Part A – ER Diagram (Dublin Logistics)

A database can be modelled as collection of Entities and Relationship between Entities. This technique is called as Entity-Relationship Modeling (E-R Modeling). An Entity is as object which is easily distinguishable from other available objects.

In this E-R Diagram we have different Entities like – COMPANY, LOCAL_DEPOT, NTNL_DEPOT, MANAGER, SUPPLIER, PRODUCT and VEHICLE. Each Entity has certain amount of properties which gives more information about Entity and it is called as Attributes.

For Example – VEHICAL is and Entity and It has 3 Attributes namely, REG_NUM, VEH_MAKE and VEH_MODEL. These attributes give additional information of vehicle such as Registration Number, Vehicle Manufacturer and Vehicle Model.

The respective E-R Diagram for Dublin Logistics is shown in below Figure 1.

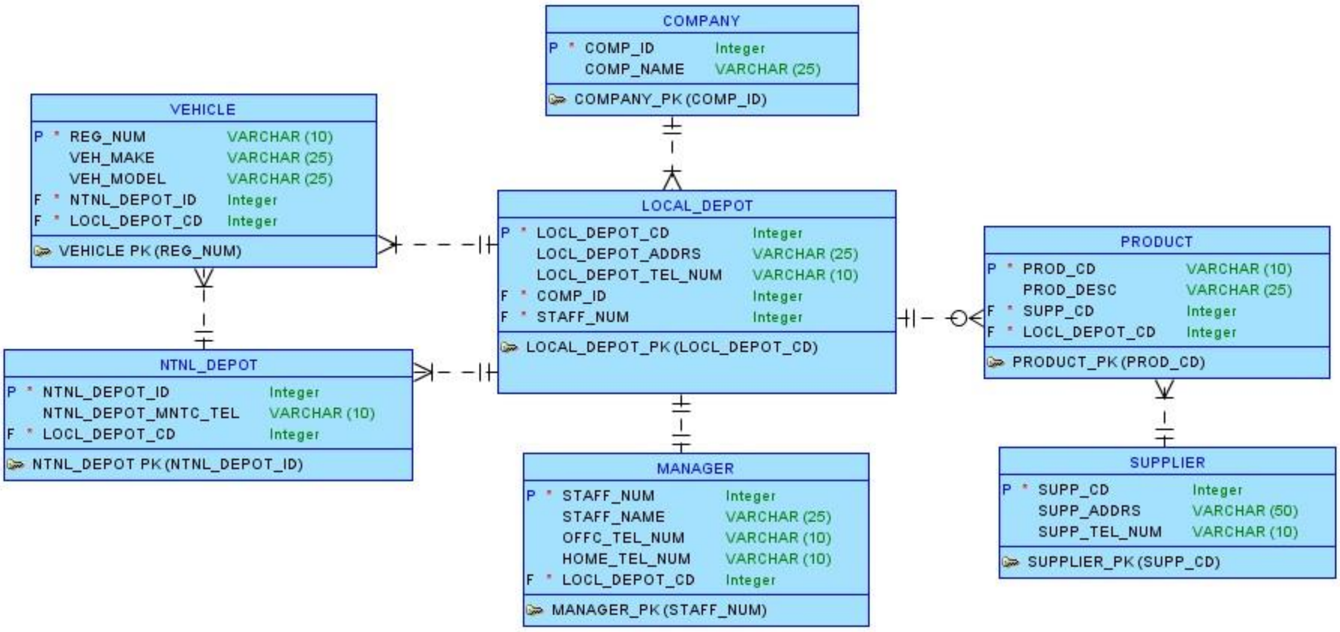


Figure 1: Dublin Logistics E-R Diagram (Logical Model)

As shown in above Figure 1, there is 1:N relationship between COMPANY and LOCAL_DEPOT. Also, LOCAL_DEPOT is having 1:N relationship VEHICLE NTNL_DEPOT and PRODUCT. On the other hand, there is 1:1 relationship between LOCAL_DEPOT and MANAGER as one depot can have only one manager. Further, SUPPLIER is having 1:N relationship with PRODUCT, as supplier can provide different products. Also, there is 1:N relationship between NTNL_DEPOT and VEHICLE as national depot will be having 10 or more vehicles for delivering products.

Once E-R Diagram is ready we can Engineer to Relational Model which is shown in Figure 2 below.

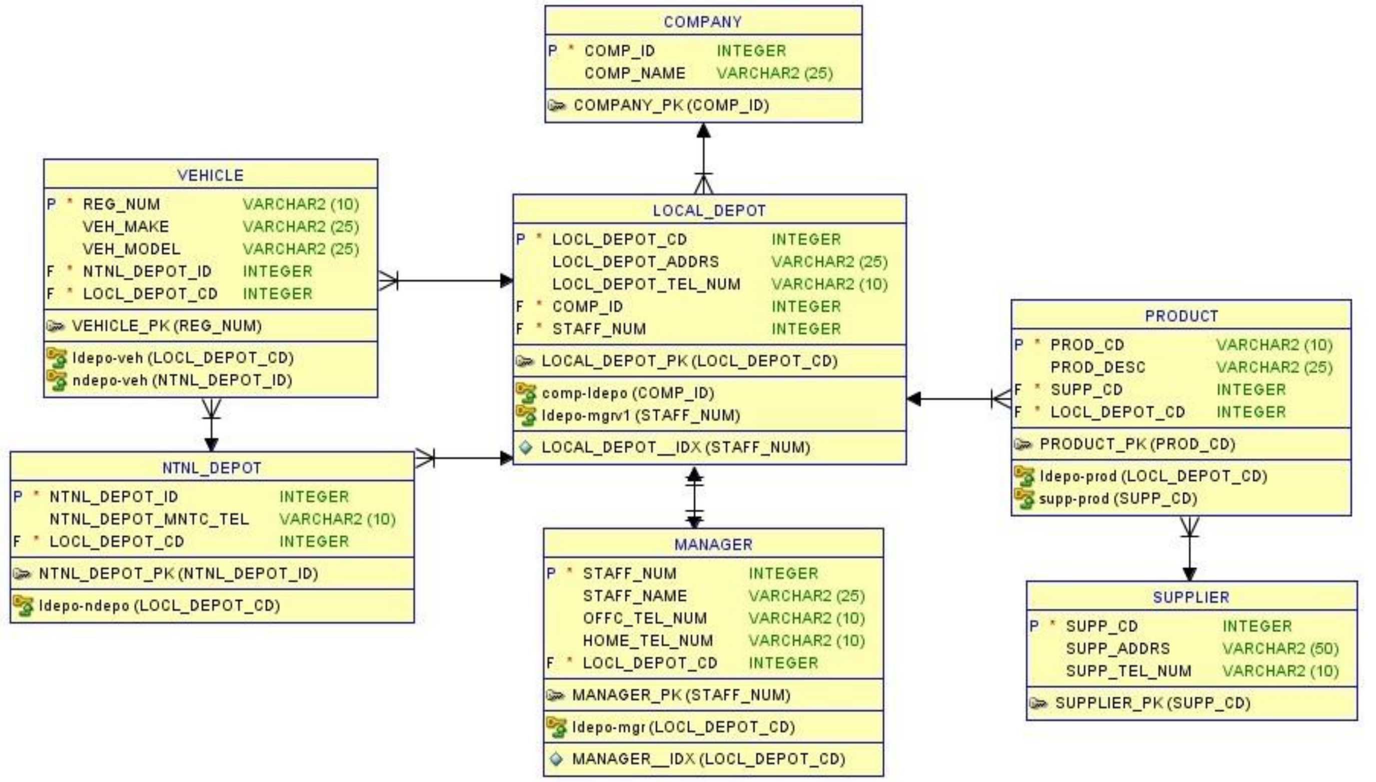
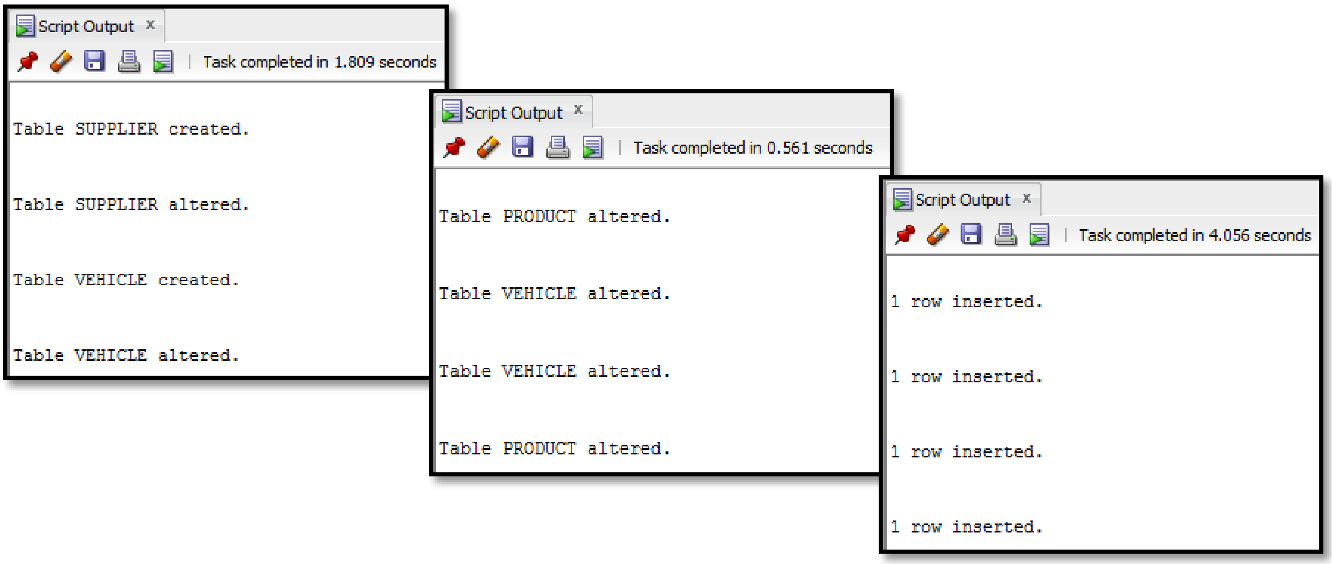
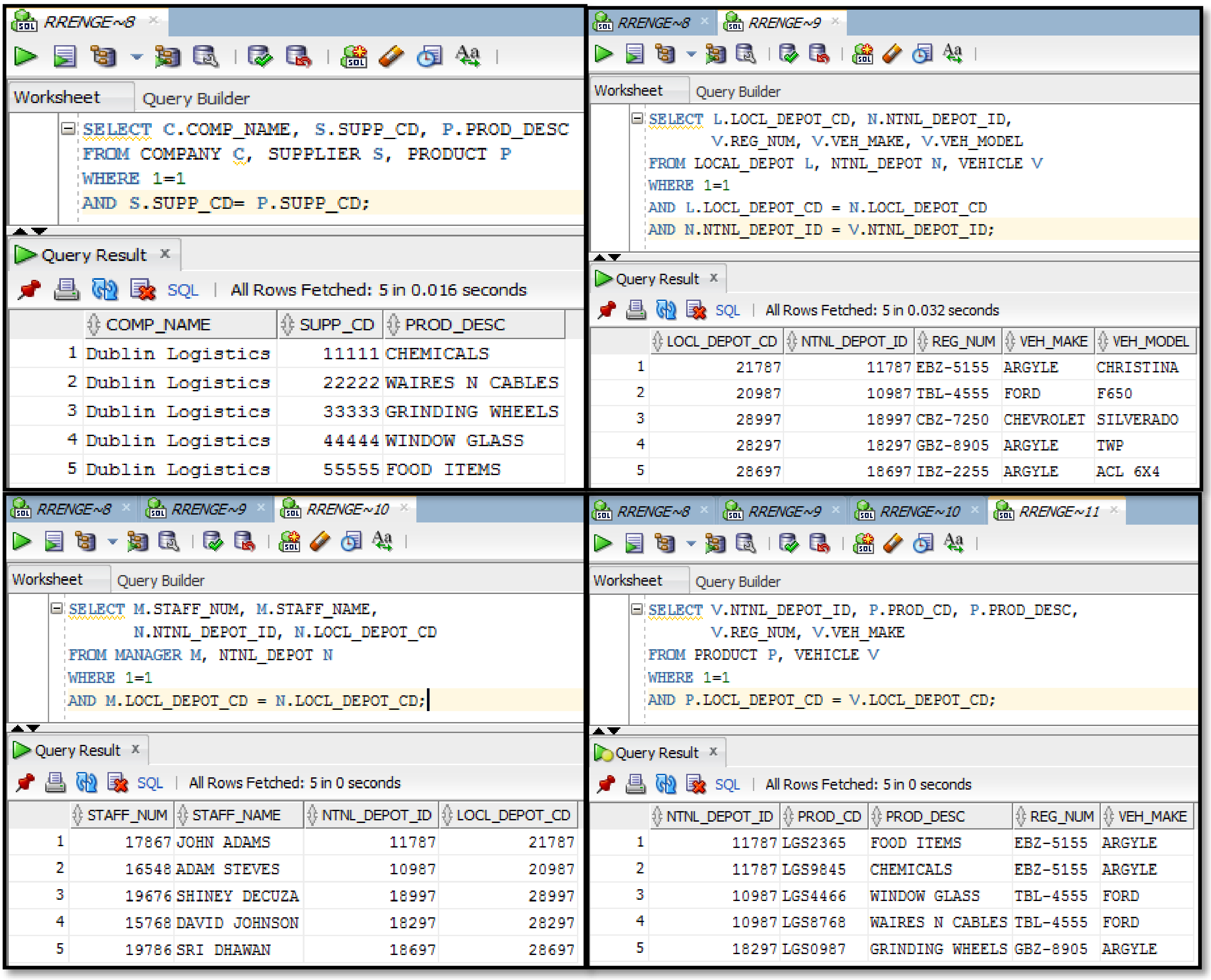


Figure 2: Dublin Logistics E-R Diagram (Relational Model)

Now, Next step is to generate the DDL Statements from this Relational Model. With the help of SQL Developer, respective tables along with its references can be created easily by executing those DDL scripts. Once structure is ready in oracle schema, data should be inserted into those tables.





```sql
SELECT C.COMP_NAME, S.SUPP_CD, P.PROD_DESC
FROM COMPANY C, SUPPLIER S, PRODUCT P
WHERE 1=1
AND S.SUPP_CD= P.SUPP_CD;
```

All Rows Fetched: 5 in 0.016 seconds

	COMP_NAME	SUPP_CD	PROD_DESC
1	Dublin Logistics	11111	CHEMICALS
2	Dublin Logistics	22222	WAIRES N CABLES
3	Dublin Logistics	33333	GRINDING WHEELS
4	Dublin Logistics	44444	WINDOW GLASS
5	Dublin Logistics	55555	FOOD ITEMS

```sql
SELECT L.LOCL_DEPOT_CD, N.NTNL_DEPOT_ID,
       V.REG_NUM, V.VEH_MAKE, V.VEH_MODEL
FROM LOCAL_DEPOT L, NTNL_DEPOT N, VEHICLE V
WHERE 1=1
AND L.LOCL_DEPOT_CD = N.LOCL_DEPOT_CD
AND N.NTNL_DEPOT_ID = V.NTNL_DEPOT_ID;
```

All Rows Fetched: 5 in 0.032 seconds

	LOCL_DEPOT_CD	NTNL_DEPOT_ID	REG_NUM	VEH_MAKE	VEH_MODEL
1	21787	11787	EBZ-5155	ARGYLE	CHRISTINA
2	20987	10987	TBL-4555	FORD	F650
3	28997	18997	CBZ-7250	CHEVROLET	SILVERADO
4	28297	18297	GBZ-8905	ARGYLE	TWP
5	28697	18697	IBZ-2255	ARGYLE	ACL 6X4

```sql
SELECT M.STAFF_NUM, M.STAFF_NAME,
       N.NTNL_DEPOT_ID, N.LOCL_DEPOT_CD
FROM MANAGER M, NTNL_DEPOT N
WHERE 1=1
AND M.LOCL_DEPOT_CD = N.LOCL_DEPOT_CD;
```

All Rows Fetched: 5 in 0 seconds

	STAFF_NUM	STAFF_NAME	NTNL_DEPOT_ID	LOCL_DEPOT_CD
1	17867	JOHN ADAMS	11787	21787
2	16548	ADAM STEVES	10987	20987
3	19676	SHINEY DECUZA	18997	28997
4	15768	DAVID JOHNSON	18297	28297
5	19786	SRI DHAWAN	18697	28697

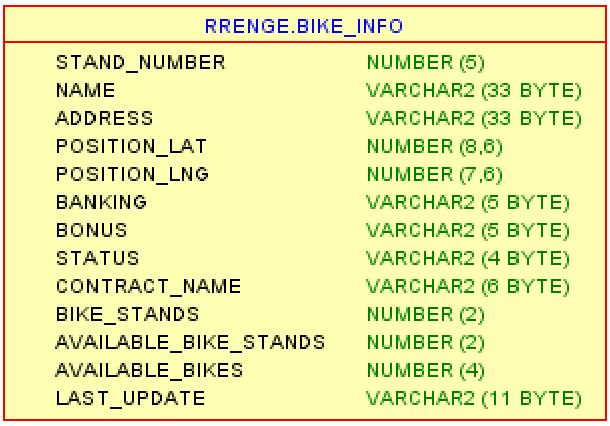
```sql
SELECT V.NTNL_DEPOT_ID, P.PROD_CD, P.PROD_DESC,
       V.REG_NUM, V.VEH_MAKE
FROM PRODUCT P, VEHICLE V
WHERE 1=1
AND P.LOCL_DEPOT_CD = V.LOCL_DEPOT_CD;
```

All Rows Fetched: 5 in 0 seconds

	NTNL_DEPOT_ID	PROD_CD	PROD_DESC	REG_NUM	VEH_MAKE
1	11787	LGS2365	FOOD ITEMS	EBZ-5155	ARGYLE
2	11787	LGS9845	CHEMICALS	EBZ-5155	ARGYLE
3	10987	LGS4466	WINDOW GLASS	TBL-4555	FORD
4	10987	LGS8768	WAIRES N CABLES	TBL-4555	FORD
5	18297	LGS0987	GRINDING WHEELS	GBZ-8905	ARGYLE

Part B – SQL Statistical and Analytical Functions

For performing this part of assignment, Dublin Bikes dataset is used, which is downloaded from website and stored in data frame in R Studio and then it is saved in Bike_Info.csv file and then it can be easily loaded into Oracle schema. While loading CSV file into Oracle schema first column is renamed as STAND_NUMBER, originally it was NUMBER and was creating issues while loading data in oracle as number is treated as datatype in oracle therefore it cannot be a column name for any of the table.



RRENGE.BIKE_INFO

STAND_NUMBER	NUMBER (5)
NAME	VARCHAR2 (33 BYTE)
ADDRESS	VARCHAR2 (33 BYTE)
POSITION_LAT	NUMBER (8,6)
POSITION_LNG	NUMBER (7,6)
BANKING	VARCHAR2 (5 BYTE)
BONUS	VARCHAR2 (5 BYTE)
STATUS	VARCHAR2 (4 BYTE)
CONTRACT_NAME	VARCHAR2 (6 BYTE)
BIKE_STANDS	NUMBER (2)
AVAILABLE_BIKE_STANDS	NUMBER (2)
AVAILABLE_BIKES	NUMBER (4)
LAST_UPDATE	VARCHAR2 (11 BYTE)

Figures 2.1: BIKE_INFO

The various Statistical and Analytical functions are used to analyse the BIKE_INFO dataset.

1) RANK()

RANK () function is used to identify the rank in given ordered partition. There can be same rank to more than one rows, therefore next rank is skipped if this happens. As shown in below screenshot, rank 7 is allocated two times therefore 8 is skipped and directly rank 9 is assigned, Similarly, rank 9 is populated three times therefore rank 10 and rank 11 are skipped and rank 12 is assigned to upcoming row.

NAME	AVAILABLE_BIKES	RANK
HEUSTON STATION (CENTRAL)	40	1
EXCISE WALK	39	2
HEUSTON BRIDGE (NORTH)	37	3
GUILD STREET	36	4
TAL	32	5
NOR	30	6
CUS	29	7
FOW	29	7
PEA	28	9
CUSTOM HOUSE QUAY	28	9
SANDWITH STREET	28	9
SOUTH DOCK ROAD	27	12
DEVERELL PLACE	26	13

Query Result 13 SQL

```
RRENGE : SELECT NAME, AVAILABLE_BIKES,
RANK() OVER(ORDER BY AVAILABLE_BIKES DESC) RANK
FROM BIKE_INFO
ORDER BY RANK
```

Copy..

```sql
-- 1. RANK FUNCTION
SELECT NAME, AVAILABLE_BIKES,
RANK() OVER(ORDER BY AVAILABLE_BIKES DESC) RANK
FROM BIKE_INFO
ORDER BY RANK;
```

Figure 2.1.1: Rank Function Query and Output

2) DENSE_RANK()

DENSE_RANK () function is used to identify the rank in given ordered partition. There can be same rank to more than one rows, here rank is not skipped in any case. As shown in below screenshot, rank 7 is allocated two times still rank 8 is assigned.

Similarly, rank 8 and rank 11 are populated two times but none of the numbers are skipped here in case of DENSE_RANK () function.

NAME	AVAILABLE_BIKES	DENSE_RANK
HEUSTON STATION (CENTRAL)	40	1
EXCISE WALK	39	2
HEUSTON BRIDGE (NORTH)	37	3
GUILD STREET	36	4
TALBOT STREET	32	5
NO	30	6
CU	29	7
FO	29	7
PE	28	8
CU	28	8
SANDWITH STREET	28	8
SOUTH DOCK ROAD	27	9
DEVERELL PLACE	26	10
JAMES STREET	25	11
CHARLEMONT PLACE	25	11
PORTOBELLO ROAD	24	12

Query Result 14 SQL

```
RRENGE : SELECT NAME, AVAILABLE_BIKES,
DENSE_RANK() OVER(ORDER BY AVAILABLE_BIKES DESC) DENSE_RANK
FROM BIKE_INFO
ORDER BY DENSE_RANK
```

Copy..

```sql
-- 2. DENSE_RANK FUNCTION
SELECT NAME, AVAILABLE_BIKES,
DENSE_RANK() OVER(ORDER BY AVAILABLE_BIKES DESC) DENSE_RANK
FROM BIKE_INFO
ORDER BY DENSE_RANK;
```

Figure 2.1.2: Dense_Rank Function Query and Output

3) Normal Statistics Distribution

This is used to check the normality of the data. In this case Shapiro Wilks test is performed on AVAILABLE_BIKE_STANDS and AVAILABLE_BIKES to check the normality of the variables. From the results, it is identified that p value < 0.05 level significance, which clearly helps to conclude that data is not normally distributed.

```
-- 3. NORMAL DISTRIBUTION FIT
SET SERVEROUTPUT ON
DECLARE
SIG      NUMBER ;
MEAN     NUMBER := 1;
STDEV    NUMBER := 1;
BEGIN
    DBMS_OUTPUT.PUT_LINE('NORMAL DISTRIBUTION FIT STATISTICS');
    DBMS_OUTPUT.PUT_LINE('SHAPIRO_WILKS TEST RESULTS FOR AVAILABLE BIKE STANDS');
    DBMS_STAT_FUNCS.NORMAL_DIST_FIT('RRENGE','BIKE_INFO','AVAILABLE_BIKE_STANDS','SHAPIRO_WILKS',MEAN,STDEV,SIG);
    DBMS_OUTPUT.PUT_LINE('SHAPIRO_WILKS TEST RESULTS FOR AVAILABLE BIKES');
    DBMS_STAT_FUNCS.NORMAL_DIST_FIT('RRENGE','BIKE_INFO','AVAILABLE_BIKES','SHAPIRO_WILKS',MEAN,STDEV,SIG);
    DBMS_OUTPUT.PUT_LINE('P VALUE IS:' || (SIG));
END;
```

Task completed in 0.264 seconds

```
NORMAL DISTRIBUTION FIT STATISTICS
SHAPIRO_WILKS TEST RESULTS FOR AVAILABLE BIKE STANDS
W value : .9591696114927336316165416163724149778215
SHAPIRO_WILKS TEST RESULTS FOR AVAILABLE BIKES
W value : .9147611468501799378082474237393754935364
P VALUE IS:.00000750103513319994947113438601139623211856
```

Figure 2.1.3: Normal Distribution Fit Query and Output

4) Summary Statistics

As name suggests, it is used to obtain the summary of the dataset. This is used to gather the summary of the data by summarizing the data retrieved from the other columns by applying different function.

```
-- 4. SUMMARY STATISTICS FOR BIKE_STANDS
SET SERVEROUTPUT ON

DECLARE
SUMM_STAT   DBMS_STAT_FUNCS.SUMMARYTYPE;

BEGIN
    DBMS_STAT_FUNCS.SUMMARY('RRENGE', 'BIKE_INFO', 'BIKE_STANDS', 3, SUMM_STAT);
    DBMS_OUTPUT.PUT_LINE('SUMMARY STATISTICS FOR BIKE STANDS');
    DBMS_OUTPUT.PUT_LINE('MEAN OF BIKE STANDS:'||' '|| ROUND(SUMM_STAT.MEAN));
    DBMS_OUTPUT.PUT_LINE('STANDARD DEVIATION OF BIKE STANDS'||' '|| ROUND(SUMM_STAT.STDDEV));
    DBMS_OUTPUT.PUT_LINE('SMALLEST BIKE STAND HAS'||' '|| SUMM_STAT.MIN||' '||'BIKES');
    DBMS_OUTPUT.PUT_LINE('LARGEST BIKE STAND HAS'||' '|| SUMM_STAT.MAX||' '||'BIKES');
END;
```

```
SUMMARY STATISTICS FOR BIKE STANDS
MEAN OF BIKE STANDS: 31
STANDARD DEVIATION OF BIKE STANDS 8
SMALLEST BIKE STAND HAS 16 BIKES
LARGEST BIKE STAND HAS 40 BIKES
```

Figure 2.1.4: Summary Statistics Query and Output

5) Co-Variance

Co-Variance is used to identify the relationship between two different variables.

This can be achieved by using COVAR_SAMP function and can be analysed based on co-variance value. In results, it is clearly identified that all values of co-variance is negative. Therefore, there is negative relationship between AVAILABLE_BIKE_STANDS and AVAILABLE_BIKES i.e. if count of one increases then automatically other one will decrease.

```
-- 5. CO-VARIENCE
SELECT STAND_NUMBER, NAME, AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES,
COVAR_SAMP(AVAILABLE_BIKES, AVAILABLE_BIKE_STANDS) OVER(ORDER BY NAME ) AS CO_VARIANCE
FROM BIKE_INFO
ORDER BY STAND_NUMBER;
```

STAND_NUMBER	NAME	AVAILABLE_BIKE_STANDS	AVAILABLE_BIKES	CO_VARIANCE
1	CLARENDON ROW	20	11	-55.190909090909090909090909090909091
2	BLESSINGTON STREET	18	2	-4.3333333333333333333333333333333333
3	BOLT	18	1	1.7
4	GREE	[illegible]	9	-88.647897897897897897897897897897898789
5	CHAR	15	25	-66.946428571428571428571428571428571428 57
6	CHRI	12	8	-58.444444444444444444444444444444445
7	HIGH	13	16	-105.39591836734693877551020408163265 3061
8	CUST	2	28	-99.65
9	EXCHEQUER STREET	5	19	-88.760869565217391304347826086956521 73913
10	DAME STREET	1	15	-97.941176470588235294117647058823529 41175

Query Result 15 SQL
RRENGE : SELECT STAND_NUMBER, NAME, AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES,
COVAR_SAMP(AVAILABLE_BIKES, AVAILABLE_BIKE_STANDS) OVER(ORDER BY NAME)
AS CO_VARIANCE
FROM BIKE_INFO
ORDER BY STAND_NUMBER
Copy..

Figure 2.1.5: Co-Variance Query and Output

6) Linear Regression

Again, Liner Regression is used to identify the relationship between two variables.

In this case, AVAILABLE_BIKE _STANDS and AVAILABLE_BIKES are the two variables which is analysed to check the positive or negative relationship based on REG_SLOPE value obtained with REGR_SLOPE function.

```
-- 6. LINEAR REGRESSION
SELECT STAND_NUMBER, NAME, AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES,
REGR_SLOPE(AVAILABLE_BIKES,AVAILABLE_BIKE_STANDS) OVER(ORDER BY NAME)REG_SLOPE
FROM BIKE_INFO
ORDER BY STAND_NUMBER;
```

STAND_NUMBER	NAME	AVAILABLE_BIKE_STANDS	AVAILABLE_BIKES	REG_SLOPE
1	CLARENDON ROW	20	11	-0.49697118533071381794368041912246234 44662
2	BLESSINGTON STREET	18	2	-0.05327868852459016393442622950819672 13115
3	B	18	1	0.01995305164319248826291079812206572 76995
4	G	[illegible]	9	-0.63683972084092894819161444119645765 69177
5	C	15	25	-0.49544072948328267477203647416413373 86018
6	C	12	8	-0.45472228225632159066349686622001296 7366
7	H	13	16	-0.71980509343918647696356094732616743 23179
8	C	2	28	-0.63236382866208355367530407191961924 90746
9	EXCHEQUER STREET	5	19	-0.60375591482649842271293375394321766 56152
10	DAME STREET	1	15	-0.58249879739362399965015087243626186 20719

Query Result 15 SQL
RRENGE : SELECT STAND_NUMBER, NAME, AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES,
REGR_SLOPE(AVAILABLE_BIKES,AVAILABLE_BIKE_STANDS) OVER(ORDER BY NAME)
REG_SLOPE
FROM BIKE_INFO
ORDER BY STAND_NUMBER
Copy..

Figure 2.1.6: Liner Regression Query and Output

7) CROSSTAB

Crosstab or Cross tabulation is mostly used in quantitative research to identify relationship between two variables. As shown in below result, p value is 0.000000… i.e. nearly zero. Therefore it is clearly understood that there is no relationship between AVAILABLE_BIKE_STANDS and AVAILABLE_BIKES.

```sql
-- 7. CROSSTAB
SELECT STATS_CROSSTAB (AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, 'CHISQ_OBS') CHI_SQUARED,
STATS_CROSSTAB AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, 'CHISQ_SIG') P_VALUE,
STATS_CROSSTAB(AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, 'PHI_COEFFICIENT') PHI_COEFFICIENT
FROM BIKE_INFO;
```

CHI_SQUARED	P_VALUE	PHI_COEFF
1399.603174603174603174603174603178	0.00000002513625133879O985	3.74112706895017748363247982781824006207

Query Result 21 SQL

```
RRENGE : SELECT
STATS_CROSSTAB (AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, 'CHISQ_OBS') AS CHI_SQUARED,
STATS_CROSSTAB (AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, 'CHISQ_SIG') AS P_VALUE,
STATS_CROSSTAB (AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, 'PHI_COEFFICIENT') AS PHI_COEFF
FROM BIKE_INFO
```

Copy..

Figure 2.1.7: Crosstab Query and Output

8) BETWEEN

This keyword is used to obtain the result from specified range. In this case, STAND_NUMBER from 1 to 10 are populated with the help of BETWEEN keyword by specifying range of variable.

```sql
-- 8. BETWEEN
SELECT STAND_NUMBER, NAME, ADDRESS
FROM BIKE_INFO
WHERE 1=1
AND STAND_NUMBER BETWEEN 1 AND 10
ORDER BY STAND_NUMBER;
```

STAND_NUMBER	NAME	ADDRESS
1	CLARENDON ROW	Clarendon Row
2	BLESSIN	
3	BOLTON	
4	GREEK S	
5	CHARLEM	
6	CHRISTC	
7	HIGH ST	High Street
8	CUSTOM HOUSE QUAY	Custom House Quay
9	EXCHEQUER STREET	Exchequer Street
10	DAME STREET	Dame Street

Query Result SQL

```
RRENGE : SELECT STAND_NUMBER, NAME, ADDRESS
FROM BIKE_INFO
WHERE 1=1
AND STAND_NUMBER BETWEEN 1 AND 10
ORDER BY STAND_NUMBER
```

Copy..

Figure 1.2.8: Between Query and Output

9) MIN(), MAX()

MIN function is used to get the minimum value for the particular column and similarly MAX function is used to get the maximum value of the column. As shown below minimum bikes available is zero and maximum available bikes count is 40.

	MIN(AVAILABLE_BIKES)	MAX(AVAILABLE_BIKES)
1	0	40

Query Result SQL

```
RRENGE : SELECT MIN(AVAILABLE_BIKES), MAX(AVAILABLE_BIKES)
FROM BIKE_INFO
```

Copy..

Figure 2.1.9: MIN and MAX Query and Output

Part C – Machine Learning SQL

Firstly we have to create below mentioned tables and need to load data in those tables.

1) MINING_DATA_BUILD_V
2) MINING_DATA_APPLY_V

This is done with the help of queries available in Lab Exercise.

Once data loading activity is completed, then view named ANALYTICAL_VIEW should be created to combine data of above mentioned 2 tables. The query used for creating view is –

```
CREATE OR REPLACE VIEW ANALYTICAL_VIEW
AS
SELECT * FROM MINING_DATA_BUILD_V
UNION
SELECT * FROM MINING_DATA_APPLY_V;

SELECT COUNT(1) FROM ANALYTICAL_VIEW; --3000 ROWS
```

Now, in Step 3, we have to create two additional views namely TRAINING_DATA and TEST_DATA for machine learning. These views are created by splitting the view created in Step 2, i.e. ANALYTICAL_VIEW. TRAINING_DATA view contains 60% of data and on the other hand TEST_DATA view contains remaining 40% of the data from ANALYTICAL_VIEW.

The sampling of these views is done with the help of ORA_HASH function as SAMPLE function cannot be used with views. ORA_HASH usually computes hash value and generates random sample and this sample data set can be further used for machine learning training and testing data sets. Seed value is one of the parameter for ORA_HASH function which is optional parameter, this parameter helps to split data into many different data samples each time when this function is executed. The code below shows the sampling of data for machine learning.

```
--STEP 3
--TRAIN DATA
CREATE OR REPLACE VIEW TRAINING_DATA
AS
SELECT VW.*
FROM ANALYTICAL_VIEW VW
WHERE
ORA_HASH(CUST_ID,(SELECT COUNT(1) FROM ANALYTICAL_VIEW), 0) <
     (SELECT COUNT(1) FROM ANALYTICAL_VIEW) * 60 / 100;
SELECT COUNT(1) FROM TRAINING_DATA; --1803

--TEST DATA
CREATE OR REPLACE VIEW TEST_DATA
AS
SELECT VW.*
FROM ANALYTICAL_VIEW VW
WHERE
ORA_HASH(CUST_ID,(SELECT COUNT(1) FROM ANALYTICAL_VIEW), 0) >
     (SELECT COUNT(1) FROM ANALYTICAL_VIEW) * 60 / 100;
SELECT COUNT(1) FROM TEST_DATA; --1195
```

In next step, we have to build 2 machine learning models, namely Naïve Bayes and Decision Tree. Firstly, the setting table should be created before creating the machine learning model. Therefore setting table for Naïve Bayes and Decision Tree should be created as there should be separate setting tables for each models need to be created.

```
-- CREATING SETTING TABLE FOR NAIVE BAYES
CREATE TABLE naive_bayes_model_settings (
    setting_name VARCHAR2(30),
    setting_value VARCHAR2(30));
BEGIN
    INSERT INTO naive_bayes_model_settings (setting_name, setting_value)
    VALUES (dbms_data_mining.algo_name,dbms_data_mining.algo_naive_bayes);

    INSERT INTO naive_bayes_model_settings (setting_name, setting_value)
    VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);

    COMMIT;
END;
```

Similarly, we have to create the setting table for Decision tree.

```
-- CREATING SETTING TABLE FOR DECISION TREE
CREATE TABLE decision_tree_model_settings (
    setting_name VARCHAR2(30),
    setting_value VARCHAR2(30));
BEGIN
    INSERT INTO decision_tree_model_settings (setting_name, setting_value)
    VALUES (dbms_data_mining.algo_name,dbms_data_mining.algo_decision_tree);

    INSERT INTO decision_tree_model_settings (setting_name, setting_value)
    VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);

    COMMIT;
END;
```

Once done with both model setting tables, then we are ready to create the machine learning models. Firstly we have to start with Naïve Bayes, by default it is Naïve Bayes model.

```
-- CREATING NAIVE BAYES MODEL
BEGIN
    DBMS_DATA_MINING.CREATE_MODEL(
        model_name          => 'Naive_Bayes_Model',
        mining_function     => dbms_data_mining.classification,
        data_table_name     => 'TRAINING_DATA',
        case_id_column_name => 'cust_id',
        target_column_name  => 'affinity_card',
        settings_table_name => 'naive_bayes_model_settings');
END;
```

Similarly, we have to build decision tree by changing the model name and setting table and rest query will be same for creating decision tree model.

```
-- CREATING DECISION TREE MODEL
BEGIN
    DBMS_DATA_MINING.CREATE_MODEL(
        model_name          => 'Decision_Tree_Model',
        mining_function     => dbms_data_mining.classification,
        data_table_name     => 'TRAINING_DATA',
        case_id_column_name => 'cust_id',
        target_column_name  => 'affinity_card',
        settings_table_name => 'decision_tree_model_settings');
END;
```

Once model is created it can be verified by checking the user_mining_model_settings table and all_mining_model_attributes tables. The result is shown in below screenshots.

	ATTRIBUTE_NAME	ATTRIBUTE_TYPE	USAGE_TYPE	TARGET
1	AGE	NUMERICAL	ACTIVE	NO
2	HOME_THEATER_PACKAGE	NUMERICAL	ACTIVE	NO
3	CUST_GENDER	CATEGORICAL	ACTIVE	NO
4	CUST_MARITAL_STATUS	CATEGORICAL	ACTIVE	NO
5	BOOKKEEPING_APPLICATION	NUMERICAL	ACTIVE	NO
6	EDUCATION	CATEGORICAL	ACTIVE	NO
7	HOUSEHOLD_SIZE	CATEGORICAL	ACTIVE	NO
8	OCCUPATION	CATEGORICAL	ACTIVE	NO
9	Y_BOX_GAMES	NUMERICAL	ACTIVE	NO
10	YRS_RESIDENCE	NUMERICAL	ACTIVE	NO
11	AFFINITY_CARD	CATEGORICAL	ACTIVE	YES

Figure 3.2: Exploring Model Attributes

	MODEL_NAME	SETTING_NAME	SETTING_VALUE	SETTING_TYPE
1	DECISION_TREE_MODEL	ALGO_NAME	ALGO_DECISION_TREE	INPUT
2	DECISION_TREE_MODEL	PREP_AUTO	ON	INPUT
3	DECISION_TREE_MODEL	TREE_TERM_MINPCT_NODE	.05	DEFAULT
4	DECISION_TREE_MODEL	TREE_TERM_MINREC_SPLIT	20	DEFAULT
5	DECISION_TREE_MODEL	TREE_IMPURITY_METRIC	TREE_IMPURITY_GINI	DEFAULT
6	DECISION_TREE_MODEL	TREE_TERM_MINPCT_SPLIT	.1	DEFAULT
7	DECISION_TREE_MODEL	TREE_TERM_MAX_DEPTH	7	DEFAULT
8	DECISION_TREE_MODEL	TREE_TERM_MINREC_NODE	10	DEFAULT
9	NAIVE_BAYES_MODEL	ALGO_NAME	ALGO_NAIVE_BAYES	INPUT
0	NAIVE_BAYES_MODEL	PREP_AUTO	ON	INPUT
1	NAIVE_BAYES_MODEL	NABS_SINGLETON_THRESHOLD	0	DEFAULT
2	NAIVE_BAYES_MODEL	NABS_PAIRWISE_THRESHOLD	0	DEFAULT

Figure 3.1: Model Metadata Exploration

Figure 3.1 shows Model Name and its Setting Values for Decision Tree and Naïve Bayes Models which we have developed earlier in step 3. As shown in Figure 3.2 AFFINITY_CARD is target variable.

Step 5 is to evaluate the models created using the TEST_DATA. For evaluating these models, it is suitable to create the views to analyze the data. The code to create the views for decision tree and naïve bayes models is shown below.

```
--STEP 5 EVALUATING MODELS USING TEST_DATA
CREATE OR REPLACE VIEW VW_DT_RESULTS
AS
SELECT CUST_ID,
       prediction(DECISION_TREE_MODEL USING *)  predicted_value,
       prediction_probability(DECISION_TREE_MODEL USING *) probability
FROM   TEST_DATA;

CREATE OR REPLACE VIEW VW_NB_RESULTS
AS
SELECT CUST_ID,
       prediction(NAIVE_BAYES_MODEL USING *)  predicted_value,
       prediction_probability(NAIVE_BAYES_MODEL USING *) probability
FROM   TEST_DATA;
```

Step 6 is to create the random sample of 20 records from ANALYTICSL_VIEW. These record are stored in newly created table SAMPLE_DATA. This is done with the help of ROWNUM function. ROMNUM is pseudo column of oracle which generates number when the query is executed and the first record is marked as one and so on.

```
--RANDOM SAMPLE OF 20 REOCRDS
CREATE TABLE SAMPLE_DATA
AS
SELECT *
FROM ANALYTICAL_VIEW
WHERE rownum <=20;
```

In final step, LABELLED_DATA table is created with all the required columns and four additional columns representing Prediction and Probability for Naïve Bayes and Decision Tree Models.

```sql
--LABELLED DATA OF SAMPLE 20 RECORDS
CREATE TABLE LABELLED_DATA
AS
SELECT VW.*, PREDICTION(Decision_Tree_Model using *) AS DT_PREDICTION,
    PREDICTION_PROBABILITY(Decision_Tree_Model using *) AS DT_PROBABILITY
    PREDICTION(Naive_Bayes_Model using *) AS NB_PREDICTION,
    PREDICTION_PROBABILITY(Naive_Bayes_Model using *) AS NB_PROBABILITY
FROM SAMPLE_DATA VW
WHERE rownum <=20;
```

SQL | All Rows Fetched: 20 in 0.016 seconds

	CUST_ID	CUST_GENDER	AGE	DT_PREDICTION	DT_PROBABILITY	NB_PREDICTION	NB_PROBABILITY
1	100001	F	62	0	0.9870609981515711	0	0.9717018604278564
2	100002	F	41	0	0.9029850746268657	0	0.7530478835105896
3	100003	M	34	0	0.9029850746268657	0	0.9737856388092041
4	100004	F	50	0	0.9029850746268657	0	0.97086101770401
5	100005	M	46	1	0.7128378378378378	1	0.9752129912376404
6	100006	F	20	0	0.9870609981515711	0	0.999997615814209
7	100007	F	40	0	0.9029850746268657	0	0.992745041847229
8	100008	M	41	0	0.9029850746268657	0	0.9028350710868835
9	100009	M	29	1	0.7128378378378378	0	0.7830963134765625
[illegible]	[illegible]	[illegible]	28	0	0.6530214424951267	0	0.9752767086029053
[illegible]	[illegible]	[illegible]	[illegible]	0	0.9870609981515711	0	0.9996218085289001
[illegible]	[illegible]	[illegible]	35	1	0.7128378378378378	1	0.9970750212669373
[illegible]	100013	M	42	0	0.6530214424951267	1	0.9223827123641968
[illegible]	100014	F	49	0	0.9029850746268657	0	0.97086101770401
15	100015	M	44	0	0.9029850746268657	0	0.7379080057144165
16	100016	F	34	0	0.9029850746268657	0	0.975822389125824
17	100017	F	68	0	0.9029850746268657	0	0.97086101770401
18	100018	F	27	0	0.6530214424951267	0	0.9982815980911255
19	100019	M	32	0	0.6530214424951267	1	0.7482720017433167
20	100020	F	57	0	0.6078431372549019	0	0.9402749538421631

Query Result 22 SQL

RRENGE : SELECT CUST_ID, CUST_GENDER, AGE, DT_PREDICTION, DT_PROBABILITY, NB_PREDICTION, NB_PROBABILITY FROM LABELLED_DATA

Copy..

Figure 3.3: Labelled Data with Prediction and Probability Outcome

Confusion Matrix with TEST_DATA view.

```sql
--CONFUSION MATRIX NAIVE BAYES
SET SERVEROUTPUT ON
DECLARE
v_accuracy NUMBER;
BEGIN
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
accuracy => v_accuracy,
apply_result_table_name => 'VW_NB_RESULTS',
target_table_name => 'TEST_DATA',
case_id_column_name => 'cust_id',
target_column_name => 'affinity_card',
confusion_matrix_table_name => 'nb_confusion_matrix_v2',
score_column_name => 'PREDICTED_VALUE',
score_criterion_column_name => 'PROBABILITY',
cost_matrix_table_name => null,
apply_result_schema_name => null,
target_schema_name => null,
cost_matrix_schema_name => null,
score_criterion_type => 'PROBABILITY');
DBMS_OUTPUT.PUT_LINE('**** NAIVE BAYES MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
```

Query Result × | Script Output ×

Task completed in 0.398 seconds

```
**** NAIVE BAYES MODEL ACCURACY ****: .7908

PL/SQL procedure successfully completed.
```

Figure 4.1: Naive Bayes Confusion Matrix Query and Result

```
--CONFUSION MATRIX DECISION TREE
SET SERVEROUTPUT ON
DECLARE
v_accuracy NUMBER;
BEGIN
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
accuracy => v_accuracy,
apply_result_table_name => 'VW_DT_RESULTS',
target_table_name => 'TEST_DATA',
case_id_column_name => 'cust_id',
target_column_name => 'affinity_card',
confusion_matrix_table_name => 'dt_confusion_matrix',
score_column_name => 'PREDICTED_VALUE',
score_criterion_column_name => 'PROBABILITY',
cost_matrix_table_name => null,
apply_result_schema_name => null,
target_schema_name => null,
cost_matrix_schema_name => null,
score_criterion_type => 'PROBABILITY');
DBMS_OUTPUT.PUT_LINE('**** DECISION TREE MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
```

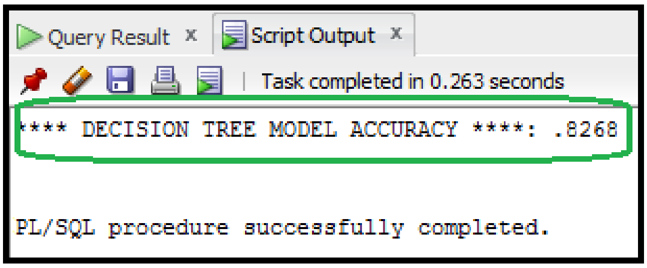
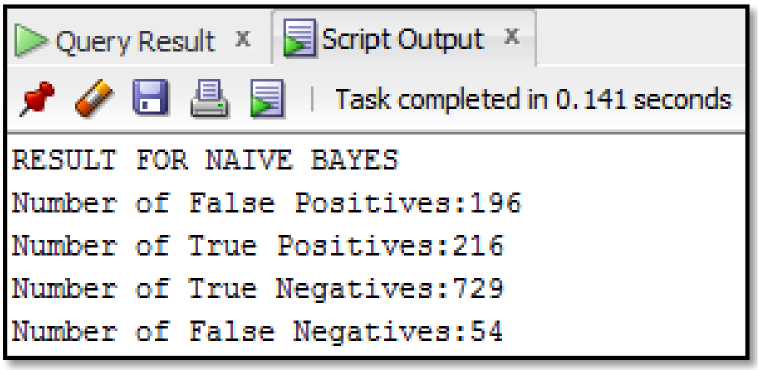


Figure 4.2: Decision Tree Confusion Matrix Query and Result

Part D – PL/SQL Code

Code for Procedures:



Precision for Naive Bayes = TP / (TP + FP)
= 216 / (216+196)
= 0.5242

Recall for Naïve Bayes = TP / (TP + FN)
= 216 / (216+54)
= 0.8

Accuracy for Naïve Bayes = (TP+TN) / (TP + TN + FP + FN)
= 216+729 / (216+54+729+196)
= 0.7907

Task completed in 0.141 seconds

```
RESULT FOR DECISION TREE
Number of False Positives:59
Number of True Positives:122
Number of True Negatives:866
Number of False Negatives:148
```

Precision for Decision Tree = TP / (TP + FP)
= 122 / (122+59)
= 0.67

Recall for Decision Tree = TP / (TP + FN)
= 122 / (122+148)
= 0.45

Accuracy for Decision Tree = (TP+TN) / (TP + TN + FP + FN)
= 122+866 / (122+59+866+148)
= 0.82

Name: Raunak Renge
Student Id: D17124381
Course Code: DT228A
Year: 2017-18
Assignment: Working With Data
Lecturer: Brendan Tierney
Username: rrenge
Password: d17124381