND CreditScore >(SELECT AVG(CreditScore)FROM churn_modelling)

For the query number 16 again we use many columns in our group by clause and we filter them by using a having command along with a comparison of a single column with another sql query

-- Query 17 Count Customer per Tenure group

SELECT CustomerId, Tenure, count(*) FROM churn_modelling GROUP BY GROUPING SETS ((CustomerId), (Tenure))

For the query number 17 we are replacing our classical group by sql command with the command group by grouping sets, which results into returning to grouped lists for the columns that are contained in the parentheses.

-- Query 18 Get Customers whose total Profit is greater than a Million

SELECT CustomerId, SUM(Balance+EstimatedSalary) as [TotalMoney] FROM churn_modelling GROUP BY CustomerId HAVING SUM(Balance+EstimatedSalary)> 100000 ORDER BY TotalMoney DESC;

The query number 18 contains a SUM function which adds the values of two other columns, this combined with the order by sql command will result also in a complex query.

-- Query 19 Tenure Percentage

select Tenure, count(*) as total_tenure, count(*) * 1.0 / sum(count(*)) over () as ratio from churn_modelling group by Tenure ;

For the last before the end query we can see a group by statement which contains a count aggregate functions along with a statistical division of two other functions; the count and the sum function. Notice also the word over which filters the data over a set of rows and does a more elastic search otherwise the percentage would not have been calculated.

-- Query 20 Get Customers with Credit Score greater than average

SELECT CustomerId FROM churn_modelling WHERE CreditScore> (SELECT AVG(CreditScore) FROM churn_modelling)

The final query contains a where clause followed by a whole select statement, that slowers the execution time because the engine will execute first a second select query within the main one.

Statistical Analysis

In this part we are going to see the final results of all those queries that we mentioned before. The analytical SQL code for each database can be found in the Appendix of this assignment. In the below image we can see the database comparison in seconds for each specific query. As we can see the Postgre-SQL database is the slowest one where as the SQL Sever is the fastest.

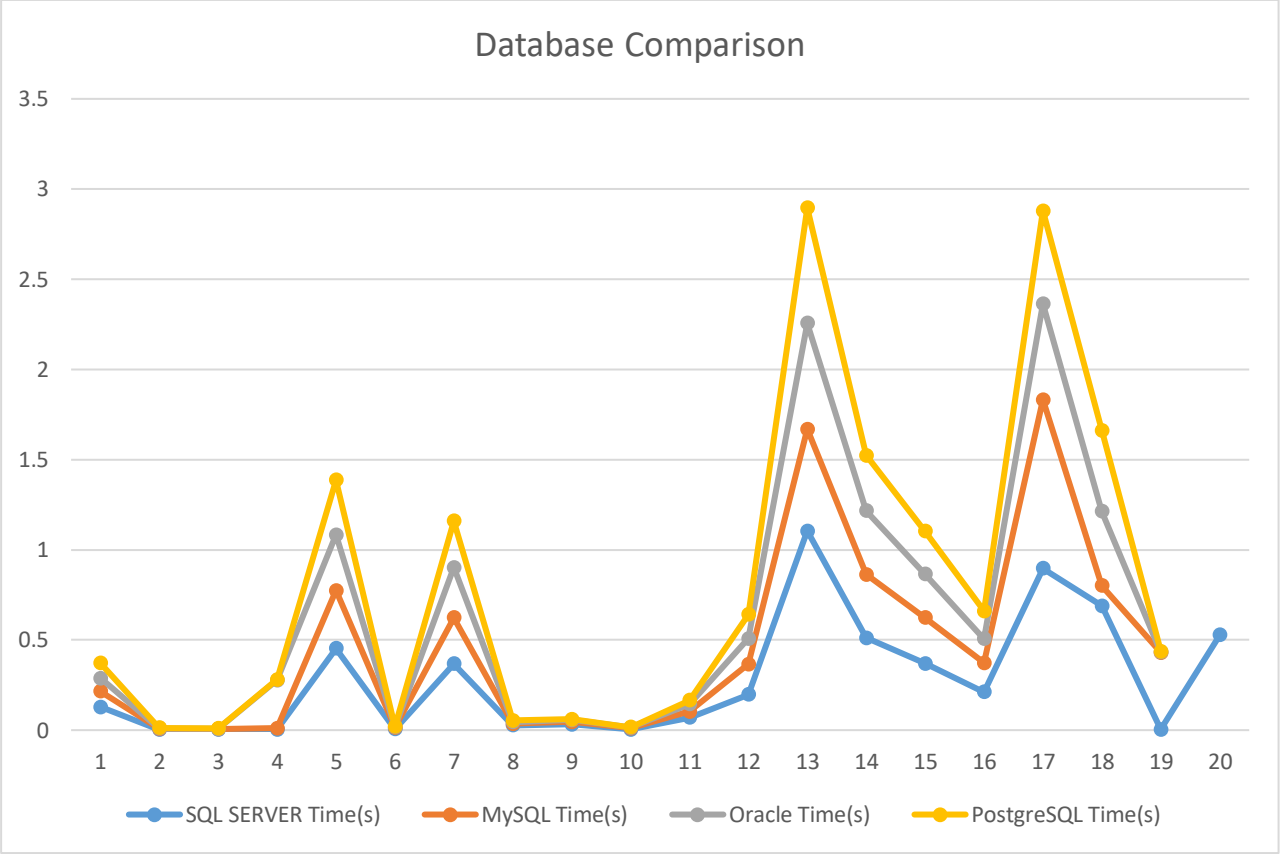

Image 27: Database Comparison Graph

Now let's put the Spark Framework in the previous graph and see how well it did:

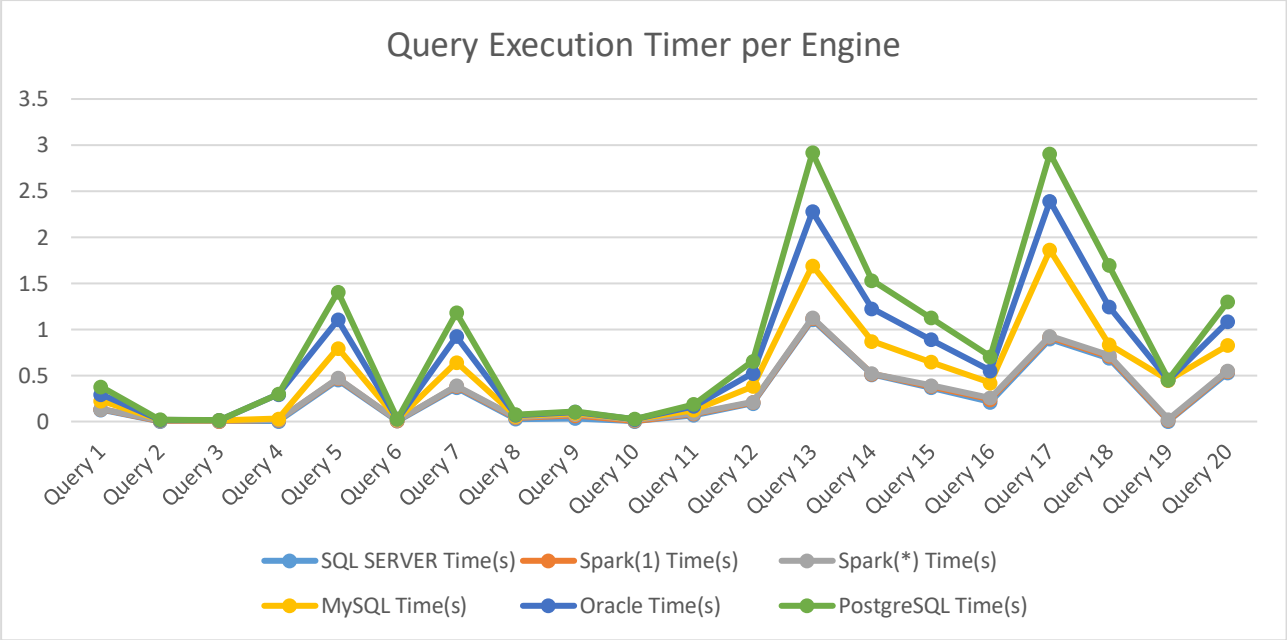

Image 28: Query Execution Time per Engine Graph

We can see very easily that in all the queries the Apache Spark Framework was the fastest (red and grey line) where as the other data storage systems did not perform so well. Of course there are some queries that were not extremely complex so the lines might overlap, so this is also another proof that for the very complex queries the time difference is significant for instance Queries 4-7 and 12-18. For more details for each specific query that overlap for example Queries 2 and 3 we have noted the time for all the databases and the Spark Framework for each specific query:

Queries	SQL SERVER Time(s)	Spark(1) Time(s)	Spark(*) Time(s)	MySQL Time(s)	Oracle Time(s)	PostgreSQL Time(s)
Query 1	0.127657	0.006479	0.001996	0.087764	0.071808	0.083787
Query 2	0.001995	0.004404	0.005163	0.004988	0.001993	0.002981
Query 3	0.001995	0.002017	0.003992	0.003988	0.001996	0.001962
Query 4	0.002991	0.012604	0.006065	0.007979	0.26686	0.001994
Query 5	0.452605	0.012021	0.008639	0.321646	0.30868	0.303162
Query 6	0.004988	0.003641	0.005195	0.008967	0.002958	0.002992
Query 7	0.36709	0.014535	0.00606	0.256336	0.279057	0.256818
Query 8	0.025904	0.014598	0.008779	0.009969	0.008976	0.007978
Query 9	0.032912	0.031357	0.014396	0.012964	0.01097	0.002992
Query 10	0.003989	0.002027	0.009546	0.00698	0.002992	0.001959
Query 11	0.068816	0.013921	0.007816	0.034905	0.039892	0.021941
Query 12	0.196292	0.010943	0.004437	0.169546	0.142619	0.131648
Query 13	1.104802	0.013112	0.009166	0.564542	0.585813	0.640084
Query 14	0.511632	0.005591	0.004598	0.350639	0.353562	0.306751
Query 15	0.369012	0.013273	0.009892	0.253833	0.241354	0.23787
Query 16	0.211435	0.02822	0.018125	0.162051	0.131647	0.154586
Query 17	0.895978	0.021214	0.008369	0.936071	0.531306	0.514897
Query 18	0.689125	0.018209	0.016401	0.111689	0.4109	0.449451
Query 19	0.003989	0.009717	0.009625	0.425121	0.003024	0.003988
Query 20	0.526575	0.018967	0.008166	0.27327	0.254867	0.217923

We have already proved that the Apache Spark Framework is way faster than the rest databases when it comes to query processing. What is left to compare now is to see if the number of cores in our machine play any significant role in the execution time.

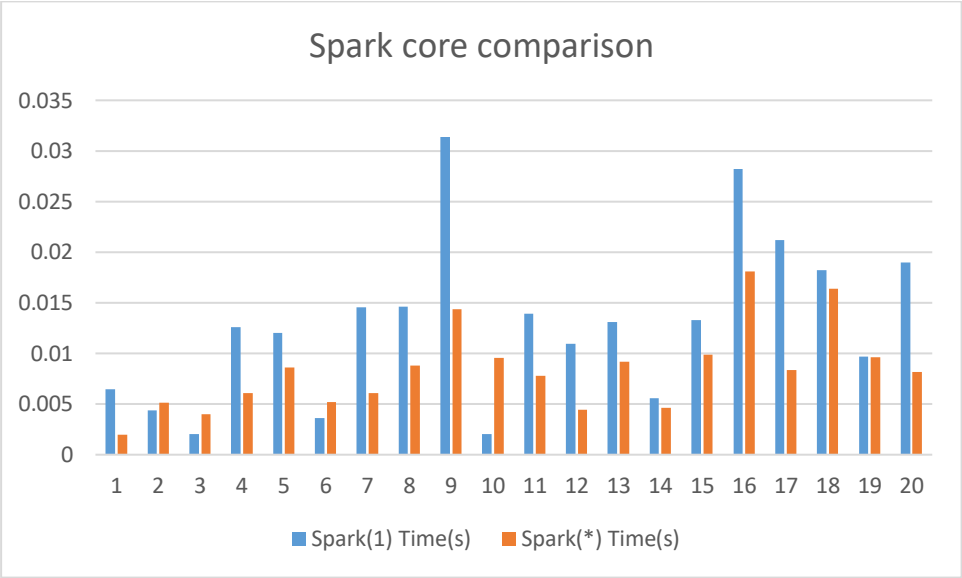

Image 29: Spark Core Comparison

As we can see, utilizing all the available cores in our machine and making them available to the spark frameworks makes it very fast. The orange columns which stand for the multi-core utilization are almost half the size of the blue one which stand for the single-core utilization, so we can conclude that indeed the Spark Framework is very fast compared to other data storage systems but the Spark Framework with multi-core utilization is fast in the scale of microsecond!

Future Work

All in all this assignment simply compared how the different databases and the Spark Framework behave on complex queries. This assignment could be easily extended by examining even more complex SQL structures like stored procedures or functions. It would be also interesting to see how a powerful business server would have behaved with these queries or even a very strong GPU instead of simply CPU.

References

L. Perkins (2018) Seven Databases in Seven Weeks, 2nd edition, The Pragmatic Bookshelf

L. Weise (2015), Advanced Data Management, de Gruyter

J. Lescovec, A. Rajaraman, J.D. Ullman (2014), Mining of Massive Datasets, Cambridge University Press

Apendix

SQL SERVER Queries

```
from datetime import datetime

import pyodbc

conn = pyodbc.connect('Driver={SQL Server};'

                      'Server=LAPTOP-4QCKGR44\MSSQLSERVER2012;'

                      'Database=ITC6107a1;'

                      'Trusted_Connection=yes;')

cursor = conn.cursor()

#Query No#1

start=datetime.now()

cursor.execute('SELECT COUNT(CustomerId) AS [Number of Exited Customers]  FROM
ITC6107A1.dbo.Churn_Modelling$ WHERE Exited=1 ')

for row in cursor:

   print(row)

print("-----------------------------------")

print ('The time for Query No#1 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#2

start=datetime.now()

cursor.execute('SELECT AVG(EstimatedSalary) As [Average Estimated Customer
Salary],MIN(EstimatedSalary) As [Minimum Estimated Customer Salary]
,MAX(EstimatedSalary) As [Maximum Estimated Customer Salary]  FROM
ITC6107A1.dbo.Churn_Modelling$')

for row in cursor:

   print(row)

print("-----------------------------------")

print ('The time for Query No#2 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#3

start=datetime.now()

cursor.execute('SELECT COUNT(CustomerID) FROM ITC6107A1.dbo.Churn_Modelling$
WHERE Exited=1 AND IsActiveMember=1 ')

for row in cursor:

   print(row)


print("-----------------------------------")

print ('The time for Query No#3 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#4

start=datetime.now()

cursor.execute(""" SELECT  (COUNT(CustomerID)*100/(SELECT COUNT(*) FROM
ITC6107A1.dbo.Churn_Modelling$ ))  AS [Total Male Percentage] FROM
ITC6107A1.dbo.Churn_Modelling$ WHERE Gender='Male'""")

for row in cursor:

   print(row)

  print("-----------------------------------")

print ('The time for Query No#4 is %s' % (datetime.now()-start))

print("-----------------------------------")


#Query No#5

start=datetime.now()

cursor.execute('SELECT COUNT (CustomerId) AS [COUNTERSUM] FROM
ITC6107A1.dbo.Churn_Modelling$ GROUP BY Age,CustomerId HAVING Age BETWEEN 20
AND 40 ')

for row in cursor:

   print(row)


print("-----------------------------------")

print ('The time for Query No#5 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#6

start=datetime.now()

cursor.execute("""SELECT COUNT(CustomerId), Geography  FROM
ITC6107A1.dbo.Churn_Modelling$  WHERE Geography='Spain' or Geography='France'
GROUP BY Geography""")

for row in cursor:
   print(row)

print("-------------------------------------")
print ('The time for Query No#6 is %s' % (datetime.now()-start))
print("-------------------------------------")


#Query No#7

start=datetime.now()

cursor.execute('SELECT CustomerId, Surname from ITC6107A1.dbo.Churn_Modelling$
WHERE NOT ( HasCrCard=0 AND IsActiveMember=0) AND (Age<40) GROUP BY
Surname,CustomerId ORDER BY Surname DESC')


for row in cursor:

   print(row)

 print("-------------------------------------")

print ('The time for Query No#7 is %s' % (datetime.now()-start))

print("-------------------------------------")

#Query No#8

start=datetime.now()

cursor.execute("""SELECT CustomerId,Surname, Geography FROM
ITC6107A1.dbo.Churn_Modelling$ WHERE Surname LIKE  'A%' AND (Geography
IN('Germany','France')) GROUP BY CustomerId,Surname,Geography ORDER BY CustomerId,
Surname DESC,Geography DESC""")

for row in cursor:

   print(row)


print("-------------------------------------")

print ('The time for Query No#8 is %s' % (datetime.now()-start))

print("-------------------------------------")
```

```
#Query No#9

start=datetime.now()

cursor.execute(""" SELECT CustomerId,Surname FROM ITC6107A1.dbo.Churn_Modelling$
WHERE Tenure IN ('2' ,'4' ,'6' ,'8') AND Geography IN('Spain' ,'Germany') AND AGE IN ( '20',
'30','40') GROUP BY CustomerId,Surname,Tenure,Geography,Age ORDER BY
CustomerId,Surname,Tenure,Geography,Age""")


for row in cursor:

   print(row)


print("-----------------------------------")

print ('The time for Query No#9 is %s' % (datetime.now()-start))

print("-----------------------------------")


#Query No#10

start=datetime.now()

cursor.execute("""SELECT COUNT(CustomerId) FROM ITC6107A1.dbo.Churn_Modelling$
WHERE (Gender='Male' AND Geography='Spain') OR(CreditScore>500)""")

for row in cursor:

   print(row)

print("-----------------------------------")

print ('The time for Query No#10 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#11

start=datetime.now()

cursor.execute(""" SELECT CustomerId,NumOfProducts FROM
ITC6107A1.dbo.Churn_Modelling$  WHERE (Balance>150000 AND Age>30) OR (
NumOfProducts  IN( '3' ,'4')) GROUP BY CustomerId,NumOfProducts  """)


for row in cursor:

   print(row)
```

```
print("-----------------------------------")

print ('The time for Query No#11 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#12

start=datetime.now()

cursor.execute('SELECT Age,CreditScore FROM ITC6107A1.dbo.Churn_Modelling$  WHERE
CreditScore>500 GROUP BY Age, CreditScore  HAVING Age>(Select AVG(Age) FROM
ITC6107A1.dbo.Churn_Modelling$ )')


for row in cursor:

   print(row)
```

```
print("-----------------------------------")

print ('The time for Query No#12 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#13

start=datetime.now()

cursor.execute('SELECT MIN (Balance) AS [Minimum Balance],MAX (Balance) AS [Maximum
Balance],   AVG (Balance) AS [Average Balance], MIN (Age) AS [Minimum Age],MAX (Age) AS
[Maximum Age],   AVG (Age) AS [Average Age]  ,MIN (CreditScore) AS [Minimum
CreditScore],MAX (CreditScore) AS [Maximum CreditScore],   AVG (CreditScore) AS [Average
CreditScore]  , MIN (Tenure) AS [Minimum Tenure],MAX (Tenure) AS [Maximum Tenure],
AVG (Tenure) AS [Average Tenure]  FROM ITC6107A1.dbo.Churn_Modelling$ GROUP BY
Balance,Age,CreditScore,Tenure')


for row in cursor:

   print(row)
```

```
print("-----------------------------------")

print ('The time for Query No#13 is %s' % (datetime.now()-start))

print("-----------------------------------")




#Query No#14

start=datetime.now()

cursor.execute('SELECT Geography,Balance,sum(Balance ) OVER(PARTITION BY Geography )
AS TotalBalPerRegion FROM ITC6107A1.dbo.Churn_Modelling$ GROUP BY Geography,
Balance')


for row in cursor:

   print(row)



print("-----------------------------------")

print ('The time for Query No#14 is %s' % (datetime.now()-start))

print("-----------------------------------")

Query No#15

start=datetime.now()

cursor.execute('SELECT CustomerId,EstimatedSalary FROM
ITC6107A1.dbo.Churn_Modelling$ GROUP BY CustomerId,EstimatedSalary  HAVING
EstimatedSalary >(SELECT  AVG(EstimatedSalary ) FROM ITC6107A1.dbo.Churn_Modelling$
)')


for row in cursor:

   print(row)



print("-----------------------------------")

print ('The time for Query No#15 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#16

start=datetime.now()

cursor.execute('SELECT CustomerId, Surname FROM ITC6107A1.dbo.Churn_Modelling$
GROUP BY CustomerId,Surname,Balance,CreditScore HAVING Balance >(SELECT
AVG(Balance )FROM ITC6107A1.dbo.Churn_Modelling$ ) AND CreditScore >(SELECT
AVG(CreditScore )FROM ITC6107A1.dbo.Churn_Modelling$ )')


for row in cursor:

   print(row)



print("-----------------------------------")

print ('The time for Query No#16 is %s' % (datetime.now()-start))

print("-----------------------------------")




#Query No#17

start=datetime.now()

cursor.execute('SELECT CustomerId, Tenure, count(*) FROM
ITC6107A1.dbo.Churn_Modelling$ GROUP BY GROUPING SETS (   ( CustomerId),  (Tenure))')


for row in cursor:

   print(row)



print("-----------------------------------")

print ('The time for Query No#17 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#18

start=datetime.now()

cursor.execute('SELECT CustomerId,  SUM(Balance+EstimatedSalary) as [TotalMoney] FROM
ITC6107A1.dbo.Churn_Modelling$ GROUP BY CustomerId HAVING
SUM(Balance+EstimatedSalary)> 100000 ORDER BY TotalMoney DESC;')

for row in cursor:

   print(row)


print("-----------------------------------")

print ('The time for Query No#18 is %s' % (datetime.now()-start))

print("-----------------------------------")


#Query No#19

start=datetime.now()

cursor.execute('select [Tenure], count(*) as total_tenure,  count(*) * 1.0 / sum(count(*))
over () as ratio from ITC6107A1.dbo.Churn_Modelling$ group by [Tenure] ;')

for row in cursor:

   print(row)


print("-----------------------------------")

print ('The time for Query No#19 is %s' % (datetime.now()-start))

print("-----------------------------------")


#Query No#20

start=datetime.now()

cursor.execute('SELECT CustomerId FROM ITC6107A1.dbo.Churn_Modelling$ WHERE
CreditScore> (SELECT AVG(CreditScore)  FROM ITC6107A1.dbo.Churn_Modelling$  )')

# for row in cursor:

#     print(row)

print("-----------------------------------")

print ('The time for Query No#20 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

Oracle Queries

```
# importing module

import cx_Oracle

from datetime import datetime

con = cx_Oracle.connect('hr/hr@localhost/orcl')

 #Query No#1

start=datetime.now()

c = con.cursor()

c.execute('SELECT COUNT(CustomerID) AS Number_of_Exited_Customers  FROM
churn_modelling WHERE Exited=1' ) # use triple quotes if you want to spread your query
across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.


print("-------------------------------------")

print ('The time for Query No#1 is %s' % (datetime.now()-start))

print("-------------------------------------")

c.close()

#Query No#2

start=datetime.now()

c = con.cursor()

c.execute("SELECT AVG(EstimatedSalary) As
Average_Est_Customer_Salary,MIN(EstimatedSalary) As Minimum_Est_Customer_Salary
,MAX(EstimatedSalary) As Maximum_Est_Customer_Salary  FROM churn_modelling") # use
triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.


print("-------------------------------------")

print ('The time for Query No#2 is %s' % (datetime.now()-start))

print("-------------------------------------")
```

```
#Query No#3

start=datetime.now()

c = con.cursor()

c.execute('SELECT COUNT(CustomerID) FROM churn_modelling WHERE Exited=1 AND
IsActiveMember=1 ') # use triple quotes if you want to spread your query across multiple
lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#3 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#4

start=datetime.now()

c = con.cursor()

c.execute(" SELECT  (COUNT(CustomerID)*100/(SELECT COUNT(*) FROM churn_modelling ))
AS Total_Male_Percentage FROM churn_modelling WHERE Gender='Male' GROUP BY
CUSTOMERID") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.

 print("-----------------------------------")

print ('The time for Query No#4 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#5

start=datetime.now()


c = con.cursor()

c.execute('SELECT COUNT (CustomerId) AS COUNTERSUM FROM churn_modelling GROUP BY Age,CustomerId HAVING Age BETWEEN 20 AND 40') # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#5 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#6

start=datetime.now()

c = con.cursor()

c.execute("SELECT COUNT(CustomerId), Geography  FROM churn_modelling  WHERE Geography='Spain' or Geography='France' GROUP BY Geography") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#6 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#7

start=datetime.now()


c = con.cursor()

c.execute("SELECT CustomerId, Surname from churn_modelling  WHERE NOT ( HasCrCard=0 AND IsActiveMember=0) AND (Age<40) GROUP BY Surname,CustomerId ORDER BY Surname DESC") # use triple quotes if you want to spread your query across multiple lines

for result in c:
```

```
    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#7 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#8

start=datetime.now()

c = con.cursor()

c.execute("SELECT CustomerId,Surname, Geography FROM churn_modelling WHERE Surname LIKE  'A%' AND (Geography IN('Germany','France')) GROUP BY CustomerId,Surname,Geography ORDER BY CustomerId, Surname DESC,Geography DESC") # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#8 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#9

start=datetime.now()

c = con.cursor()

c.execute("SELECT CustomerId,Surname FROM churn_modelling WHERE Tenure IN ('2' ,'4' ,'6' ,'8') AND Geography IN('Spain' ,'Germany') AND AGE IN ( '20', '30','40') GROUP BY CustomerId,Surname,Tenure,Geography,Age ORDER BY CustomerId,Surname,Tenure,Geography,Age") # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#9 is %s' % (datetime.now()-start))
```

```
print("-----------------------------------")


Query No#10

start=datetime.now()


c = con.cursor()

c.execute("SELECT COUNT(CustomerId) FROM churn_modelling  WHERE (Gender='Male' AND Geography='Spain') OR(CreditScore>500)") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#10 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#11

start=datetime.now()

c = con.cursor()

c.execute("SELECT CustomerId,NumOfProducts FROM churn_modelling  WHERE (Balance>150000 AND Age>30) OR ( NumOfProducts  IN( '3' ,'4')) GROUP BY CustomerId,NumOfProducts ") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#11 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

```
#Query No#12

start=datetime.now()

c = con.cursor()

c.execute('SELECT Age,CreditScore FROM churn_modelling  WHERE CreditScore>500 GROUP BY Age, CreditScore  HAVING Age>(Select AVG(Age) FROM churn_modelling )') # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#12 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#13

start=datetime.now()

c = con.cursor()

c.execute('SELECT MIN (Balance) AS Minimum_Balance,MAX (Balance) AS Maximum_Balance,   AVG (Balance) AS Average_Balance, MIN (Age) AS Minimum_Age,MAX (Age) AS Maximum_Age,   AVG (Age) AS Average_Age  , MIN (CreditScore) AS Minimum_CreditScore,MAX (CreditScore) AS Maximum_CreditScore,   AVG (CreditScore) AS Average_CreditScore  , MIN (Tenure) AS Minimum_Tenure,MAX (Tenure) AS Maximum_Tenure,   AVG (Tenure) AS Average_Tenure   FROM churn_modelling GROUP BY Balance,Age,CreditScore,Tenure') # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#13 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#14

start=datetime.now()

c = con.cursor()

c.execute('SELECT Geography,Balance,sum(Balance ) OVER(PARTITION BY Geography ) AS TotalBalPerRegion FROM churn_modelling GROUP BY Geography, Balance') # use triple quotes if you want to spread your query across multiple lines

for result in c:
```

```
    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#14 is %s' % (datetime.now()-start))

print("-----------------------------------")

Query No#15

start=datetime.now()


c = con.cursor()

c.execute('SELECT CustomerId,EstimatedSalary FROM churn_modelling GROUP BY CustomerId,EstimatedSalary  HAVING EstimatedSalary >(SELECT  AVG(EstimatedSalary ) FROM churn_modelling )') # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#15 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#16

start=datetime.now()

c = con.cursor()

c.execute('SELECT CustomerId, Surname FROM churn_modelling GROUP BY CustomerId,Surname,Balance,CreditScore HAVING Balance >(SELECT  AVG(Balance )FROM churn_modelling ) AND CreditScore >(SELECT  AVG(CreditScore )FROM churn_modelling )') # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#16 is %s' % (datetime.now()-start))
```

```
print("-----------------------------------")


#Query No#17

start=datetime.now()


c = con.cursor()

c.execute('SELECT CustomerId, Tenure, count(*) FROM churn_modelling GROUP BY
GROUPING SETS (  ( CustomerId),  (Tenure))') # use triple quotes if you want to spread your
query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.



print("-----------------------------------")

print ('The time for Query No#17 is %s' % (datetime.now()-start))

print("-----------------------------------")


Query No#18

start=datetime.now()


c = con.cursor()

c.execute('SELECT CustomerId,  SUM(Balance+EstimatedSalary) as TotalMoney FROM
churn_modelling GROUP BY CustomerId HAVING SUM(Balance+EstimatedSalary)> 100000
ORDER BY TotalMoney DESC') # use triple quotes if you want to spread your query across
multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.



print("-----------------------------------")

print ('The time for Query No#18 is %s' % (datetime.now()-start))
```

```
print("-----------------------------------")

#Query No#19

start=datetime.now()

c = con.cursor()

c.execute('select Tenure, count(*) as total_tenure,   count(*) * 1.0 / sum(count(*)) over () as ratio from churn_modelling group by Tenure ') # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.



print("-----------------------------------")

print ('The time for Query No#19 is %s' % (datetime.now()-start))

print("-----------------------------------")

#Query No#20

start=datetime.now()

c = con.cursor()

c.execute('SELECT CustomerId FROM churn_modelling WHERE CreditScore> (SELECT AVG(CreditScore)  FROM churn_modelling  )') # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.



print("-----------------------------------")

print ('The time for Query No#20 is %s' % (datetime.now()-start))

print("-----------------------------------")
```

MySQL Queries

```
from datetime import datetime

import mysql.connector

mydb = mysql.connector.connect(user='root', password='Bnft!123',

                    host='127.0.0.1', database='itc6107a1',

                    auth_plugin='mysql_native_password')

mycursor = mydb.cursor()

# Query No# 1

start=datetime.now()

mycursor.execute("SELECT COUNT(CustomerID) AS Number_of_Exited_Customers  FROM
churn_modelling WHERE Exited=1 ")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#1 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 2

start=datetime.now()

mycursor.execute("SELECT AVG(EstimatedSalary) As
Average_Estimated_Customer_Salary,MIN(EstimatedSalary) As
Minimum_Estimated_Customer_Salary ,MAX(EstimatedSalary) As
Maximum_Estimated_Customer_Salary  FROM churn_modelling")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#2 is %s' % (datetime.now()-start))

print("------------------------------------")
```

```
# Query No# 3

start=datetime.now()

mycursor.execute("SELECT COUNT(CustomerID) FROM churn_modelling WHERE Exited=1
AND IsActiveMember=1 ")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#3 is %s' % (datetime.now()-start))

print("------------------------------------")


# Query No# 4

start=datetime.now()

mycursor.execute("SELECT  (COUNT(CustomerID)*100/(SELECT COUNT(*) FROM
churn_modelling ))  AS Total_Male_Percentage FROM churn_modelling WHERE
Gender='Male'")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#4 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 5

start=datetime.now()

mycursor.execute("SELECT COUNT(CustomerId) AS COUNTERSUM FROM churn_modelling
GROUP BY Age,CustomerId HAVING Age BETWEEN 20 AND 40 ")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#5 is %s' % (datetime.now()-start))

print("------------------------------------")
```

```
# Query No# 6

start=datetime.now()

mycursor.execute("SELECT COUNT(CustomerId), Geography  FROM churn_modelling
WHERE Geography='Spain' or Geography='France' GROUP BY Geography")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#6 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 7

start=datetime.now()

mycursor.execute("SELECT CustomerId, Surname from churn_modelling  WHERE NOT (
HasCrCard=0 AND IsActiveMember=0) AND (Age<40) GROUP BY Surname,CustomerId
ORDER BY Surname DESC")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#7 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 8

start=datetime.now()

mycursor.execute("SELECT CustomerId,Surname, Geography FROM churn_modelling WHERE
Surname LIKE  'A%' AND (Geography IN('Germany','France')) GROUP BY
CustomerId,Surname,Geography ORDER BY CustomerId, Surname DESC,Geography DESC")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#8 is %s' % (datetime.now()-start))

print("------------------------------------")
```

```
# Query No# 9

start=datetime.now()

mycursor.execute("SELECT CustomerId,Surname FROM churn_modelling WHERE Tenure IN
('2' ,'4' ,'6' ,'8') AND Geography IN('Spain' ,'Germany') AND AGE IN ( '20', '30','40') GROUP BY
CustomerId,Surname,Tenure,Geography,Age ORDER BY
CustomerId,Surname,Tenure,Geography,Age")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#9 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 10

start=datetime.now()

mycursor.execute("SELECT COUNT(CustomerId) FROM churn_modelling  WHERE
(Gender='Male' AND Geography='Spain') OR(CreditScore>500)")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#10 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 11

start=datetime.now()

mycursor.execute("SELECT CustomerId,NumOfProducts FROM churn_modelling  WHERE
(Balance>150000 AND Age>30) OR ( NumOfProducts  IN( '3' ,'4')) GROUP BY
CustomerId,NumOfProducts ")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#11 is %s' % (datetime.now()-start))

print("------------------------------------")
```

```
# Query No# 12

start=datetime.now()

mycursor.execute("SELECT Age,CreditScore FROM churn_modelling  WHERE
CreditScore>500 GROUP BY Age, CreditScore  HAVING Age>(Select AVG(Age) FROM
churn_modelling )")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#12 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 13

start=datetime.now()

mycursor.execute("  SELECT MIN(Balance) AS Minimum_Balance,MAX(Balance) AS
Maximum_Balance,   AVG(Balance) AS Average_Balance, MIN(Age) AS
Minimum_Age,MAX(Age) AS Maximum_Age,   AVG(Age) AS Average_Age  , MIN(CreditScore)
AS Minimum_CreditScore,MAX(CreditScore) AS Maximum_CreditScore,   AVG(CreditScore)
AS Average_CreditScore  , MIN(Tenure) AS Minimum_Tenure,MAX(Tenure) AS
Maximum_Tenure,   AVG(Tenure) AS Average_Tenure   FROM churn_modelling GROUP BY
Balance,Age,CreditScore,Tenure")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")

print ('The time for Query No#13 is %s' % (datetime.now()-start))

print("------------------------------------")

# Query No# 14

start=datetime.now()

mycursor.execute("SELECT Geography,Balance,sum(Balance ) OVER(PARTITION BY
Geography ) AS TotalBalPerRegion FROM churn_modelling GROUP BY Geography, Balance")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("------------------------------------")
```

```
print ('The time for Query No#14 is %s' % (datetime.now()-start))

print("-----------------------------------")

# Query No# 15

start=datetime.now()

mycursor.execute("SELECT CustomerId,EstimatedSalary FROM churn_modelling GROUP BY
CustomerId,EstimatedSalary  HAVING EstimatedSalary >(SELECT  AVG(EstimatedSalary )
FROM churn_modelling )")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("-----------------------------------")

print ('The time for Query No#15 is %s' % (datetime.now()-start))

print("-----------------------------------")

# Query No# 16

start=datetime.now()

mycursor.execute("SELECT CustomerId, Surname FROM churn_modelling GROUP BY
CustomerId,Surname,Balance,CreditScore HAVING Balance >(SELECT  AVG(Balance )FROM
churn_modelling ) AND CreditScore >(SELECT  AVG(CreditScore )FROM churn_modelling )")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("-----------------------------------")

print ('The time for Query No#16 is %s' % (datetime.now()-start))

print("-----------------------------------")

# Query No# 17

start=datetime.now()

mycursor.execute("SELECT CustomerId, Tenure, count(*) FROM churn_modelling GROUP BY
CustomerId, Tenure WITH ROLLUP")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)


print("-----------------------------------")
```

```
print ('The time for Query No#17 is %s' % (datetime.now()-start))

print("-----------------------------------")


# Query No# 18

start=datetime.now()

mycursor.execute("SELECT COUNT(CustomerID) AS Number_of_Exited_Customers  FROM
churn_modelling WHERE Exited=1 GROUP BY CustomerId")

myreslt = mycursor.fetchall()

for x in myresult:

  print(x)

  print("-----------------------------------")

print ('The time for Query No#18 is %s' % (datetime.now()-start))

print("-----------------------------------")

# Query No# 19

start=datetime.now()

mycursor.execute("SELECT CustomerId,  SUM(Balance+EstimatedSalary) as TotalMoney
FROM churn_modelling GROUP BY CustomerId HAVING
SUM(Balance+EstimatedSalary)>100000 ORDER BY TotalMoney DESC ")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

print("-----------------------------------")

print ('The time for Query No#19 is %s' % (datetime.now()-start))

print("-----------------------------------")

# Query No# 20

start=datetime.now()

mycursor.execute("SELECT CustomerId FROM churn_modelling WHERE CreditScore>
(SELECT AVG(CreditScore)  FROM churn_modelling  )")

myresult = mycursor.fetchall()

for x in myresult:

  print(x)

  print("-----------------------------------")  print ('The time for Query No#20 is %s' %
(datetime.now()-start))print("-----------------------------------")
```

Postgre SQL Queries

```
import psycopg2

from datetime import datetime

connection = psycopg2.connect(database="ITC6107A1", user="sysadmin",
password="Bnft!123", host="127.0.0.1", port=5432)

#Query No#1

start=datetime.now()

c = connection.cursor()

c.execute('SELECT COUNT(customerid) AS Number_of_exited_Customers  FROM
churn_modeling WHERE exited=1 ') # use triple quotes if you want to spread your query
across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#1 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#2

start=datetime.now()

c = connection.cursor()

c.execute('SELECT AVG(estimatedsalary) As
Average_Estimated_Customer_Salary,MIN(estimatedsalary) As
Minimum_Estimated_Customer_Salary ,MAX(estimatedsalary) As
Maximum_Estimated_Customer_Salary  FROM churn_modeling') # use triple quotes if you
want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#2 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()
```

```
#Query No#3

start=datetime.now()

c = connection.cursor()

c.execute('SELECT COUNT(customerid) FROM churn_modeling WHERE exited=1 AND
isactivemember=1 ') # use triple quotes if you want to spread your query across multiple
lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.

print("-------------------------------------")

print ('The time for Query No#3 is %s' % (datetime.now()-start))

print("-------------------------------------")

c.close()

#Query No#4

start=datetime.now()

c = connection.cursor()

c.execute("SELECT  (COUNT(customerid)*100/(SELECT COUNT(*) FROM churn_modeling ))
AS Total_Male_Percentage FROM churn_modeling WHERE gender='Male' ") # use triple
quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.

print("-------------------------------------")

print ('The time for Query No#4 is %s' % (datetime.now()-start))

print("-------------------------------------")

c.close()

#Query No#5

start=datetime.now()


c = connection.cursor()

c.execute('SELECT COUNT (customerid) AS COUNTERSUM FROM churn_modeling GROUP BY
age,customerid HAVING age BETWEEN 20 AND 40 ') # use triple quotes if you want to spread
your query across multiple lines

for result in c:
```

```
    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#5 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#6

start=datetime.now()

c = connection.cursor()

c.execute("SELECT COUNT(customerid), geography  FROM churn_modeling  WHERE geography='Spain' or geography='France' GROUP BY geography ") # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#6 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#7

start=datetime.now()

c = connection.cursor()

c.execute('SELECT customerid, surname from churn_modeling  WHERE NOT ( hascrcard=0 AND isactivemember=0) AND (age<40) GROUP BY surname,customerid ORDER BY surname DESC') # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#7 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()
```

Query No#8

```
start=datetime.now()

c = connection.cursor()

c.execute("SELECT customerid,surname, geography FROM churn_modeling WHERE surname LIKE  'A%' AND (geography IN('Germany','France')) GROUP BY customerid,surname,geography ORDER BY customerid, surname DESC,geography DESC") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#8 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#9

start=datetime.now()

c = connection.cursor()

c.execute("SELECT customerid,surname FROM churn_modeling WHERE tenure IN ('2' ,'4' ,'6' ,'8') AND geography IN('Spain' ,'Germany') AND age IN ( '20', '30','40') GROUP BY customerid,surname,tenure,geography,age ORDER BY customerid,surname,tenure,geography,age") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#9 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()
```

```
#Query No#10

start=datetime.now()

c = connection.cursor()

c.execute("SELECT COUNT(customerid) FROM churn_modeling  WHERE (gender='Male' AND geography='Spain') OR(creditscore>500)") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#10 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#11

start=datetime.now()

c = connection.cursor()

c.execute("SELECT customerid,numofproducts FROM churn_modeling  WHERE (Balance>150000 AND age>30) OR ( numofproducts  IN( '3' ,'4')) GROUP BY customerid,numofproducts ") # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#11 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#12

start=datetime.now()

c = connection.cursor()

c.execute('SELECT age,creditscore FROM churn_modeling  WHERE creditscore>500 GROUP BY age, creditscore  HAVING age>(Select AVG(age) FROM churn_modeling )') # use triple quotes if you want to spread your query across multiple lines

for result in c:
```

```
    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#12 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#13

start=datetime.now()

c = connection.cursor()

c.execute('SELECT MIN (Balance) AS Minimum_Balance,MAX (Balance) AS Maximum_Balance,   AVG (Balance) AS Average_Balance, MIN (age) AS Minimum_age,MAX (age) AS Maximum_age,   AVG (age) AS Average_age  , MIN (creditscore) AS Minimum_creditscore,MAX (creditscore) AS Maximum_creditscore,   AVG (creditscore) AS Average_creditscore  , MIN (tenure) AS Minimum_tenure,MAX (tenure) AS Maximum_tenure,   AVG (tenure) AS Average_tenure   FROM churn_modeling GROUP BY Balance,age,creditscore,tenure') # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.

print("-----------------------------------")

print ('The time for Query No#13 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#14

start=datetime.now()

c = connection.cursor()

c.execute('SELECT geography,Balance,sum(Balance ) OVER(PARTITION BY geography ) AS TotalBalPerRegion FROM churn_modeling GROUP BY geography, Balance') # use triple quotes if you want to spread your query across multiple lines

for result in c:

    print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.
```

```
print("-----------------------------------")

print ('The time for Query No#14 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#15

start=datetime.now()

c = connection.cursor()

c.execute('SELECT customerid,estimatedsalary FROM churn_modeling GROUP BY
customerid,estimatedsalary  HAVING estimatedsalary >(SELECT  AVG(estimatedsalary )
FROM churn_modeling )') # use triple quotes if you want to spread your query across
multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#15 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#16

start=datetime.now()


c = connection.cursor()

c.execute('SELECT customerid, surname FROM churn_modeling GROUP BY
customerid,surname,Balance,creditscore HAVING Balance >(SELECT  AVG(Balance )FROM
churn_modeling ) AND creditscore >(SELECT  AVG(creditscore )FROM churn_modeling )') #
use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#16 is %s' % (datetime.now()-start))
```

```
print("-----------------------------------")

c.close()


#Query No#17

start=datetime.now()


c = connection.cursor()

c.execute('SELECT customerid, tenure, count(*) FROM churn_modeling GROUP BY
GROUPING SETS (  ( customerid),  (tenure))') # use triple quotes if you want to spread your
query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#17 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#18

start=datetime.now()

c = connection.cursor()

c.execute('SELECT customerid,  SUM(Balance+estimatedsalary) as TotalMoney FROM
churn_modeling GROUP BY customerid HAVING SUM(Balance+estimatedsalary)> 100000
ORDER BY TotalMoney DESC;') # use triple quotes if you want to spread your query across
multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll
need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#18 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()
```

```
#Query No#19

start=datetime.now()

c = connection.cursor()

c.execute('select tenure, count(*) as total_tenure,   count(*) * 1.0 / sum(count(*)) over () as ratio from churn_modeling group by tenure') # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#19 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()

#Query No#20

start=datetime.now()


c = connection.cursor()

c.execute('SELECT customerid FROM churn_modeling WHERE creditscore> (SELECT AVG(creditscore)  FROM churn_modeling  )') # use triple quotes if you want to spread your query across multiple lines

for result in c:

   print (result) # this only shows the first two columns. To add an additional column you'll need to add , '-', row[2], etc.


print("-----------------------------------")

print ('The time for Query No#20 is %s' % (datetime.now()-start))

print("-----------------------------------")

c.close()
```

Apache Spark Queries – same code for 1 core or multi core only the line : sc = SparkContext('local[*]', 'FinalProject')

changes from local[*] to local[1] for one core utilization

```
# ==== Apache Spark Queries ===
from datetime import datetime
from pyspark.sql import SparkSession
import sys
from pyspark import SparkContext,SparkConf
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import pandas as pd

df = pd.read_csv("C:\\Users\\30694\\Downloads\\Churn_Modelling.csv")
def kill_current_spark_context():
   SparkContext.getOrCreate().stop()
kill_current_spark_context()
sc = SparkContext('local[*]', 'FinalProject')
spark=SparkSession.builder.appName("spark sql test2").getOrCreate()
spark_df = spark.createDataFrame(df)
  spark_df.registerTempTable("Table")

# Query No# 1
start=datetime.now()
query_result=spark.sql("SELECT COUNT(CustomerID) AS  ExitedCustomers FROM Table
WHERE Exited=1 ").show()
# query_result.show()

print ('The time for Query No#1 is %s' % (datetime.now()-start))
```

```
# Query No# 2

start=datetime.now()

query_result=spark.sql("SELECT AVG(EstimatedSalary) As
Average_Estimated_Customer_Salary,MIN(EstimatedSalary) As
Minimum_Estimated_Customer_Salary ,MAX(EstimatedSalary) As
Maximum_Estimated_Customer_Salary  FROM Table")

print ('The time for Query No#2 is %s' % (datetime.now()-start))


# Query No# 3

start=datetime.now()

query_result=spark.sql("SELECT COUNT(CustomerID) FROM Table WHERE Exited=1 AND
IsActiveMember=1 ")

print ('The time for Query No#3 is %s' % (datetime.now()-start))

# Query No# 4

start=datetime.now()

query_result=spark.sql("SELECT  (COUNT(CustomerID)*100/(SELECT COUNT(*) FROM Table
))  AS Total_Male_Percentage FROM Table WHERE Gender='Male'")

print ('The time for Query No#4 is %s' % (datetime.now()-start))

# Query No# 5

start=datetime.now()

query_result=spark.sql("SELECT COUNT (CustomerId) AS COUNTERSUM FROM Table GROUP
BY Age,CustomerId HAVING Age BETWEEN 20 AND 40 ")

print ('The time for Query No#5 is %s' % (datetime.now()-start))

# Query No# 6

start=datetime.now()

query_result=spark.sql("SELECT COUNT(CustomerId), Geography  FROM Table  WHERE
Geography='Spain' or Geography='France' GROUP BY Geography ")

print ('The time for Query No#6 is %s' % (datetime.now()-start))

# Query No# 7

start=datetime.now()

query_result=spark.sql("SELECT CustomerId, Surname from Table  WHERE NOT (
HasCrCard=0 AND IsActiveMember=0) AND (Age<40)GROUP BY Surname,CustomerId ORDER
BY Surname DESC ")

print ('The time for Query No#7 is %s' % (datetime.now()-start))
```

```
# Query No# 8

start=datetime.now()

query_result=spark.sql("SELECT CustomerId,Surname, Geography FROM Table WHERE
Surname LIKE  'A%' AND (Geography IN('Germany','France'))GROUP BY
CustomerId,Surname,Geography ORDER BY CustomerId, Surname DESC,Geography DESC")

print ('The time for Query No#8 is %s' % (datetime.now()-start))

# Query No# 9

start=datetime.now()

query_result=spark.sql("SELECT CustomerId,Surname FROM Table WHERE Tenure IN ('2' ,'4'
,'6' ,'8') AND Geography IN('Spain' ,'Germany') AND AGE IN ( '20', '30','40') GROUP BY
CustomerId,Surname,Tenure,Geography,Age ORDER BY
CustomerId,Surname,Tenure,Geography,Age ")

print ('The time for Query No#9 is %s' % (datetime.now()-start))

# Query No# 10

start=datetime.now()

query_result=spark.sql("SELECT COUNT(CustomerId) FROM Table  WHERE (Gender='Male'
AND Geography='Spain') OR(CreditScore>500)")

print ('The time for Query No#10 is %s' % (datetime.now()-start))

# Query No# 11

start=datetime.now()

query_result=spark.sql("SELECT CustomerId,NumOfProducts FROM Table  WHERE
(Balance>150000 AND Age>30) OR ( NumOfProducts  IN( '3' ,'4')) GROUP BY
CustomerId,NumOfProducts ")

print ('The time for Query No#11 is %s' % (datetime.now()-start))


# Query No# 12

start=datetime.now()

query_result=spark.sql("SELECT Age,CreditScore FROM Table  WHERE CreditScore>500
GROUP BY Age,CreditScore  HAVING Age>(Select AVG(Age) FROM Table)")

print ('The time for Query No#12 is %s' % (datetime.now()-start))

# Query No# 13

start=datetime.now()

query_result=spark.sql(""" SELECT MIN (Balance) AS Minimum_Balance,MAX (Balance) AS
Maximum_Balance,   AVG (Balance) AS Average_Balance,
```

```
 MIN (Age) AS Minimum_Age,MAX (Age) AS Maximum_Age,   AVG (Age) AS Average_Age  ,

 MIN (CreditScore) AS Minimum_CreditScore,MAX (CreditScore) AS Maximum_CreditScore,
AVG (CreditScore) AS Average_CreditScore  ,

 MIN (Tenure) AS Minimum_Tenure,MAX (Tenure) AS Maximum_Tenure,   AVG (Tenure) AS
Average_Tenure

   FROM Table GROUP BY Balance,Age,CreditScore,Tenure""")

print ('The time for Query No#13 is %s' % (datetime.now()-start))


# Query No# 14

start=datetime.now()

query_result=spark.sql("SELECT Geography,Balance,sum(Balance ) OVER(PARTITION BY
Geography ) AS TotalBalPerRegion FROM Table GROUP BY Geography, Balance")


print ('The time for Query No#14 is %s' % (datetime.now()-start))


# Query No# 15

start=datetime.now()

query_result=spark.sql("SELECT CustomerId,EstimatedSalary FROM Table GROUP BY
CustomerId,EstimatedSalary  HAVING EstimatedSalary >(SELECT  AVG(EstimatedSalary )
FROM Table )")

print ('The time for Query No#15 is %s' % (datetime.now()-start))

# Query No# 16

start=datetime.now()

query_result=spark.sql("SELECT CustomerId, Surname FROM Table GROUP BY
CustomerId,Surname,Balance,CreditScore HAVING Balance >(SELECT  AVG(Balance )FROM
Table ) AND CreditScore >(SELECT  AVG(CreditScore )FROM Table )")


print ('The time for Query No#16 is %s' % (datetime.now()-start))
```

```
# Query No# 17

start=datetime.now()

query_result=spark.sql("SELECT CustomerId, Tenure, count(*) FROM Table GROUP BY
GROUPING SETS (  ( CustomerId),  (Tenure))")


print ('The time for Query No#17 is %s' % (datetime.now()-start))

# Query No# 18

start=datetime.now()

query_result=spark.sql("""SELECT CustomerId,

      SUM(Balance+EstimatedSalary) as TotalMoney FROM Table

   GROUP BY CustomerId

 HAVING    SUM(Balance+EstimatedSalary)  > 100000

ORDER BY   TotalMoney DESC""")

print ('The time for Query No#18 is %s' % (datetime.now()-start))

# Query No# 19

start=datetime.now()

query_result=spark.sql("""select Tenure, count(*) as total_tenure,

      count(*) * 1.0 / sum(count(*)) over () as ratio

from Table

group by Tenure """)

print ('The time for Query No#19 is %s' % (datetime.now()-start))


 Query No# 20

start=datetime.now()

query_result=spark.sql("SELECT CustomerId FROM Table WHERE CreditScore> (SELECT
AVG(CreditScore)  FROM Table  )")

# print(query_result)

# for row in query_result:

#     print(row)

print ('The time for Query No#20 is %s' % (datetime.now()-start))
```

Code for Data Visualization

```
import matplotlib.pyplot as plt # plotting

import numpy as np # linear algebra

import pandas as pd

import seaborn as sns



sns.set_context("paper", rc={"font.size":18,"axes.titlesize":18,"axes.labelsize":25})

df = pd.read_csv("C:\\Users\\30694\\Downloads\\Churn_Modelling.csv")

sns.set(rc={'figure.figsize':(11.7,8.27),"font.size":20,"axes.titlesize":20,"axes.labelsize":20},st
yle="white")



nRow, nCol = df.shape

print(f'There are {nRow} rows and {nCol} columns')



if pd.isnull(df).sum().all()==0:

   print ('There are no missing values')

else:

   print ('There are missing values')

for col in df:

   print(df[col].unique())



print(df['Exited'].value_counts())

labels = 'Exited', 'Retained'

sizes = [df.Exited[df['Exited']==1].count(), df.Exited[df['Exited']==0].count()]
```

```
explode = (0, 0.1)
fig1, ax1 = plt.subplots(figsize=(10, 8))
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.title("Proportion of customer churned and retained", size = 20)
plt.show()

# Get unique count for each variable
df.nunique()

fig, axarr = plt.subplots(2, 2, figsize=(20, 12))
sns.countplot(x='Geography', hue = 'Exited',data = df, ax=axarr[0][0])
sns.countplot(x='Gender', hue = 'Exited',data = df, ax=axarr[0][1])
sns.countplot(x='HasCrCard', hue = 'Exited',data = df, ax=axarr[1][0])
sns.countplot(x='IsActiveMember', hue = 'Exited',data = df, ax=axarr[1][1])


 # Relations based on the continuous data attributes
fig, axarr = plt.subplots(3, 2, figsize=(20, 12))
sns.boxplot(y='CreditScore',x = 'Exited', hue = 'Exited',data = df, ax=axarr[0][0])
sns.boxplot(y='Age',x = 'Exited', hue = 'Exited',data = df , ax=axarr[0][1])
sns.boxplot(y='Tenure',x = 'Exited', hue = 'Exited',data = df, ax=axarr[1][0])
sns.boxplot(y='Balance',x = 'Exited', hue = 'Exited',data = df, ax=axarr[1][1])
sns.boxplot(y='NumOfProducts',x = 'Exited', hue = 'Exited',data = df, ax=axarr[2][0])
sns.boxplot(y='EstimatedSalary',x = 'Exited', hue = 'Exited',data = df, ax=axarr[2][1])
```