Advanced Database Management Systems

Banking Enterprise Database Design

Group Project (Group 15)

Sai Teja Reddy Garlapati **U91837993**

Pandu Ranga Madasu **U51811049**

Sanjana Kunduru **U36356862**

TABLE OF CONTENTS

1. Introduction

The database designed in this project is a banking enterprise database. This database is designed in such a way that it can serve the important and basic requirements of a bank and can be linked to a web page used by the bank employees and also a part of it accessible to the customers. The employees can log in and update the details about the customers like their accounts and transactions, loans and credit cards details. The customers can log in to their net banking accounts and check their transactions.

1.1 Basic Requirements:

- The banking database will have the information about different branches, their customers, employees working, accounts, loans and credit cards details.
- Different transactions like in bank account transactions, loan payments, credit card transactions are recorded.
- Net banking facility is provided to the customers.

1.2 Assumptions:

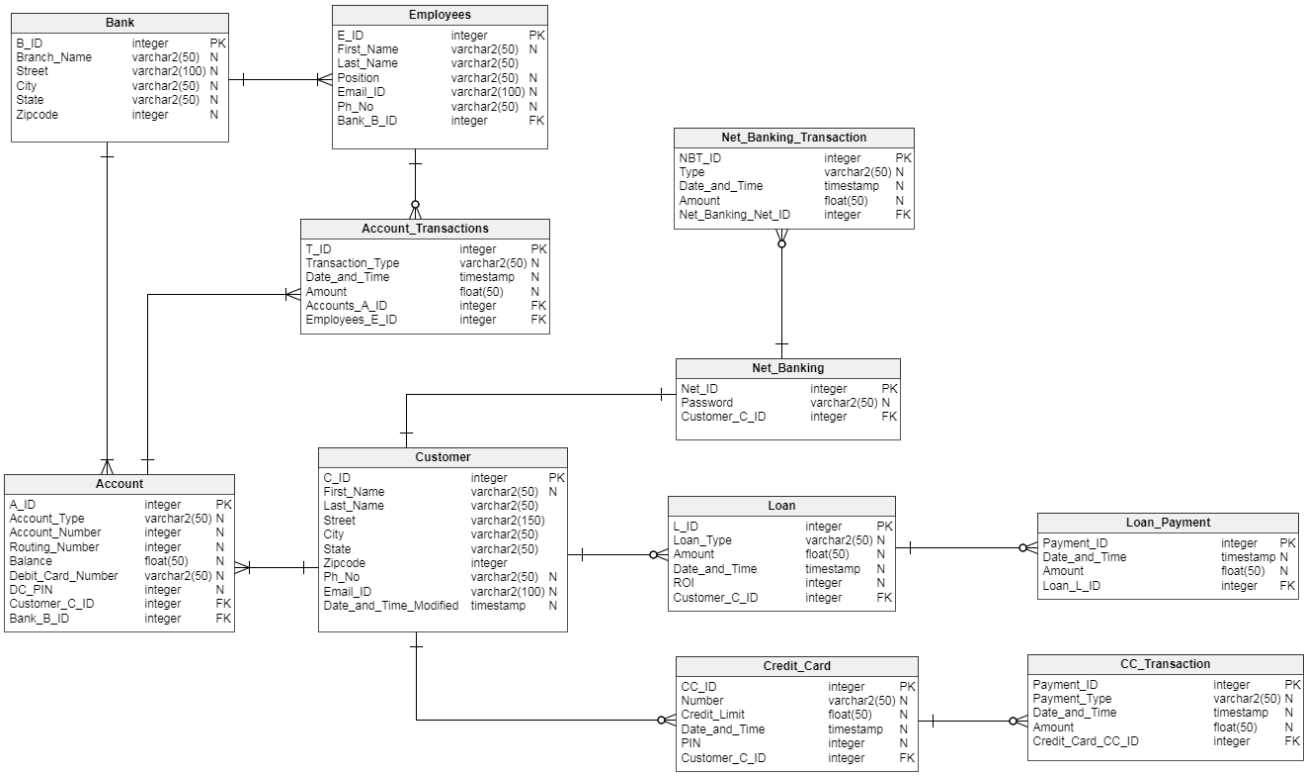
- There are no joint accounts i.e. each account is linked to only one customer.
- Each customer has only one account and thus will have only one net banking account.
- It is not necessary that every customer has taken a loan and a credit card.
- It is not necessary that every customer who has net banking have performed net banking transactions.

2. Logical Database Design

This section includes the entity-relationship diagram (ERD) and data dictionaries of the banking database design and the conceptual design behind this.

2.1 Conceptual Design:

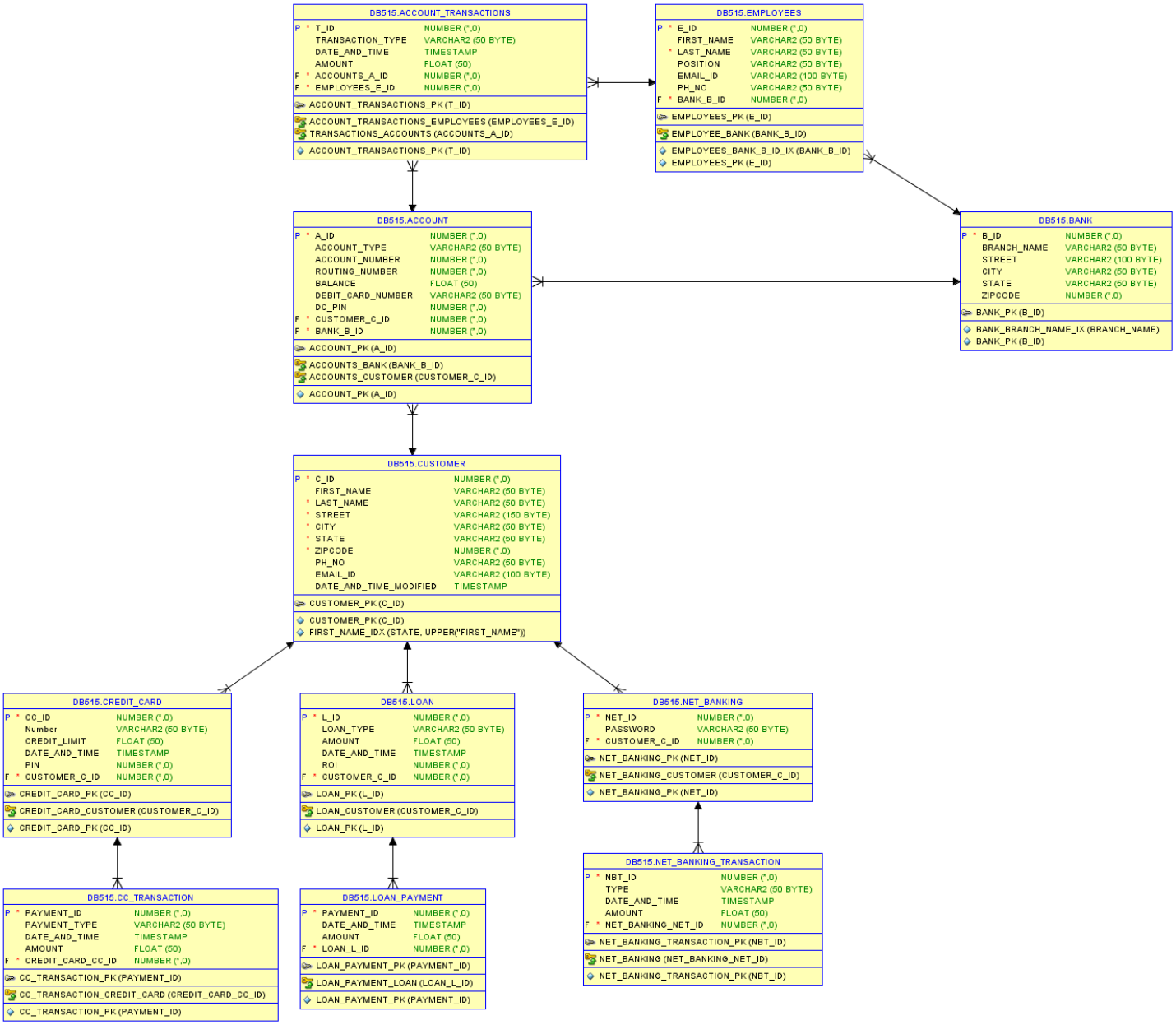
Initially, based on the assumptions and requirements of the enterprise, a conceptual model is designed which is platform independent i.e. irrespective of the database management system version. The following is the conceptual diagram of Banking enterprise database that is created using "*Vertabelo*"



2.2 Logical Design:

Once the requirements are finalized, all the tables and their associated views, indexes etc. are created based on the conceptual design using Oracle SQL Developer (in this project). The logical design is created automatically as per the adb_black schema and the tables used in developing this database. The following diagram displays the logical design developed based on the tables created and used along with its constraints.

File → Data Modeler → Import → Data Dictionary.

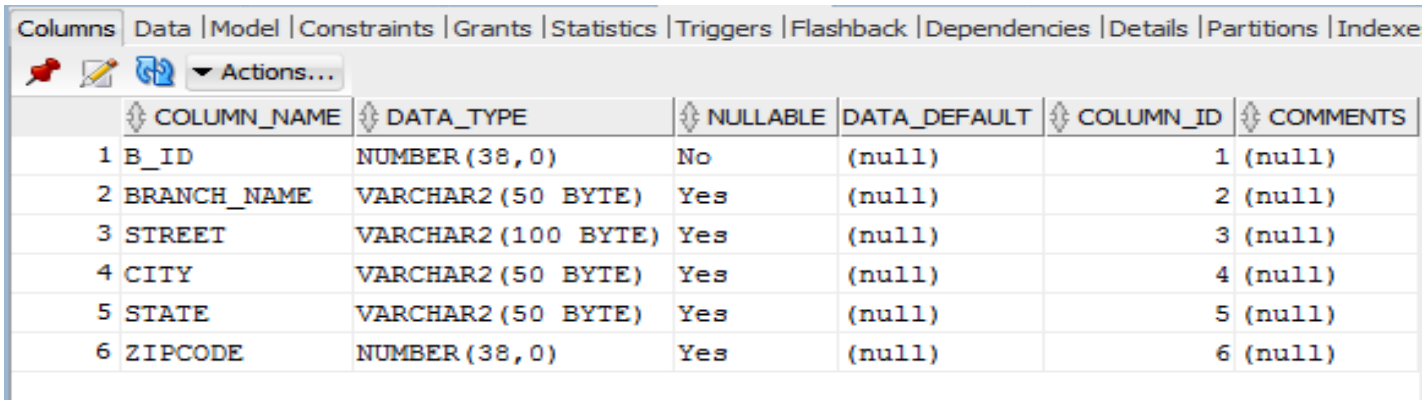


3. Physical Database Design

Physical Database design refers to the conversion of relations in the logical data base into corresponding database objects. As part of the project we are using Oracle SQL Developer for the creation and development of the database objects based on the logical model developed. Physical design involved in creating various database objects like tables, indexes and views etc. based on entities and relations in logical design.

3.1 Table: Bank

```
CREATE TABLE DB515.Bank (
    B_ID integer  NOT NULL,
    Branch_Name varchar2(50)  NULL,
    Street varchar2(100)  NULL,
    City varchar2(50)  NULL,
    State varchar2(50)  NULL,
    Zipcode integer  NULL,
    CONSTRAINT Bank_pk PRIMARY KEY (B_ID)
) ;
```



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	B_ID	NUMBER(38,0)	No	(null)	1	(null)
2	BRANCH_NAME	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	STREET	VARCHAR2(100 BYTE)	Yes	(null)	3	(null)
4	CITY	VARCHAR2(50 BYTE)	Yes	(null)	4	(null)
5	STATE	VARCHAR2(50 BYTE)	Yes	(null)	5	(null)
6	ZIPCODE	NUMBER(38,0)	Yes	(null)	6	(null)

3.2 Table: Employees

```
CREATE TABLE DB515.Employees (
    E_ID integer  NOT NULL,
    First_Name varchar2(50)  NULL,
    Last_Name varchar2(50)  NOT NULL,
    Position varchar2(50)  NULL,
    Email_ID varchar2(100)  NULL,
    Ph_No varchar2(50)  NULL,
    Bank_B_ID integer  NOT NULL,
    CONSTRAINT Employees_pk PRIMARY KEY (E_ID)
) ;
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	E_ID	NUMBER(38,0)	No	(null)	1	(null)
2	FIRST_NAME	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	LAST_NAME	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	POSITION	VARCHAR2(50 BYTE)	Yes	(null)	4	(null)
5	EMAIL_ID	VARCHAR2(100 BYTE)	Yes	(null)	5	(null)
6	PH_NO	VARCHAR2(50 BYTE)	Yes	(null)	6	(null)
7	BANK_B_ID	NUMBER(38,0)	No	(null)	7	(null)

Foreign Keys:

Name	Column	Mapping
Employee_Bank	Bank_B_ID	Bank.B_ID

```
ALTER TABLE DB515.Employees ADD CONSTRAINT Employee_Bank
    FOREIGN KEY (Bank_B_ID)
    REFERENCES Bank (B_ID);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	EMPLOYEES_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	EMPLOYEE_BANK	Foreign_Key	(null)	DB515	BANK	BANK_PK	NO ACTION	ENABLED	NOT DEFERRABLE
3	SYS_C0065666	Check	"E_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
4	SYS_C0065667	Check	"LAST_NAME" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
5	SYS_C0065668	Check	"BANK_B_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

3.3 Table: Account

```
CREATE TABLE DB515.Account (
    A_ID integer  NOT NULL,
    Account_Type varchar2(50)  NULL,
    Account_Number integer  NULL,
    Routing_Number integer  NULL,
    Balance float(50)  NULL,
    Debit_Card_Number varchar2(50)  NULL,
    DC_PIN integer  NULL,
    Customer_C_ID integer  NOT NULL,
    Bank_B_ID integer  NOT NULL,
    CONSTRAINT Account_pk PRIMARY KEY (A_ID)
) ;
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	A_ID	NUMBER(38,0)	No	(null)	1	(null)
2	ACCOUNT_TYPE	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	ACCOUNT_NUMBER	NUMBER(38,0)	Yes	(null)	3	(null)
4	ROUTING_NUMBER	NUMBER(38,0)	Yes	(null)	4	(null)
5	BALANCE	FLOAT	Yes	(null)	5	(null)
6	DEBIT_CARD_NUMBER	VARCHAR2(50 BYTE)	Yes	(null)	6	(null)
7	DC_PIN	NUMBER(38,0)	Yes	(null)	7	(null)
8	CUSTOMER_C_ID	NUMBER(38,0)	No	(null)	8	(null)
9	BANK_B_ID	NUMBER(38,0)	No	(null)	9	(null)

Foreign Keys:

Name	Column	Mapping
Accounts_Bank	Bank_B_ID	Bank.B_ID
Accounts_Bank	Customer_C_ID	Customer.C_ID

```
ALTER TABLE DB515.Account ADD CONSTRAINT Accounts_Bank
    FOREIGN KEY (Bank_B_ID)
    REFERENCES Bank (B_ID);

ALTER TABLE DB515.Account ADD CONSTRAINT Accounts_Customer
    FOREIGN KEY (Customer_C_ID)
    REFERENCES Customer (C_ID);
```

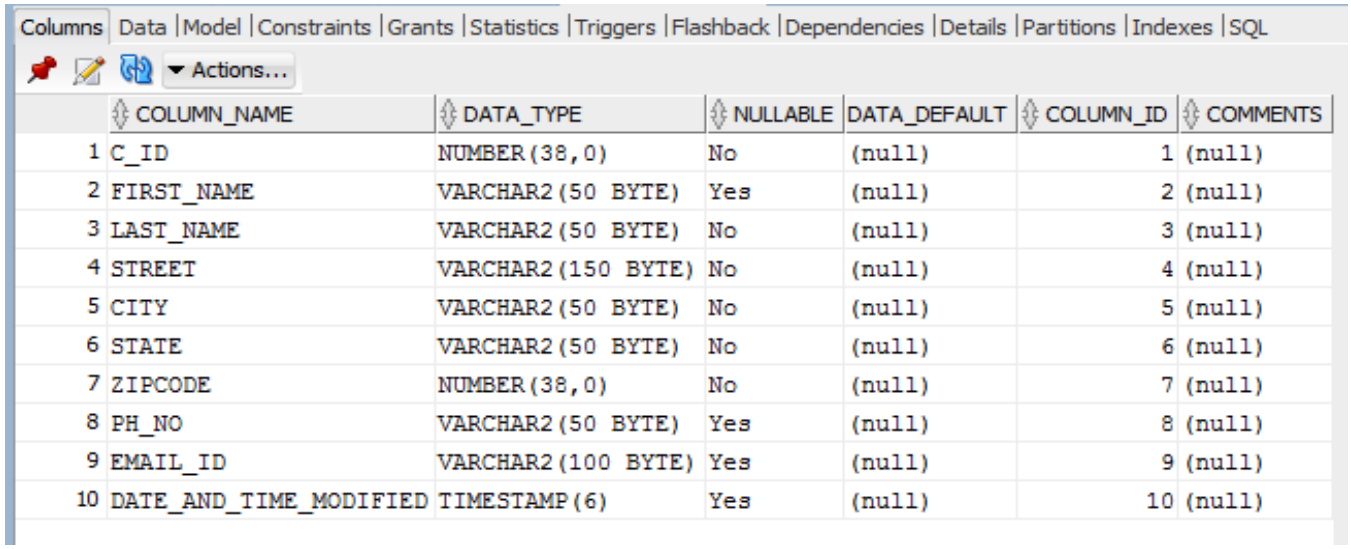
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEF
1	ACCOUNTS_BANK	Foreign_Key	(null)	DB515	BANK	BANK_PK	NO ACTION	ENABLED	NOT I
2	ACCOUNTS_CUSTOMER	Foreign_Key	(null)	DB515	CUSTOMER	CUSTOMER_PK	NO ACTION	ENABLED	NOT I
3	ACCOUNT_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT I
4	SYS_C0065643	Check	"A_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT I
5	SYS_C0065644	Check	"CUSTOMER_C_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT I
6	SYS_C0065645	Check	"BANK_B_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT I

3.4 Table: Customer

```
CREATE TABLE DB515.Customer (
    C_ID integer  NOT NULL,
    First_Name varchar2(50)  NULL,
    Last_Name varchar2(50)  NOT NULL,
    Street varchar2(150)  NOT NULL,
    City varchar2(50)  NOT NULL,
    State varchar2(50)  NOT NULL,
    Zipcode integer  NOT NULL,
    Ph_No varchar2(50)  NULL,
    Email_ID varchar2(100)  NULL,
    Date_and_Time_Modified timestamp  NULL,
    CONSTRAINT Customer_pk PRIMARY KEY (C_ID)
) ;
```



Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	C_ID	NUMBER(38,0)	No	(null)	1	(null)
2	FIRST_NAME	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	LAST_NAME	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	STREET	VARCHAR2(150 BYTE)	No	(null)	4	(null)
5	CITY	VARCHAR2(50 BYTE)	No	(null)	5	(null)
6	STATE	VARCHAR2(50 BYTE)	No	(null)	6	(null)
7	ZIPCODE	NUMBER(38,0)	No	(null)	7	(null)
8	PH_NO	VARCHAR2(50 BYTE)	Yes	(null)	8	(null)
9	EMAIL_ID	VARCHAR2(100 BYTE)	Yes	(null)	9	(null)
10	DATE_AND_TIME_MODIFIED	TIMESTAMP(6)	Yes	(null)	10	(null)

3.5 Table: Account_Transactions

```
CREATE TABLE DB515.Account_Transactions (
    T_ID integer  NOT NULL,
    Transaction_Type varchar2(50)  NULL,
    Date_and_Time timestamp  NULL,
    Amount float(50)  NULL,
    Accounts_A_ID integer  NOT NULL,
    Employees_E_ID integer  NOT NULL,
    CONSTRAINT Account_Transactions_pk PRIMARY KEY (T_ID)
) ;
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | S

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	T_ID	NUMBER(38,0)	No	(null)	1	(null)
2	TRANSACTION_TYPE	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	DATE_AND_TIME	TIMESTAMP(6)	Yes	(null)	3	(null)
4	AMOUNT	FLOAT	Yes	(null)	4	(null)
5	ACCOUNTS_A_ID	NUMBER(38,0)	No	(null)	5	(null)
6	EMPLOYEES_E_ID	NUMBER(38,0)	No	(null)	6	(null)

Foreign Keys:

Name	Column	Mapping
Account_Transactions_Employees	Employees_E_ID	Employees.E_ID
Transactions_Accounts	Accounts_A_ID	Account.A_ID

```
ALTER TABLE DB515.Account_Transactions ADD CONSTRAINT Account_Transactions_Employees
    FOREIGN KEY (Employees_E_ID)
    REFERENCES Employees (E_ID);


ALTER TABLE DB515.Account_Transactions ADD CONSTRAINT Transactions_Accounts
    FOREIGN KEY (Accounts_A_ID)
    REFERENCES Account (A_ID);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME
1	ACCOUNT_TRANSACTIONS_EMPLOYEES	Foreign_Key	(null)	DB515	EMPLOYEES	EMPLOYEES_PK
2	ACCOUNT_TRANSACTIONS_PK	Primary_Key	(null)	(null)	(null)	(null)
3	SYS_C0065647	Check	"T_ID" IS NOT NULL	(null)	(null)	(null)
4	SYS_C0065648	Check	"ACCOUNTS_A_ID" IS NOT NULL	(null)	(null)	(null)
5	SYS_C0065649	Check	"EMPLOYEES_E_ID" IS NOT NULL	(null)	(null)	(null)
6	TRANSACTIONS_ACCOUNTS	Foreign_Key	(null)	DB515	ACCOUNT	ACCOUNT_PK

3.6 Table: Credit_Card

```
CREATE TABLE DB515.Credit_Card (
    CC_ID integer  NOT NULL,
    "Number" varchar2(50)  NULL,
    Credit_Limit float(50)  NULL,
    Date_and_Time timestamp  NULL,
    PIN integer  NULL,
    Customer_C_ID integer  NOT NULL,
    CONSTRAINT Credit_Card_pk PRIMARY KEY (CC_ID)
) ;
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexe

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CC_ID	NUMBER(38,0)	No	(null)	1	(null)
2	Number	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	CREDIT_LIMIT	FLOAT	Yes	(null)	3	(null)
4	DATE_AND_TIME	TIMESTAMP(6)	Yes	(null)	4	(null)
5	PIN	NUMBER(38,0)	Yes	(null)	5	(null)
6	CUSTOMER_C_ID	NUMBER(38,0)	No	(null)	6	(null)

Foreign Keys:

Name	Column	Mapping
Credit_Card_Customer	Customer_C_ID	Customer.C_ID

```
ALTER TABLE DB515.Credit_Card ADD CONSTRAINT Credit_Card_Customer
    FOREIGN KEY (Customer_C_ID)
    REFERENCES Customer (C_ID);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R
1	CREDIT_CARD_CUSTOMER	Foreign_Key	(null)	DB515	CUSTOMER	CUS
2	CREDIT_CARD_PK	Primary_Key	(null)	(null)	(null)	(nu
3	SYS_C0065656	Check	"CC_ID" IS NOT NULL	(null)	(null)	(nu
4	SYS_C0065657	Check	"CUSTOMER_C_ID" IS NOT NULL	(null)	(null)	(nu

3.7 Table: CC_Transaction

```
CREATE TABLE DB515.CC_Transaction (
    Payment_ID integer  NOT NULL,
    Payment_Type varchar2(50)  NULL,
    Date_and_Time timestamp  NULL,
    Amount float(50)  NULL,
    Credit_Card_CC_ID integer  NOT NULL,
    CONSTRAINT CC_Transaction_pk PRIMARY KEY (Payment_ID)
) ;
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Index

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PAYMENT_ID	NUMBER(38,0)	No	(null)	1	(null)
2	PAYMENT_TYPE	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	DATE_AND_TIME	TIMESTAMP(6)	Yes	(null)	3	(null)
4	AMOUNT	FLOAT	Yes	(null)	4	(null)
5	CREDIT_CARD_CC_ID	NUMBER(38,0)	No	(null)	5	(null)

Foreign Keys:

Name	Column	Mapping
CC_Transaction_Credit_Card	Credit_Card_CC_ID	Credit_Card.CC_ID

```
ALTER TABLE DB515.CC_Transaction ADD CONSTRAINT CC_Transaction_Credit_Card
    FOREIGN KEY (Credit_Card_CC_ID)
    REFERENCES Credit_Card (CC_ID);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME
1	CC_TRANSACTION_CREDIT_CARD	Foreign_Key	(null)	DB515	CREDIT_CARD
2	CC_TRANSACTION_PK	Primary_Key	(null)	(null)	(null)
3	SYS_C0065653	Check	"PAYMENT_ID" IS NOT NULL	(null)	(null)
4	SYS_C0065654	Check	"CREDIT_CARD_CC_ID" IS NOT NULL	(null)	(null)

3.8 Table: Loan

```
CREATE TABLE DB515.Loan (
    L_ID integer  NOT NULL,
    Loan_Type varchar2(50)  NULL,
    Amount float(50)  NULL,
    Date_and_Time timestamp  NULL,
    ROI integer  NULL,
    Customer_C_ID integer  NOT NULL,
    CONSTRAINT Loan_pk PRIMARY KEY (L_ID)
) ;
```

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	L_ID	NUMBER(38,0)	No	(null)	1	(null)
2	LOAN_TYPE	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	AMOUNT	FLOAT	Yes	(null)	3	(null)
4	DATE_AND_TIME	TIMESTAMP(6)	Yes	(null)	4	(null)
5	ROI	NUMBER(38,0)	Yes	(null)	5	(null)
6	CUSTOMER_C_ID	NUMBER(38,0)	No	(null)	6	(null)

Foreign Keys:

Name	Column	Mapping
Loan_Customer	Customer_C_ID	Customer.C_ID

```
ALTER TABLE DB515.Loan ADD CONSTRAINT Loan_Customer
    FOREIGN KEY (Customer_C_ID)
    REFERENCES Customer (C_ID);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME
1	LOAN_CUSTOMER	Foreign_Key	(null)	DB515	CUSTOMER	CUSTOMER_PK
2	LOAN_PK	Primary_Key	(null)	(null)	(null)	(null)
3	SYS_C0065670	Check	"L_ID" IS NOT NULL	(null)	(null)	(null)
4	SYS_C0065671	Check	"CUSTOMER_C_ID" IS NOT NULL	(null)	(null)	(null)

3.9 Table: Loan_Payment

```
CREATE TABLE DB515.Loan_Payment (
    Payment_ID integer  NOT NULL,
    Date_and_Time timestamp  NULL,
    Amount float(50)  NULL,
    Loan_L_ID integer  NOT NULL,
    CONSTRAINT Loan_Payment_pk PRIMARY KEY (Payment_ID)
) ;
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Pa

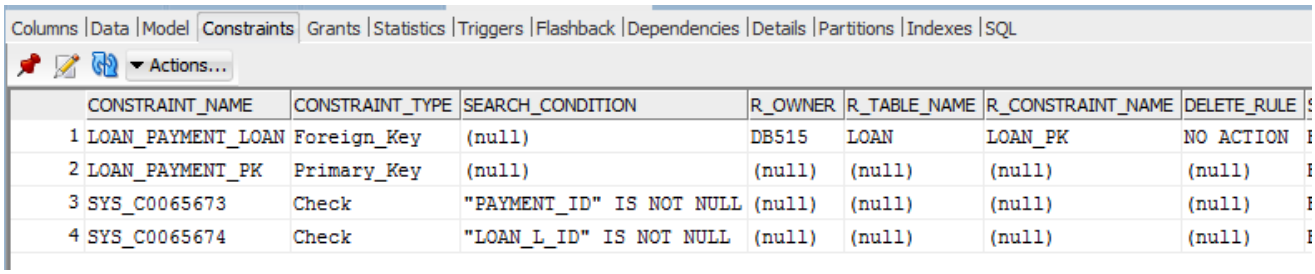
Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PAYMENT_ID	NUMBER(38,0)	No	(null)	1	(null)
2	DATE_AND_TIME	TIMESTAMP(6)	Yes	(null)	2	(null)
3	AMOUNT	FLOAT	Yes	(null)	3	(null)
4	LOAN_L_ID	NUMBER(38,0)	No	(null)	4	(null)

Foreign Keys:

Name	Column	Mapping
Loan_Payment_Loan	Loan_L_ID	Loan.L_ID

```
ALTER TABLE Loan_Payment ADD CONSTRAINT Loan_Payment_Loan
    FOREIGN KEY (Loan_L_ID)
    REFERENCES Loan (L_ID);
```



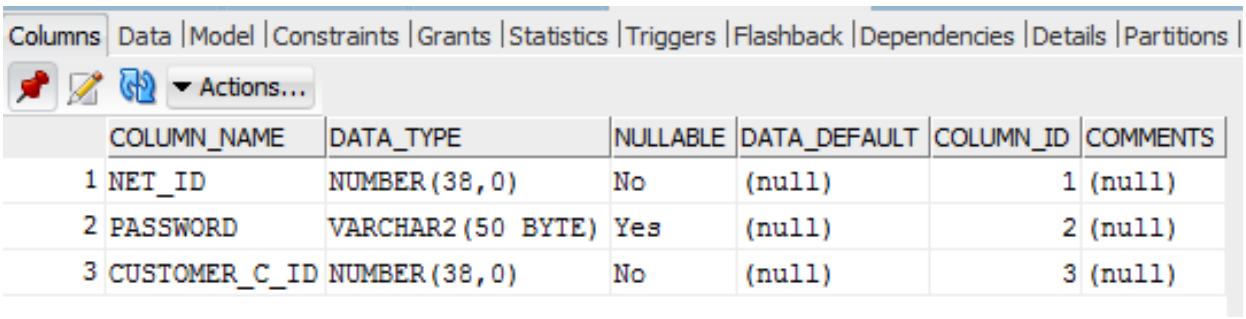
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE
1	LOAN_PAYMENT_LOAN	Foreign_Key	(null)	DB515	LOAN	LOAN_PK	NO ACTION
2	LOAN_PAYMENT_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)
3	SYS_C0065673	Check	"PAYMENT_ID" IS NOT NULL	(null)	(null)	(null)	(null)
4	SYS_C0065674	Check	"LOAN_L_ID" IS NOT NULL	(null)	(null)	(null)	(null)

3.10 Table: Net_Banking

```
CREATE TABLE Net_Banking (
    Net_ID integer  NOT NULL,
    Password varchar2(50)  NULL,
    Customer_C_ID integer  NOT NULL,
    CONSTRAINT Net_Banking_pk PRIMARY KEY (Net_ID)
) ;
```



Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions |

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	NET_ID	NUMBER(38,0)	No	(null)	1	(null)
2	PASSWORD	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	CUSTOMER_C_ID	NUMBER(38,0)	No	(null)	3	(null)

Foreign Keys:

Name	Column	Mapping
Net_Banking_Customer	Customer_C_ID	Customer.C_ID

```
ALTER TABLE Net_Banking ADD CONSTRAINT Net_Banking_Customer
    FOREIGN KEY (Customer_C_ID)
    REFERENCES Customer (C_ID);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME
1	NET_BANKING_CUSTOMER	Foreign_Key	(null)	DB515	CUSTOMER	CUSTOMER_PK
2	NET_BANKING_PK	Primary_Key	(null)	(null)	(null)	(null)
3	SYS_C0065676	Check	"NET_ID" IS NOT NULL	(null)	(null)	(null)
4	SYS_C0065677	Check	"CUSTOMER_C_ID" IS NOT NULL	(null)	(null)	(null)

3.11 Table: Net_Banking_Transaction

```
CREATE TABLE Net_Banking_Transaction (
    NBT_ID integer  NOT NULL,
    Type varchar2(50)  NULL,
    Date_and_Time timestamp  NULL,
    Amount float(50)  NULL,
    Net_Banking_Net_ID integer  NOT NULL,
    CONSTRAINT Net_Banking_Transaction_pk PRIMARY KEY (NBT_ID)
) ;
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexe

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	NBT_ID	NUMBER(38,0)	No	(null)	1	(null)
2	TYPE	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3	DATE_AND_TIME	TIMESTAMP(6)	Yes	(null)	3	(null)
4	AMOUNT	FLOAT	Yes	(null)	4	(null)
5	NET_BANKING_NET_ID	NUMBER(38,0)	No	(null)	5	(null)

Foreign Keys:

Name	Column	Mapping
Net_Banking	Net_Banking_Net_ID	Net_Banking.Net_ID

```
ALTER TABLE Net_Banking_Transaction ADD CONSTRAINT Net_Banking
    FOREIGN KEY (Net_Banking_Net_ID)
    REFERENCES Net_Banking (Net_ID);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME
1	NET_BANKING	Foreign_Key	(null)	DB515	NET_BANKING	NET_BANKING_PK
2	NET_BANKING_TRANSACTION_PK	Primary_Key	(null)	(null)	(null)	(null)
3	SYS_C0065679	Check	"NBT_ID" IS NOT NULL	(null)	(null)	(null)
4	SYS_C0065680	Check	"NET_BANKING_NET_ID" IS NOT NULL	(null)	(null)	(null)

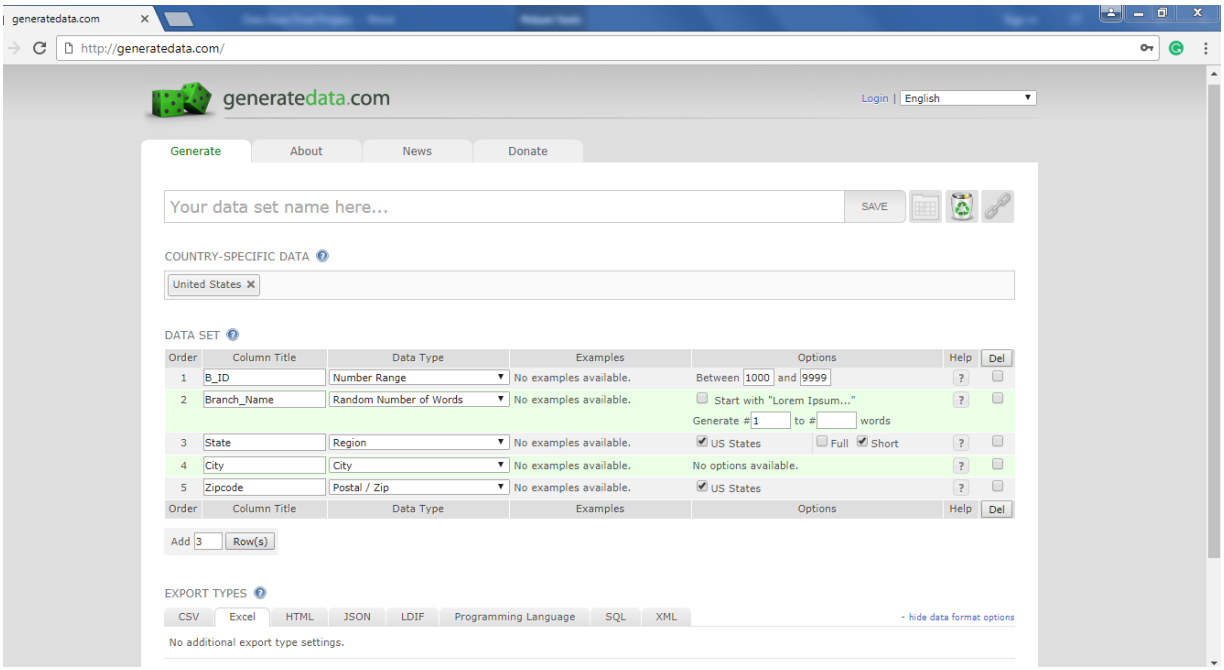
4. Data generation and Loading

The data is generated using data generating website http://generatedata.com/ and is loaded into the database. The steps followed in the database generation and loading are as follows:

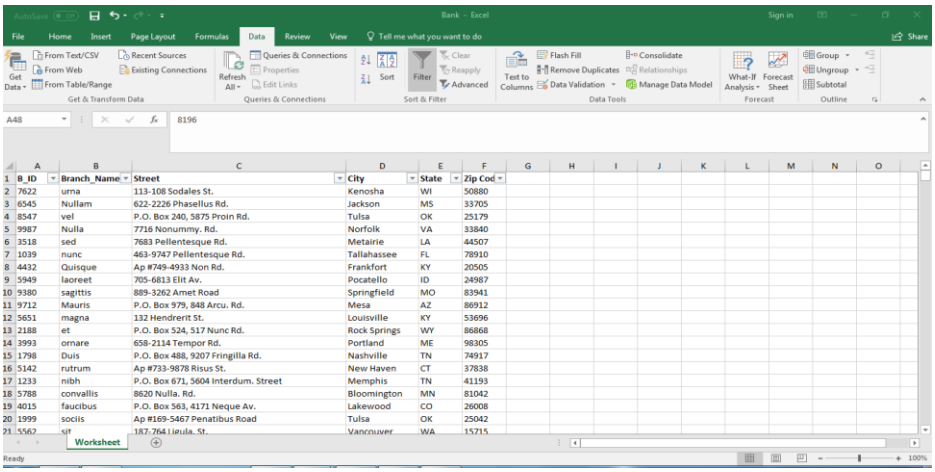
4.1 Step1:

By providing some parameters like number of records, column names and their respective data types into the http://generatedata.com/ website, data is generated in different formats like XML, JSON, CSV, Excel etc. Here, data is generated randomly by giving respective parameters to each column and is downloaded in the excel format (.xlxs) i.e. Microsoft Excel Worksheet.

- Following is the screenshot of http://generatedata.com/ website with various parameters defined for the generation of data that is to be loaded into Bank table of the database:



- Following is the MS Excel worksheet downloaded from http://generatedata.com/ website for Bank table of the database.



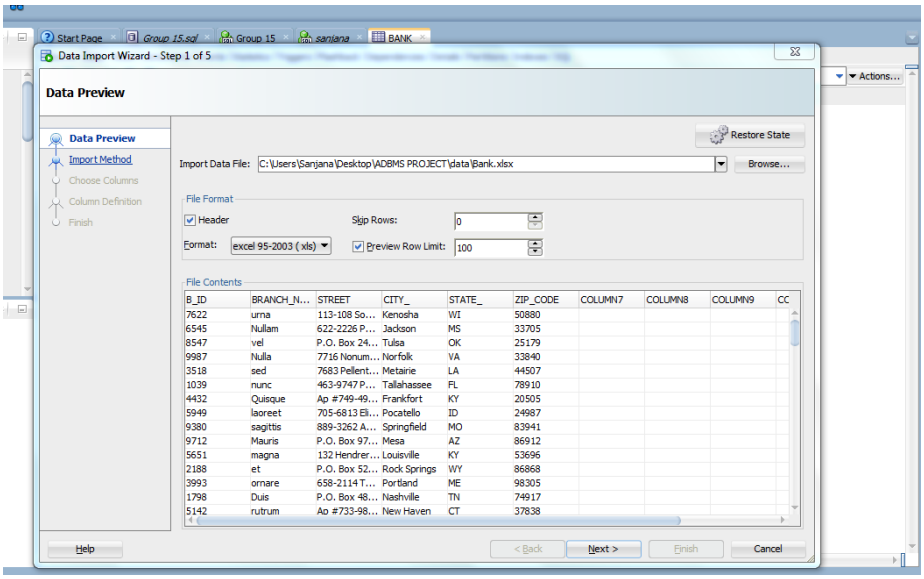
4.2 Step2:

There are some duplicate entries in the data. Some of these entries from the Excel sheet are deleted and some are replaced by other entries according to the database requirement.

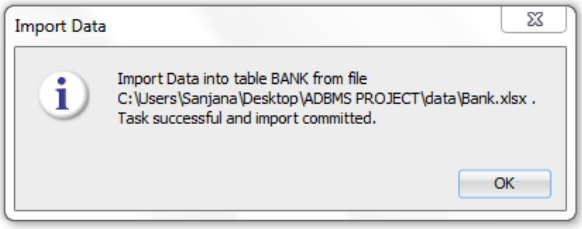
4.3 Step3:

These edited excel sheets are used to populate the database by using import wizard in the data Tab as seen in the following images:

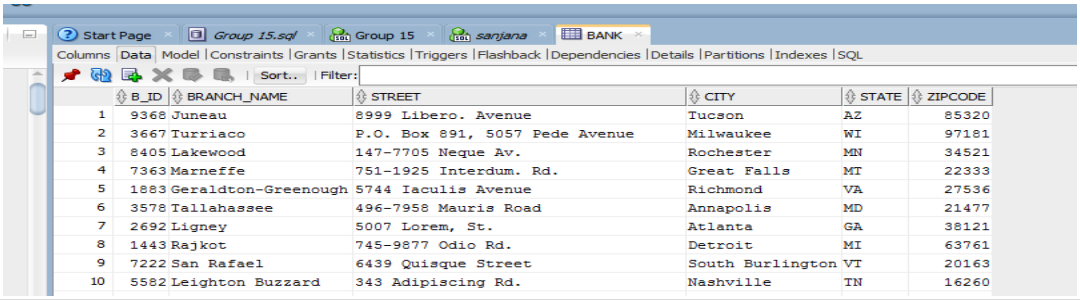
- Importing the excel sheet:



- Confirmation:



- Populated Bank table:



4.4 Number of records:

The following table gives the total number of records in each table:

Tables	Number of Records
Bank	503
Customer	10000
Employees	5000
Account	10000
Account_Transactions	14983
Loan	10000
Loan_Payment	10000
Net_Banking	5000
Net_Banking_Transactions	10000
Credit_Card	8500
Credit_Card_Transactions	10003

5. Performance Tuning

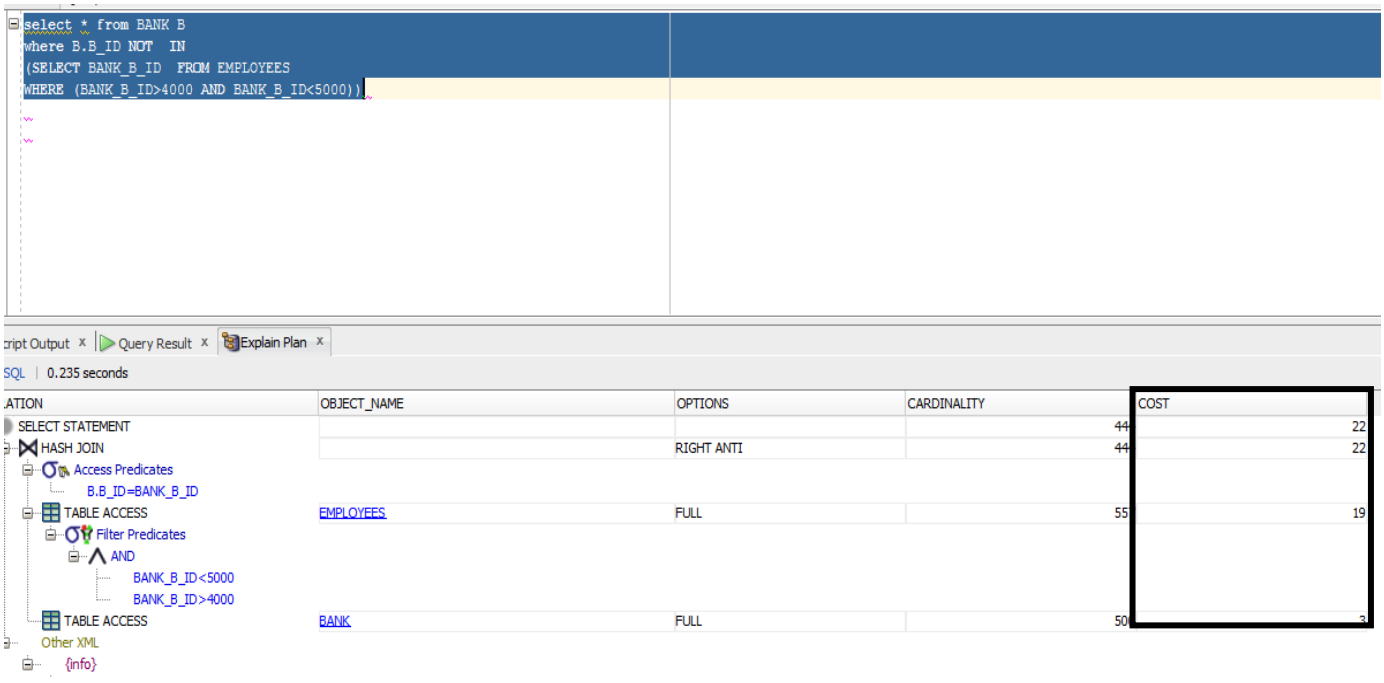
5.1 Indexing

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. The only columns that have indexes are Primary keys and columns having unique constraints. The columns that we need to index are Foreign keys as they are so commonly selected while querying and used in joins.

The below query selects list of records in Bank table when Bank ID's in Employees table are greater than 4000 and less than 5000.

```
select * from BANK B
where B.B_ID NOT  IN
(SELECT BANK_B_ID  FROM EMPLOYEES
WHERE (BANK_B_ID>4000 AND BANK_B_ID<5000));
```

- **Before Indexing:**

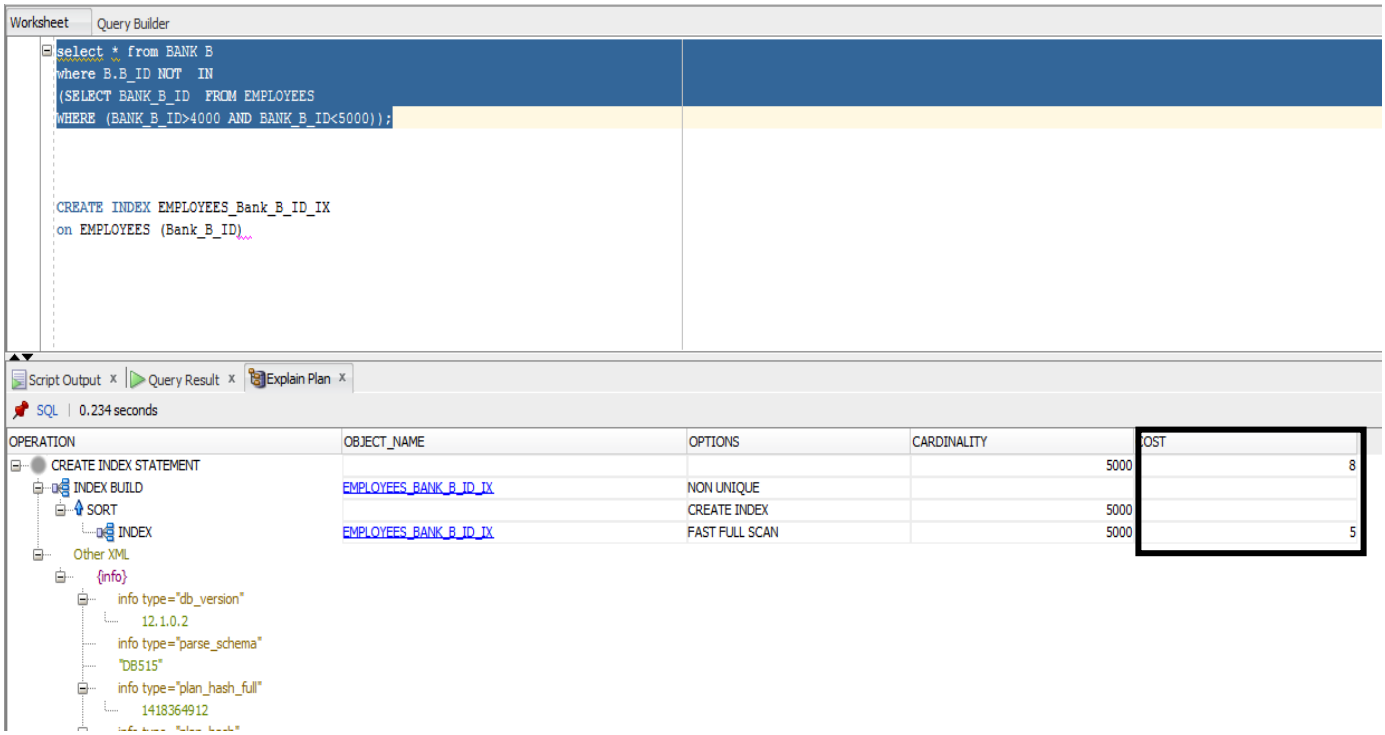


- **Create Index:**

```
CREATE INDEX EMPLOYEES_Bank_B_ID_IX
on EMPLOYEES (Bank_B_ID)
```

➢ **After Indexing:**

It can be observed that after creating the Index both Cost of the query has been decreased.



5.2 Function Based Indexing:

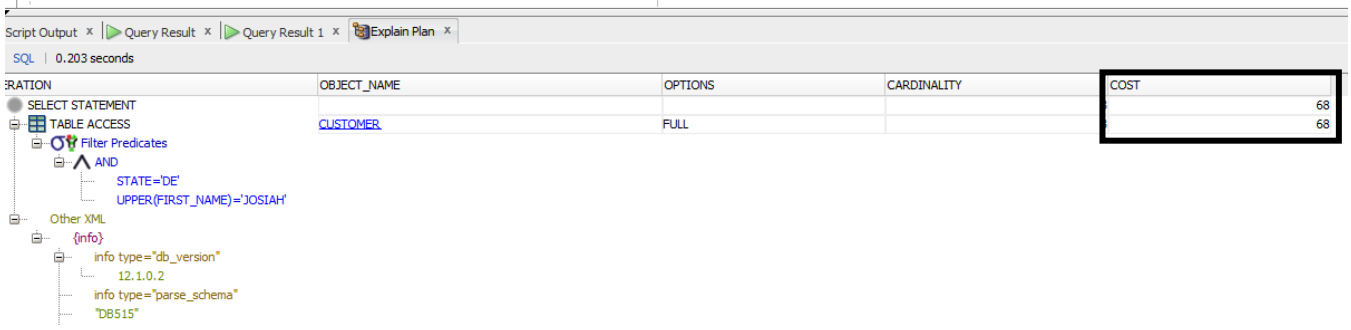
Indexing a column will not be useful when we use functions with columns. Rather than indexing a column, you index the function on that column, storing the product of the function, not the original column data. When a query is passed to the server that could benefit from that index, the query is rewritten to allow the index to be used.

```
Select * from customer
where UPPER(FIRST_NAME) = 'JOSIAH'
AND STATE ='DE';
```

This query uses a function UPPER to check where condition

- **Before Function Indexing:**

```
Select * from customer
where UPPER(FIRST_NAME) = 'JOSIAH'
AND STATE ='DE';
```



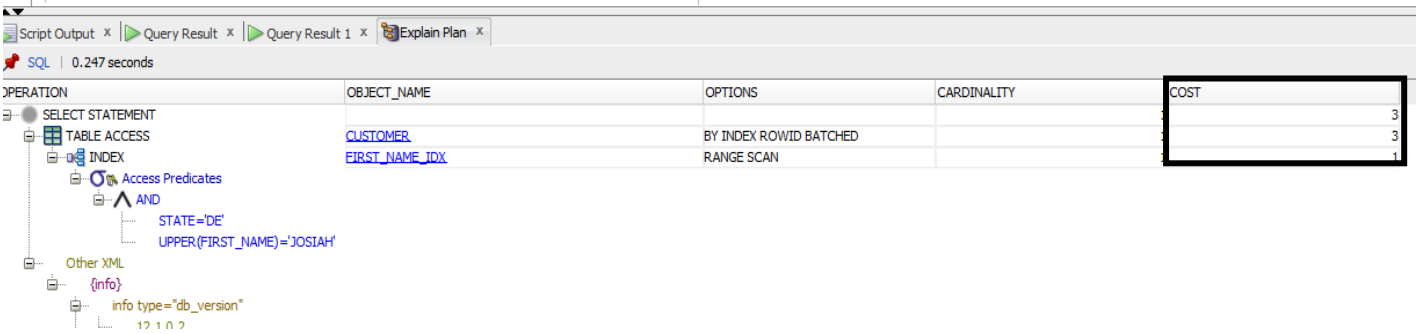
- **Create Index:**

```
CREATE INDEX first_name_idx ON
customer (STATE, UPPER(FIRST_NAME));
```

- **After Function Indexing:**

It can be observed that after Function Indexing Cost has been drastically decreased.

```
Select * from customer
where UPPER(FIRST_NAME) = 'JOSIAH'
AND STATE ='DE';
```

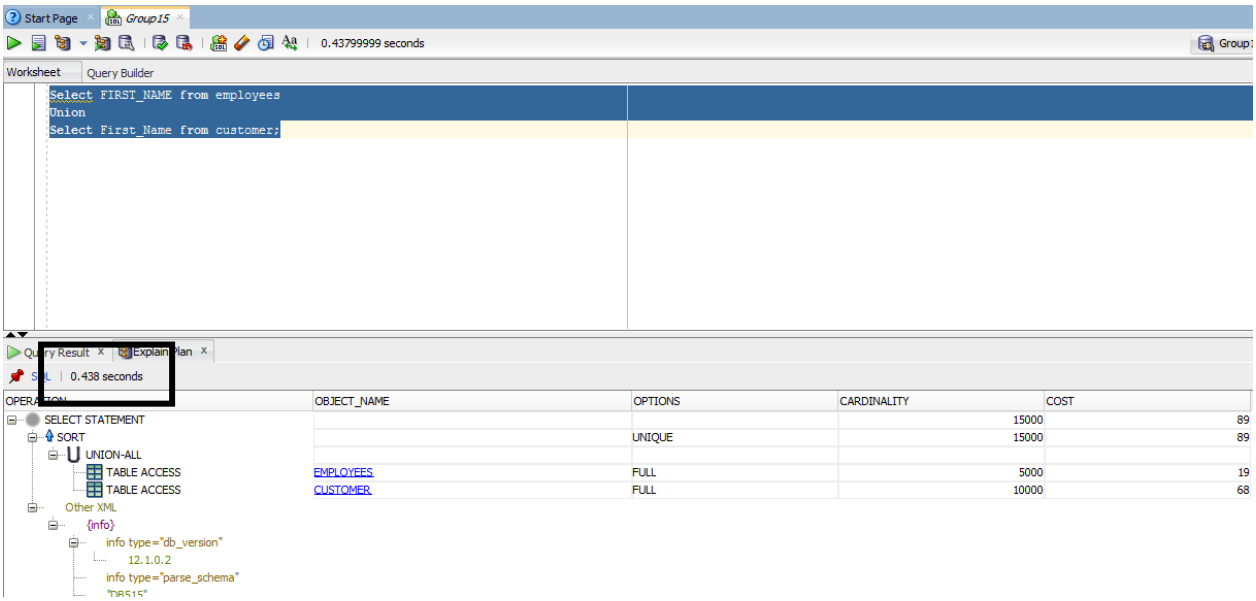


5.3 Use UNION ALL in place of UNION:

- **Union:**

This query selects First Name of Customer and Employees from their respective tables

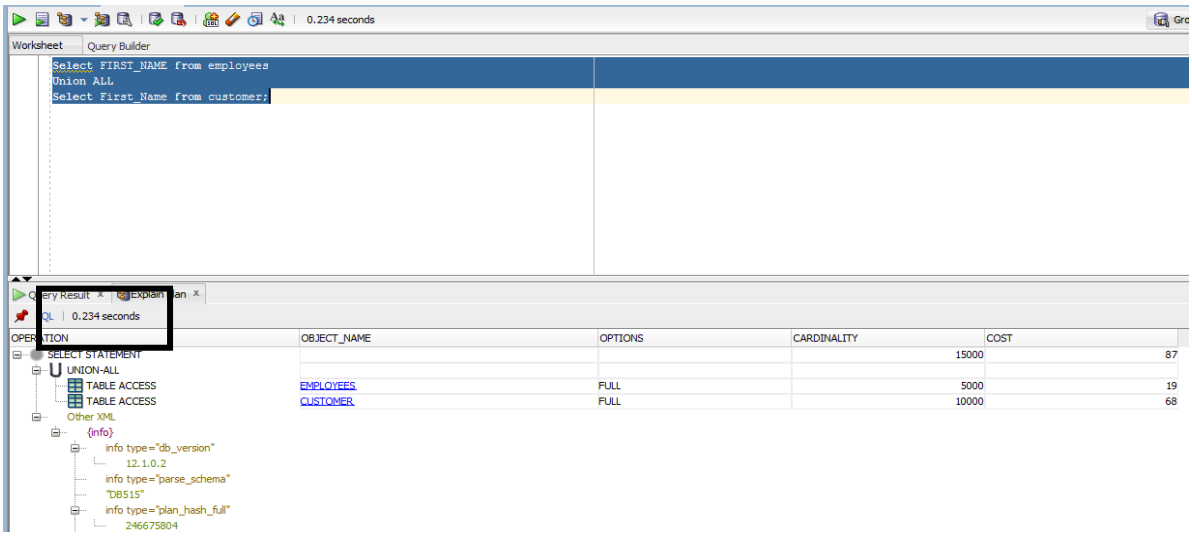
```
Select FIRST_NAME from employees
Union
Select First_Name from customer;
```



- **Union All:**

```
Select FIRST_NAME from employees
Union All
Select First_Name from customer;
```

It can be observed that time taken for Union All is low when compared to Union

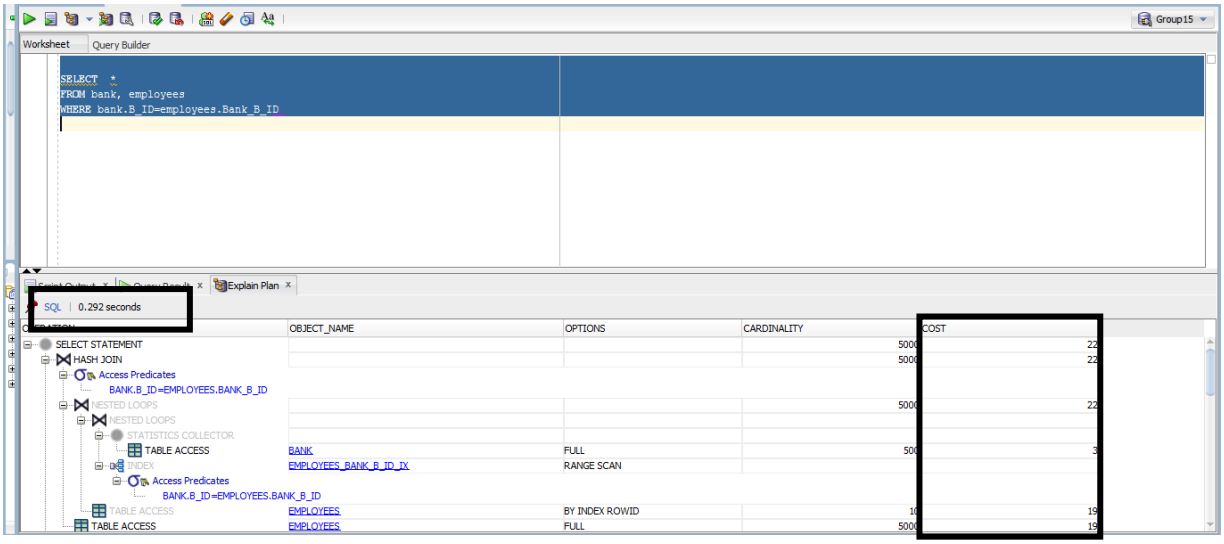


5.4 Parallel Processing:

Parallel SQL enables a SQL statement to be processed by multiple threads or processes simultaneously. Parallel execution performs these operations in parallel using multiple **parallel processes.**
The query used here retrieves the list of bank and employees based on Bank Id's

- **Without Parallelism:**

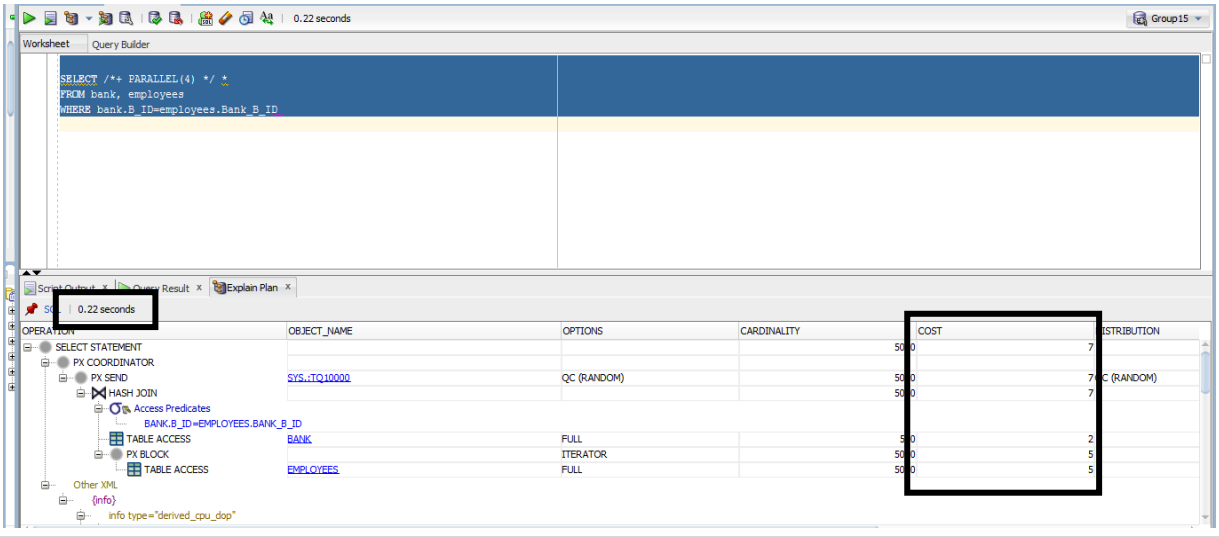
```
SELECT *
FROM bank, employees
WHERE bank.B_ID=employees.Bank_B_ID
```



- **With Parallelism:**

```
SELECT /*+ PARALLEL(4) */ *
FROM bank, employees
WHERE bank.B_ID=employees.Bank_B_ID
```

We can observe that there is considerable reduce in Time as well as cost with the use of Parallel processing. This will be extremely useful when we are dealing with bulk amounts of data.



5.5 Table Partitioning:

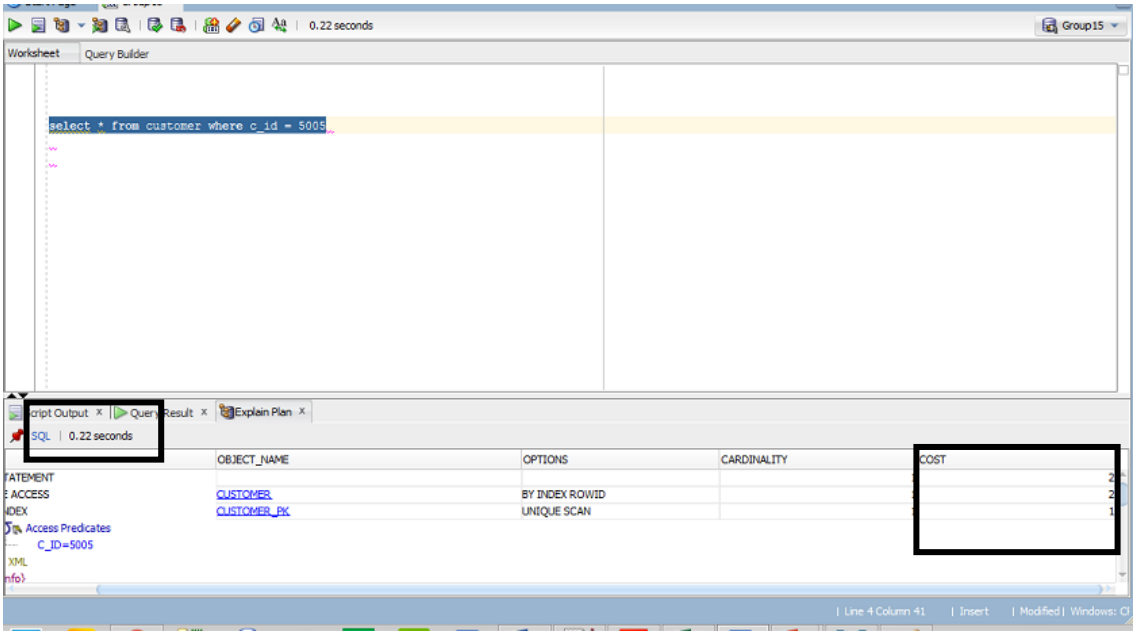
Partitioning allows tables and indexes to be partitioned into smaller, more manageable units, providing database administrators with the ability to pursue a "divide and conquer" approach to data management. With partitioning, maintenance operations can be focused on particular portions of tables.

- **Query:**

A new table is created to illustrate Table Partitioning. The query involved is as follows:

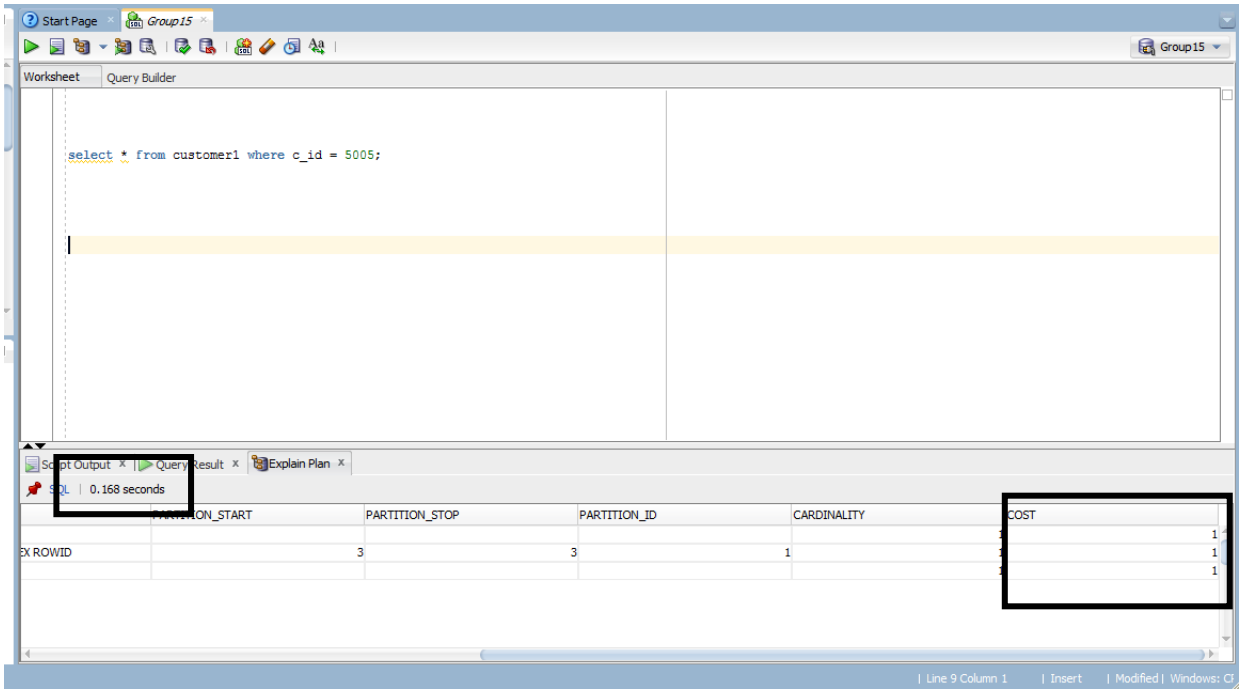
```
CREATE TABLE customer1 (
    c_id                    INTEGER NOT NULL,
    first_name              VARCHAR2(50) NULL,
    last_name               VARCHAR2(50) NOT NULL,
    street                  VARCHAR2(150) NOT NULL,
    city                    VARCHAR2(50) NOT NULL,
    state                   VARCHAR2(50) NOT NULL,
    zipcode                 INTEGER NOT NULL,
    ph_no                   VARCHAR2(50) NULL,
    email_id                VARCHAR2(100) NULL,
    date_and_time_modified  TIMESTAMP NULL,
    CONSTRAINT customer_pk_4 PRIMARY KEY ( c_id )
)
    PARTITION BY RANGE ( c_id ) ( PARTITION p1
        VALUES LESS THAN ( 2000 ),
    PARTITION p2
        VALUES LESS THAN ( 4000 ),
    PARTITION p3
        VALUES LESS THAN ( 6000 ),
    PARTITION p4
        VALUES LESS THAN ( 8000 ),
    PARTITION p5
        VALUES LESS THAN ( 10000 ),
    PARTITION p6
        VALUES LESS THAN ( 12000 )
    );
```

- **Table without Partition:**



- **Table with Partition:**

It can be clearly observed that after implementing partitioning, cost has reduced from 2 to 1 and time to execute the query has reduced from 0.22 to 0.168. This performance would be much higher for complex queries.



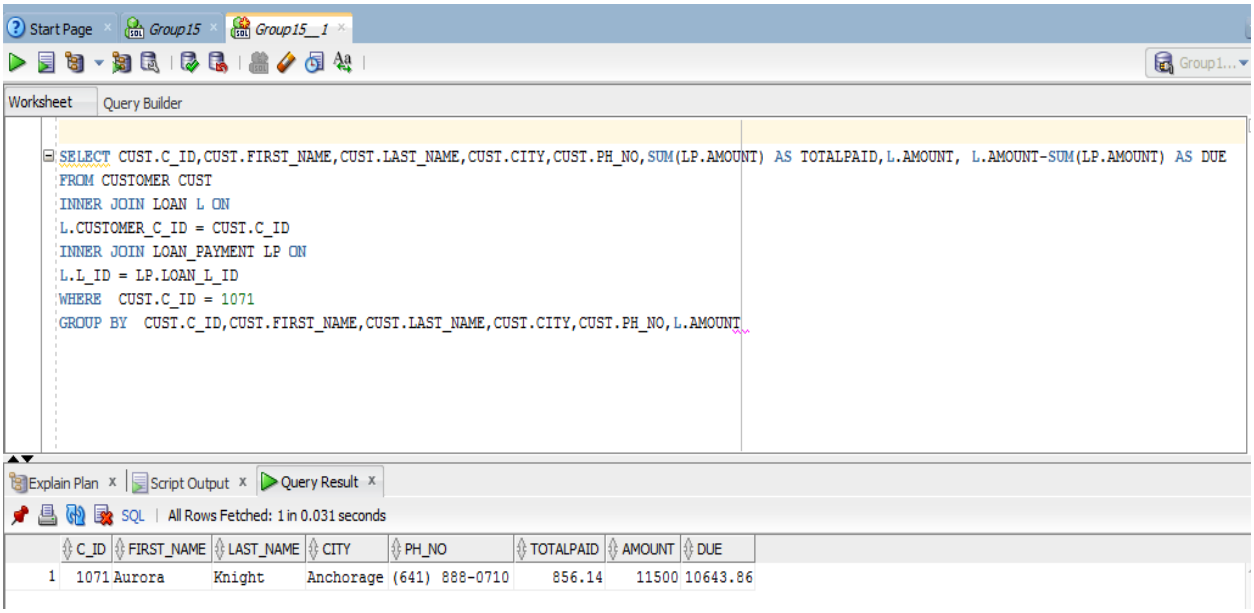
6. Querying

This section covers some useful and interesting queries which demonstrates some of the questions that can be answered by the database. These questions are executed by some internal users for statistical and data mining purposes.

6.1 QUERY 1:

For a given customer ID, return Customer details such as: ID, First Name, Last_Name, City, Phone Number, Loan Amount, Total Paid and Due.

```
SELECT CUST.c_id,
       CUST.first_name,
       CUST.last_name,
       CUST.city,
       CUST.ph_no,
       Sum(LP.amount)              AS TOTALPAID,
       L.amount,
       L.amount - Sum(LP.amount) AS DUE
FROM   customer CUST
       INNER JOIN loan L
               ON L.customer_c_id = CUST.c_id
       INNER JOIN loan_payment LP
               ON L.l_id = LP.loan_l_id
WHERE  CUST.c_id = 1071
GROUP  BY CUST.c_id,
          CUST.first_name,
          CUST.last_name,
          CUST.city,
          CUST.ph_no,
          L.amount
```



6.2 QUERY 2:

Given a customer Id find the bank in which he/she opened an account.

```
SELECT *
FROM   bank B
WHERE  B.b_id = (SELECT bank_b_id
                 FROM   employees
                 WHERE  e_id = (SELECT DISTINCT employees_e_id
                                FROM   account_transactions
                                WHERE  accounts_a_id = (SELECT a_id
                                                        FROM   account
                                                        WHERE
                                       customer_c_id = 1071)))
```

```
SELECT *
FROM   bank B
WHERE  B.b_id = (SELECT bank_b_id
                 FROM   employees
                 WHERE  e_id = (SELECT DISTINCT employees_e_id
                                FROM   account_transactions
                                WHERE  accounts_a_id = (SELECT a_id
                                                        FROM   account
                                                        WHERE
                                       customer_c_id = 1071)));
```

Explain Plan | Script Output | Query Result

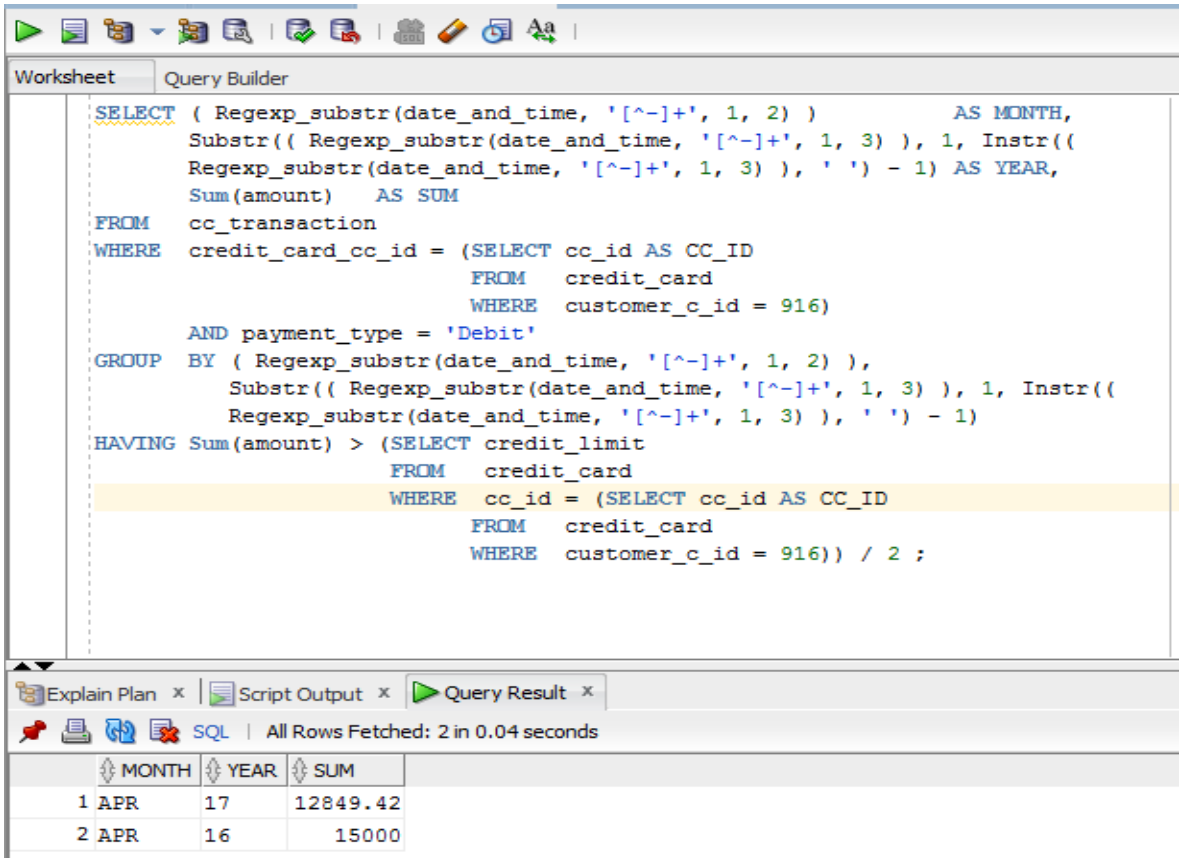
SQL | All Rows Fetched: 1 in 0.04 seconds

	B_ID	BRANCH_NAME	STREET	CITY	STATE	ZIPCODE
1	9861	Blankenberge	P.O. Box 553, 8879 Est Rd.	Jackson	MS	70888

6.3 QUERY 3:

Given a customer Id find all the months in which he/she has used more than 50% of their credit limit. This can be used in analyzing the credit card use and predict whether a customer is a potentially risk customer.

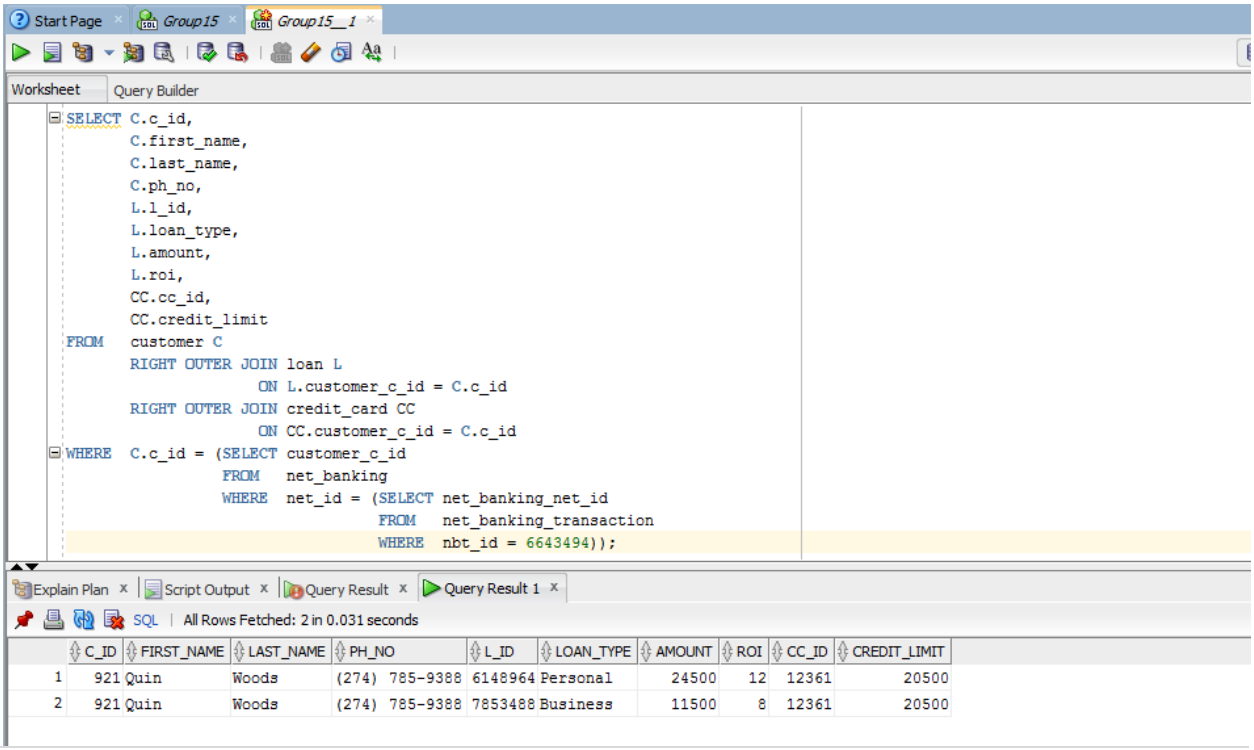
```
SELECT ( Regexp_substr(date_and_time, '[^-]+', 1, 2) )           AS MONTH,
       Substr(( Regexp_substr(date_and_time, '[^-]+', 1, 3) ), 1, Instr((
       Regexp_substr(date_and_time, '[^-]+', 1, 3) ), ' ') - 1) AS YEAR,
       Sum(amount)   AS SUM
FROM   cc_transaction
WHERE  credit_card_cc_id = (SELECT cc_id AS CC_ID
                            FROM   credit_card
                            WHERE  customer_c_id = 916)
       AND payment_type = 'Debit'
GROUP  BY ( Regexp_substr(date_and_time, '[^-]+', 1, 2) ),
          Substr(( Regexp_substr(date_and_time, '[^-]+', 1, 3) ), 1, Instr((
          Regexp_substr(date_and_time, '[^-]+', 1, 3) ), ' ') - 1)
HAVING Sum(amount) > (SELECT credit_limit
                      FROM   credit_card
                      WHERE  cc_id = (SELECT cc_id AS CC_ID
                                      FROM   credit_card
                                      WHERE  customer_c_id = 916)) / 2
```



6.4 QUERY 4:

Given a Net Banking Transaction ID display Customer details, Loan and Credit card details (if customer is having).

```
SELECT C.c_id,
       C.first_name,
       C.last_name,
       C.ph_no,
       L.l_id,
       L.loan_type,
       L.amount,
       L.roi,
       CC.cc_id,
       CC.credit_limit
FROM   customer C
       RIGHT OUTER JOIN loan L
                     ON L.customer_c_id = C.c_id
       RIGHT OUTER JOIN credit_card CC
                     ON CC.customer_c_id = C.c_id
WHERE  C.c_id = (SELECT customer_c_id
                 FROM   net_banking
                 WHERE  net_id = (SELECT net_banking_net_id
                                  FROM   net_banking_transaction
                                  WHERE  nbt_id = 6643494));
```

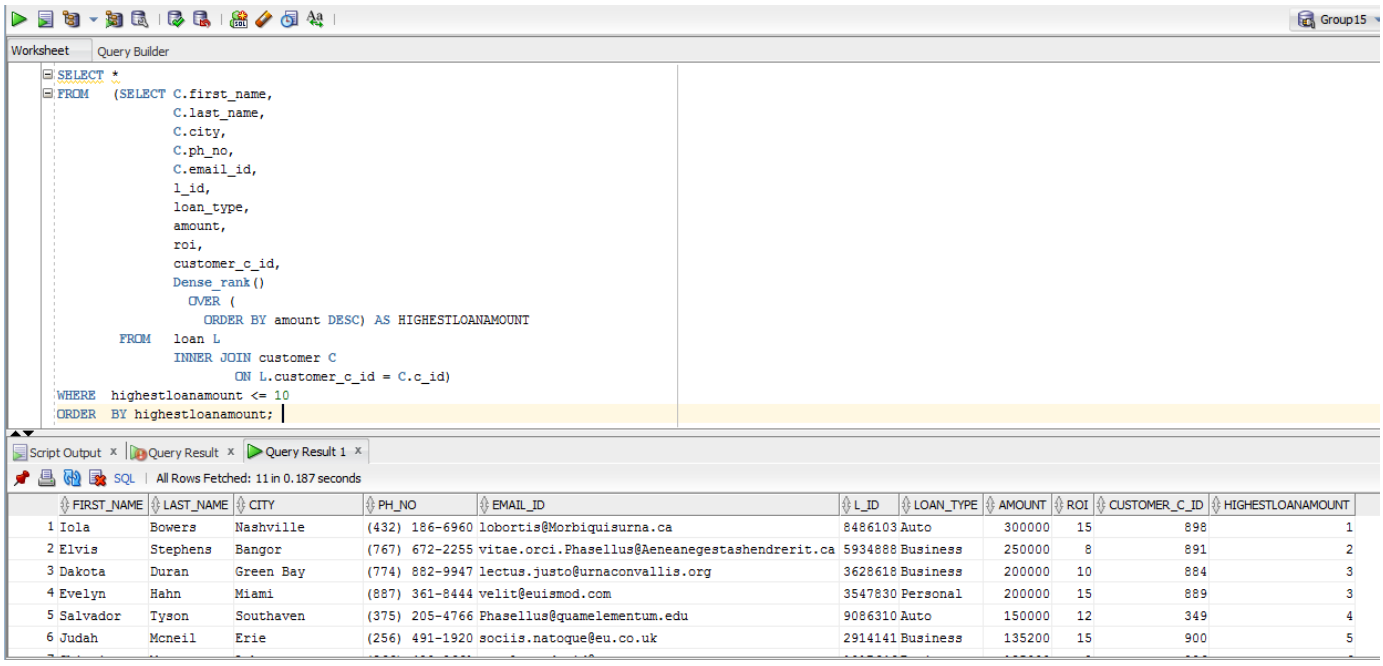


	C_ID	FIRST_NAME	LAST_NAME	PH_NO	L_ID	LOAN_TYPE	AMOUNT	ROI	CC_ID	CREDIT_LIMIT
1	921	Quin	Woods	(274) 785-9388	6148964	Personal	24500	12	12361	20500
2	921	Quin	Woods	(274) 785-9388	7853488	Business	11500	8	12361	20500

6.5 QUERY 5:

Give the details of the customers who have taken Top 10 loans (i.e. highest loan amount) from the bank. This data can be used to monitor the monthly payments of the customers who has taken huge amount of loans.

```
SELECT *
FROM   (SELECT C.first_name,
               C.last_name,
               C.city,
               C.ph_no,
               C.email_id,
               l_id,
               loan_type,
               amount,
               roi,
               customer_c_id,
               Dense_rank()
                 OVER (
                   ORDER BY amount DESC) AS HIGHESTLOANAMOUNT
        FROM   loan L
               INNER JOIN customer C
                       ON L.customer_c_id = C.c_id)
WHERE  highestloanamount <= 10
ORDER  BY highestloanamount;
```

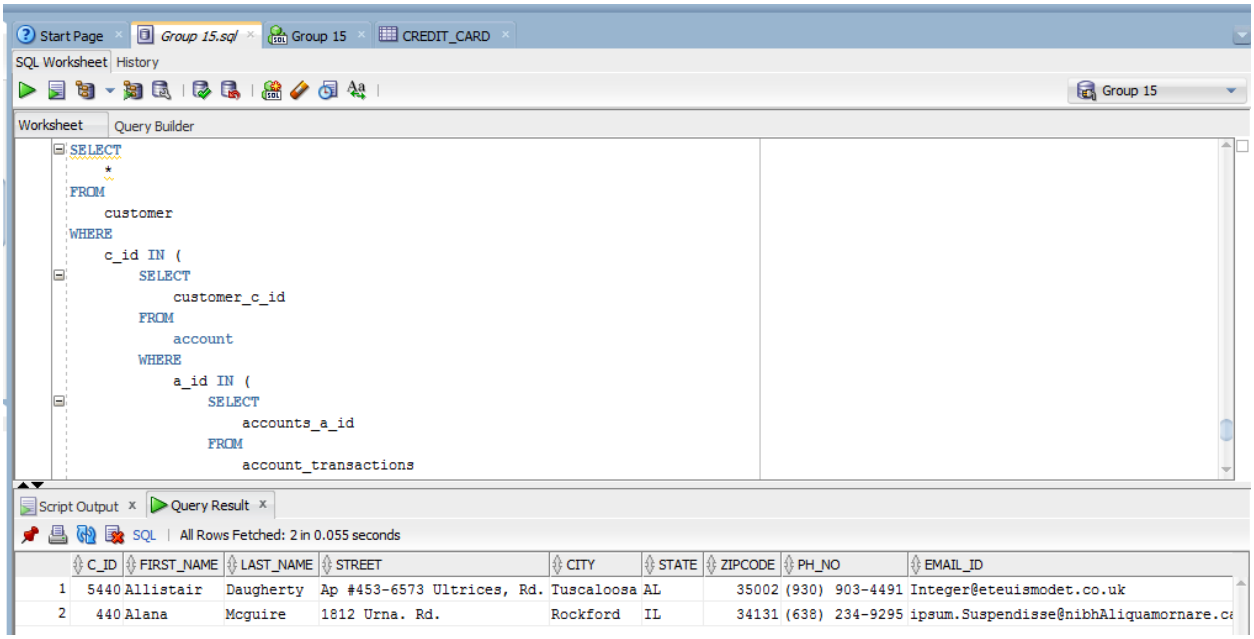


	FIRST_NAME	LAST_NAME	CITY	PH_NO	EMAIL_ID	L_ID	LOAN_TYPE	AMOUNT	ROI	CUSTOMER_C_ID	HIGHESTLOANAMOUNT
1	Iola	Bowers	Nashville	(432) 186-6960	lobortis@Morbiquisurna.ca	8486103	Auto	300000	15	898	1
2	Elvis	Stephens	Bangor	(767) 672-2255	vitae.orci.Phasellus@Aeneanegestashendrerit.ca	5934888	Business	250000	8	891	2
3	Dakota	Duran	Green Bay	(774) 882-9947	lectus.justo@urnaconvallis.org	3628618	Business	200000	10	884	3
4	Evelyn	Hahn	Miami	(887) 361-8444	velit@euismod.com	3547830	Personal	200000	15	889	3
5	Salvador	Tyson	Southaven	(375) 205-4766	Phasellus@quamelementum.edu	9086310	Auto	150000	12	349	4
6	Judah	Mcneil	Erie	(256) 491-1920	sociis.natoque@eu.co.uk	2914141	Business	135200	15	900	5

6.6 QUERY 6:

This query returns all the customers whom an employee is serving if the first name and last name of that employee is given.

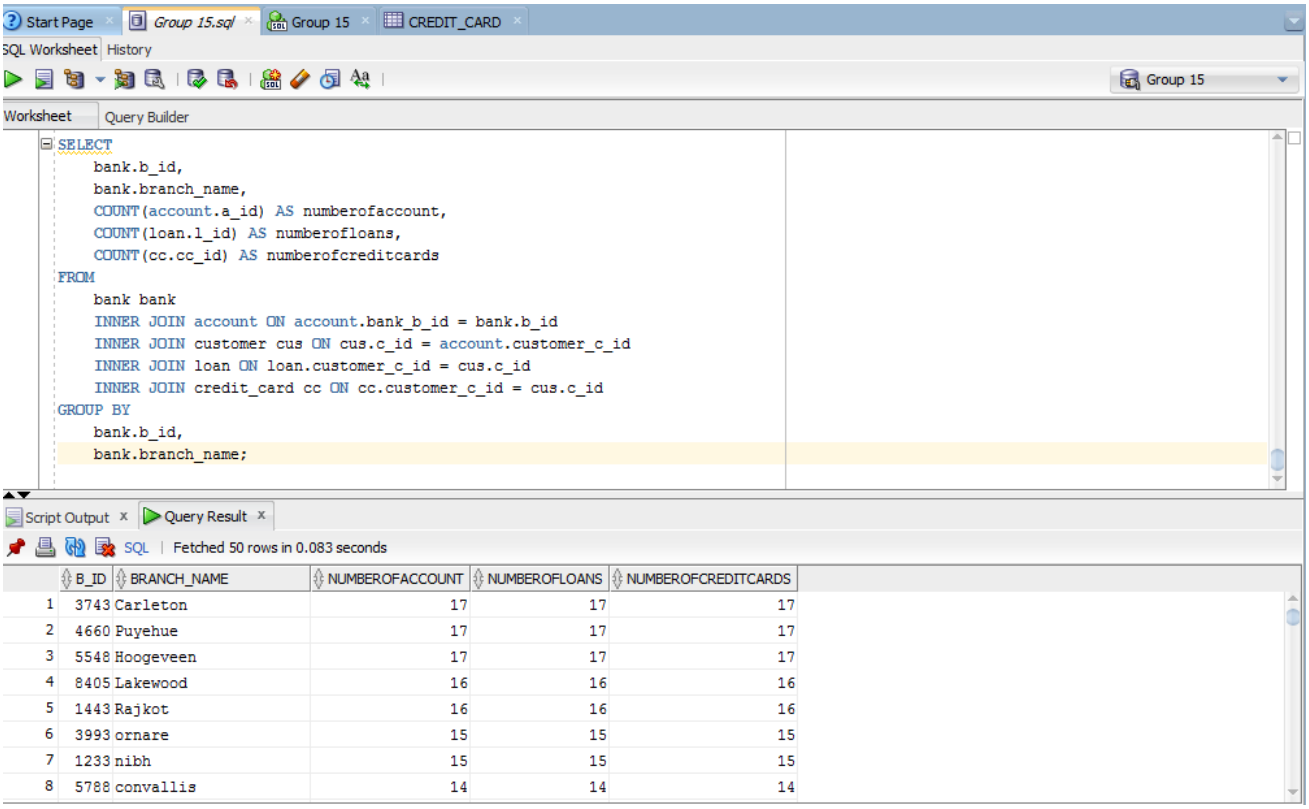
```
SELECT
    *
FROM
    customer
WHERE
   c_id IN (
       SELECT
           customer_c_id
       FROM
           account
       WHERE
           a_id IN (
               SELECT
                   accounts_a_id
               FROM
                   account_transactions
               WHERE
                   employees_e_id = (
                       SELECT
                           e_id
                       FROM
                           employees
                       WHERE
                               first_name = 'Xandra'
                           AND
                               last_name = 'Little'
                   )
           )
   );
```



6.7 QUERY 7:

This query returns total number of Accounts, Loans and Credit Cards issued by each and every branch of the Bank.

```
SELECT
    bank.b_id,
    bank.branch_name,
    COUNT(account.a_id) AS numberofaccount,
    COUNT(loan.l_id) AS numberofloans,
    COUNT(cc.cc_id) AS numberofcreditcards
FROM
    bank
    INNER JOIN account ON account.bank_b_id = bank.b_id
    INNER JOIN customer cus ON cus.c_id = account.customer_c_id
    INNER JOIN loan ON loan.customer_c_id = cus.c_id
    INNER JOIN credit_card cc ON cc.customer_c_id = cus.c_id
GROUP BY
    bank.b_id,
    bank.branch_name;
```



Start Page | Group 15.sql | Group 15 | CREDIT_CARD

SQL Worksheet | History

Group 15

Worksheet | Query Builder

```
SELECT
    bank.b_id,
    bank.branch_name,
    COUNT(account.a_id) AS numberofaccount,
    COUNT(loan.l_id) AS numberofloans,
    COUNT(cc.cc_id) AS numberofcreditcards
FROM
    bank bank
    INNER JOIN account ON account.bank_b_id = bank.b_id
    INNER JOIN customer cus ON cus.c_id = account.customer_c_id
    INNER JOIN loan ON loan.customer_c_id = cus.c_id
    INNER JOIN credit_card cc ON cc.customer_c_id = cus.c_id
GROUP BY
    bank.b_id,
    bank.branch_name;
```

Script Output | Query Result

SQL | Fetched 50 rows in 0.083 seconds

	B_ID	BRANCH_NAME	NUMBEROFACCOUNT	NUMBEROFLOANS	NUMBEROFCREDITCARDS
1	3743	Carleton	17	17	17
2	4660	Puyehue	17	17	17
3	5548	Hoogeveen	17	17	17
4	8405	Lakewood	16	16	16
5	1443	Rajkot	16	16	16
6	3993	ornare	15	15	15
7	1233	nibh	15	15	15
8	5788	convallis	14	14	14

7. DBA Scripts

7.1 Memory allocations:

It displays the memory allocations for the current database sessions. In this case, there are two data base sessions running and the DBA script and the memory allocations for each of those is as follows:

```
SET LINESIZE 200


COLUMN username FORMAT A20
COLUMN module FORMAT A20


SELECT NVL(a.username,'(oracle)') AS username,
       a.module,
       a.program,
       Trunc(b.value/1024) AS memory_kb
FROM   v$session a,
       v$sesstat b,
       v$statname c
WHERE  a.sid = b.sid
AND    b.statistic# = c.statistic#
AND    c.name = 'session pga memory'
AND    a.program IS NOT NULL
ORDER BY b.value DESC;
```

- **Output:**

	USERNAME	MODULE	PROGRAM	MEMORY_KB
1	DB515	SQL Developer	SQL Developer	2928
2	DB515	SQL Developer	SQL Developer	1729

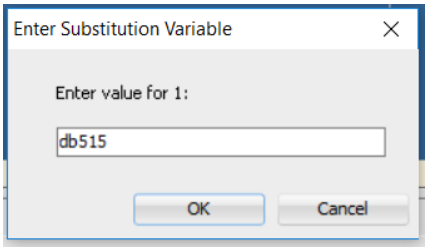
7.2 Table Indexes:

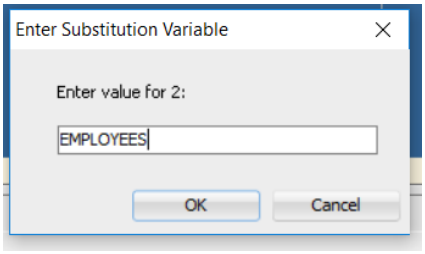
It displays the information about the columns and indexes for a specified table.

- Here, the indexes for the table Employees are retrieved as follows:

```
SET LINESIZE 500 PAGESIZE 1000 VERIFY OFF

COLUMN index_name      FORMAT A30
COLUMN column_name     FORMAT A30
COLUMN column_position FORMAT 99999

SELECT a.index_name,
       a.column_name,
       a.column_position
FROM   all_ind_columns a,
       all_indexes b
WHERE  b.owner      = UPPER('&1')
AND    b.table_name = UPPER('&2')
AND    b.index_name = a.index_name
AND    b.owner      = a.index_owner
ORDER BY 1,3;
```

- The owner in this case is our group username 'db515' and the table is 'Employees'





- **Output:**

And the below indexes are retrieved for this table:

	INDEX_NAME	COLUMN_NAME	COLUMN_POSITION
1	EMPLOYEES_BANK_B_ID_IX	BANK_B_ID	1
2	EMPLOYEES_PK	E_ID	1

7.3 Redo log files Script:

It displays the information about the redo log files.

```
SET LINESIZE 200
COLUMN member FORMAT A50
COLUMN first_change# FORMAT 99999999999999999999
COLUMN next_change# FORMAT 99999999999999999999

SELECT l.thread#,
       lf.group#,
       lf.member,
       TRUNC(l.bytes/1024/1024) AS size_mb,
       l.status,
       l.archived,
       lf.type,
       lf.is_recovery_dest_file AS rdf,
       l.sequence#,
       l.first_change#,
       l.next_change#
FROM   v$logfile lf
       JOIN v$log l ON l.group# = lf.group#
ORDER BY l.thread#,lf.group#, lf.member;
```

- **Output:**

The redo log files information such as the status, threads, changes, type and sequence are retrieved as follows:

	THREAD#	GROUP#	MEMBER	SIZE_MB	STATUS	ARCHIVED	TYPE	RDF	SEQUENCE#	FIRST_CHANGE#	NEXT_CHANGE#
1	1	1	D:\APP\ORACLE\ORADATA\CDB9\REDO01.LOG	50	INACTIVE	NO	ONLINE	NO	10300	102057087	102061054
2	1	2	D:\APP\ORACLE\ORADATA\CDB9\REDO02.LOG	50	INACTIVE	NO	ONLINE	NO	10301	102061054	102087573
3	1	3	D:\APP\ORACLE\ORADATA\CDB9\REDO03.LOG	50	CURRENT	NO	ONLINE	NO	10302	102087573	281474976710655

7.4 Cache-hit ratio Script:

It displays the Cache-hit ratio for the database. It varies depending on the system, but the minimum ratio should be 89%.

```
PROMPT
PROMPT Hit ratio should exceed 89%

SELECT Sum(Decode(a.name, 'consistent gets', a.value, 0)) "Consistent Gets",
       Sum(Decode(a.name, 'db block gets', a.value, 0)) "DB Block Gets",
       Sum(Decode(a.name, 'physical reads', a.value, 0)) "Physical Reads",
       Round(((Sum(Decode(a.name, 'consistent gets', a.value, 0)) +
         Sum(Decode(a.name, 'db block gets', a.value, 0)) -
         Sum(Decode(a.name, 'physical reads', a.value, 0))  )/
           (Sum(Decode(a.name, 'consistent gets', a.value, 0)) +
             Sum(Decode(a.name, 'db block gets', a.value, 0))))
             *100,2) "Hit Ratio %"
FROM   v$sysstat a;
```

- **Output:**

The information such as consistent gets, physical reads and the hit ratio appears as follows:

	Consistent Gets	DB Block Gets	Physical Reads	Hit Ratio %
1	16091895170	65372797	14512688	99.91

7.5 Library Cache Script:

It displays the library cache statistics

```
SET LINESIZE 500
SET PAGESIZE 1000
SET VERIFY OFF

SELECT a.namespace "Name Space",
       a.gets "Get Requests",
       a.gethits "Get Hits",
       Round(a.gethitratio,2) "Get Ratio",
       a.pins "Pin Requests",
       a.pinhits "Pin Hits",
       Round(a.pinhitratio,2) "Pin Ratio",
       a.reloads "Reloads",
       a.invalidations "Invalidations"
FROM   v$librarycache a
ORDER BY 1;

SET PAGESIZE 14
SET VERIFY ON
```

- **Output:**

All the information such as the get requests, get hits, pin requests, pin hits etc. are displayed for all the name spaces.

	Name Space	Get Requests	Get Hits	Get Ratio	Pin Requests	Pin Hits	Pin Ratio	Reloads	Invalidations
1	ACCOUNT_STATUS	21043	19973	0.95	0	0	1	0	0
2	APP CONTEXT	3	0	0	5	2	0.4	0	0
3	AUDIT POLICY	11262	11048	0.98	11262	11047	0.98	0	0
4	BODY	274604	271214	0.99	923303	918534	0.99	1192	1
5	CLUSTER	92755	92301	1	203918	203463	1	1	0
6	DBINSTANCE	1	0	0	0	0	1	0	0
7	DBLINK	22412	22310	1	0	0	1	0	0
8	DIRECTORY	9	6	0.67	9	6	0.67	0	0
9	EDITION	90706	90696	1	173800	173777	1	4	0
10	HINTSET OBJECT	220	161	0.73	220	155	0.7	0	0
11	INDEX	85338	78126	0.92	81033	63047	0.78	5699	0
12	JAVA DATA	17	2	0.12	930	914	0.98	0	0
13	JAVA RESOURCE	11	2	0.18	11	2	0.18	0	0
14	JAVA SOURCE	11	2	0.18	11	2	0.18	0	0
15	OBJECT ID	9159	0	0	0	0	1	0	0

8. Database Programming

This section highlights the stored procedures and functions that were created to serve some of the real-time applications of the banking domain.

8.1 Stored Procedure:

8.1.1 Stored Procedure to update password:

In the real time, during net banking, if a user forgets his/her password or wants to update his/her password then, her/she will select “Forgot Password” Tab and updates it. System does this by executing the following Stored Procedure.

This stored procedure takes in 2 parameters, Net_ID and Customer_ID, and validates if they match with each other and with the database.

- If they match, it updates the password in the database accordingly and returns “Successfully Updated”.
- If they do not match, then it returns “Enter valid Net_Id and Customer_Id”

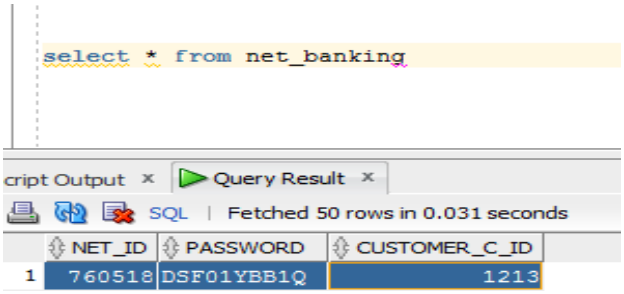
SP_UPDATEPASSOWORD:

```
CREATE OR REPLACE PROCEDURE sp_updatepassword (
    p_net_id        INT,
    p_customer_id   INT,
    p_password      VARCHAR2,
    p_retval        OUT VARCHAR2
)
    AS
BEGIN
    UPDATE net_banking
        SET
            password = p_password
    WHERE
            net_id = p_net_id
        AND
            customer_c_id = p_customer_id;

    IF
        ( SQL%rowcount >= 1 )
    THEN
        p_retval := 'Successfully Updated';
    ELSE
        p_retval := 'Enter valid Net_Id and Customer_Id';
    END IF;

END sp_updatepassword;
```

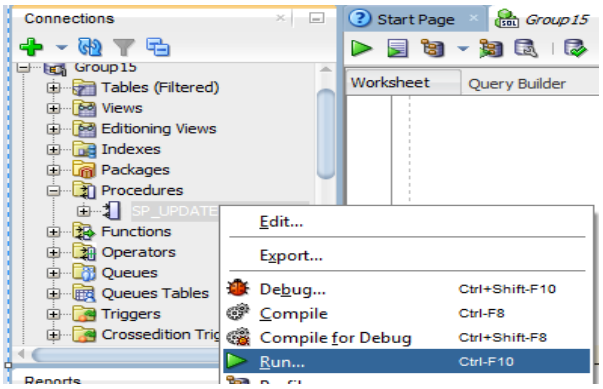
- **Illustration:**

The following example illustrates how the above stored procedure is implemented:

- Let us consider below are the user details to update their password based on Net_ID and Customer_ID:

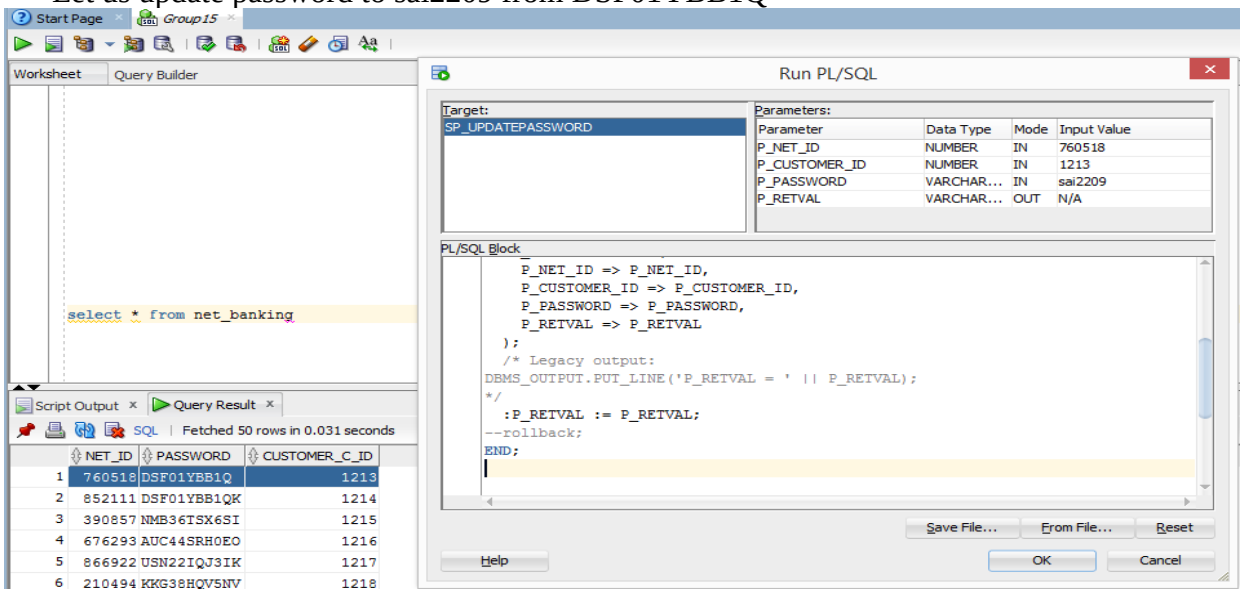


- Run the SP_UPDATEPASSWORD

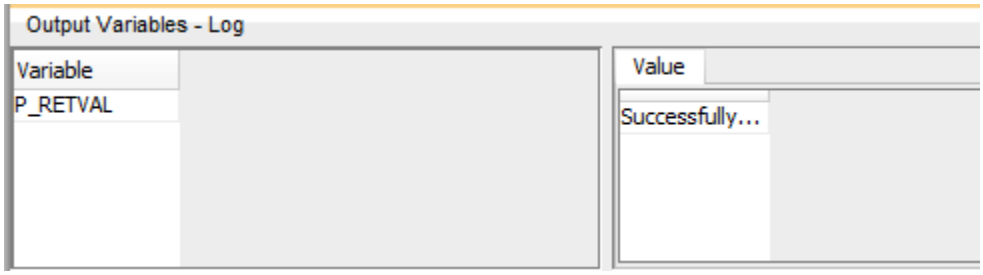


- Enter Valid details:

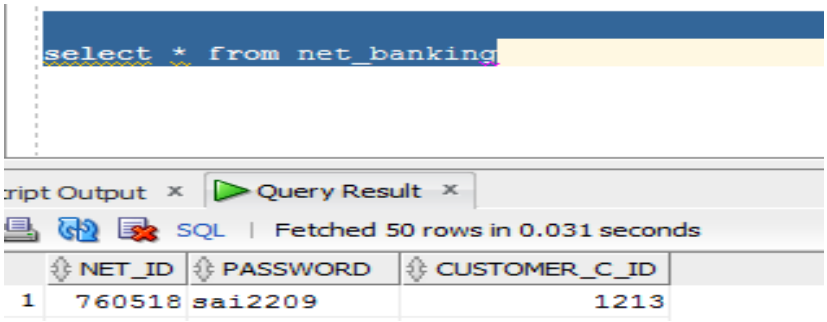
Let us update password to sai2209 from DSF01YBB1Q



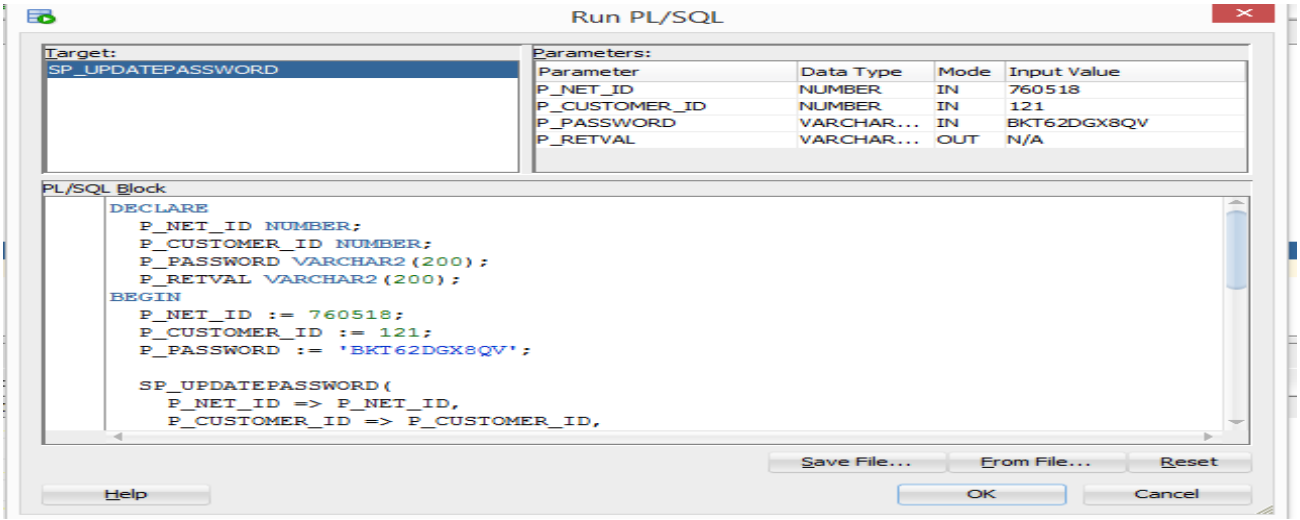
- Successfully Executed:



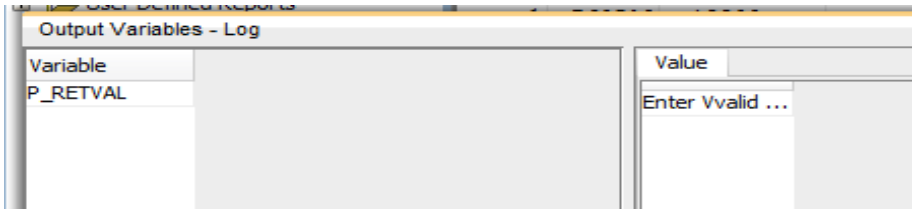
- Password updated in database



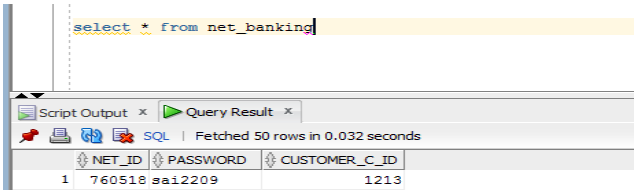
- If invalid credentials are entered**:**



- It fails to update the password:



- Password is not updated in the database due to invalid customer ID.



8.1.2 Stored Procedure to insert values into the BANK table:

This stored procedure checks whether a given branch already exists in the database and returns:

- 'Successfully Inserted' after inserting values into the database, if it do not exist.
- 'Bank Already Exists' if given branch already exists in the database.

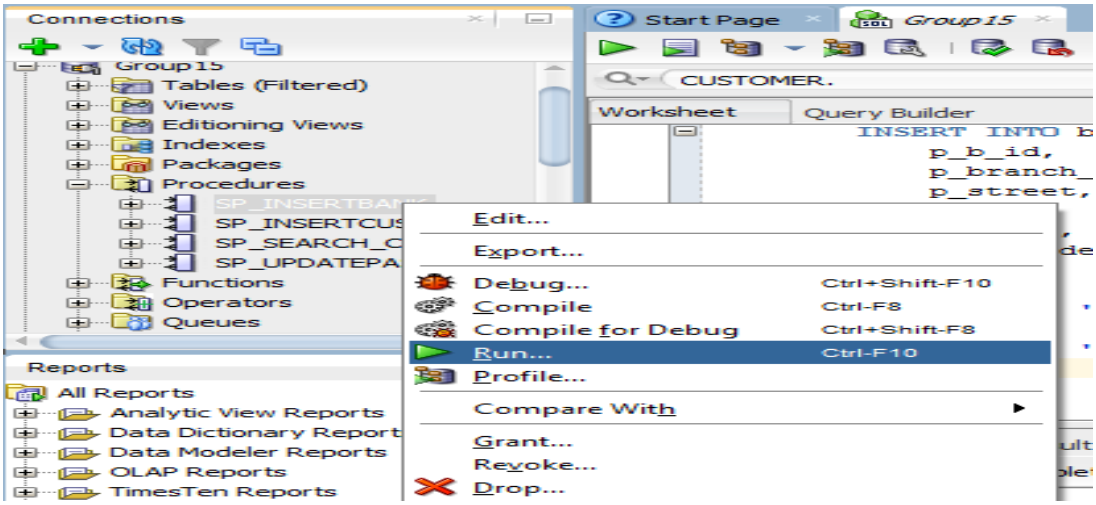
SP_INSERTBANK:

```
CREATE OR REPLACE PROCEDURE sp_insertbank (
    p_b_id          INT,
    p_branch_name   VARCHAR2,
    p_street        VARCHAR2,
    p_city          VARCHAR2,
    p_state         VARCHAR2,
    p_zipcode       INT,
    p_retval        OUT VARCHAR2
) AS
    bankexists   NUMBER;
BEGIN
    SELECT
        COUNT(*)
    INTO
        bankexists
    FROM
        bank
    WHERE
        b_id = p_b_id;

    IF
        bankexists = 0
    THEN
        INSERT INTO bank VALUES (
            p_b_id,
            p_branch_name,
            p_street,
            p_city,
            p_state,
            p_zipcode
        );

        p_retval := 'Successfully Inserted';
    ELSE
        p_retval := 'Bank Already Exists';
    END IF;

END sp_insertbank;
```

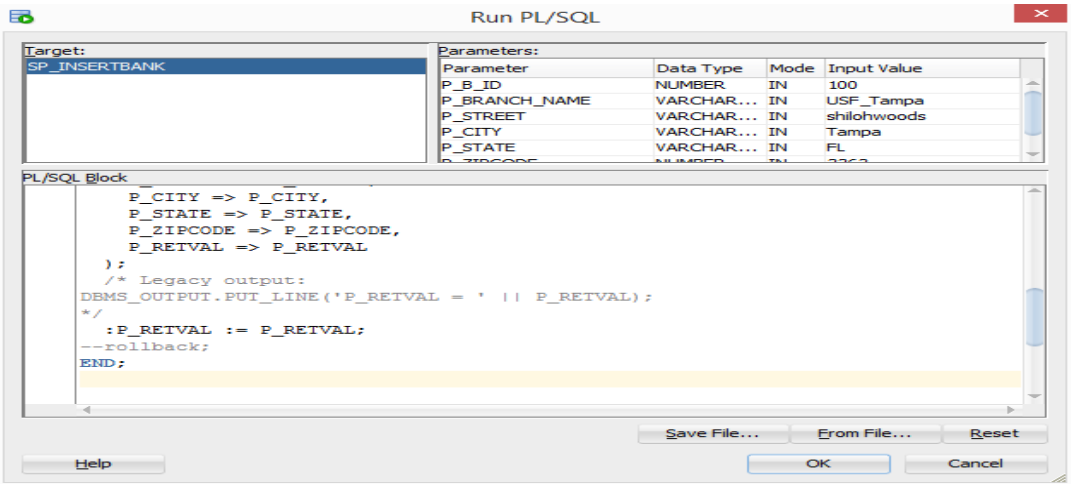
- **Illustration:**

The following example illustrates how the above stored procedure is implemented:

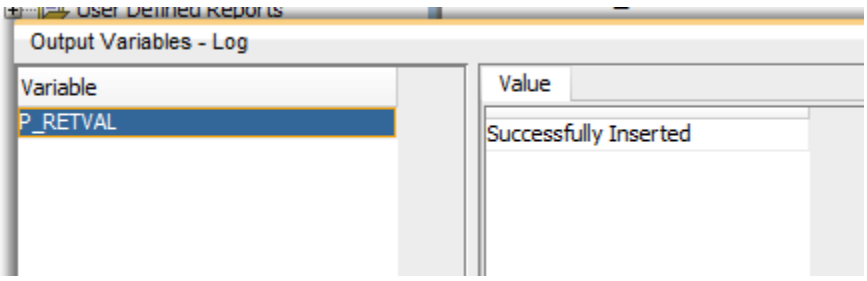
- Run Stored Procedure:



- Insert New data (if bank does not exist in database):



- Stored Procedure returned **"Successfully Inserted"**



- Record successfully inserted into bank table

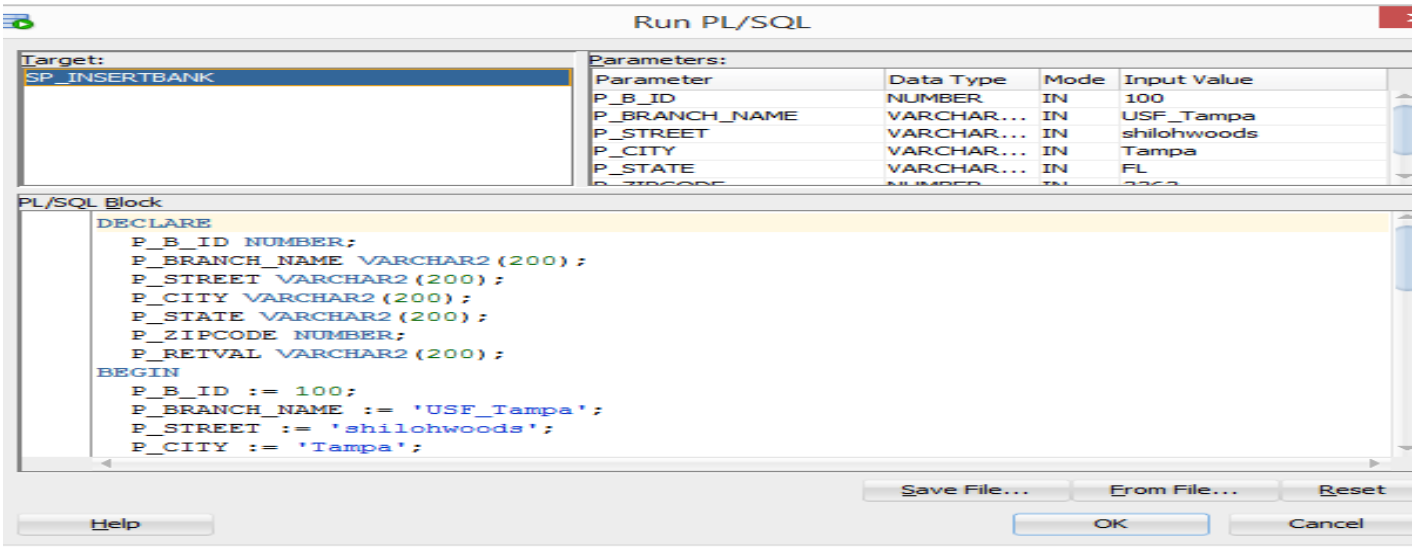
```
select * from bank where b_id =100
```

ript Output × | Query Result ×

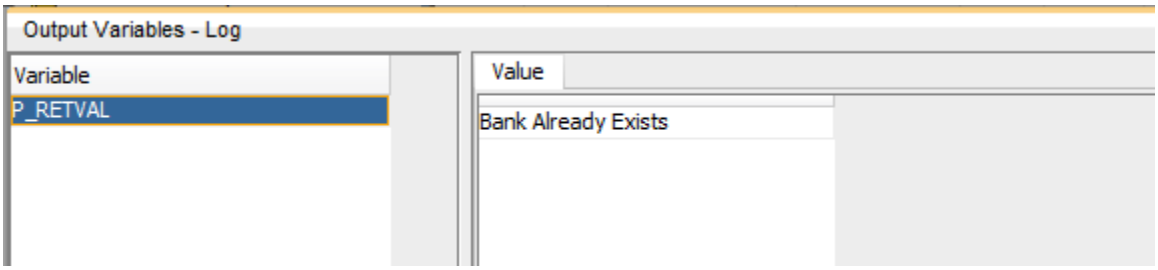
SQL | All Rows Fetched: 1 in 0.031 seconds

	B_ID	BRANCH_NAME	STREET	CITY	STATE	ZIPCODE
1	100	USF_Tampa	shilohwoods	Tampa	FL	3363

- Let us try to enter the same record again:



- Stored Procedure returned **"Bank Already Exists"** and does not allow to insert the record



8.2 Functions:

8.2.1. Function to authenticate User credentials to login:

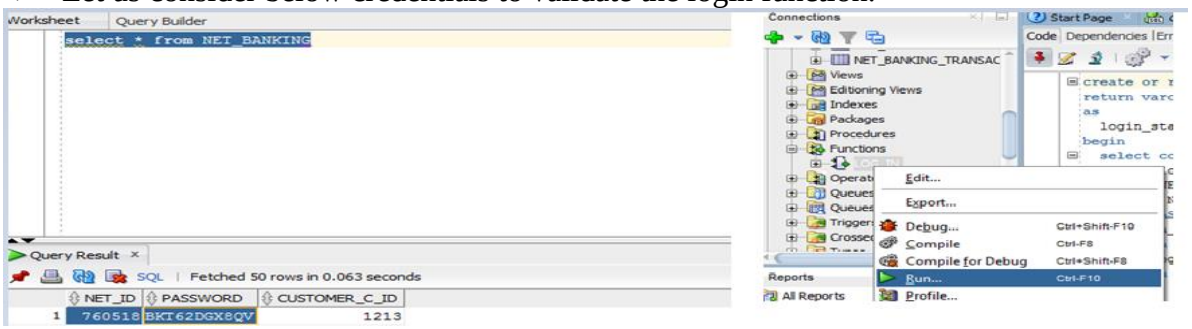
Below is the function which authenticates Net_Id and Password of a user to login into the system. Function accepts 2 parameters, Net_Id and Password and then returns:

8 'Login successful!' - if Net_Id and Password matches
9 'Wrong username or password!' – if Net_Id and Password do not match
10 'Multiple Users!!!' – if multiple users with same credentials are found (ideally not possible)

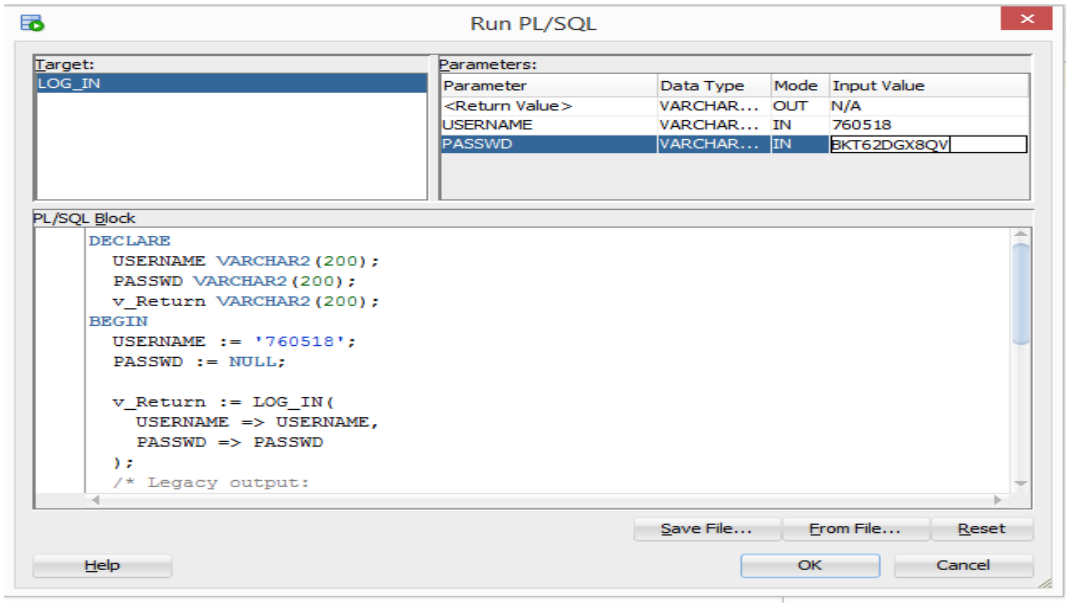
```
CREATE FUNCTION log_in (
    username   IN VARCHAR2,
    passwd     IN VARCHAR2
) RETURN VARCHAR2 AS
    login_status   NUMBER;
BEGIN
    SELECT
        COUNT(*)
    INTO
        login_status
    FROM
        net_banking
    WHERE
            net_id = username
        AND
            password = passwd;

    IF
        login_status = 0
    THEN
        RETURN 'Wrong username or password!';
    ELSIF login_status = 1 THEN
        RETURN 'Login successful!';
    ELSE
        RETURN 'Multiple Users!!!';
    END IF;

END;
```

- **Function Execution:**

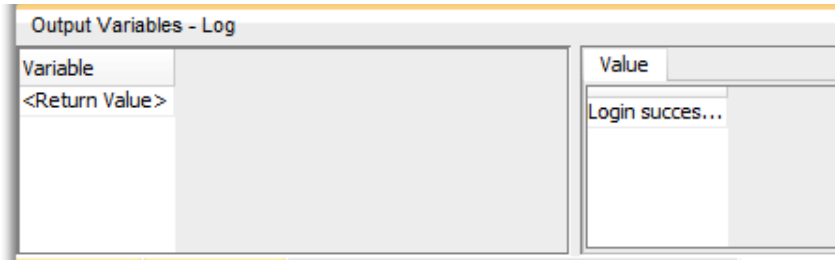
  - Let us consider below credentials to validate the login function.



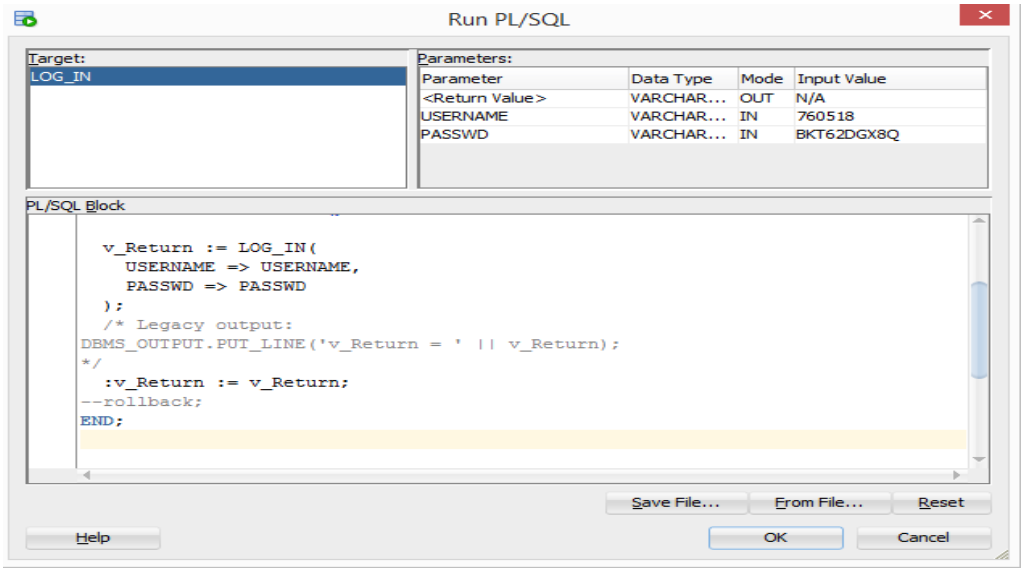
- Valid credentials are entered:



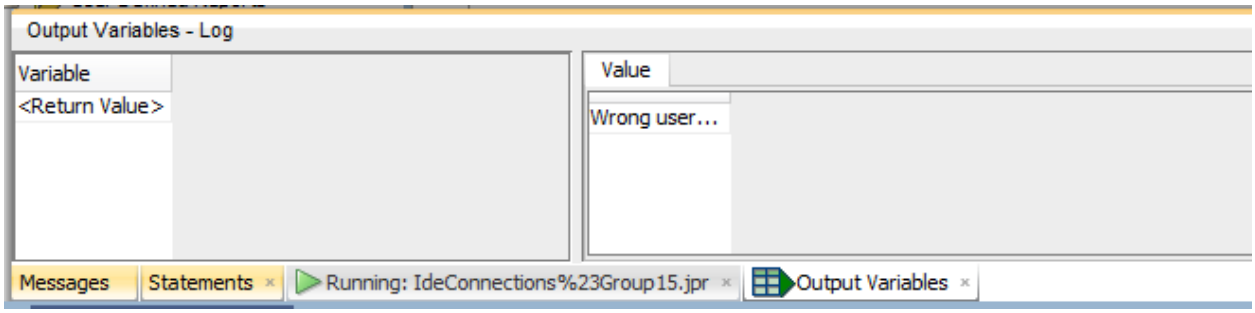
- Login is successful:



- If invalid credentials are entered:



- Login fails and returns: “Wrong username or password”



8.2.2 Function to allow the Credit Card transactions by checking remaining credit limits.

This function takes in 2 parameters, Customer_ID and requested Amount, and returns:

- “Transaction success!”- if user has enough funds in his credit limit.
- “Insufficient Funds!!!” – if user does not have enough funds for the current month.

```
CREATE OR REPLACE FUNCTION check_credit_balance (
    amount        IN INT,
    customer_id   IN INT
) RETURN VARCHAR2 AS
    amountspent   NUMBER;
    totalcredit   NUMBER;
BEGIN
    SELECT
        SUM(amount) AS sum
    INTO
        amountspent
    FROM
        cc_transaction
    WHERE
            credit_card_cc_id = (
                SELECT
                    cc_id AS cc_id
                FROM
                    credit_card
                WHERE
                    customer_c_id = customer_id
            )
        AND
            payment_type = 'Debit'
        AND
            regexp_substr(
                date_and_time,
                '[^-]+',
                1,
                2
            ) = regexp_substr(
                (
                    SELECT
```

```
                    systimestamp
                FROM
                    dual
            ),
            '[^-]+',
            1,
            2
        )
    AND
        substr(
            (regexp_substr(
                date_and_time,
                '[^-]+',
                1,
                3
            ) ),
            1,
            instr(
                (regexp_substr(
                    date_and_time,
                    '[^-]+',
                    1,
                    3
                ) ),
                ' '
            ) - 1
        ) = substr(
            (regexp_substr(
                (
                    SELECT
                        systimestamp
                    FROM
                        dual
                ),
                '[^-]+',
                1,
                3
            ) ),
            1,
            instr(
                (regexp_substr(
                    date_and_time,
                    '[^-]+',
                    1,
                    3
                ) ),
                ' '
            ) - 1
        )
GROUP BY
    ( regexp_substr(
        date_and_time,
        '[^-]+',
        1,
        2
    ) ),
```

```
        substr(
            (regexp_substr(
                date_and_time,
                '[^-]+',
                1,
                3
            ) ),
            1,
            instr(
                (regexp_substr(
                    date_and_time,
                    '[^-]+',
                    1,
                    3
                ) ),
                ' '
            ) - 1
        );

    SELECT
        credit_limit
    INTO
        totalcredit
    FROM
        credit_card
    WHERE
        customer_c_id = customer_id;

    IF
        amount < totalcredit - amountspent
    THEN
        RETURN 'TRANSACTION SUCCESS!';
    ELSE
        RETURN 'INSUFFICIENT FUNDS!!!';
    END IF;
END;
```

- **Function Execution**
  - Let us consider customer id = 927 having credit limit of 23500;

```
select * from credit_card where customer_c_id = 927;
```

cript Output | Query Result

SQL | All Rows Fetched: 1 in 0.032 seconds

	CC_ID	Number	CREDIT_LIMIT	DATE_AND_TIME	PIN	CUSTOMER_C_ID
1	36065	3427 476111 30025	23500	03-OCT-16 09.38.04.000000000 AM	5035	927

- From credit card transaction table, it can be observed that user has spent 2300 in the current month(01-NOV-17)

```
select * from cc_transaction where credit_card_cc_id = 36065;
```

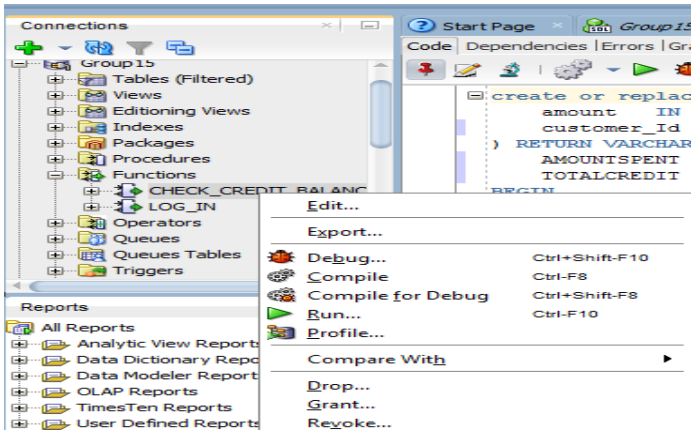
Query Result | SQL | All Rows Fetched: 3 in 0.031 seconds

	PAYMENT_ID	PAYMENT_TYPE	DATE_AND_TIME	AMOUNT	CREDIT_CARD_CC_ID
1	1214003	Debit	16-JUN-17 09.17.26.000000000 AM	4299.9	36065
2	8612445	Debit	13-MAY-17 09.28.07.000000000 AM	2576.48	36065
3	121400	Debit	01-NOV-17 09.17.26.000000000 AM	23000	36065

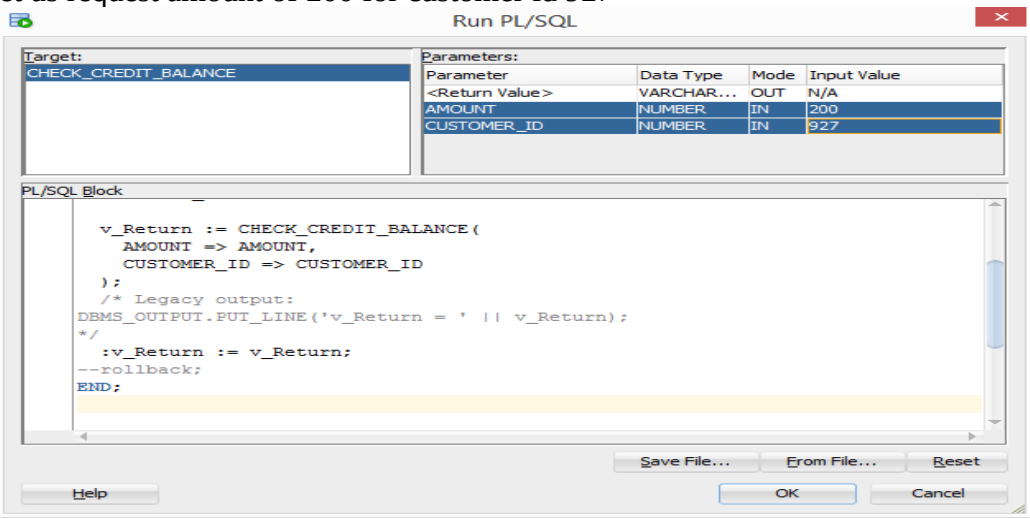
Remaining credit limit = credit limit – amount spent
= 23500 – 23000 = 500

- If user requests for money less than 500 our function should let user to use the amount returning "Transaction successful"
- Else function should return -"Insufficient Funds"

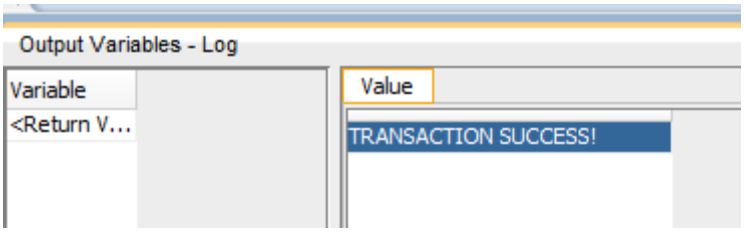
- Let us run the function:



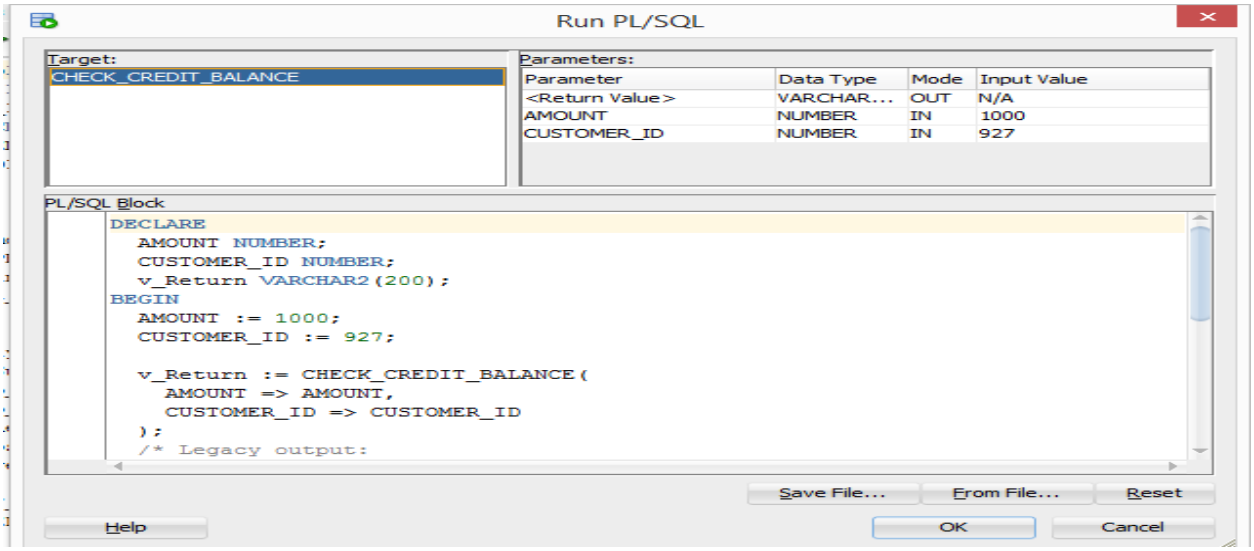
- Let us request amount of 200 for customer id 927



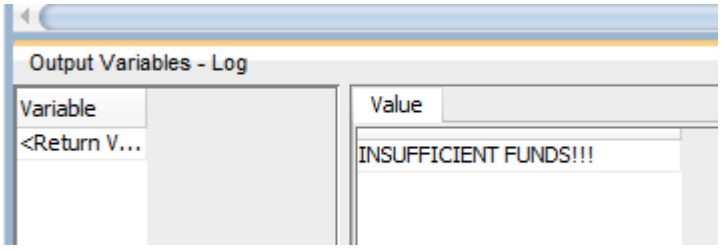
- As the requested amount is less than the remaining credit card limit, the transaction will be successful.



- Now, let us request amount of 1000



- Our function has returned “INSUFFICIENT FUNDS!!!” as the requested amount exceeds the remaining credit limit. (we had credit limit remaining of 500 and we requested for 1000.)



9. Evaluation Table

Topic / Section	Description	Evaluation
Logical database design	The logical design section should include entity-relationship diagrams (ERDs) and data dictionaries for your database design, as well as any design assumptions. There should also be a complete ERD for your entire project. There is no expectation that you implement all of your design, just indicate the areas built.	**15**
Physical database design	This section should cover implementation-level issues. For instance, discuss predicted usage and indexing strategies that support expected activities. In addition, you may wish to discuss architecture issues, including distributed database issues (even though you may not implement anything in these areas). Artifacts could include capacity planning, storage subsystems, and data placement (e.g., tablespace / file system arrangements), indexing strategies, transaction usage maps, etc.	**15**
Data generation and loading	Describe the queries, stored procedures, desktop tools (e.g., MS Excel) that were used to populate the database. You may have used queries with mod function, data arithmetic, number sequences, lookup tables, and even data from the Web. Any / all of these are interesting additions to the project. You must create and populate at least five tables from your design. Two of those tables must include at least 10,000 records a piece. Include a count of the number of rows inserted into each table.	**10**
Performance tuning	In this section, highlight any experiments run as part of the project related to performance tuning. Experiments with different indexing strategies, optimizer changes, transaction isolation levels, function-based indexes, and table partitioning can all be interesting. Remember to look at different types of queries (e.g., point, range, scan), execution plans, and I/O burden. For each experiment include the following: (1) purpose of the experiment, (2) steps followed to run the experiment, (3) key results (include screenshots, figures, and/or tables to help highlight results), and (4) a discussion of the results that explains what happened and why.	**15**
Querying	In this section, create queries that highlight the types of questions that can be answered by the database. These queries should demonstrate your skills in query writing. (Analytic SQL extensions may be explored for this section.)	**20**
DBA scripts	During the semester, we looked at example DBA scripts that query the system catalog (a good way to explore the database engine). Provide DBA scripts that are helpful for reporting on database objects, indexes, constraints, physical storage, data files, etc. For each script provide the following: (1) SQL / PL/SQL code, (2) description of why the script is useful, (3) how the script could be used, and (4) some sample results from executing the script.	**10**
Database programming	For this section, highlight any stored procedures, functions, or triggers that were created that are not included in the data generation and loading topic.	**15**
Database security	Database security is an important area of interest that can also be investigated. Though you are limited on the implementation side, you can develop a security policy and discuss how you would implement various aspects using authentication strategies, roles, profiles, and even auditing features.	**0**
Interface design / Data visualization	Though interface issues are not typically the focus of the project, you are free to add emphasis here. You can do everything from sketches and mock-ups, to using HTML and other web-enabled tools to build an interface. You can also experiment with creating visualizations for your data using a variety of freely-available tools such as Tableau Public.	**0**