School of Computing and Mathematical Sciences

COMP-1848– Data Warehousing and Business Intelligence (2021/22)

Student Name: Rambabu GaneshKumar
Student ID: 001185508
Date: 08/01/2022
Module Leader: Hooman Oroojeni
Group:2
Team Members: Rojsun Parameswaran,Srihari Venkatesan,Illker Ozturk.

Contents

Table of Figures

Introduction:

The problem statement in the coursework states that you must design a star schema for storing the measurements of various types of water sensors based on different parameters, and that the data must be extracted from access to Oracle SQL developer. After being transformed and loaded into the data mart, it should be fed into the ML algorithm for predicting future water sensor measurements.

Overview:

Data warehousing is a system that retrieves data from source system to staging area and transform the data and Load the data into specific data mart which is small version of data warehouse that justifying a particular problem which would also be in dimensional data model by which it can further used to predict business problems

Extracting data from Ms access to Oracle SQL developer:

Merging multiple tables into single table so that it would be useful for transferring data from MS Access to Oracle SQL developer. Below figure consists converting tables from year 2000 to 2016 as single table waterquality.

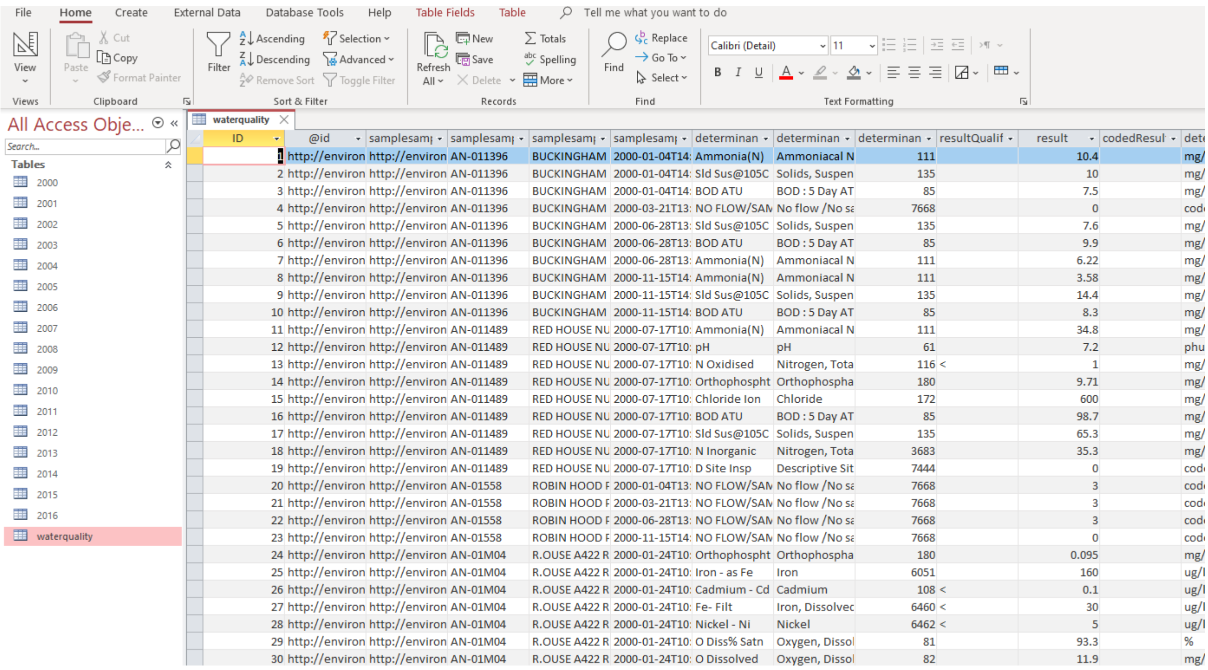


Figure 1-Exporting data as single Table

On right clicking the Merged table Select export->ODBC Database and name the table as water quality.

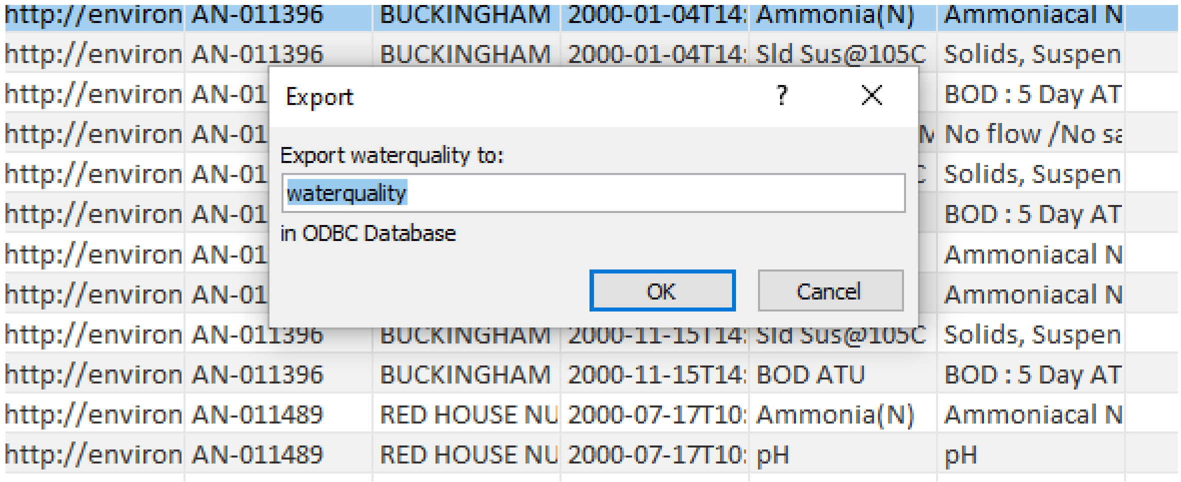


Figure 2-Exporting the Merged Table

Selecting the Data Source Oracle in Client64.

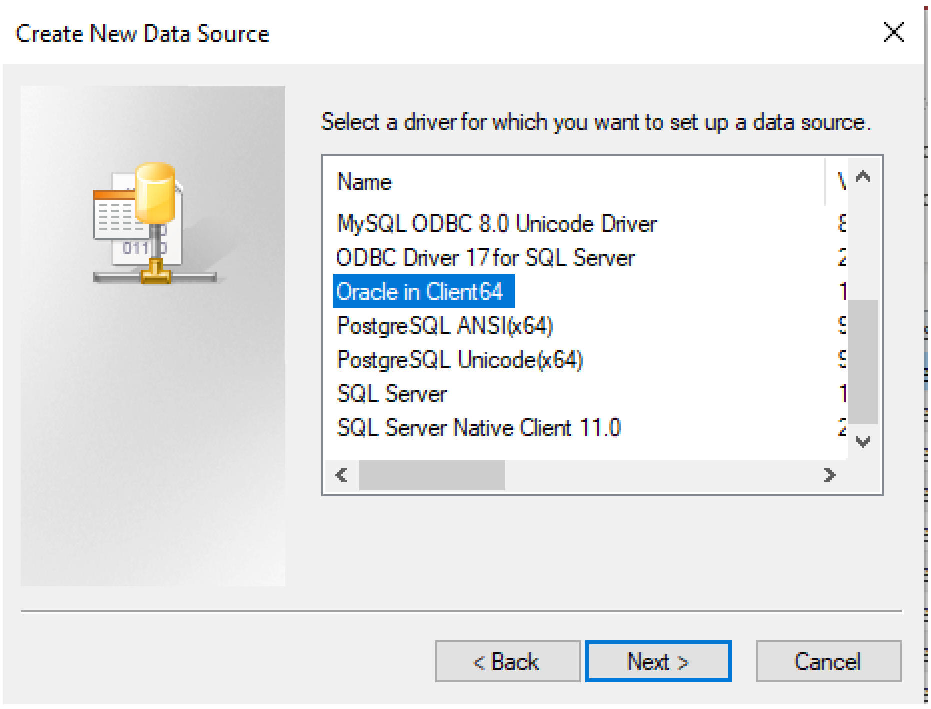


Figure 3-Data Source

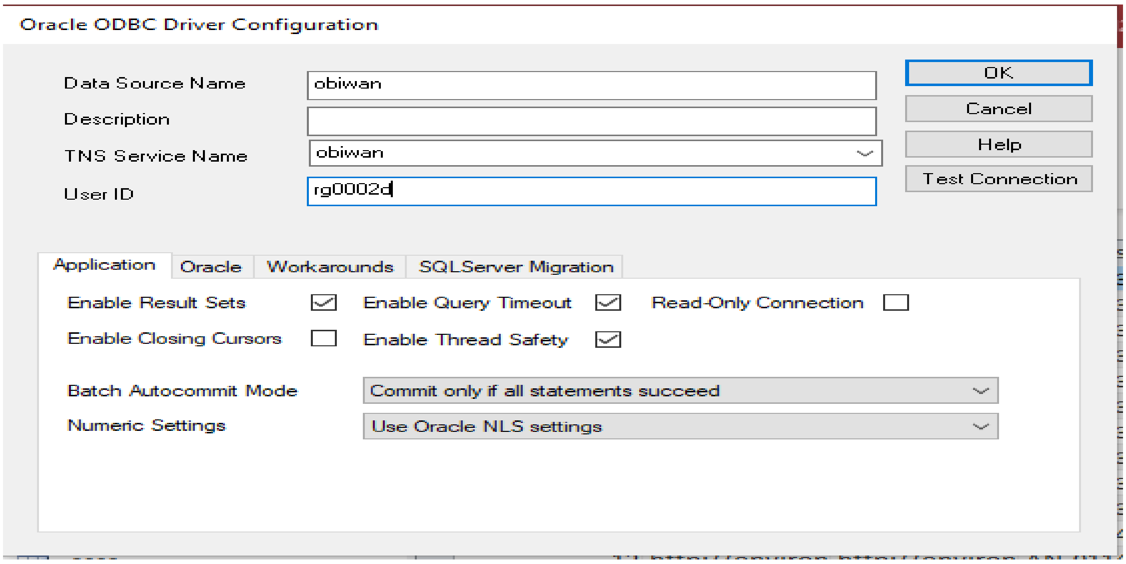


Figure 4-Credentials

Once Data is Exported from Ms Access using **DESC** key for describing the table columns and their data types .This table will be the staging area from where the data will further transformed and populated in Fact and Dimension table.

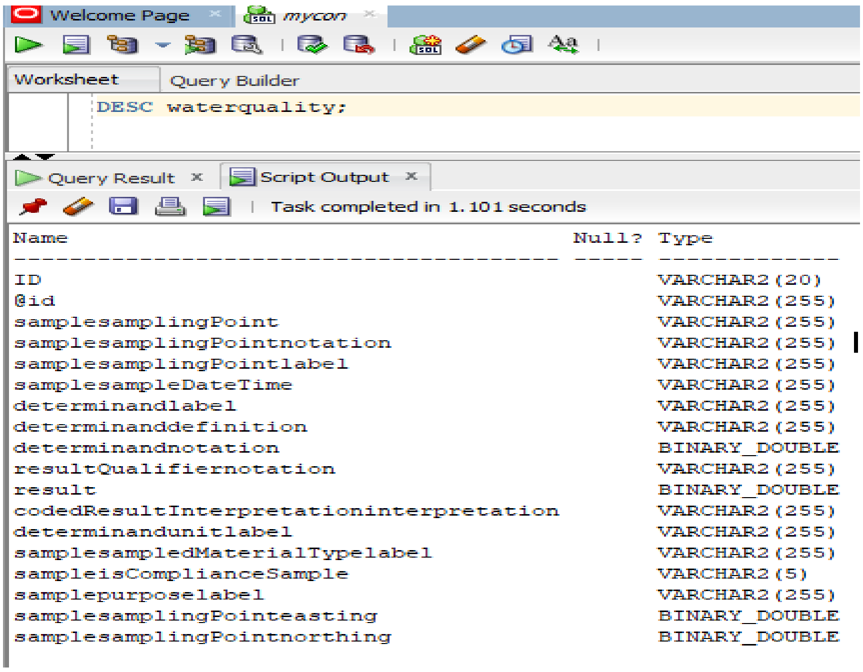


Figure 5-Verifying the Extracted Data

Star Schema Design :

Star schema is a dimensional data model with Fact and dimension tables, Here fact table and dimension table are connected with each other using referential integrity.

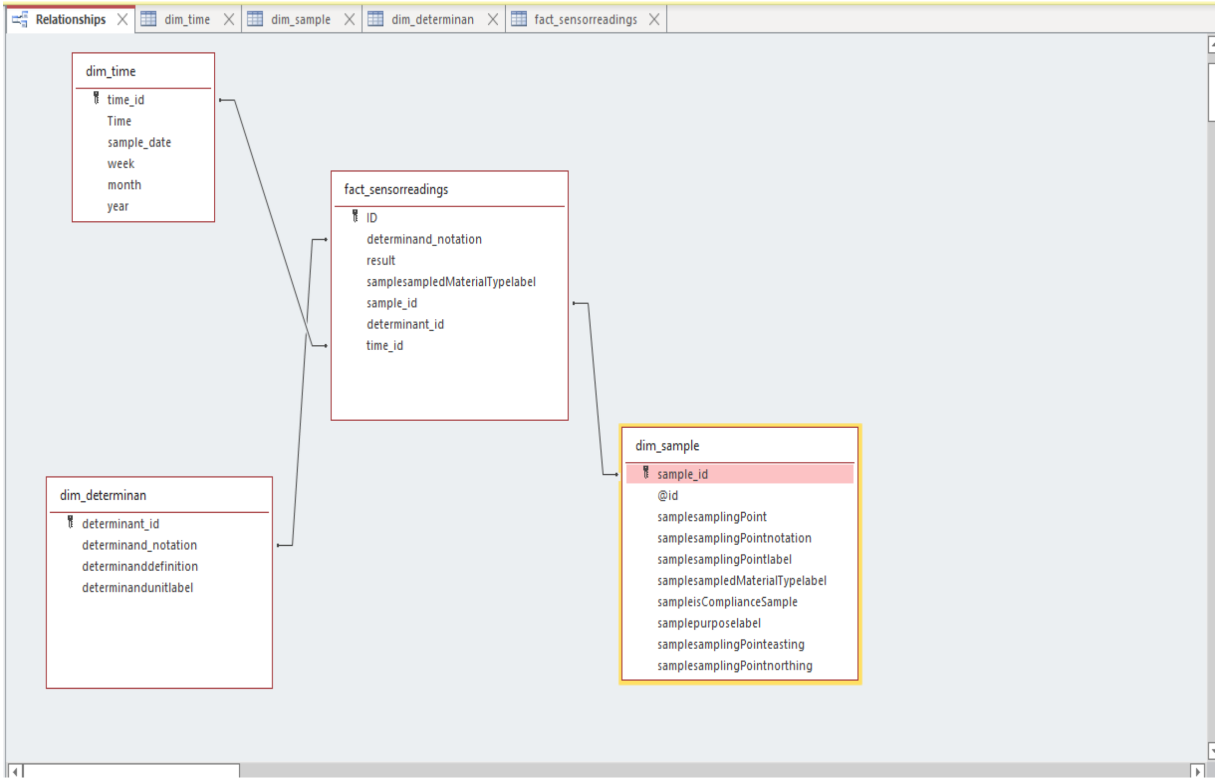


Figure 6-Star Schema Design

Verifying how null values and Unique values in the columns of newly extracted water quality tables using the below command shown in figure

```
---VERIFYING NULL AND NOT NULL VALUES IN THE TABLE
select column_name,nullable,num_distinct,num_nulls from all_tab_columns

where owner='RG0002D' and table_name='WATERQUALITY' order by column_id;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 18 in 0.411 seconds

	COLUMN_NAME	NULLABLE	NUM_DISTINCT	NUM_NULLS
1	ID	Y	24928	0
2	@id	Y	24796	0
3	samplesamplingPoint	Y	114	0
4	samplesamplingPointnotation	Y	114	0
5	samplesamplingPointlabel	Y	114	0
6	samplesampleDateTime	Y	2027	0
7	determinandlabel	Y	350	0
8	determinanddefinition	Y	349	0
9	determinandnotation	Y	350	0
10	resultQualifiernotation	Y	2	19447
11	result	Y	4239	0
12	codedResultInterpretationinterpretation	Y	0	24931
13	determinandunitlabel	Y	15	0
14	samplesampledMaterialTypelabel	Y	11	0
15	sampleisComplianceSample	Y	2	0
16	samplepurposelabel	Y	14	0
17	samplesamplingPointeasting	Y	112	0
18	samplesamplingPointnorthing	Y	114	0

Figure 7-Verifying Null values

Based on the above result it seems multiple columns *codedResultInterpretationinterpretation* and *resultQualifiernotation* which consists of too many nullvalues which can be dropped from the waterquality table since it wont be useful for the Analytic process.

By executing below command shown in figure ,the columns with excessive null records are removed.

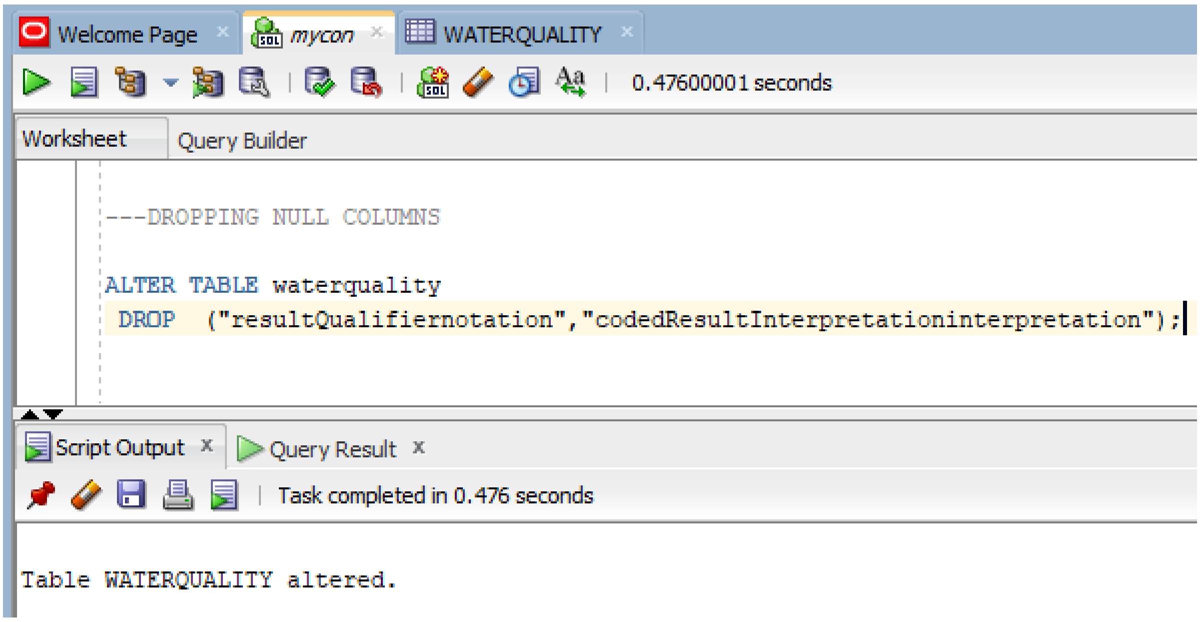


Figure 8-Dropping columns

Describing the water quality table after removing the null columns .

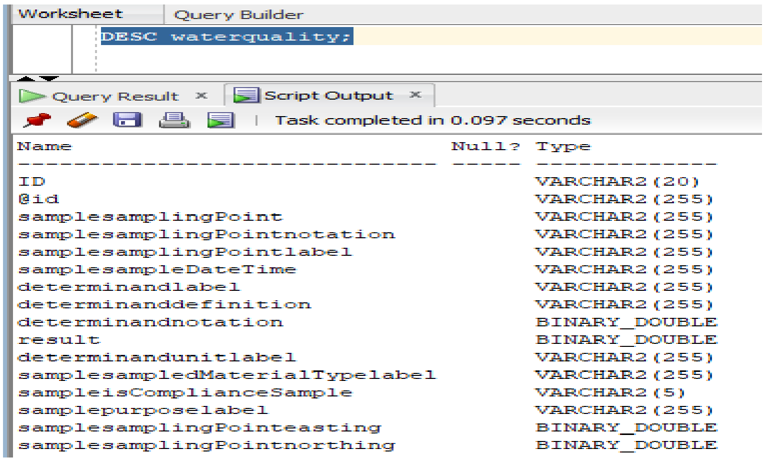


Figure 9-Describing after removing

Based on the design, Implementing sql query for creating DIM_SAMPLE shown in below figure, Using surrogate key SAMPLE_ID as Primary key which is also auto generated one.

Star Schema Implementation:

Based the Star schema design now it is implemented by creating fact and dimension tables, Firstly creating DIM_SAMPLE based on the sql query shown in figure, this table classify only determinant data from the staging area *SAMPLE_ID* will act as a PRIMARY KEY ,which will have all sample related value.

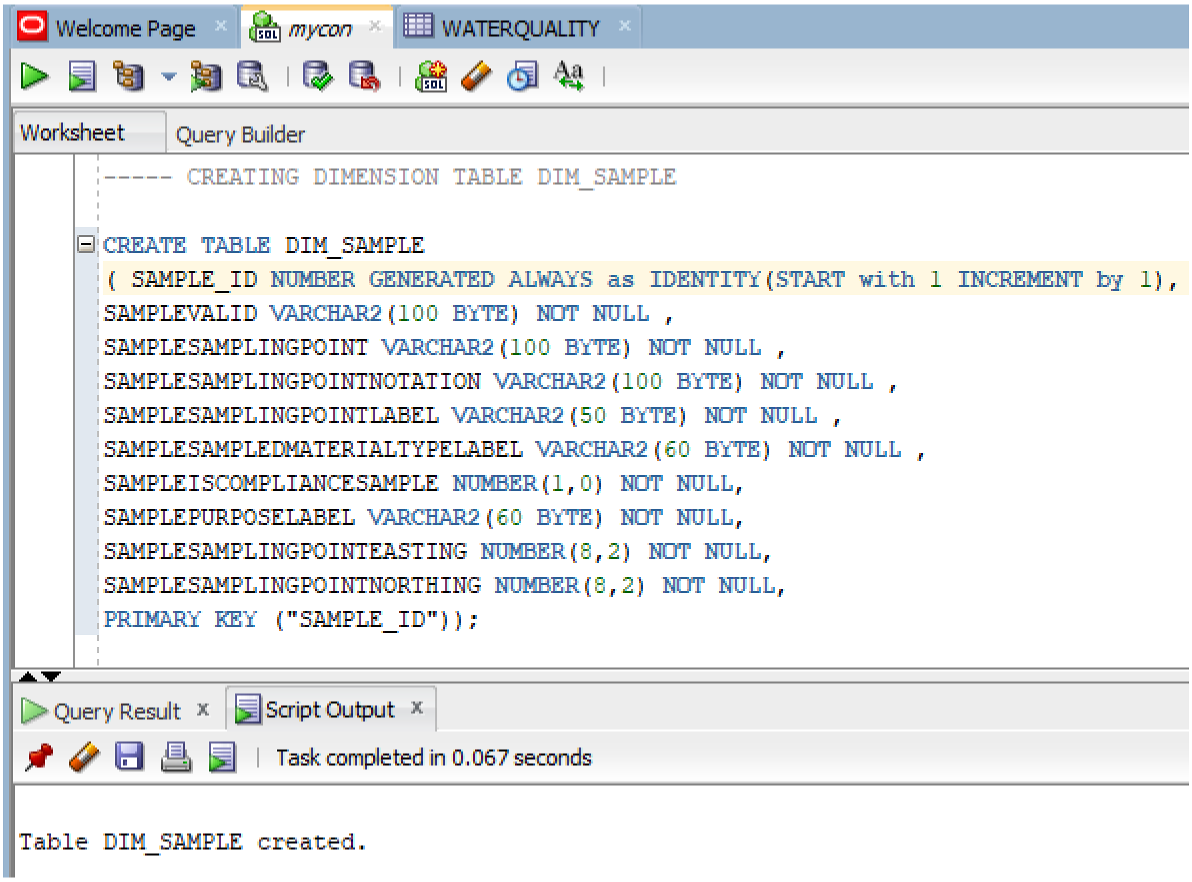


Figure 10-Creating Sample Dimension

In the above query columns are created with appropriate data types which is different from the staging area as part of Data Cleansing and maintain Data integrity NOT NULL and PRIMARY KEY are part of maintaining data integrity. Creating Surrogate key SAMPLE_ID which autogenerates start from 1.

Creating table Determinant dimension based on the sql query shown in figure, this table classify only determinant data from the staging area *DETERMINANTNOTATION* will act as a PRIMARY KEY since I has unique value specific for each water sensors(*DETERMINANDLABEL*).

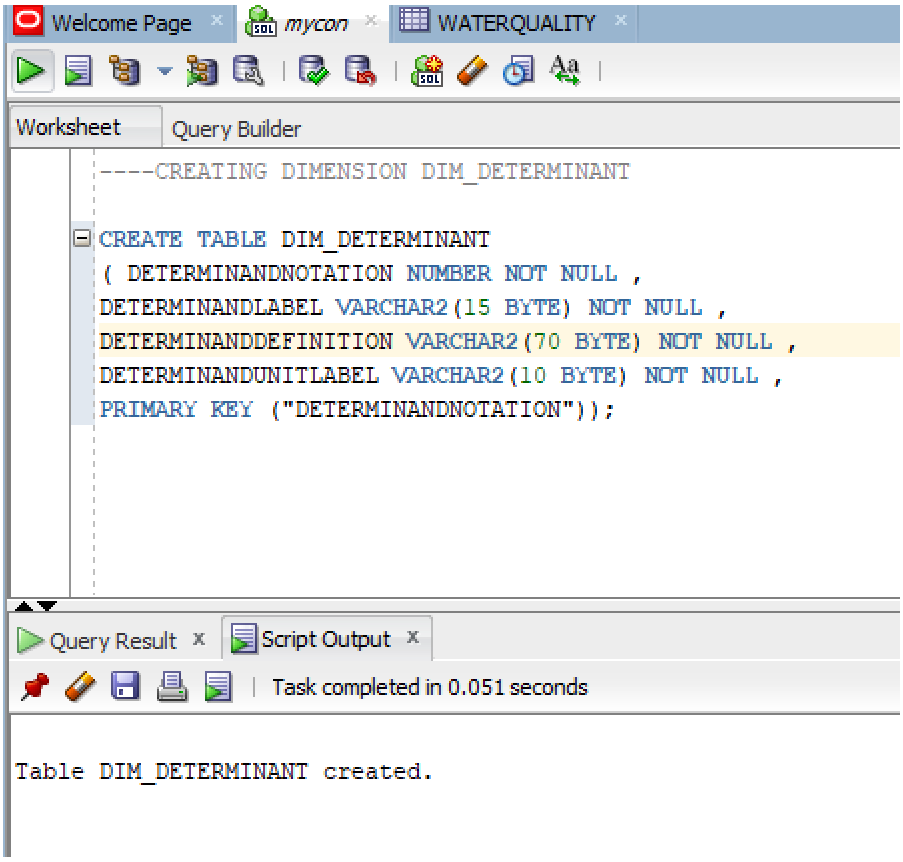


Figure 11-Creating Determinant Dimension

Implementing Time Dimension table DIM_TIME by the following query shown in figure which has different Date and time formats which would be a important for Analysing the data timely. TIME_ID will be the PRIMARY KEY .

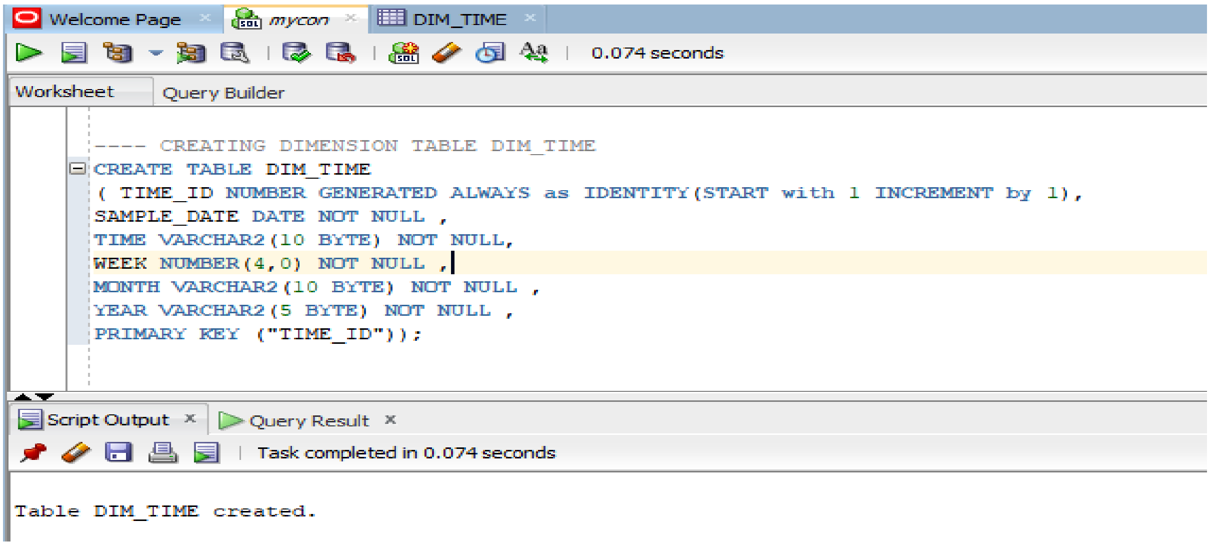


Figure 12-Creating Time Dimension

Implementing the Fact table factwatersensor with PRIMARY KEY ID and enforcing referential integrity with all dimension table using foreign keys and mapping to delete a record in the fact table, whenever a data is removed from determinant table.This fact water sensor table is referenced by SAMPLE_ID,TIME_ID,DETERMINANTNOTATION are the FOREIGNKEY .By applying this reference fact table will now have direct relationship with the dimension table

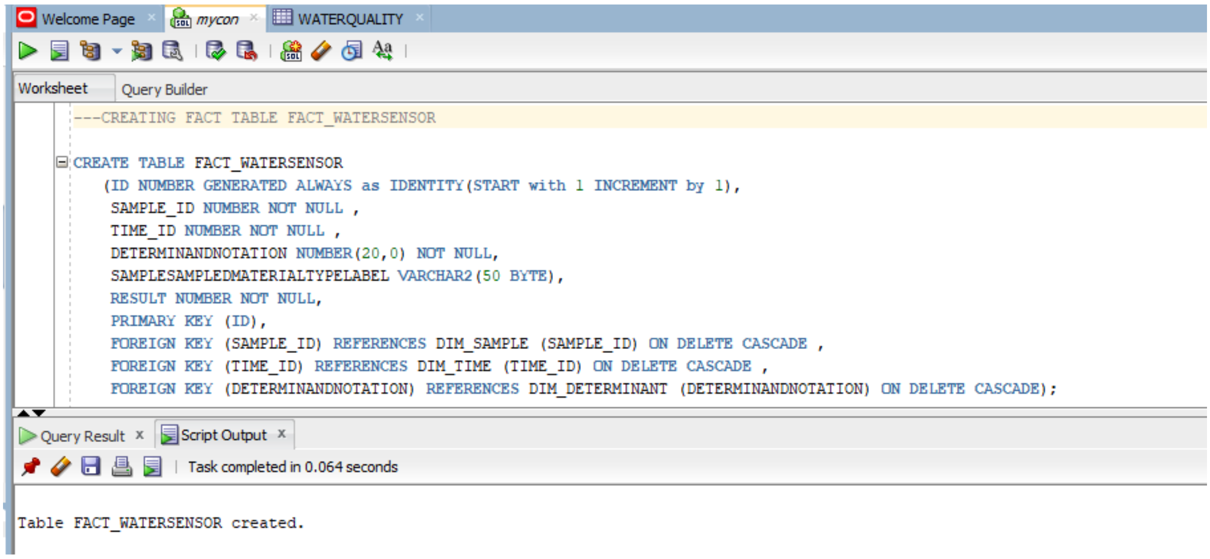


Figure 13-Creating FACT Table

After creating fact and dimension tables, also changing column data types based on the data from the staging area to verify all the data integrity before feeding the data, The Star Schema designed is implemented which is show in below figure

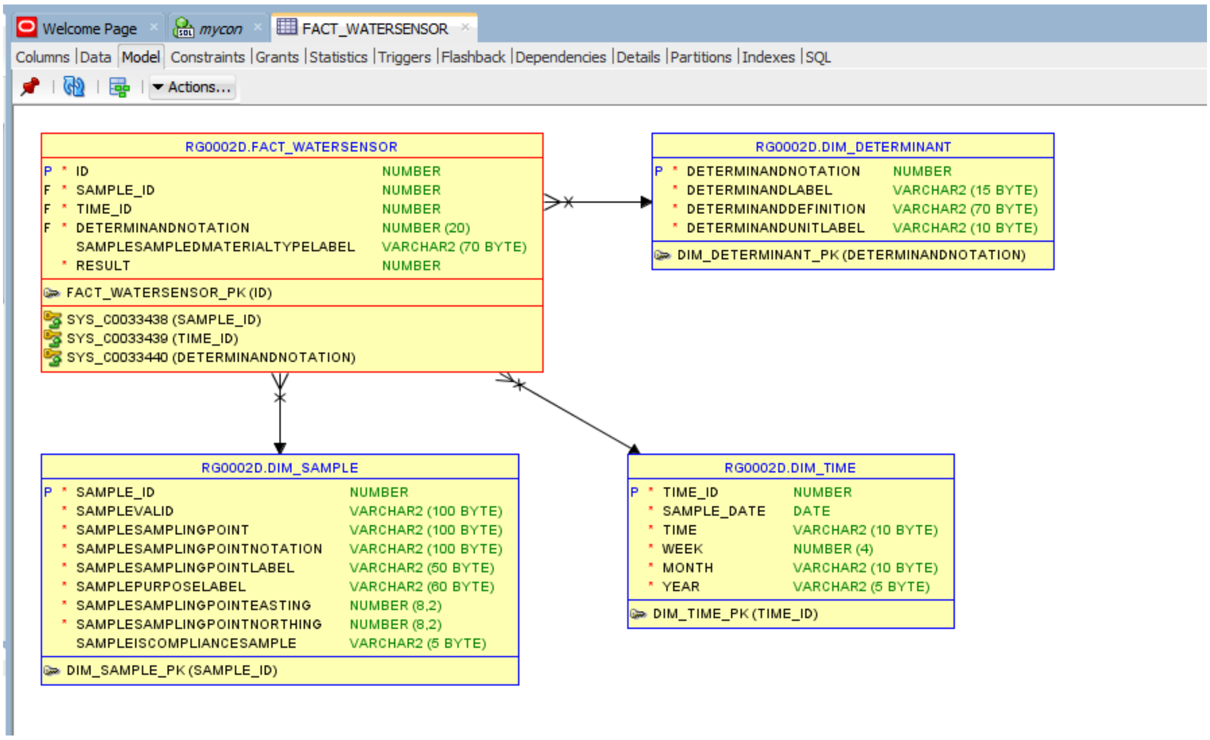


Figure 14-Star Schema Implementation

Transforming and Loading Data:

Since the sample dimension table DIM_SAMPLE is created, now the data can be populated from staging area using which all related to sample by Cursor based on the query shown below

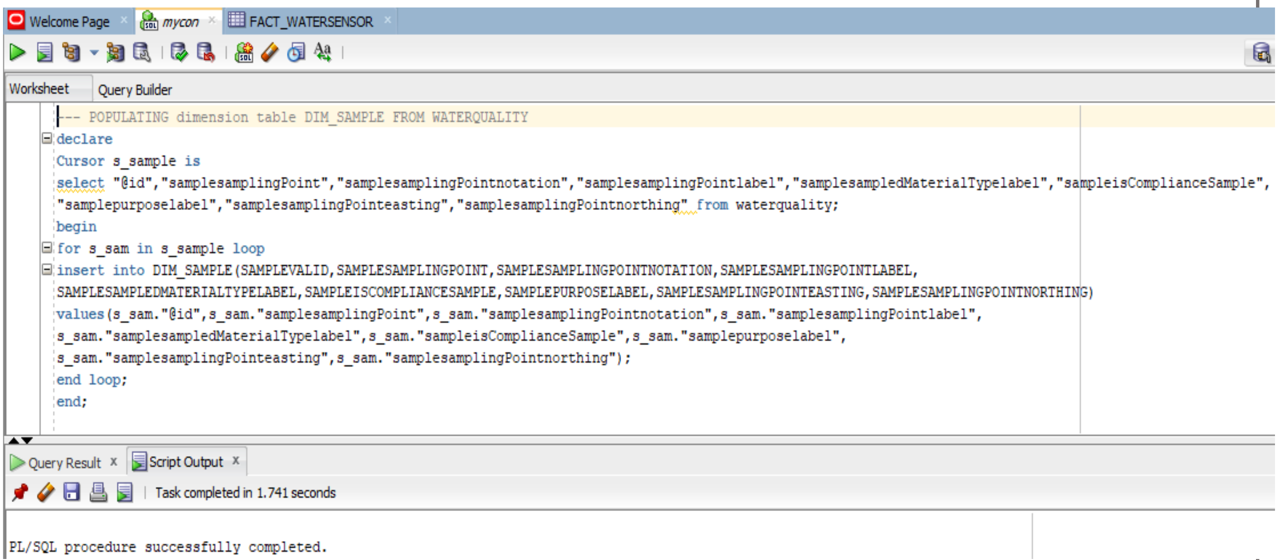


Figure 15-Populating Sample Dimension

Verifying how many are populated in *DIM_SAMPLE* by using the count keyword

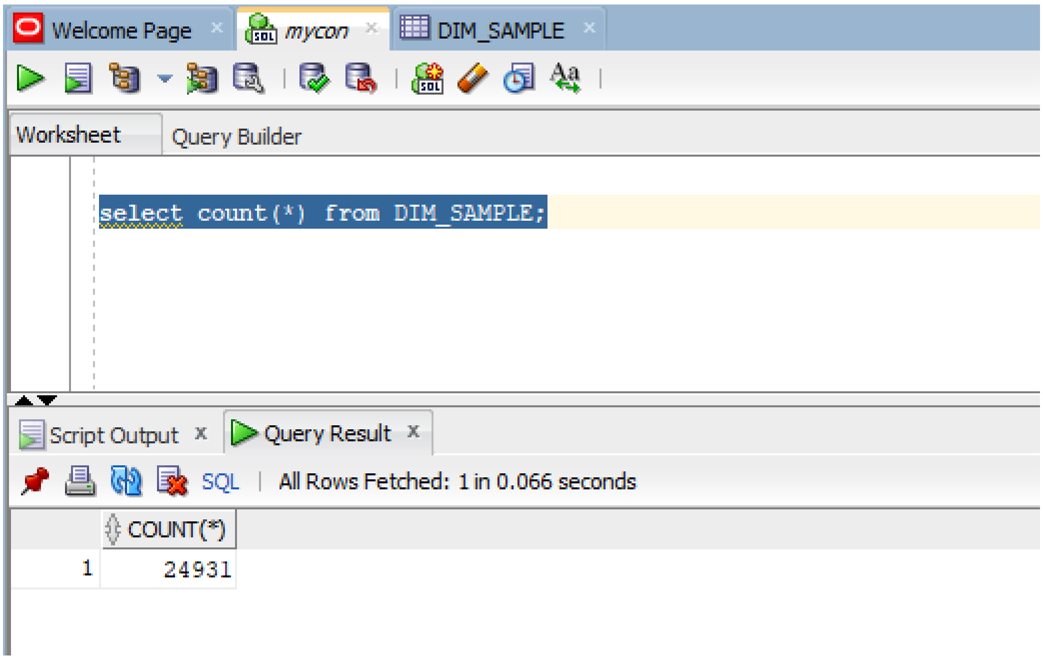


Figure 16-Verifying populated Sample data

Time dimension table is populated from column *"samplesampleDateTime"* from staging area waterquality which would be transformed using TO DATE function for changing incoming date value of varchar data type to date data type which would be further used for other date related columns like TIME,MONTH and year using TO CHAR and TO NUMBER.

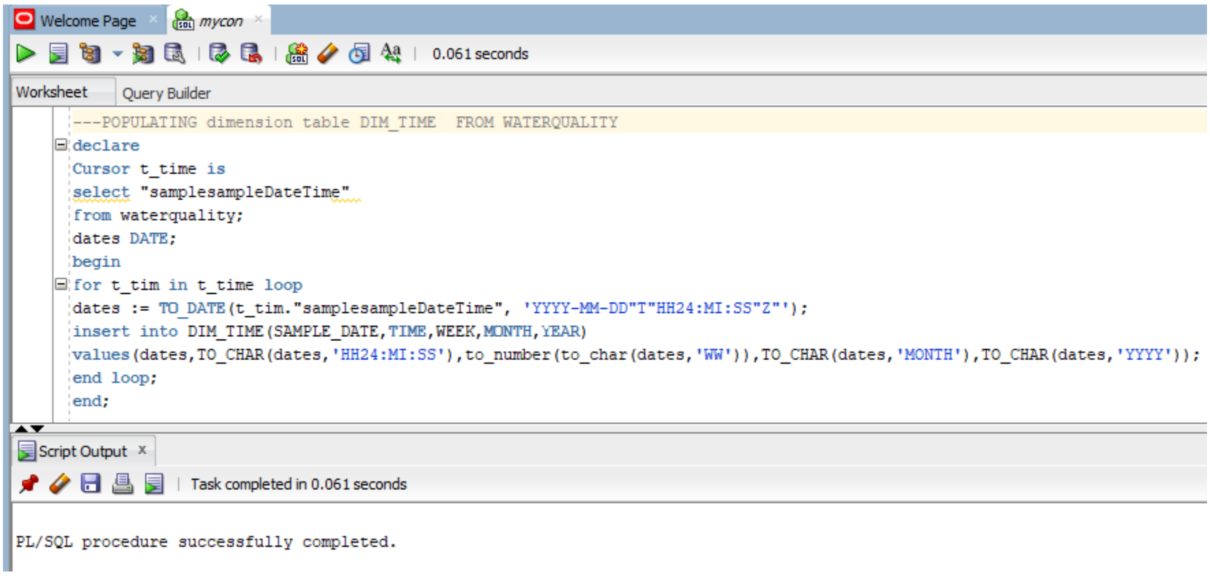


Figure 17- Populating Time Dimension

Verifying the populated records in DIM_TIME

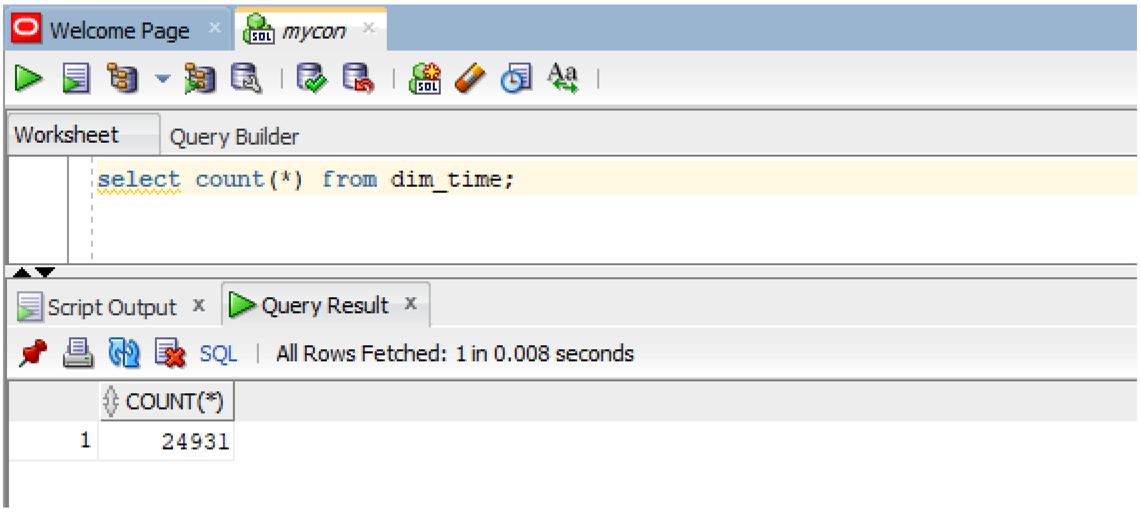


Figure 18-Verifying TIme Dimension Data

Similarly determinant dimension also populated from staging area using similar approach but fetching the distinct sensor type would be more informative and ensures the unique information to the specific sensor type which shown in below figure.

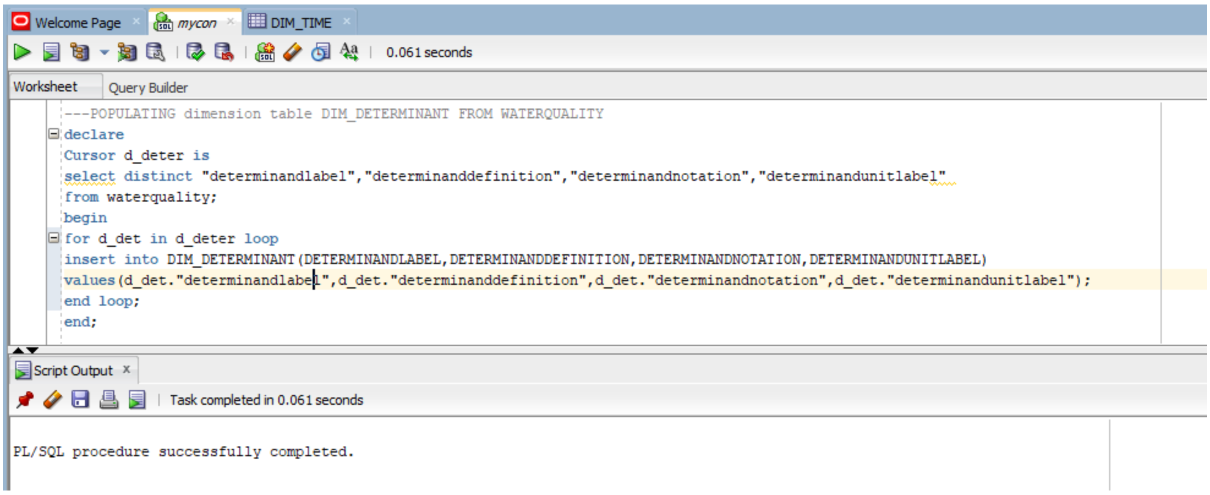


Figure 19-Populating Determinant Table

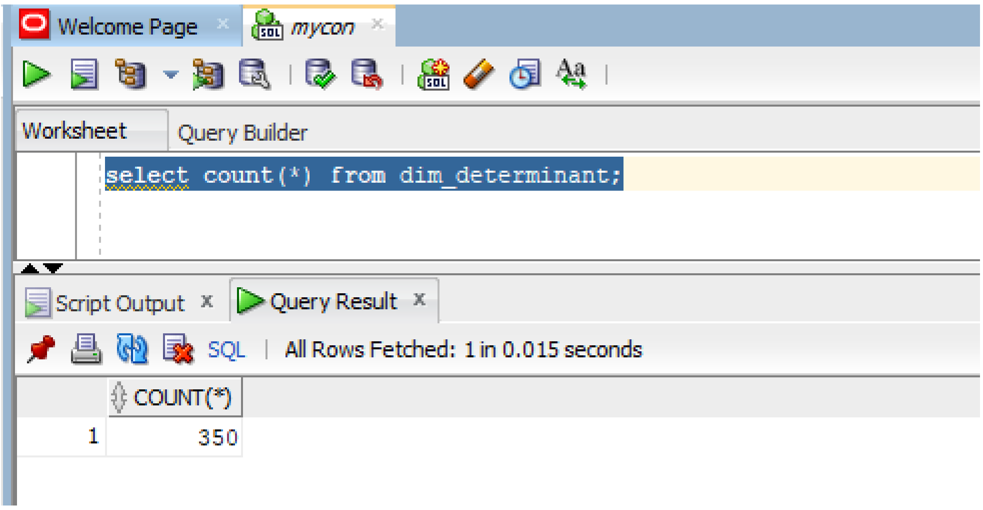


Figure 20-verifying Determinant Dimension

Once all the dimension tables are populated ,now the fact table need to be populated since it has referential constraint with the dimension tables. For inserting the data, Cursor function is used by selecting the data using the conditions for fetching it from the staging which would reference all the dimension table values and provides the result ,which is inserted into the fact table.

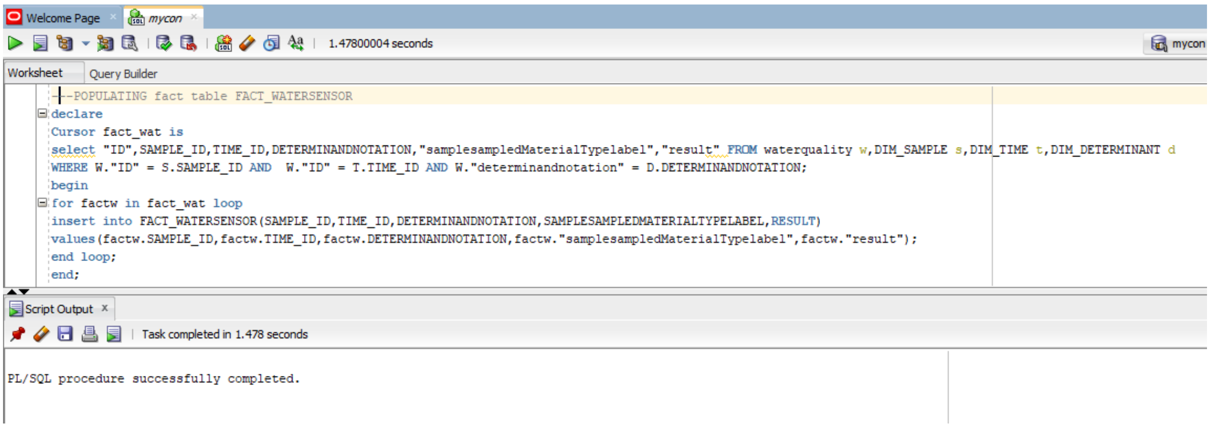


Figure 21-Populating Fact Water Sensor

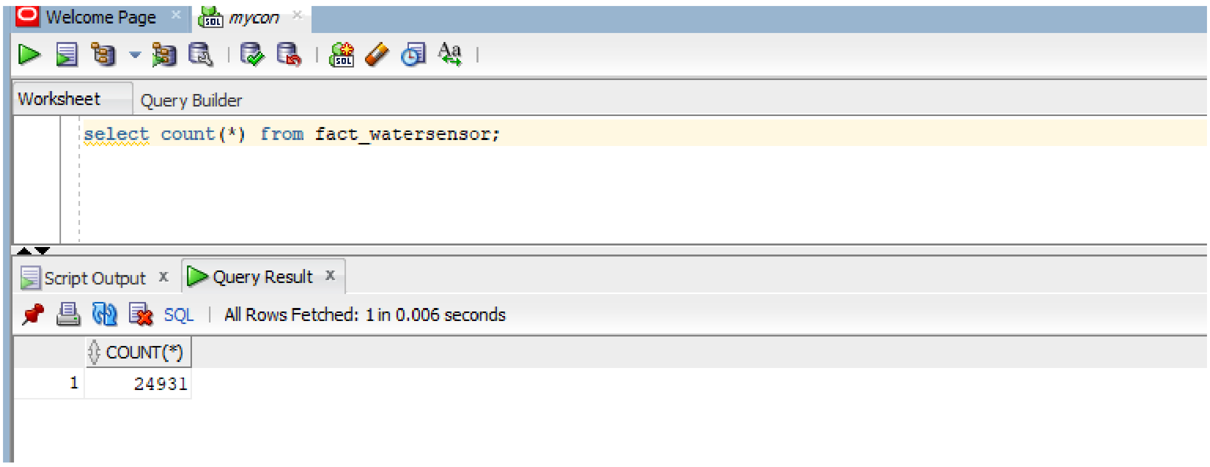


Figure 22-Verifying the Fact Tabl

Statistical Queries:

The list of water sensors measured by type of it by month

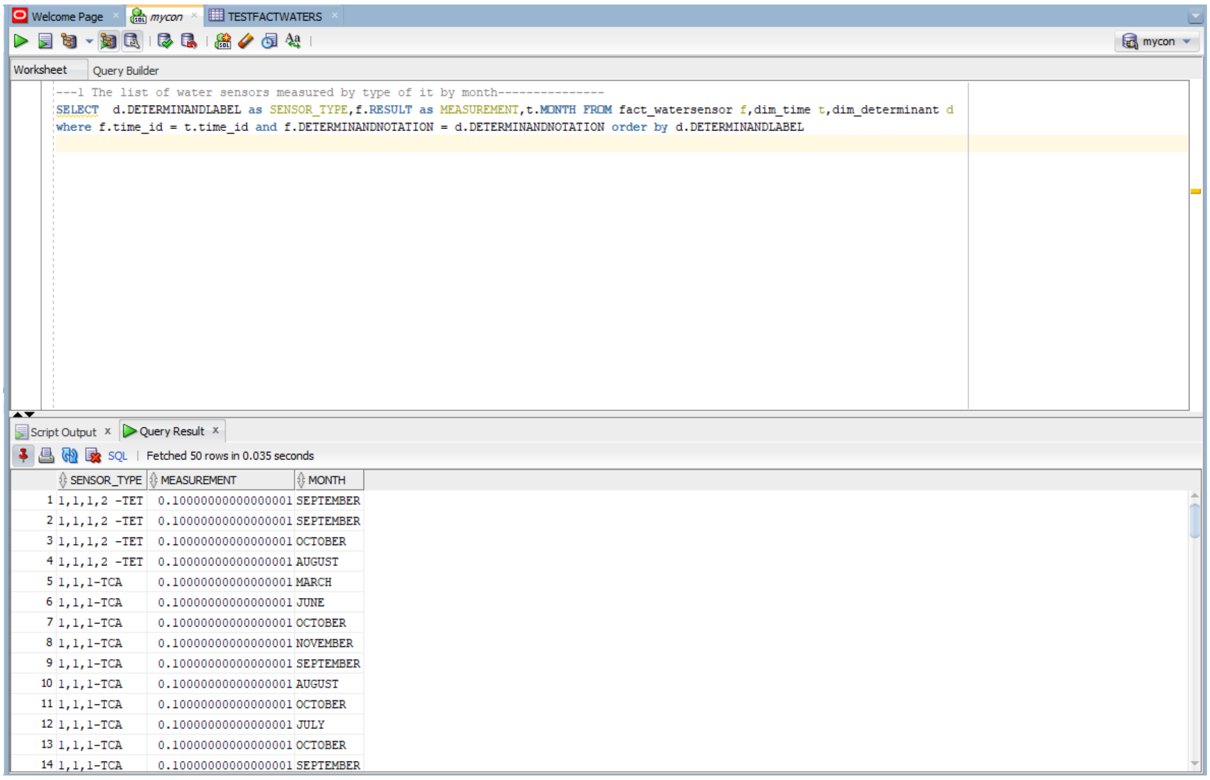


Figure 23-Type of sensor measurement by month

The number of sensor measurements collected by type of sensor by week

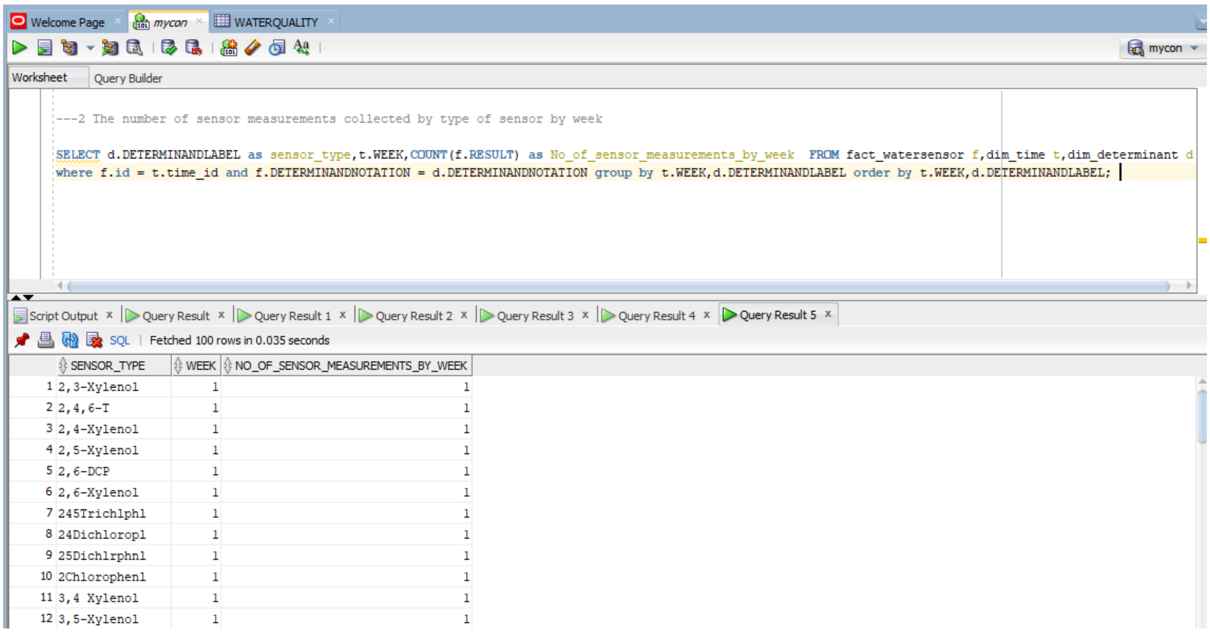
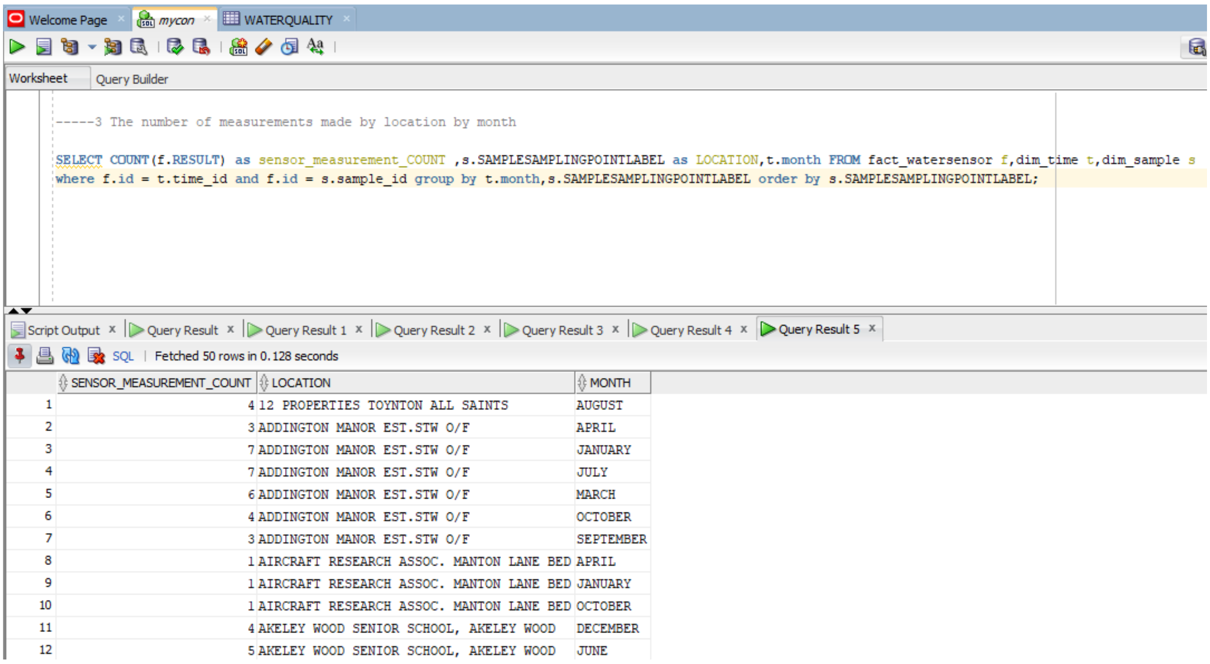


Figure 24- Type of sensor measurement by week

The number of measurements made by location by month



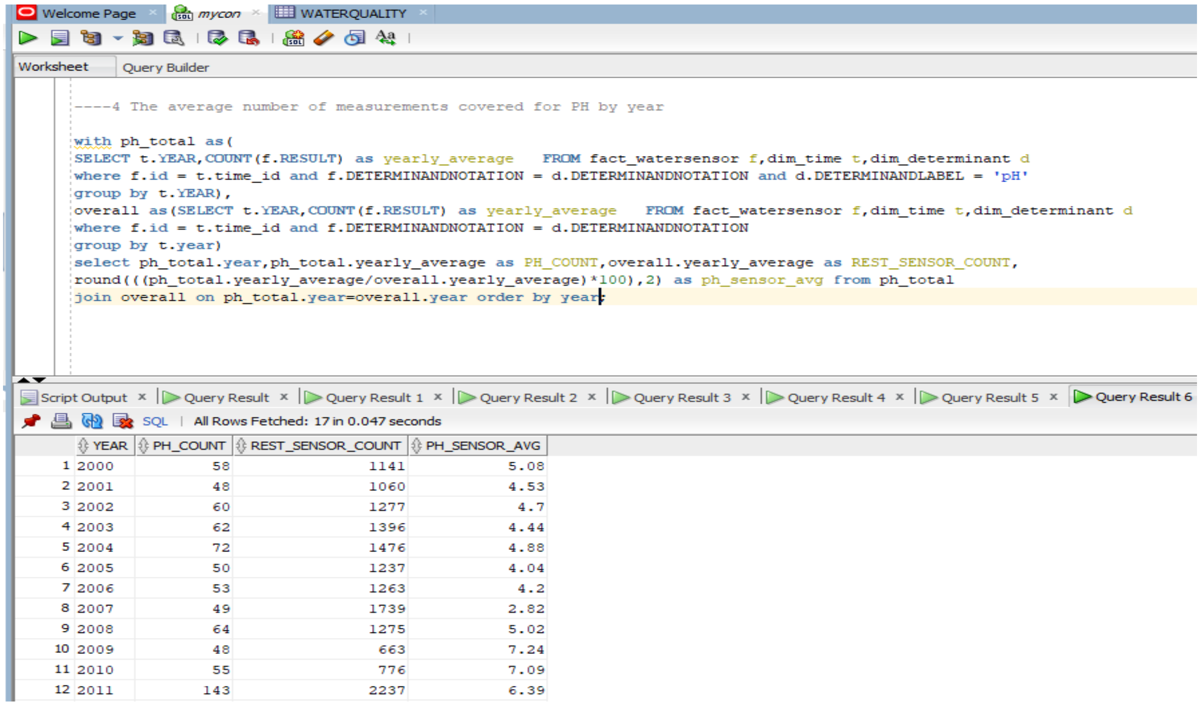
```sql
-----3 The number of measurements made by location by month

SELECT COUNT(f.RESULT) as sensor_measurement_COUNT ,s.SAMPLESAMPLINGPOINTLABEL as LOCATION,t.month FROM fact_watersensor f,dim_time t,dim_sample s
where f.id = t.time_id and f.id = s.sample_id group by t.month,s.SAMPLESAMPLINGPOINTLABEL order by s.SAMPLESAMPLINGPOINTLABEL;
```

Fetched 50 rows in 0.128 seconds

	SENSOR_MEASUREMENT_COUNT	LOCATION	MONTH
1	4	12 PROPERTIES TOYNTON ALL SAINTS	AUGUST
2	3	ADDINGTON MANOR EST.STW O/F	APRIL
3	7	ADDINGTON MANOR EST.STW O/F	JANUARY
4	7	ADDINGTON MANOR EST.STW O/F	JULY
5	6	ADDINGTON MANOR EST.STW O/F	MARCH
6	4	ADDINGTON MANOR EST.STW O/F	OCTOBER
7	3	ADDINGTON MANOR EST.STW O/F	SEPTEMBER
8	1	AIRCRAFT RESEARCH ASSOC. MANTON LANE BED	APRIL
9	1	AIRCRAFT RESEARCH ASSOC. MANTON LANE BED	JANUARY
10	1	AIRCRAFT RESEARCH ASSOC. MANTON LANE BED	OCTOBER
11	4	AKELEY WOOD SENIOR SCHOOL, AKELEY WOOD	DECEMBER
12	5	AKELEY WOOD SENIOR SCHOOL, AKELEY WOOD	JUNE

Figure 25- No of Measurements based on location & Month

The average number of measurements covered for PH by year



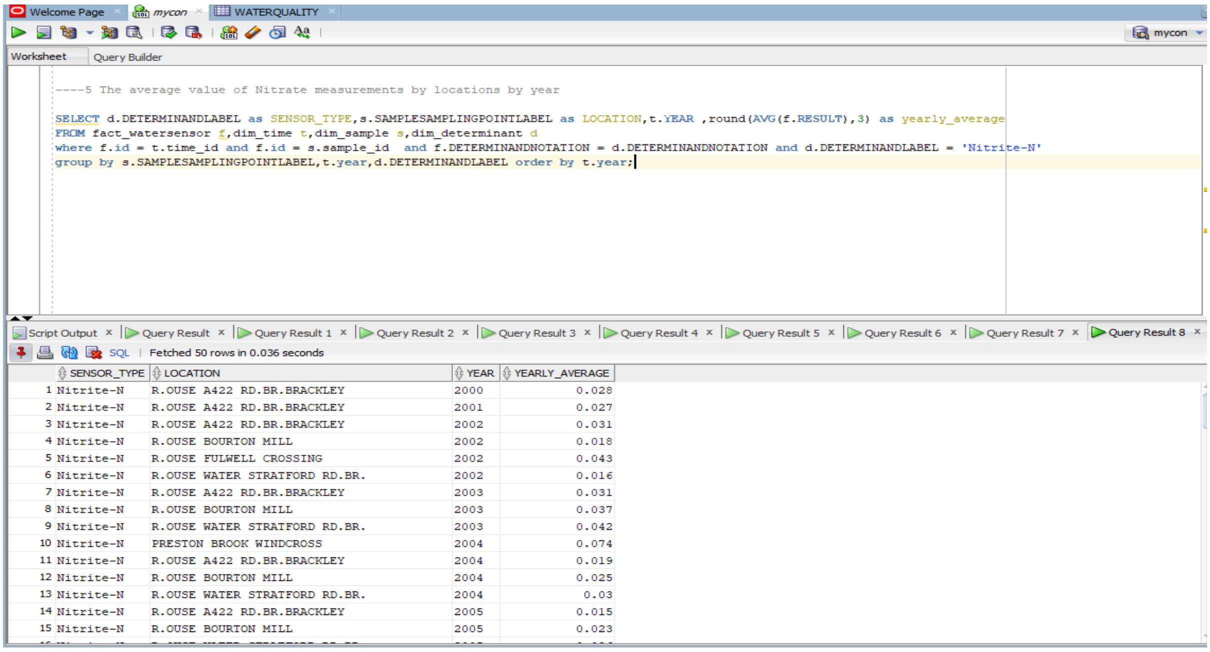
```sql
----4 The average number of measurements covered for PH by year

with ph_total as(
SELECT t.YEAR,COUNT(f.RESULT) as yearly_average   FROM fact_watersensor f,dim_time t,dim_determinant d
where f.id = t.time_id and f.DETERMINANDNOTATION = d.DETERMINANDNOTATION and d.DETERMINANDLABEL = 'pH'
group by t.YEAR),
overall as(SELECT t.YEAR,COUNT(f.RESULT) as yearly_average   FROM fact_watersensor f,dim_time t,dim_determinant d
where f.id = t.time_id and f.DETERMINANDNOTATION = d.DETERMINANDNOTATION
group by t.year)
select ph_total.year,ph_total.yearly_average as PH_COUNT,overall.yearly_average as REST_SENSOR_COUNT,
round(((ph_total.yearly_average/overall.yearly_average)*100),2) as ph_sensor_avg from ph_total
join overall on ph_total.year=overall.year order by year
```

All Rows Fetched: 17 in 0.047 seconds

	YEAR	PH_COUNT	REST_SENSOR_COUNT	PH_SENSOR_AVG
1	2000	58	1141	5.08
2	2001	48	1060	4.53
3	2002	60	1277	4.7
4	2003	62	1396	4.44
5	2004	72	1476	4.88
6	2005	50	1237	4.04
7	2006	53	1263	4.2
8	2007	49	1739	2.82
9	2008	64	1275	5.02
10	2009	48	663	7.24
11	2010	55	776	7.09
12	2011	143	2237	6.39

Figure 26-Average count of pH measurements based on year

The average value of Nitrate measurements by locations by year



```sql
----5 The average value of Nitrate measurements by locations by year

SELECT d.DETERMINANDLABEL as SENSOR_TYPE,s.SAMPLESAMPLINGPOINTLABEL as LOCATION,t.YEAR ,round(AVG(f.RESULT),3) as yearly_average
FROM fact_watersensor f,dim_time t,dim_sample s,dim_determinant d
where f.id = t.time_id and f.id = s.sample_id  and f.DETERMINANDNOTATION = d.DETERMINANDNOTATION and d.DETERMINANDLABEL = 'Nitrite-N'
group by s.SAMPLESAMPLINGPOINTLABEL,t.year,d.DETERMINANDLABEL order by t.year;
```

Fetched 50 rows in 0.036 seconds

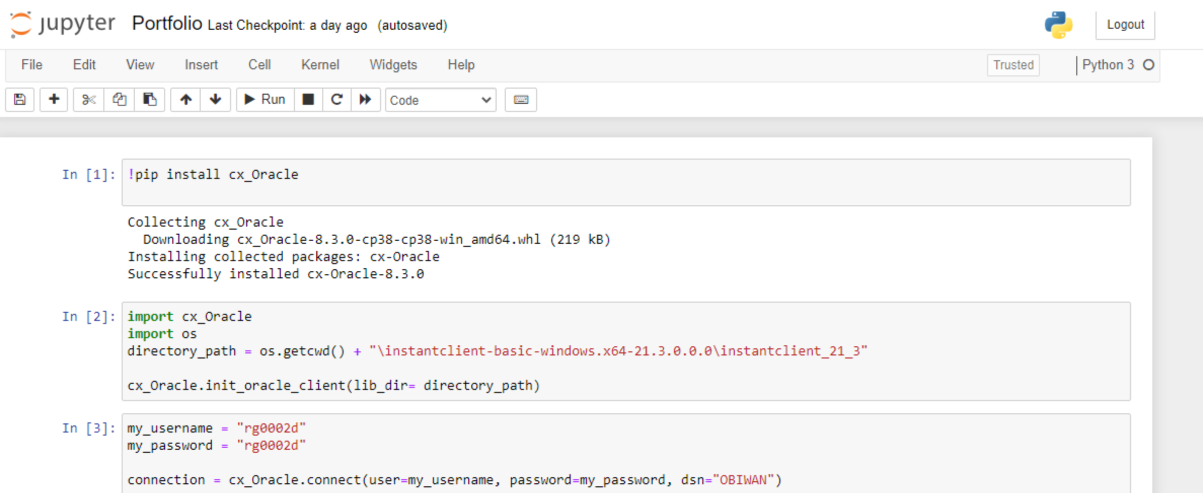
	SENSOR_TYPE	LOCATION	YEAR	YEARLY_AVERAGE
1	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	2000	0.028
2	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	2001	0.027
3	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	2002	0.031
4	Nitrite-N	R.OUSE BOURTON MILL	2002	0.018
5	Nitrite-N	R.OUSE FULWELL CROSSING	2002	0.043
6	Nitrite-N	R.OUSE WATER STRATFORD RD.BR.	2002	0.016
7	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	2003	0.031
8	Nitrite-N	R.OUSE BOURTON MILL	2003	0.037
9	Nitrite-N	R.OUSE WATER STRATFORD RD.BR.	2003	0.042
10	Nitrite-N	PRESTON BROOK WINDCROSS	2004	0.074
11	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	2004	0.019
12	Nitrite-N	R.OUSE BOURTON MILL	2004	0.025
13	Nitrite-N	R.OUSE WATER STRATFORD RD.BR.	2004	0.03
14	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	2005	0.015
15	Nitrite-N	R.OUSE BOURTON MILL	2005	0.023

Figure 27-Average Nitrate measurement per year

Analysis Using Python:

After transformed data populated in the fact and dimension table, the data is used for predicting the future values by specific sensor by establishing connection between oracle SQL developer and python.

By executing below mentioned steps in a python notebook, connection can be established to the oracle server.



```
In [1]: !pip install cx_Oracle

Collecting cx_Oracle
  Downloading cx_Oracle-8.3.0-cp38-cp38-win_amd64.whl (219 kB)
Installing collected packages: cx-Oracle
Successfully installed cx-Oracle-8.3.0

In [2]: import cx_Oracle
import os
directory_path = os.getcwd() + "\instantclient-basic-windows.x64-21.3.0.0.0\instantclient_21_3"

cx_Oracle.init_oracle_client(lib_dir= directory_path)

In [3]: my_username = "rg0002d"
my_password = "rg0002d"

connection = cx_Oracle.connect(user=my_username, password=my_password, dsn="OBIWAN")
```

Figure 28-Connection Parameters

Importing the libraries that are required to Analyse the data for predicting the future values. Using connection cursor function to fetch the average measurement of 'pH' water sensor based the year, address and the material type of the sensor also whether its compliance is True or False and assigned it to a variable which are described in the below code.

```
In [24]: import numpy as np
         import pandas as pd
         from pandas import DataFrame
         import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib inline
```

```
In [27]: with connection.cursor() as cursor:
             cursor.execute("SELECT d.DETERMINANDLABEL as SENSOR_TYPE,s.SAMPLESAMPLINGPOINTLABEL as LOCATION,f.samplesampledmaterialtypela
             df = DataFrame(cursor.fetchall())
             df.columns = [x[0] for x in cursor.description]
             print("I got %d lines " % len(df))

         I got 143 lines
```

Figure 29-Required Modules

After fetching the data ,now analysing the datatype of the dataset and structure of the dataset below.

```
In [6]: df.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 121 entries, 0 to 120
        Data columns (total 6 columns):
         #   Column                          Non-Null Count  Dtype
        ---  ------                          --------------  -----
         0   SENSOR_TYPE                     121 non-null    object
         1   LOCATION                        121 non-null    object
         2   SAMPLESAMPLEDMATERIALTYPELABEL  121 non-null    object
         3   SAMPLEISCOMPLIANCESAMPLE        121 non-null    object
         4   YEAR                            121 non-null    object
         5   YEARLY_AVERAGE                  121 non-null    float64
        dtypes: float64(1), object(5)
        memory usage: 5.8+ KB
```

```
In [7]: df
```

Out[7]:

	SENSOR_TYPE	LOCATION	SAMPLESAMPLEDMATERIALTYPELABEL	SAMPLEISCOMPLIANCESAMPLE	YEAR	YEARLY_AVERAGE
0	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	RIVER / RUNNING SURFACE WATER	FALSE	2000	0.03
1	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	RIVER / RUNNING SURFACE WATER	FALSE	2001	0.03
2	Nitrite-N	R.OUSE A422 RD.BR.BRACKLEY	RIVER / RUNNING SURFACE WATER	FALSE	2002	0.03
3	Nitrite-N	R.OUSE BOURTON MILL	RIVER / RUNNING SURFACE WATER	FALSE	2002	0.02
4	Nitrite-N	R.OUSE FULWELL CROSSING	RIVER / RUNNING SURFACE WATER	FALSE	2002	0.04
...	...	...	...	...	...	...
116	Nitrite-N	R.TOVE CAPPENHAM BRIDGE	RIVER / RUNNING SURFACE WATER	FALSE	2016	0.03
117	Nitrite-N	R.TOVE COSGROVE PARK FT.BR.	RIVER / RUNNING SURFACE WATER	FALSE	2016	0.04
118	Nitrite-N	SILVERSTONE BK.A43 RD.BR.	ANY WATER	FALSE	2016	0.03
119	Nitrite-N	SILVERSTONE BK.A43 RD.BR.	RIVER / RUNNING SURFACE WATER	FALSE	2016	0.13
120	Nitrite-N	SULGRAVE BK.WESTON RD.BR.	RIVER / RUNNING SURFACE WATER	FALSE	2016	0.01

121 rows × 6 columns

Figure 30-Verifying the dataset

Validating whether the fetched data has null columns are not

```
In [8]: df.isna().sum()

Out[8]: SENSOR_TYPE                      0
        LOCATION                         0
        SAMPLESAMPLEDMATERIALTYPELABEL   0
        SAMPLEISCOMPLIANCESAMPLE         0
        YEAR                             0
        YEARLY_AVERAGE                   0
        dtype: int64
```

Figure 31-Validating Null Columns

Validating the yearly average values how it have spreaded in the data set using normal distribution graph

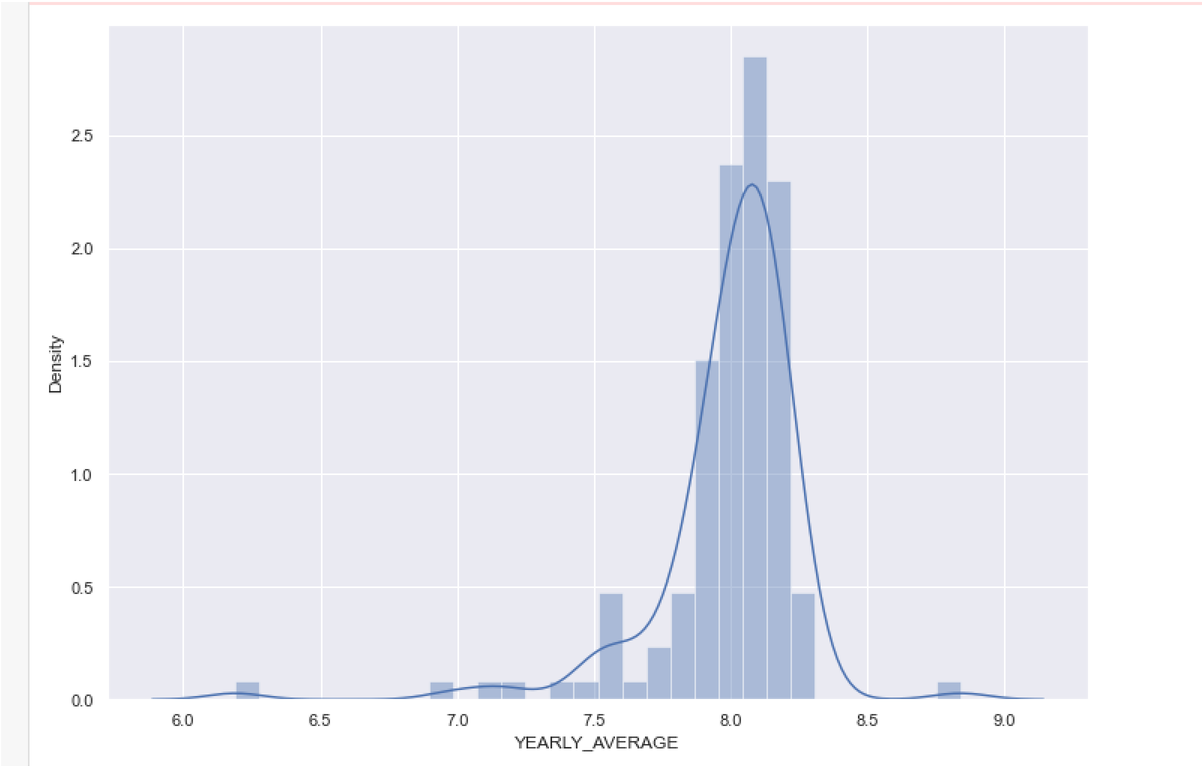


Figure 32-Average measurements per year value spread

The dataset has more qualitative data column values which is now transformed quantitative data using Label encoding which can now passed to ML algorithms for predicting the future values.

In [14]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['LOCATION'] = le.fit_transform(df['LOCATION'])
df['SAMPLESAMPLEDMATERIALTYPELABEL'] = le.fit_transform(df['SAMPLESAMPLEDMATERIALTYPELABEL'])
df['SAMPLEISCOMPLIANCESAMPLE'] = le.fit_transform(df['SAMPLEISCOMPLIANCESAMPLE'])
df
```

Out[14]:

	LOCATION	SAMPLESAMPLEDMATERIALTYPELABEL	SAMPLEISCOMPLIANCESAMPLE	YEAR	YEARLY_AVERAGE
0	32	4	0	2000	0.03
1	32	4	0	2001	0.03
2	32	4	0	2002	0.03
3	34	4	0	2002	0.02
4	35	4	0	2002	0.04
...	...	...	...	...	...
116	40	4	0	2016	0.03
117	41	4	0	2016	0.04
118	45	0	0	2016	0.03
119	45	4	0	2016	0.13
120	47	4	0	2016	0.01

Figure 33-Data Conversion

Linear Regression model is a statistical method, which predicts the dependent variable with a set of independent variable For measuring the efficiency of linear regression there are some of performance metrices which can be applied that is R square value, The transformed dataset is now split into two parts where y is the target column data and X contains the factors,which used to predict the values. Since there is mismatch between the range of values between the features MinMax Scaler is applied to optimise the X data.

Using sklearn library the data is further split into training and test data with training size with 70%of the data and test data has 30 %.These data are now passed in Linear Regression linear regression library function and getting the prediction values.

The predicted value is now store in variable y_pred which is further used for validating.

```
In [12]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         df['SAMPLESAMPLEDMATERIALTYPELABEL'] = le.fit_transform(df['SAMPLESAMPLEDMATERIALTYPELABEL'])
         df['LOCATION'] = le.fit_transform(df['LOCATION'])
         df['SAMPLEISCOMPLIANCESAMPLE'] = le.fit_transform(df['SAMPLEISCOMPLIANCESAMPLE'])

In [28]: X = df.iloc[:,:-1].values  #independent variable array
         y = df.iloc[:,4].values #dependent variable vector

In [29]: from sklearn.preprocessing import MinMaxScaler
         X = MinMaxScaler().fit_transform(X)

In [21]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.70, random_state=8)

In [22]: from sklearn.linear_model import LinearRegression
         regressor = LinearRegression()
         regressor.fit(X_train,y_train) #actually produces the linear eqn for the data
         print(regressor.intercept_)
         print(regressor.coef_)

         8.243231978180656
         [ 0.1257519  -0.18971538 -0.51190797 -0.17162799]

In [23]: y_pred = regressor.predict(X_test)
         y_pred

Out[23]: array([7.9950616 , 8.04223138, 7.99513141, 7.97573949, 8.25068092,
                7.99300002, 8.02297907, 8.06155349, 8.02311869, 8.06134406,
                8.13450934, 8.06368487, 8.07661282, 7.98433485, 8.09586512,
                7.89893971, 8.00365696, 8.13024657, 7.999464  , 8.13877212,
                8.09806632, 7.95662681, 8.11305584, 7.62339875, 7.94583025,
                7.99520122, 7.96714413, 7.97566968, 7.96927552, 8.02077788,
                8.0808756 , 7.94369886, 8.00771031, 7.78425966, 7.49694877,
                7.94163728, 8.05061731, 8.01410448, 7.88828277, 7.52671839,
                8.07274873, 8.05082674, 7.97787088])
```

Figure 34-Linear Regression Model

Below scatterplot consists of the data points of both predicted and test value

```
In [24]: import seaborn as sns
         sns.regplot(x=y_test,y=y_pred)

Out[24]: <AxesSubplot:>
```

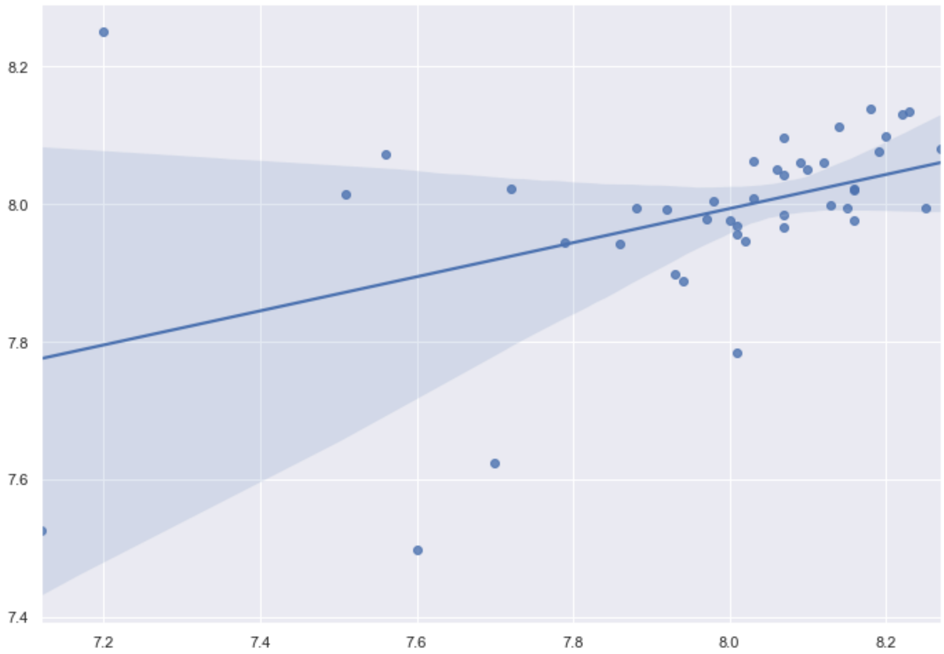


Figure 35-Scatter plot for the predicted values

As the model have predicted the results, For evaluating the models performance calculating Mean Square Error and Mean absolute error. Mean square error for measuring the distance of data points from the regression line ,where Mean absolute error is individual prediction errors for the absolute values in the test set. RMSE is also a validating parameter for validating   the difference between actual values and predicted values.

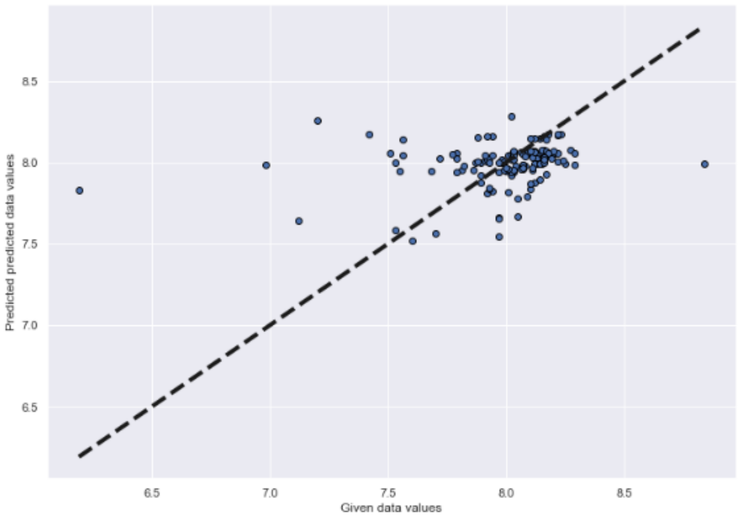
```
In [25]:
from sklearn import metrics
from sklearn.metrics import r2_score
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score is {}'.format(r2_score(y_test, y_pred)))

MAE: 0.13989675738146
MSE: 0.05334645972185372
RMSE: 0.23096852539221382
R2 score is 0.17510257924101968
```

Figure 36-Evaluation

Hyper tuning:

Cross Validation is a validating parameter used in machine learning regression approaches. When separating data for training and testing, the cross validation function reserves a set of data and tests the predicted data against it as part of the validation process. It examines the model's quality and determines whether the model is too big or too little, among other things. K fold is a cross validation strategy that divides the model into two sections, k and k-1, for training and k-n th values for testing. Below code demonstrates performing k fold cross validation for the dataset where the regression line results are more optimised than using manually.

In [26]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_predict
regressor = LinearRegression()
# cross_val_predict returns an array of the same size as `y` where each entry
# is a prediction obtained by cross validation:
predicted_dat = cross_val_predict(regressor, X, y, cv=10)

fig, ax = plt.subplots()
ax.scatter(y, predicted_dat, edgecolors=(0, 0, 0))
ax.plot([y.min(), y.max()], [y.min(), y.max()], "k--", lw=4)
ax.set_xlabel("Given data values")
ax.set_ylabel("Predicted predicted data values")
plt.show()
```



As per the Above model the results which predicted are pretty much average because of more number of outliers and there is not enough data for machine to train. On hyper tuning the model with K-Fold cross validation it gives better result than the manual one.

Advantages:

- It is simpler model used for predicting the dependent variable comparatively with other ML models.
- Since the algorithm mechanism is simpler, it easy to implement computations

- One of the primary reasons for Linear regression's popularity is its ability to determine the relative impact of predictor variables on the predicted value when the predictors are independent of one another.

Disadvantage:

- Not optimal choice for bigger problems.
- More outliers
- Couldn't predict the importance of the feature

Summary:

The above project is complete process for exporting data to sql and cleanse the data and stored in a dimensional model which further transferred to python for analytics using ML libraries.By which the course work comprises of a complete end to end Etl project with Analytics.

Group Work Partitioning:

GROUP ID: 2

Queries Part done by myself - **Rambabu Ganeshkumar**

- Queries (10%)

Below Specification have implemented by **Rojsun parameshwaran, Srhari venkatesan and Ilker Ozturk**

- Designing star schema including the Time dimension (10%)
- ETL: export the data from a Microsoft Access database into Oracle (5%)