COMP1848

Data Warehousing and Business Intelligence

Name: Nazri

Student ID: 001278084

MSc Big Data and Business Intelligence

School of Computing and Mathematical Science

Table of Contents

Introduction: Design and Implementation of a Water Quality Data Warehouse 3
Access Oracle servers using SQL Developer 4
Designing Water Quality Data Warehouse 5
Star Schema Design 6
BUS Plan 7
Extract Transform Load 8
Extract Process: Export Data from Microsoft Access to Oracle 9
Staging Area 13
Steps for Creating Staging Area: 14
Data Cleansing 17
Steps for Cleaning 18
Building the Warehouse – Loading the Data 21
Steps of Data Loading 23
Connection of Oracle and Python 36
SQL Queries 38
Conclusion 46
References 47

Introduction: Design and Implementation of a Water Quality Data Warehouse

Water quality monitoring is a critical aspect of environmental stewardship, ensuring the health and safety of our natural water resources. Traditional monitoring methods, however, present challenges such as cost, time consumption, and limited spatial coverage. This coursework endeavours to address these issues by proposing a comprehensive solution through the design and implementation of a data mart/data warehouse using the Oracle Database Management System (DBMS). The primary goal is to revolutionize water quality monitoring by shifting from periodic, resource-intensive sampling to a real-time, data-driven approach.

The traditional approach to water quality monitoring involves manual sampling, subsequent lab testing, and investigation. This process is not only costly and time-consuming but also provides data only at the specific points of sampling. Leveraging technology, we aim to provide a more efficient and cost-effective solution by introducing remote monitoring through a data warehouse. This approach allows for real-time data access, trend analysis, and alerts, empowering stakeholders to make informed decisions promptly.

The dataset originates from the Department for Environment Food & Rural Affairs, accessed through the Environment Agency's Water Quality Archive. The dataset, spanning from 2000 to 2016, encompasses measurements from various sampling points across England, including coastal or estuarine waters, rivers, lakes, ponds, canals, and groundwaters. Despite the wealth of data, concerns have been raised about data quality, necessitating careful consideration during the design and implementation of the data warehouse.

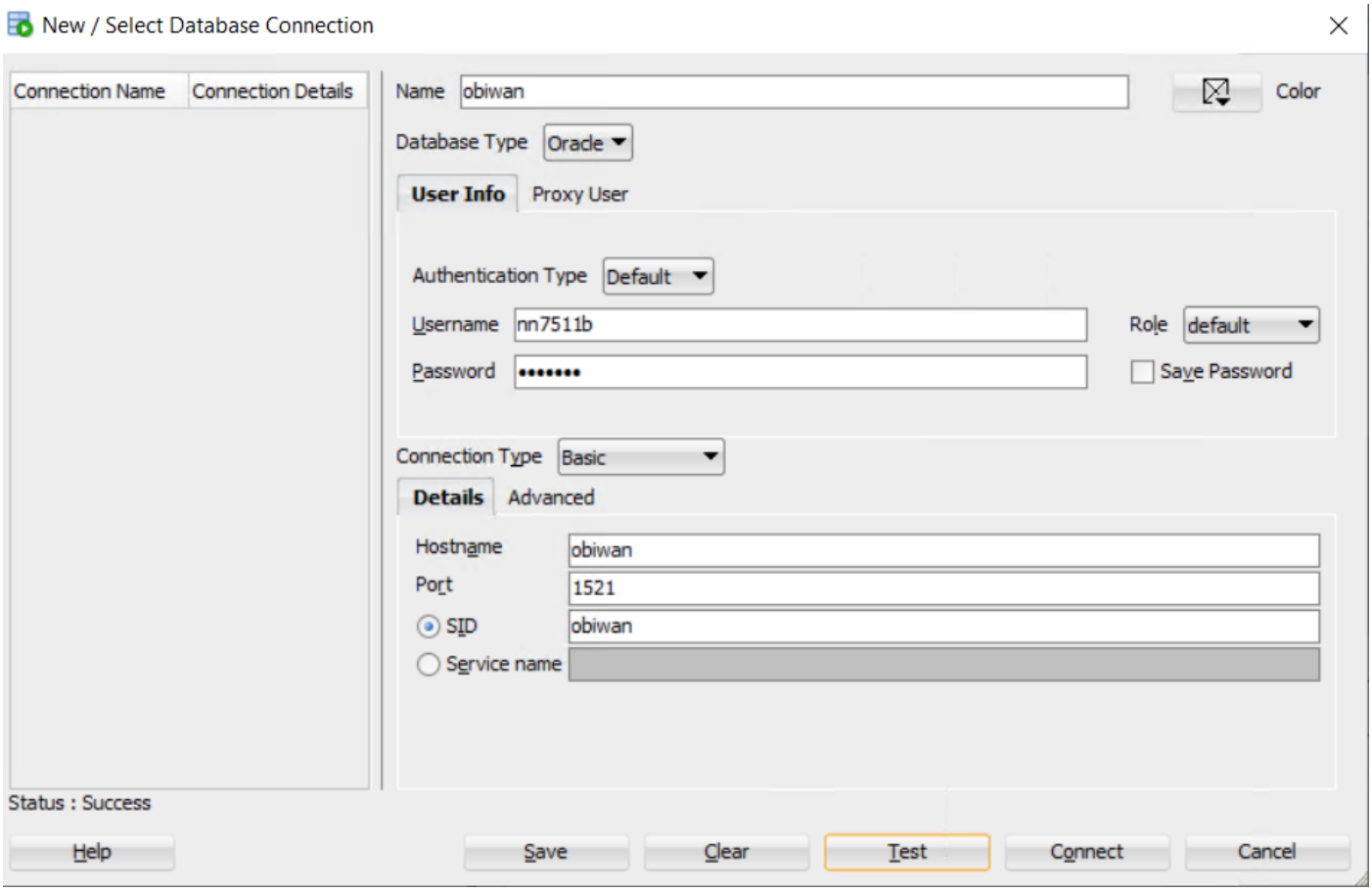
In response to the challenges posed by traditional monitoring methods and the growing need for timely and comprehensive water quality information, our coursework focuses on developing a data warehouse that can address specific queries related to water sensor measurements, location-based statistics, and yearly trends in water parameters.

This report will detail the step-by-step process of designing the data warehouse, exporting data from Microsoft Access to Oracle, implementing ETL processes, cleansing the dataset, and building the necessary tables and relationships. Additionally, we will explore the integration between Oracle and Python for advanced data analysis. The aim is to provide a robust solution that not only meets the specific requirements outlined but also serves as a foundation for scalable and adaptable water quality monitoring systems in the future.

Access Oracle servers using SQL Developer

In the context of the coursework, Oracle SQL Developer provides a user-friendly environment for accessing and managing the Oracle Database, facilitating tasks such as designing tables, executing SQL queries, and overseeing the ETL processes integral to building the water quality monitoring data warehouse.

To access Oracle servers using SQL Developer, open the SQL Developer program on your Virtual Desktop, right-click on "Connections," select "New Connection", enter the details (Connection Name: obiwan, Username: your Oracle username, Password: your Oracle password, Hostname: obiwan, Port: 1521, SID: obiwan), and click the 'Test and Connect' button.



Designing Water Quality Data Warehouse

A data warehouse is a large, centralized repository of integrated data collected from various sources within an organization. It is designed to support business intelligence and decision-making processes by providing a unified and historical view of data. The design of a data warehouse involves structuring the data in a way that facilitates efficient querying, reporting, and analysis.

Key Concepts:

ETL (Extract, Transform, Load): The ETL process involves extracting data from source systems, transforming it to fit the data warehouse schema, and loading it into the warehouse. Transformation includes cleaning, aggregating, and structuring the data according to business needs.

Data Warehouse Architecture: Data warehouses often follow one of two architectures: the Inmon or Kimball approach. Inmon's architecture focuses on creating a normalized data warehouse with a clear distinction between operational and informational systems. Kimball's approach, on the other hand, emphasizes a dimensional model (like star or snowflake schema) for simplicity and ease of querying.

Star Schema: A star schema is a type of dimensional model commonly used in data warehousing. It consists of a central fact table connected to multiple dimension tables. The fact table contains measurable data (facts), and dimension tables provide context to those facts. The relationships form a star-like structure, making it easy to understand and query.

Dimensional Modeling: Dimensional modeling is a design technique used in data warehousing to structure data for easy querying and reporting. It involves creating dimensions (descriptive attributes) and facts (measurable data). The star schema is a popular dimensional modeling technique.

Steps in Data Warehouse Design:

Requirements Gathering: Understand the business requirements and determine the key performance indicators (KPIs) that the data warehouse should support.

Data Source Identification: Identify and integrate data from relevant sources, such as operational databases, spreadsheets, and external data feeds.

ETL Process Design: Plan the Extract, Transform, and Load processes to bring data into the warehouse. This includes data cleansing, transformation, and loading into the data warehouse.

Data Modeling: Create a logical and physical data model. Dimensional modeling techniques, such as star schemas or snowflake schemas, are often used to organize data for optimal querying.

Database Design: Implement the physical database structure based on the data model. This involves creating tables, indexes, and relationships.

Metadata Management: Establish a robust metadata management system to document the meaning and origin of data elements. This ensures data lineage and provides a reference for users.

Testing: Conduct thorough testing to ensure the accuracy and performance of the data warehouse. This includes validating ETL processes, querying performance, and data consistency.

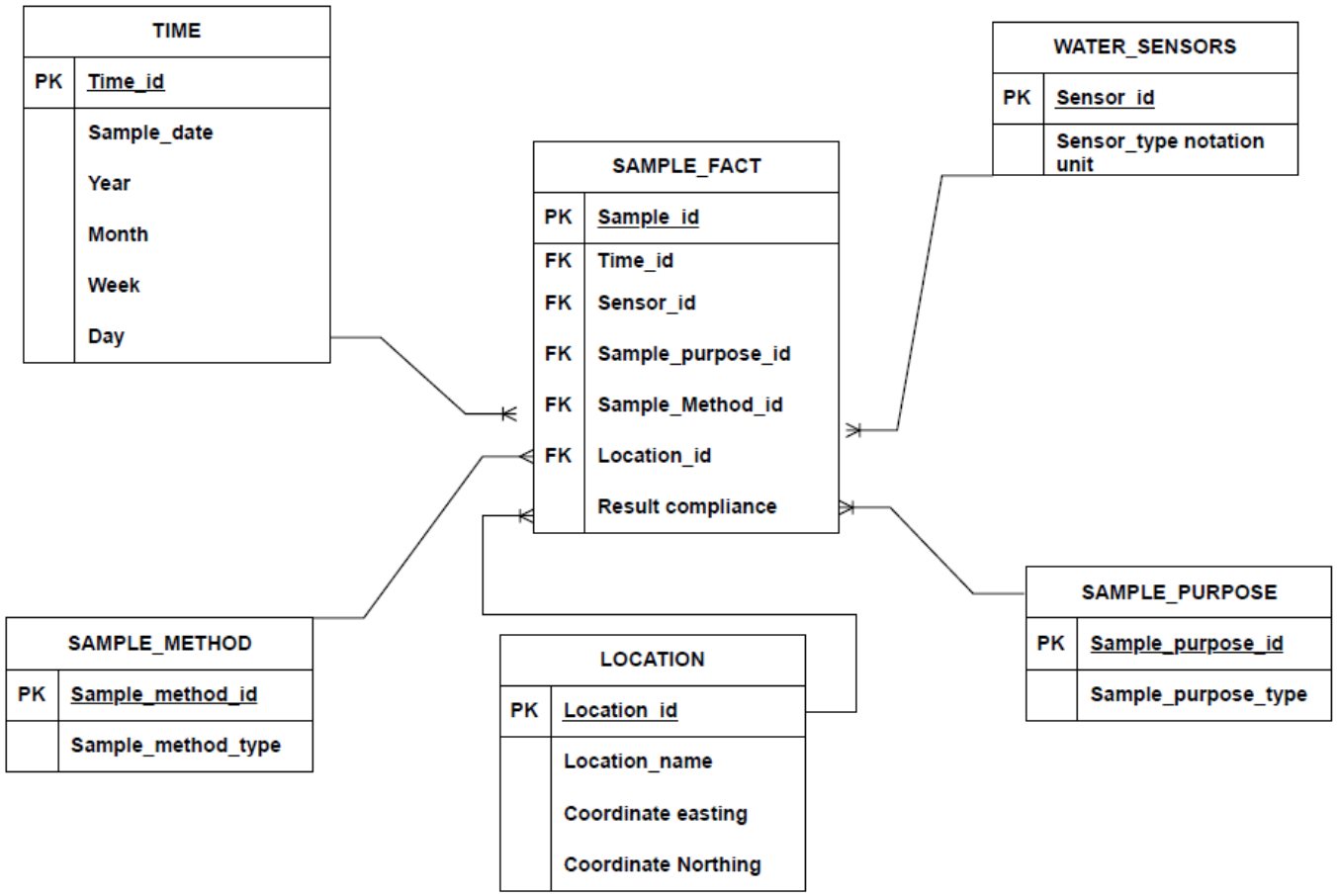
Deployment: Deploy the data warehouse for production use. This involves migrating data from the staging environment to the live environment.

Maintenance and Evolution: Regularly maintain and update the data warehouse to accommodate changing business needs. This may involve modifying the data model, updating ETL processes, and incorporating new data sources.

In summary, data warehouse design is a holistic process that involves understanding business requirements, integrating data from various sources, modeling data for optimal querying, and implementing a robust database structure. Techniques such as star schema and ETL processes play a crucial role in achieving an efficient and effective data warehouse.

Star Schema Design

The star schema is a widely used approach in data warehousing for its simplicity and efficiency in querying data. In the context of water quality monitoring, the star schema is well-suited to capture the essential relationships between business processes and their associated dimensions. Here's a breakdown of the star schema design:



Fact Table: **SAMPLE_FACT**

The central fact table contains key performance indicators related to water quality samples, such as results, compliance which are the relevant identifiers. Foreign keys establish relationships with dimension tables to provide additional context to the measurements.

Dimension Tables:

TIME Dimension:

- Represents the temporal aspect of water quality measurements.
- Enables analysis based on time-related criteria such as month, week, day, and year.

WATER_SENSORS Dimension:

- Contains details about different types of water sensors.
- Facilitates analysis based on the type of sensor used for each measurement.

SAMPLE_METHOD Dimension:

- Captures the methods employed during the sampling process.
- Useful for understanding how samples are collected and processed.

SAMPLE_PURPOSE Dimension:

- Describes the purpose behind each water quality sample.
- Provides insights into the reasons for taking specific measurements.

LOCATION Dimension:

- Represents the geographical locations where samples are taken.
- Supports spatial analysis based on the sampling point's name, coordinates, and other attributes.

Justification:

Performance: The star schema's denormalized structure enhances query performance. It simplifies joins and aggregations, leading to faster query execution.

Simplicity: The simplicity of the star schema makes it easy to understand and maintain. It aligns well with the coursework's scope, allowing for efficient implementation and querying.

Analytical Flexibility: Business analysts can easily navigate and analyze data using the star schema. The clear separation of dimensions and facts facilitates straightforward exploration of water quality measurements from different perspectives.

Scalability: The star schema can easily accommodate additional dimensions or facts, making it scalable for future requirements.

BUS Plan

In the context of a data warehouse design, a BUS matrix or BUS plan is a visual representation that illustrates the alignment of business processes with corresponding dimensions within the data warehouse. It helps to map how different dimensions relate to specific business processes, providing a clear understanding of the data model's structure and how it serves the business needs.

Extract Transform Load

ETL stands for Extract, Transform, Load, and it refers to a process of data integration used in data warehousing and business intelligence. The ETL process is crucial for collecting, cleaning, and consolidating data from various sources into a central data repository, typically a data warehouse or data mart. Let's break down each step of the ETL process:

Extract

The extraction phase involves retrieving data from multiple source systems, which could include databases, spreadsheets, flat files, APIs, or other data repositories.

Purpose: Extracting data allows you to access and gather information from diverse sources, preparing it for further processing.

Transform

The transformation phase involves cleaning, structuring, and enriching the extracted data to make it suitable for analysis and reporting.

Processes:

- Cleaning: Removing or correcting errors, handling missing values, and dealing with inconsistencies.
- Structuring: Reformatting data to fit the desired schema or data model.
- Enriching: Adding derived or calculated fields, aggregating data, or creating new variables.

Purpose: Transformation ensures that the data is consistent, accurate, and aligned with the requirements of the target system.

Load

The loading phase involves storing the transformed data into the target data warehouse or data mart.

Processes:

- Inserts: Adding new records to the data warehouse.
- Updates: Modifying existing records based on changes in the source data.
- Deletes: Removing obsolete or irrelevant records.

Purpose: Loading completes the ETL cycle by populating the central repository with clean, organized data that is ready for analysis.

Key Considerations:

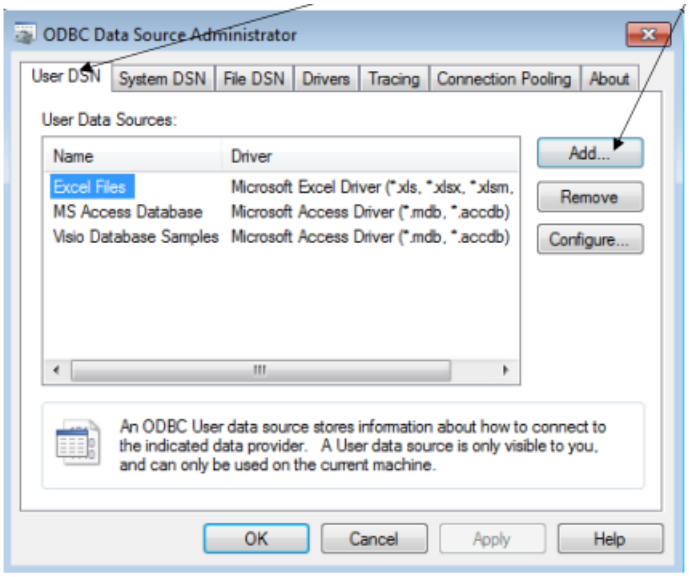
- **Automation**: ETL processes are often automated to ensure efficiency, consistency, and repeatability.
- **Scalability**: ETL systems should be scalable to handle increasing data volumes and evolving business requirements.
- **Data Quality**: Maintaining data quality throughout the ETL process is crucial for producing reliable insights.
- **Metadata Management**: Keeping track of metadata (data about data) helps in understanding the origin, transformation rules, and usage of each data element.

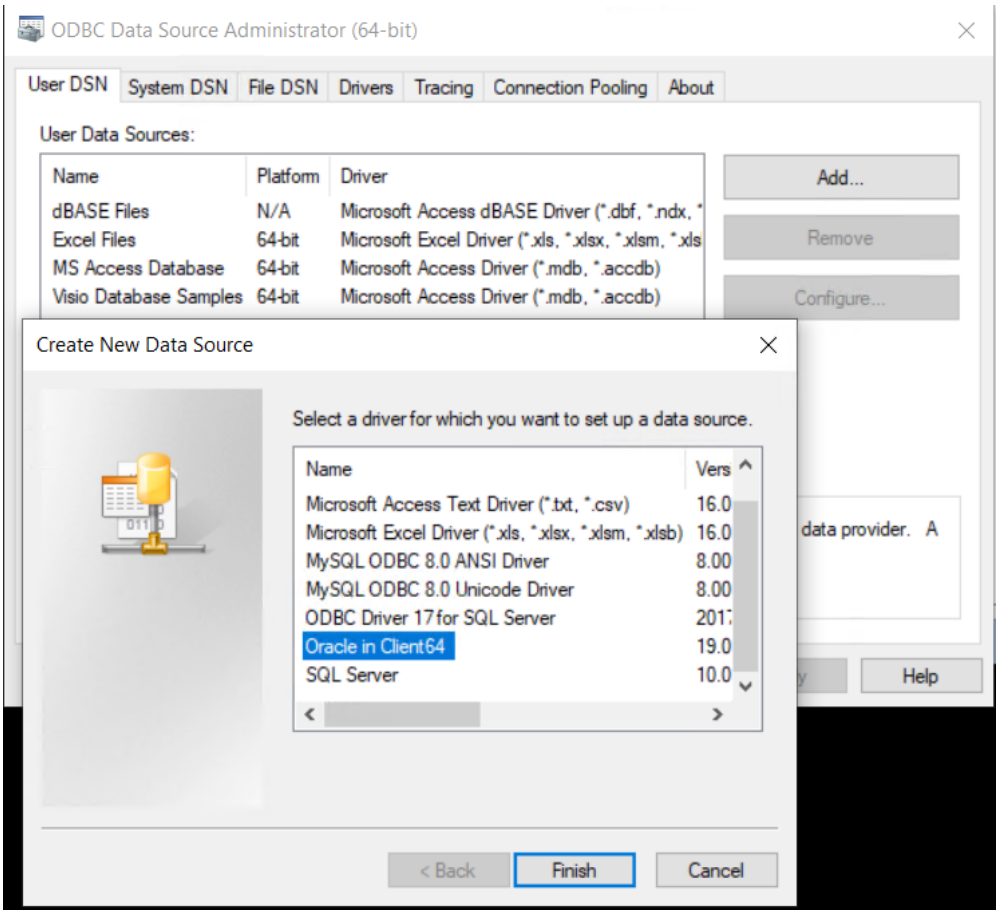
The ETL process plays a vital role in the success of data warehousing and business intelligence initiatives, enabling organizations to derive meaningful insights from their data and support informed decision-making.

Extract Process: Export Data from Microsoft Access to Oracle

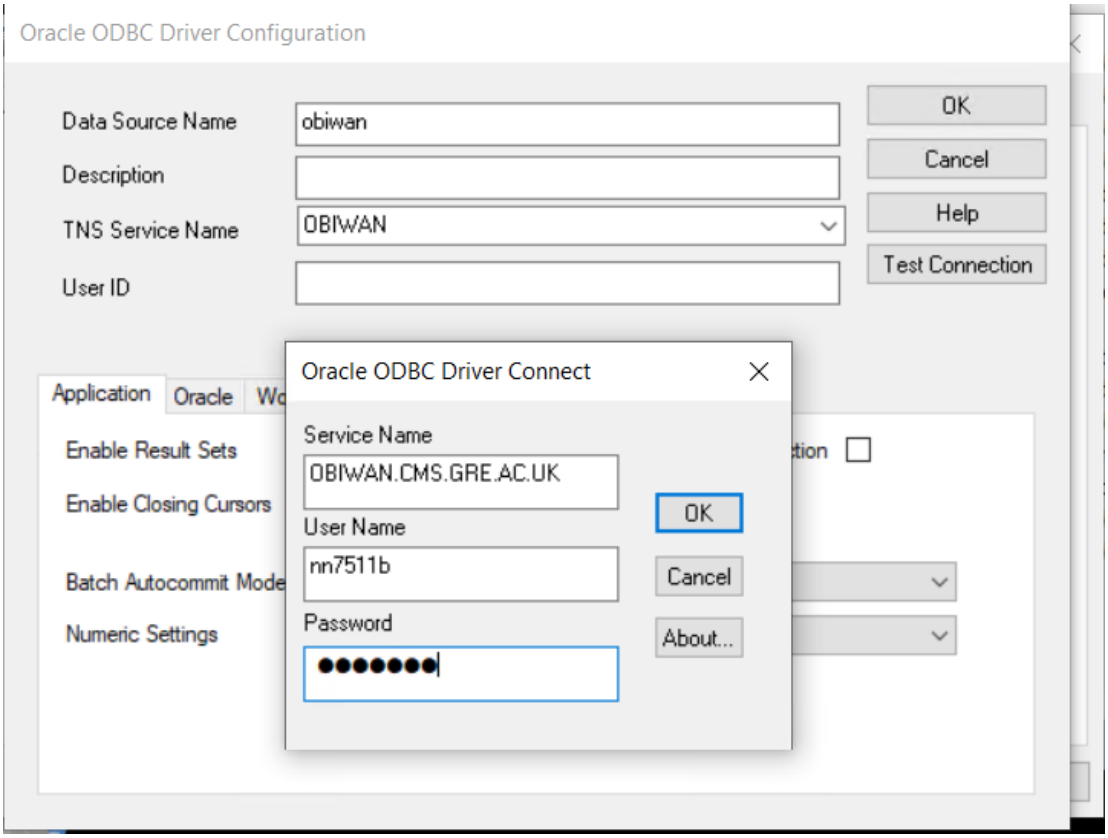
Here we have used ODBC Data Sources 64 bit to export the data from Microsoft access to oracle. For that we have created as new datasouce ‘obiwan’. Its steps are as follows

- Open ODBC Data sources 64 bit -> Go to User DSN in the new window which appears-> Click Add button-> select Oracle in Client64 in a screen which appears -> Click Finish

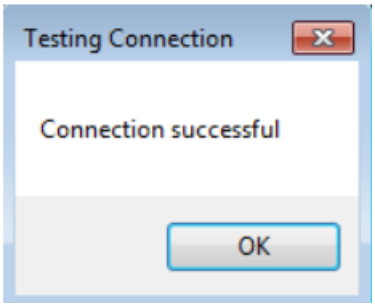




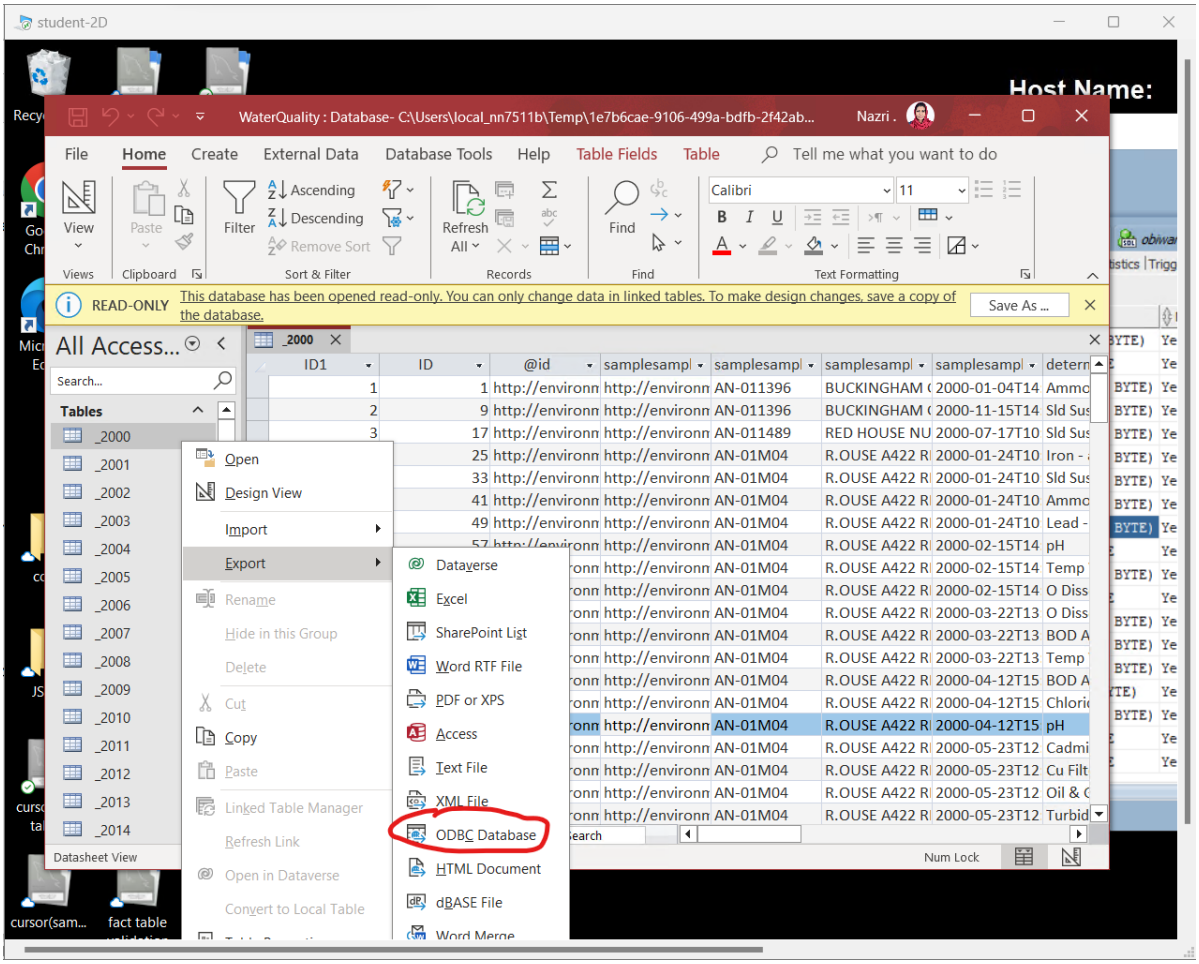
- Enter the Data Source Name ‘Obiwan’ -> TNS Service Name ‘OBIWAN’ -> Enter your own details on new dialog box which appears->click OK button

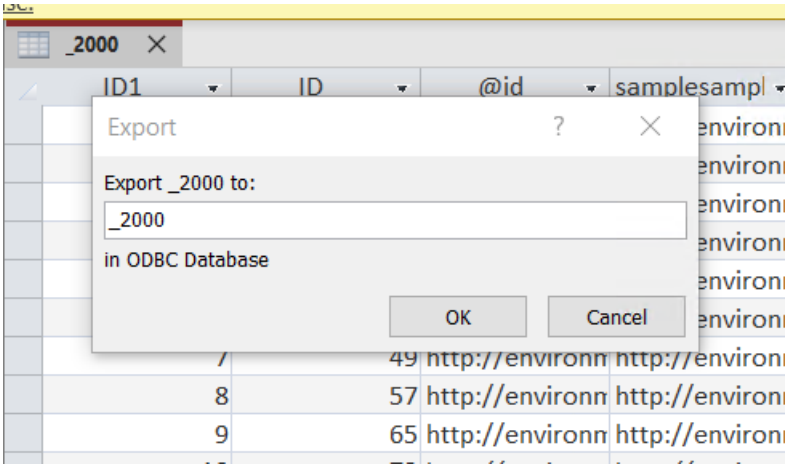


- If the connection is successful shows the below message

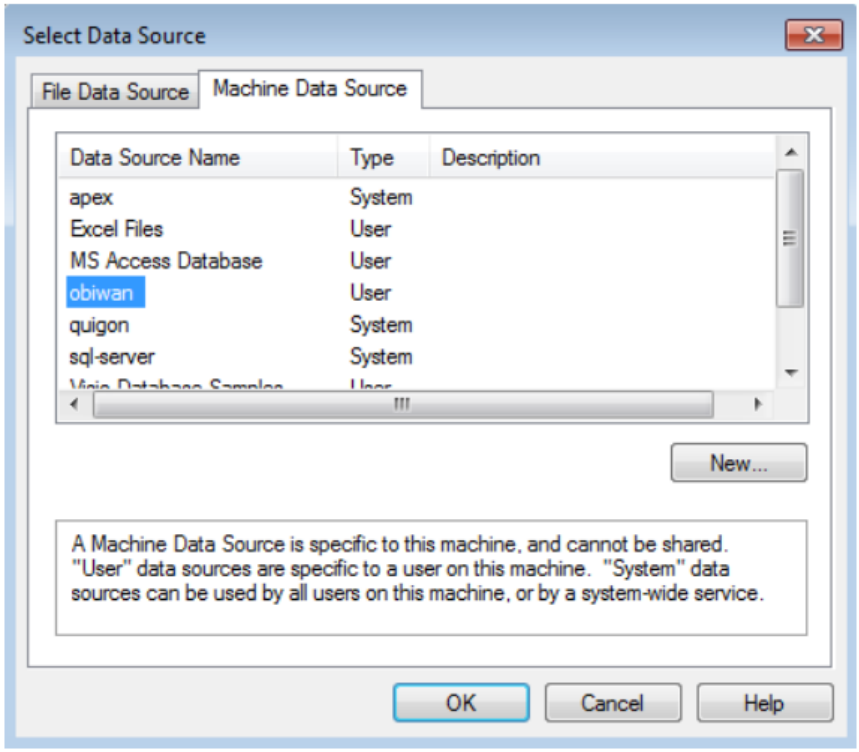


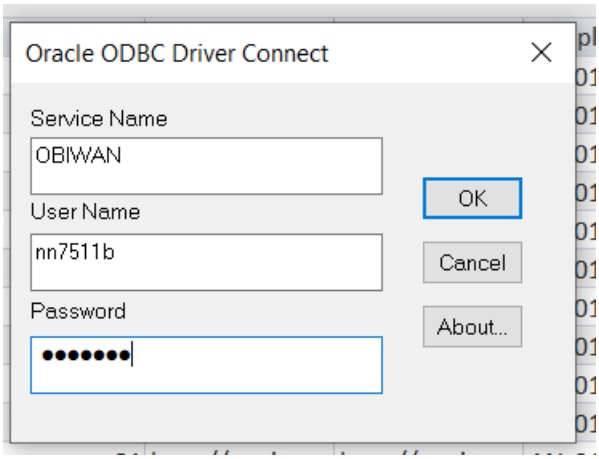
- Now open our Water Quality dataset in MS access-> Right click on the table names -> Select Export -> select ODBC Database-> Click Ok on the new screen



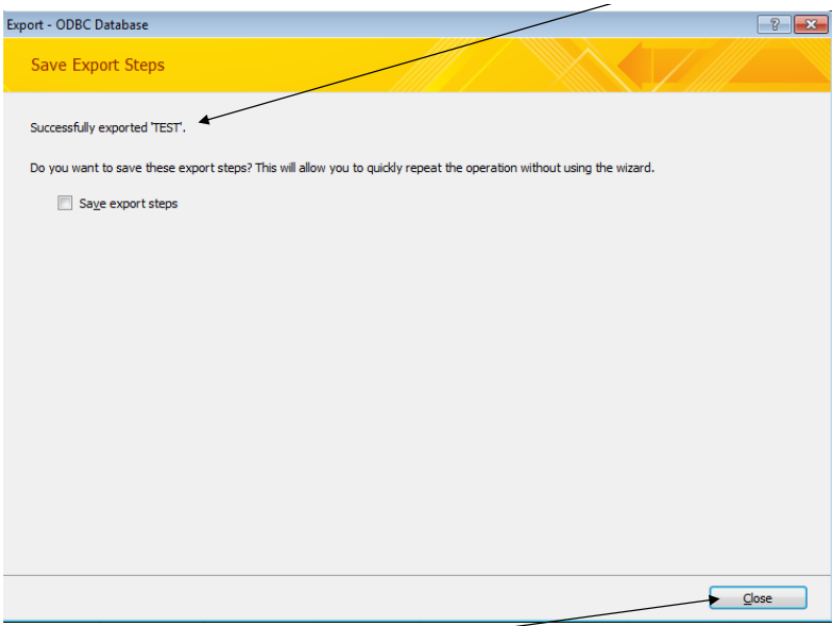


- The following screen should appear, Select Machine Data Source Tab as shown. Choose ‘obiwan’ Data source Name and the type user details and then click the OK button. Now You can upload the dataset successfully.





- Now the following window appears which means that you have successfully exported table the dataset -> Then Click Close



- Upload each dataset (_2000 to _2016) with the above steps.

Staging Area

In the context of data warehousing, a staging area is an interim storage location where raw data is temporarily held and transformed before being loaded into the data warehouse. The staging area plays a crucial role in the Extract, Transform, Load (ETL) process, facilitating the transition from the source system to the data warehouse.

Functions of the Staging Area:

- **Data Extraction:** The staging area serves as the initial landing zone for data extracted from source systems, such as databases, flat files, or external APIs.
- **Raw Data Storage:** Raw data from source systems is stored in its original format in the staging area. This preserves the integrity of the source data for auditing and validation purposes.
- **Data Transformation:** Transformation processes are applied to the raw data in the staging area to clean, enrich, and reformat it as needed for the data warehouse. This may involve tasks such as data cleansing, data validation, and handling data type conversions.
- **Error Handling:** The staging area allows for the identification and handling of data quality issues and errors. Any discrepancies or inconsistencies in the raw data can be addressed before loading into the data warehouse.
- **Data Integration:** Data from different source systems can be integrated and harmonized in the staging area, ensuring that it conforms to the desired structure and standards before being loaded into the data warehouse.
- **Historical Data:** Staging areas often accommodate the storage of historical data, allowing for the tracking of changes over time and supporting the creation of historical snapshots in the data warehouse.

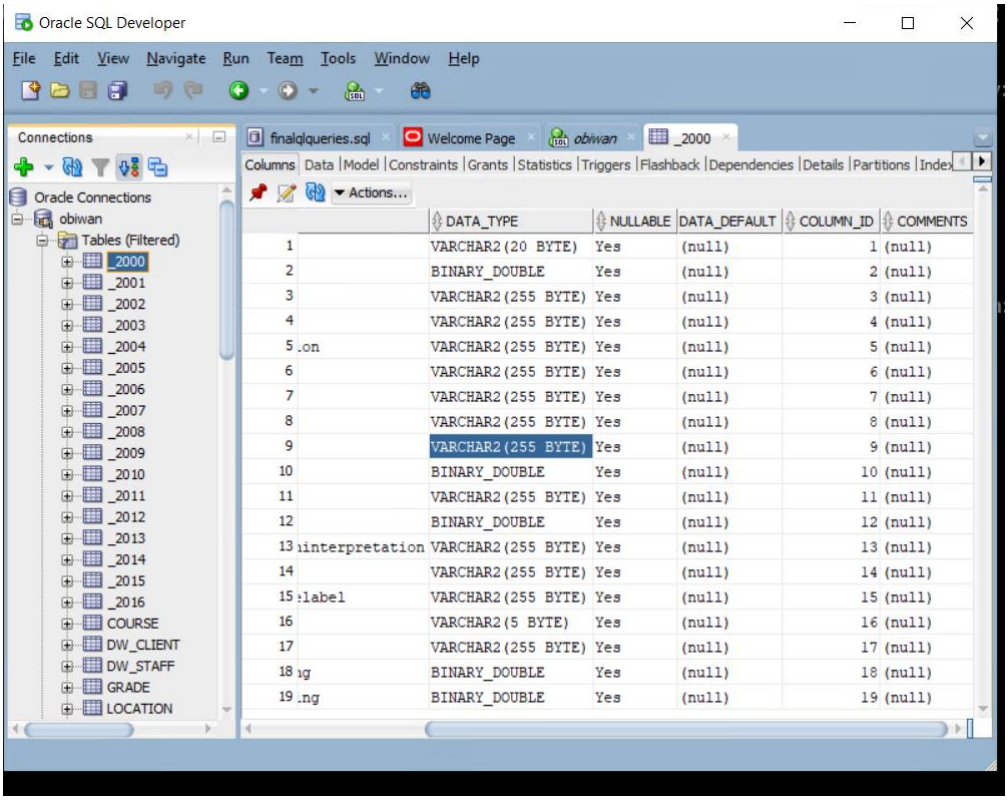
Purposes of the Staging Area:

- **Isolation of ETL Processes**: The staging area provides a buffer between the source systems and the data warehouse, isolating the ETL processes from the operational systems. This minimizes the impact on the source systems during data extraction.
- **Data Validation**: Before data is loaded into the data warehouse, the staging area allows for thorough validation and quality checks. Any anomalies or inconsistencies can be identified and rectified before the data reaches the production environment.
- **Performance Optimization**: Staging areas help optimize performance by enabling parallel processing and reducing the load on the data warehouse during the ETL process.
- **Auditability:** Staging areas contribute to the auditability of the ETL process. The storage of raw and transformed data allows for traceability and accountability in case of issues or discrepancies.

Overall, the staging area acts as a crucial intermediate step in the data integration pipeline, ensuring that only clean, validated, and transformed data is loaded into the data warehouse for analytical purposes.

Steps for Creating Staging Area:

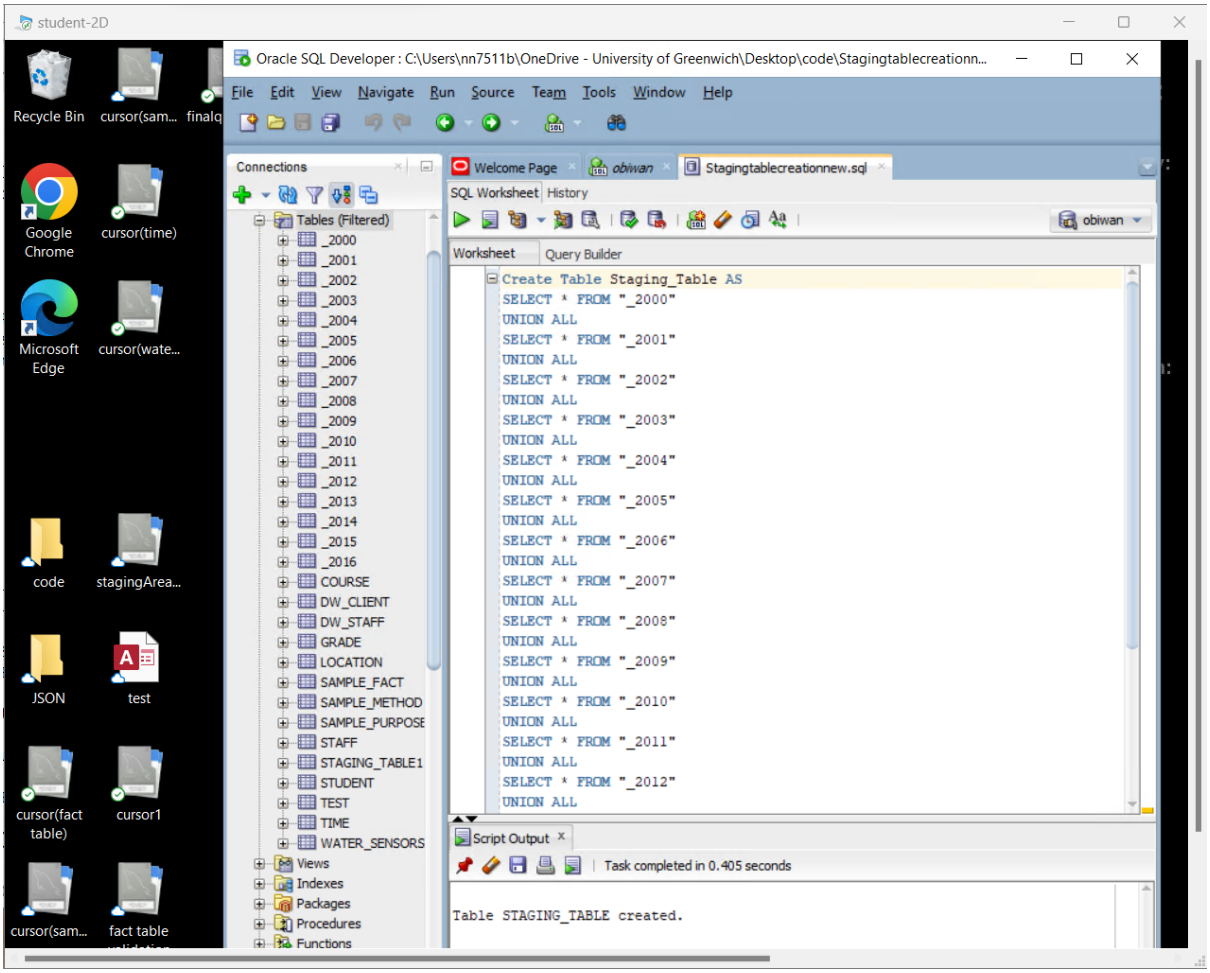
- Now we can see our exported tables in Obiwan connection of SQL Developer.



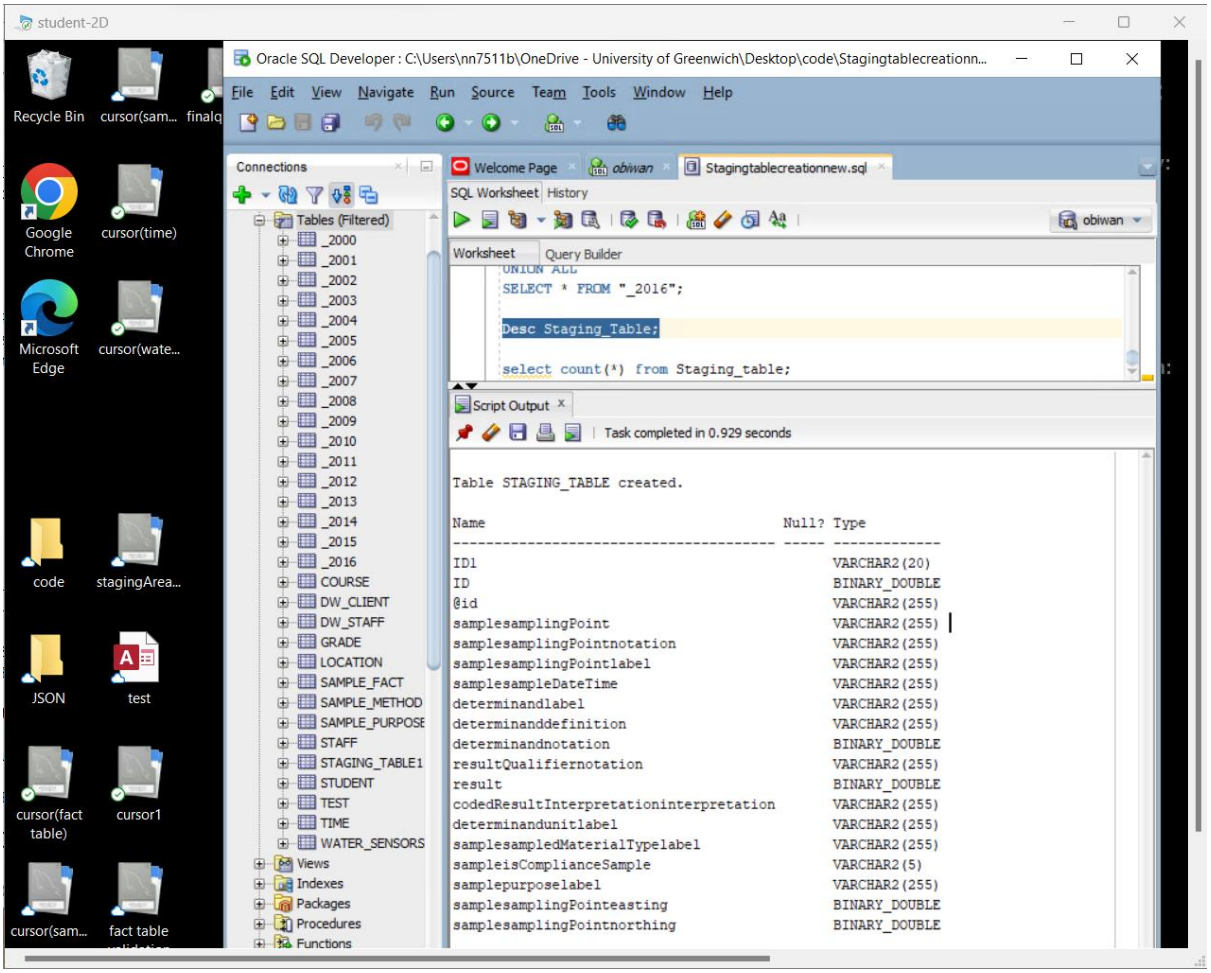
- Create a staging area for initial data storage by combining all the dataset tables. For this purpose we use ‘union all’ function

```
Create Table Staging_Table AS

SELECT * FROM "_2000"
UNION ALL
SELECT * FROM "_2001"
UNION ALL
SELECT * FROM "_2002"
UNION ALL
SELECT * FROM "_2003"
UNION ALL
SELECT * FROM "_2004"
UNION ALL
SELECT * FROM "_2005"
UNION ALL
SELECT * FROM "_2006"
UNION ALL
SELECT * FROM "_2007"
UNION ALL
SELECT * FROM "_2008"
UNION ALL
SELECT * FROM "_2009"
UNION ALL
SELECT * FROM "_2010"
UNION ALL
SELECT * FROM "_2011"
```

```
UNION ALL
SELECT * FROM "_2012"
UNION ALL
SELECT * FROM "_2013"
UNION ALL
SELECT * FROM "_2014"
UNION ALL
SELECT * FROM "_2015"
UNION ALL
SELECT * FROM "_2016";
```

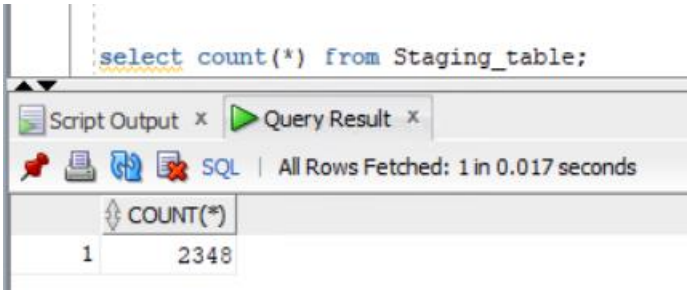


- The structure of the staging_table can be found with the following query: Desc Staging_Table;



There are 2348 datas combined from all the tables in this Staging_table

Select count(*) from Staging_table



Data Cleansing

Data cleansing, also known as data cleaning or data scrubbing, is a crucial step in the data preparation process within a data warehouse environment. It involves identifying and rectifying errors, inconsistencies, and inaccuracies in the raw data to ensure that the data used for analysis and reporting is accurate, reliable, and conforms to the desired standards. Here are key aspects and techniques involved in data cleansing:

Key Aspects of Data Cleansing:

- **Identifying Errors**: Analyse the raw data to identify errors, missing values, outliers, and inconsistencies. Common errors include duplicate records, incorrect data types, and invalid values.
- **Handling Missing Values**: Develop strategies to handle missing or null values. This may involve imputing missing values based on statistical methods, using default values, or removing records with missing values.
- **Standardizing Data:** Ensure consistency in data by standardizing formats, units, and representations. For example, standardize date formats, address formats, and units of measurement.
- **Removing Duplicates**: Identify and eliminate duplicate records to avoid inaccuracies in analytical results. Duplicates can arise due to data entry errors, system glitches, or data integration processes.
- **Correcting Inaccuracies:** Correct inaccuracies in the data by validating against predefined rules or reference data. This may involve cross-referencing data with external sources or business rules.
- **Handling Outliers**: Address outliers that can skew analytical results. Depending on the context, outliers may be corrected, removed, or treated separately in the analysis.
- **Validating Referential Integrity:** Ensure referential integrity by validating foreign key relationships between tables. This involves checking that values in foreign key columns correspond to valid primary key values in related tables.
- **Dealing with Inconsistencies:** Resolve inconsistencies in data definitions, naming conventions, and units of measurement. This is particularly important when integrating data from diverse sources.

Importance of Data Cleansing:

- **Accuracy and Reliability**: Ensures that the data used for analysis is accurate, reliable, and free from errors, leading to trustworthy insights.
- **Consistency:** Promotes consistency in data formats, units, and representations, facilitating standardized reporting and analysis.
- **Improved Decision-Making:** Supports informed decision-making by providing clean and consistent data that reflects the true state of the business.
- **Data Integration:** Facilitates the integration of data from diverse sources, ensuring that data conforms to a common standard.
- **Compliance:** Helps organizations comply with regulatory requirements by maintaining accurate and auditable data.

In summary, data cleansing is an essential step in the data preparation process, contributing to the overall quality and reliability of the data stored in a data warehouse. It ensures that the data accurately represents the real-world entities it is meant to describe, leading to more meaningful and accurate analyses.

Steps for Cleaning

- Duplicate check using the main columns without "samplesampleDateTime"

```
SELECT "samplesamplingPointlabel", "determinanddefinition",  "result", "determinandunitlabel",
"samplesampledMaterialTypelabel", "sampleisComplianceSample",  "samplepurposelabel",
COUNT(*) AS "DuplicateCount" FROM staging_table
GROUP BY "samplesamplingPointlabel", "determinanddefinition",  "result", "determinandunitlabel",
"samplesampledMaterialTypelabel", "sampleisComplianceSample",  "samplepurposelabel"
HAVING COUNT(*) > 1 order by "DuplicateCount" desc;
```

Found some duplicate records

- Duplicate check using the main columns with "samplesampleDateTime"

```
SELECT "samplesamplingPointlabel", "samplesampleDateTime","determinanddefinition",  "result",
"determinandunitlabel",
"samplesampledMaterialTypelabel", "sampleisComplianceSample",  "samplepurposelabel",
COUNT(*) AS "DuplicateCount" FROM staging_table
GROUP BY "samplesamplingPointlabel", "samplesampleDateTime", "determinanddefinition",  "result",
"determinandunitlabel","samplesampledMaterialTypelabel", "sampleisComplianceSample",
"samplepurposelabel" HAVING COUNT(*) > 1
```

Found no duplicate records. This means that for every single row samplesampleDateTime column is different

- Check duplicates in the @id columns

```
SELECT "@id", COUNT(*) AS "DuplicateCount" FROM staging_table GROUP BY "@id" HAVING
COUNT(*) > 1;
```

No duplicate records are found. This means that we can consider this as a primary key to check if there is any missing or null values across the data

- Removing the url part of @id column as this column does not have any duplicates.

```
UPDATE staging_table SET "@id" = SUBSTR("@id", INSTR("@id", '/', -1) + 1);
```

- Removing the columns id, id1 as this column have the duplicates from merging all tables from _2000 to _2016

```
ALTER TABLE staging_table DROP COLUMN ID;
```

```
ALTER TABLE staging_table DROP COLUMN ID1;
```

- check missing values on the @id column

```
select count(*) from staging_table where "@id" is null;
```

No missing/null values are found

- Drop all unwanted columns according to the star schema schema

```
ALTER table "STAGING_TABLE" DROP ("@id" , "samplesamplingPoint", "determinandlabel",
"resultQualifiernotation","codedResultInterpretationinterpretation", "samplesamplingPointnotation");
```

- Check the missing values in each column

```
SELECT
  COUNT(CASE WHEN "samplesamplingPointlabel" IS NULL THEN 1 END) AS
SAMPLING_POINT_LABEL_missing_count,
   COUNT(CASE WHEN "samplesampleDateTime" IS NULL THEN 1 END) AS
sampleDateTime_missing_count,
  COUNT(CASE WHEN "determinanddefinition" IS NULL THEN 1 END) AS
DETERMINAND_DEFINITION_missing_count,
  COUNT(CASE WHEN "determinandnotation" IS NULL THEN 1 END) AS
DETERMINAND_NOTATION_missing_count,
  COUNT(CASE WHEN "result" IS NULL THEN 1 END) AS RESULT_missing_count,
  COUNT(CASE WHEN "determinandunitlabel" IS NULL THEN 1 END) AS
DETERMINAND_UNIT_missing_count,
  COUNT(CASE WHEN "samplesampledMaterialTypelabel" IS NULL THEN 1 END) AS
SAMPLE_MATERIAL_TYPE_missing_count,
  COUNT(CASE WHEN "samplepurposelabel" IS NULL THEN 1 END) AS
SAMPLE_PURPOSE_LABEL_missing_count,
  COUNT(CASE WHEN "sampleisComplianceSample" IS NULL THEN 1 END) AS
sampleisComplianceSample_count,
  COUNT(CASE WHEN "samplesamplingPointeasting" IS NULL THEN 1 END) AS
SAMPLING_POINT_EASTING_missing_count,
  COUNT(CASE WHEN "samplesamplingPointnorthing" IS NULL THEN 1 END) AS
SAMPLING_POINT_NORTHING_missing_count
FROM Staging_table;
```

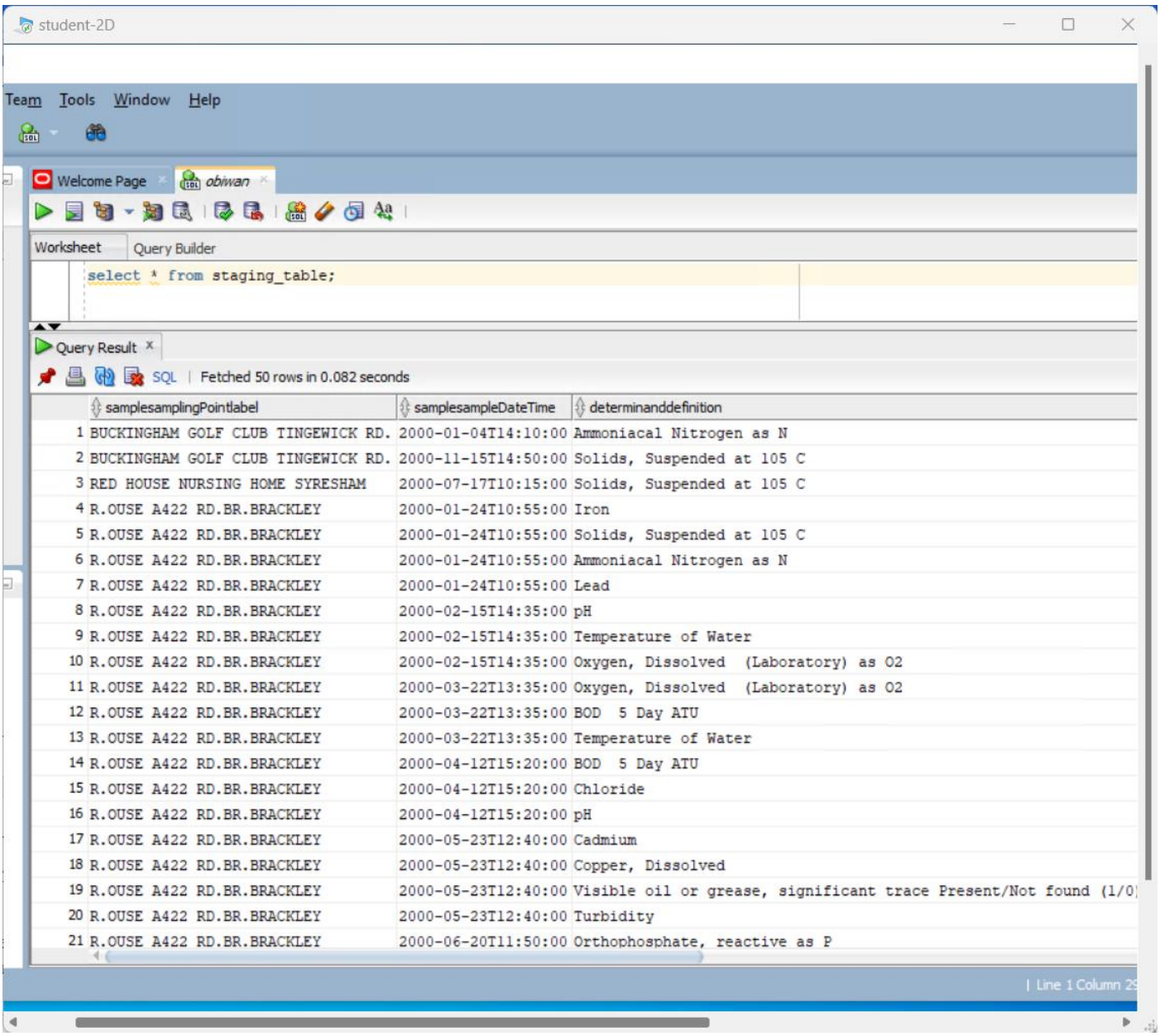
Found no records. That means there is no missing or null values in each column.

- Remove colons (‘:’ , ‘:-‘)from determinanddefinition column

```
UPDATE staging_table
SET "determinanddefinition" = REPLACE("determinanddefinition", ':', '');
```

```
UPDATE staging_table

SET "determinanddefinition" = REPLACE("determinanddefinition", ':-', '');
```

- Display the records in staging table
  select * from staging_table;



	samplesamplingPointlabel	samplesampleDateTime	determinanddefinition
1	BUCKINGHAM GOLF CLUB TINGEWICK RD.	2000-01-04T14:10:00	Ammoniacal Nitrogen as N
2	BUCKINGHAM GOLF CLUB TINGEWICK RD.	2000-11-15T14:50:00	Solids, Suspended at 105 C
3	RED HOUSE NURSING HOME SYRESHAM	2000-07-17T10:15:00	Solids, Suspended at 105 C
4	R.OUSE A422 RD.BR.BRACKLEY	2000-01-24T10:55:00	Iron
5	R.OUSE A422 RD.BR.BRACKLEY	2000-01-24T10:55:00	Solids, Suspended at 105 C
6	R.OUSE A422 RD.BR.BRACKLEY	2000-01-24T10:55:00	Ammoniacal Nitrogen as N
7	R.OUSE A422 RD.BR.BRACKLEY	2000-01-24T10:55:00	Lead
8	R.OUSE A422 RD.BR.BRACKLEY	2000-02-15T14:35:00	pH
9	R.OUSE A422 RD.BR.BRACKLEY	2000-02-15T14:35:00	Temperature of Water
10	R.OUSE A422 RD.BR.BRACKLEY	2000-02-15T14:35:00	Oxygen, Dissolved (Laboratory) as O2
11	R.OUSE A422 RD.BR.BRACKLEY	2000-03-22T13:35:00	Oxygen, Dissolved (Laboratory) as O2
12	R.OUSE A422 RD.BR.BRACKLEY	2000-03-22T13:35:00	BOD 5 Day ATU
13	R.OUSE A422 RD.BR.BRACKLEY	2000-03-22T13:35:00	Temperature of Water
14	R.OUSE A422 RD.BR.BRACKLEY	2000-04-12T15:20:00	BOD 5 Day ATU
15	R.OUSE A422 RD.BR.BRACKLEY	2000-04-12T15:20:00	Chloride
16	R.OUSE A422 RD.BR.BRACKLEY	2000-04-12T15:20:00	pH
17	R.OUSE A422 RD.BR.BRACKLEY	2000-05-23T12:40:00	Cadmium
18	R.OUSE A422 RD.BR.BRACKLEY	2000-05-23T12:40:00	Copper, Dissolved
19	R.OUSE A422 RD.BR.BRACKLEY	2000-05-23T12:40:00	Visible oil or grease, significant trace Present/Not found (1/0)
20	R.OUSE A422 RD.BR.BRACKLEY	2000-05-23T12:40:00	Turbidity
21	R.OUSE A422 RD.BR.BRACKLEY	2000-06-20T11:50:00	Orthophosphate, reactive as P

Building the Warehouse – Loading the Data

Data loading is the process of transferring cleaned and transformed data from the staging area into the data warehouse. This crucial step in the Extract, Transform, Load (ETL) process ensures that the data warehouse is populated with accurate, consistent, and well-organized data that can be used for business intelligence and analytical purposes. Here are key aspects and techniques involved in data loading:

1. Loading Strategies:

- Full Loading: Entire datasets are loaded into the data warehouse. This is suitable for smaller datasets or when the entire dataset needs to be refreshed.
- Incremental Loading: Only new or changed data since the last load is added to the data warehouse. This reduces the load time and resources required.
- Delta Loading: Only the changes (additions, updates, deletions) since the last load are loaded into the data warehouse. This is a subset of incremental loading and is particularly useful for large datasets.

2. Load Techniques:

- Bulk Loading: Involves loading large volumes of data at once, usually through direct interfaces provided by the database management system (DBMS). It is efficient for initial data loads.
- Row-by-Row Loading: Involves inserting data row by row. This method may be used for small datasets or when dealing with real-time data.
- Parallel Loading: Involves loading data into the data warehouse in parallel, utilizing multiple processes or threads. This improves loading performance, especially for large datasets.

3. Data Loading Tools:

- ETL Tools: Specialized ETL tools like Informatica, Talend, and Microsoft SSIS are commonly used for managing the extraction, transformation, and loading processes. These tools provide visual interfaces for designing and orchestrating complex data workflows.
- Database Load Utilities: Database management systems often provide native utilities for bulk data loading. For example, Oracle has SQL*Loader, and SQL Server has BULK INSERT.
- Custom Scripts: Depending on the complexity and specific requirements, custom scripts written in SQL, Python, or other programming languages may be used for data loading.

4. Loading into Dimensions and Facts:

- Dimension Tables: Data from dimension tables, which contain descriptive attributes, is loaded first. This ensures that foreign key relationships are established for fact tables.
- Fact Tables: Once dimension tables are loaded, data is loaded into fact tables, which contain quantitative metrics. Fact tables often involve more complex loading processes, especially if surrogate keys are used.

5. Error Handling and Logging:

- Implement mechanisms for error handling during the loading process. Record information about errors, and log them for review and troubleshooting.

6. Data Validation:

- After loading data, perform validation checks to ensure that the data in the warehouse aligns with the expected results. This involves comparing counts, aggregates, and key relationships.

7. Indexes and Constraints:

- Consider disabling or dropping indexes and constraints during the data loading process to improve performance. These can be re-enabled or recreated after the load is complete.

8. Monitoring and Optimization:

- Monitor the data loading process for performance and efficiency. Optimize queries, indexing, and other factors to enhance loading speed and resource utilization.

9. Load Scheduling:

- Implement scheduling mechanisms to automate and schedule data loading jobs based on business requirements. This ensures timely updates of the data warehouse.

10. Historical Loading:

- If historical data is part of the data warehouse design, implement procedures to load and manage historical records, ensuring a comprehensive view of data changes over time.

Importance of Data Loading:

- Timeliness: Enables the data warehouse to be updated with the latest information for real-time or near-real-time reporting.
- Consistency: Ensures that data loaded into the warehouse is consistent and conforms to predefined standards.
- Performance: Optimizes loading processes for efficient use of resources, minimizing the impact on operational systems.
- Availability: Makes data available for analytical purposes, supporting business intelligence and decision-making.
- Accuracy: Ensures that accurate and validated data is stored in the data warehouse, providing a reliable basis for analysis.
- Scalability: Accommodates the growth of data over time by providing strategies for incremental or delta loading.
- Auditability: Supports auditing and tracking of changes, ensuring accountability and traceability of data transformations.

In summary, effective data loading is essential for maintaining the integrity and reliability of a data warehouse. It involves careful planning, consideration of loading strategies, use of appropriate tools, and ongoing monitoring and optimization efforts.

Steps of Data Loading

- Creation of Fact and Dimension Tables

TIME Dimension Table

```
CREATE TABLE TIME (
   time_id NUMBER NOT NULL,
   sample_date DATE NOT NULL,  --@sampleSampleDateTime
   year NUMBER NOT NULL,
   month NUMBER(2) NOT NULL,
   week NUMBER(2) NOT NULL,
   day NUMBER(2) NOT NULL,
   CONSTRAINT PK_TIME PRIMARY KEY (time_id)
);
```

Water_sensors Dimension Table

```
CREATE TABLE WATER_SENSORS (
   sensor_id NUMBER NOT NULL,
   sensor_type VARCHAR(100) NOT NULL,   --@determinandDefinition
   notation NUMBER(10) NOT NULL,      --@determinandnotation
   unit VARCHAR(10) NOT NULL,         --@determinantUnitLabel
   CONSTRAINT PK_WATER_SENSORS PRIMARY KEY (sensor_id)
);
```

Sample_Method Dimension Table

```
   create table SAMPLE_METHOD (
   sample_method_id NUMBER NOT NULL,
   sample_method_type VARCHAR(50) NOT NULL,   -- @samplesampleMaterialTypeLabel
   CONSTRAINT PK_SAMPLE_METHOD PRIMARY KEY (sample_method_id)
);
```

Sample_Purpose Dimension Table

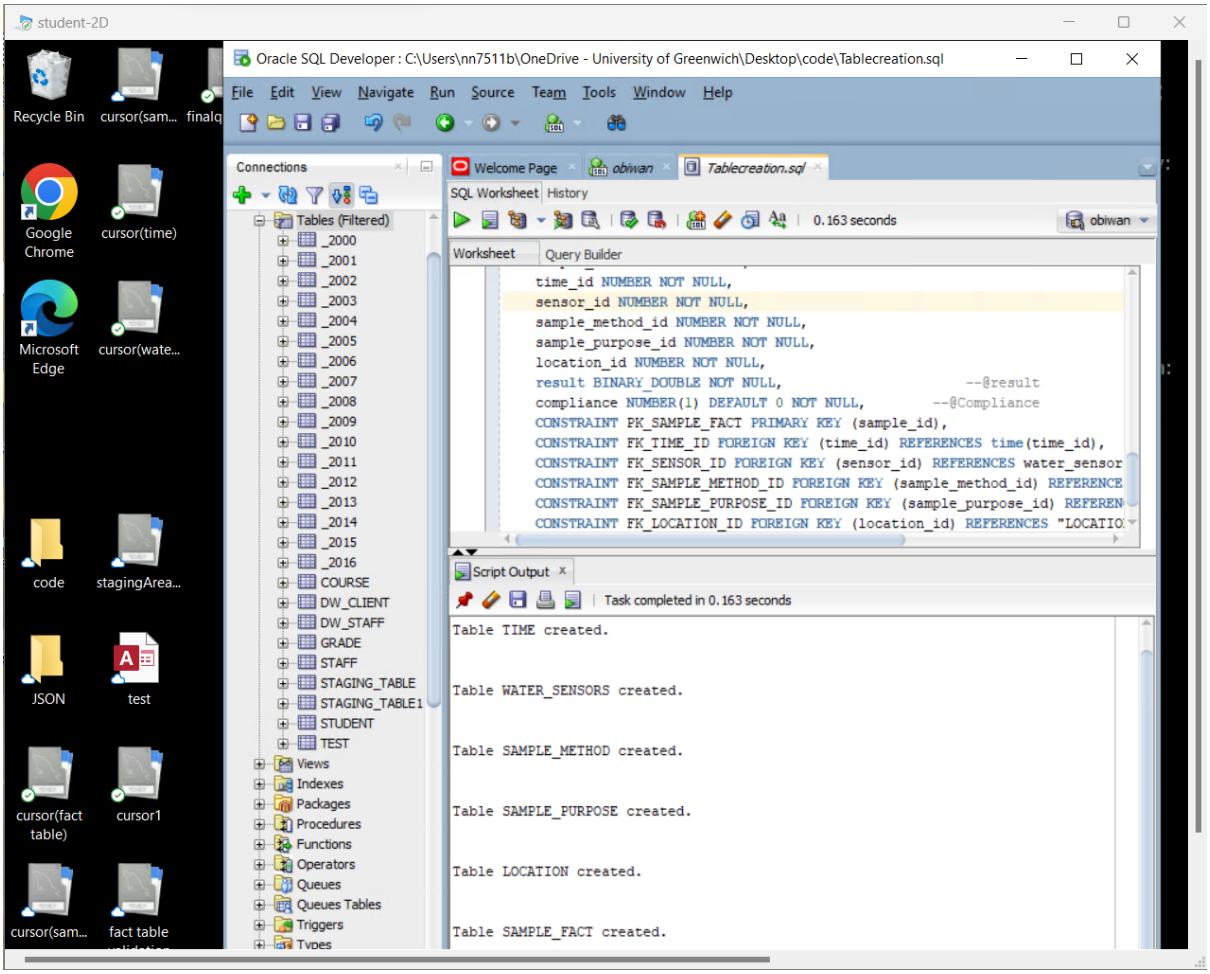
```
create table SAMPLE_PURPOSE (
   sample_purpose_id NUMBER NOT NULL,
   sample_purpose_type VARCHAR(100) NOT NULL,   -- @samplePurposeLabel
   CONSTRAINT PK_SAMPLE_PURPOSE PRIMARY KEY (sample_purpose_id)
);
```

Location Dimension Table

```
create table LOCATION (
   location_id Number NOT NULL,
   location_name VARCHAR(100) NOT NULL, --@sampleSamplingPointLabel
   coordinate_easting NUMBER NOT NULL, --@samplesampleEasting
   coordinate_northing NUMBER NOT NULL,--@samplesampleNorthings
   CONSTRAINT PK_LOCATION PRIMARY KEY (location_id)
);
```

Sample_Fact Fact Table

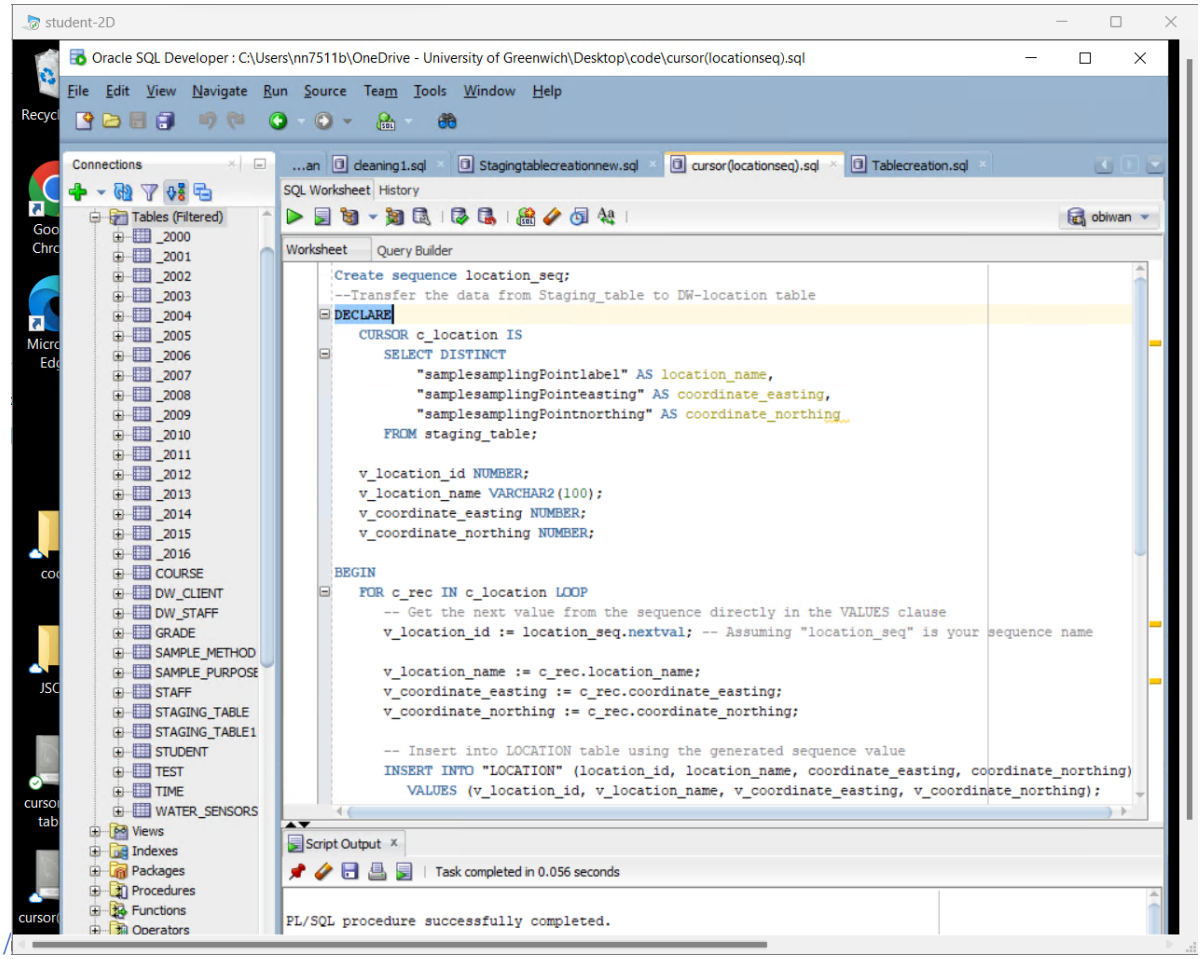
```
create table SAMPLE_FACT (
 sample_id NUMBER NOT NULL,
 time_id NUMBER NOT NULL,
 sensor_id NUMBER NOT NULL,
 sample_method_id NUMBER NOT NULL,
 sample_purpose_id NUMBER NOT NULL,
 location_id NUMBER NOT NULL,
 result BINARY_DOUBLE NOT NULL,                    --@result
 compliance NUMBER(1) DEFAULT 0 NOT NULL,          --@Compliance
 CONSTRAINT PK_SAMPLE_FACT PRIMARY KEY (sample_id),
 CONSTRAINT FK_TIME_ID FOREIGN KEY (time_id) REFERENCES time(time_id),
 CONSTRAINT FK_SENSOR_ID FOREIGN KEY (sensor_id) REFERENCES water_sensors(sensor_id),
 CONSTRAINT FK_SAMPLE_METHOD_ID FOREIGN KEY (sample_method_id) REFERENCES
sample_method(sample_method_id),
 CONSTRAINT FK_SAMPLE_PURPOSE_ID FOREIGN KEY (sample_purpose_id) REFERENCES sample_purpose
(sample_purpose_id),
 CONSTRAINT FK_LOCATION_ID FOREIGN KEY (location_id) REFERENCES "LOCATION"(location_id),
 CONSTRAINT CHECK_COMPLIANCE CHECK (compliance in (0,1))
);
```



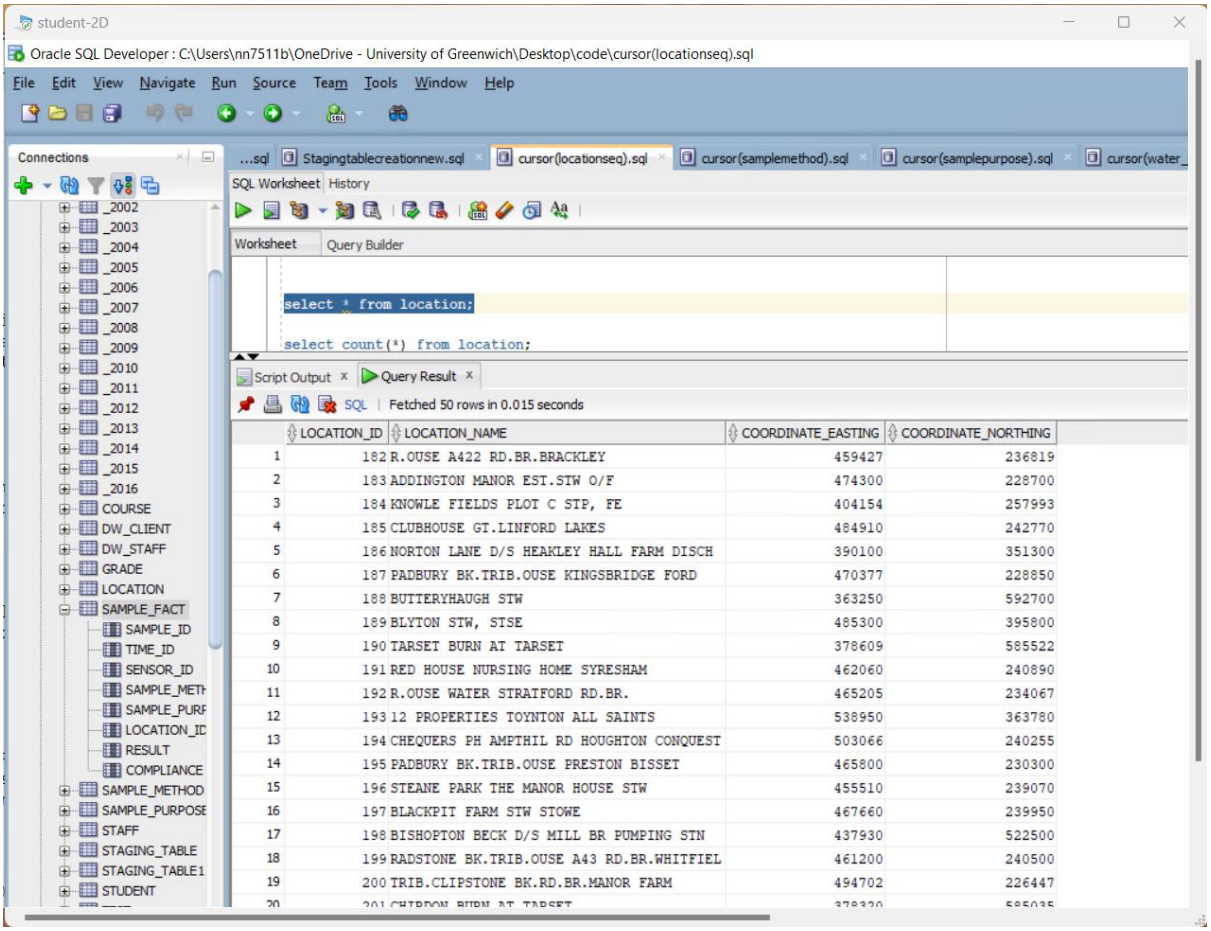
- Load data to the location table using cursor

```
Create sequence location_seq;
--Transfer the data from Staging_table to DW-location table
DECLARE
```

```
  CURSOR c_location IS
    SELECT DISTINCT
       "samplesamplingPointlabel" AS location_name,
       "samplesamplingPointeasting" AS coordinate_easting,
       "samplesamplingPointnorthing" AS coordinate_northing
    FROM staging_table;

  v_location_id NUMBER;
  v_location_name VARCHAR2(100);
  v_coordinate_easting NUMBER;
  v_coordinate_northing NUMBER;

BEGIN
  FOR c_rec IN c_location LOOP
    -- Get the next value from the sequence directly in the VALUES clause
    v_location_id := location_seq.nextval; -- Assuming "location_seq" is your sequence name

    v_location_name := c_rec.location_name;
    v_coordinate_easting := c_rec.coordinate_easting;
    v_coordinate_northing := c_rec.coordinate_northing;

    -- Insert into LOCATION table using the generated sequence value
    INSERT INTO "LOCATION" (location_id, location_name, coordinate_easting,
coordinate_northing)
      VALUES (v_location_id, v_location_name, v_coordinate_easting, v_coordinate_northing);
  END LOOP;
END;
```

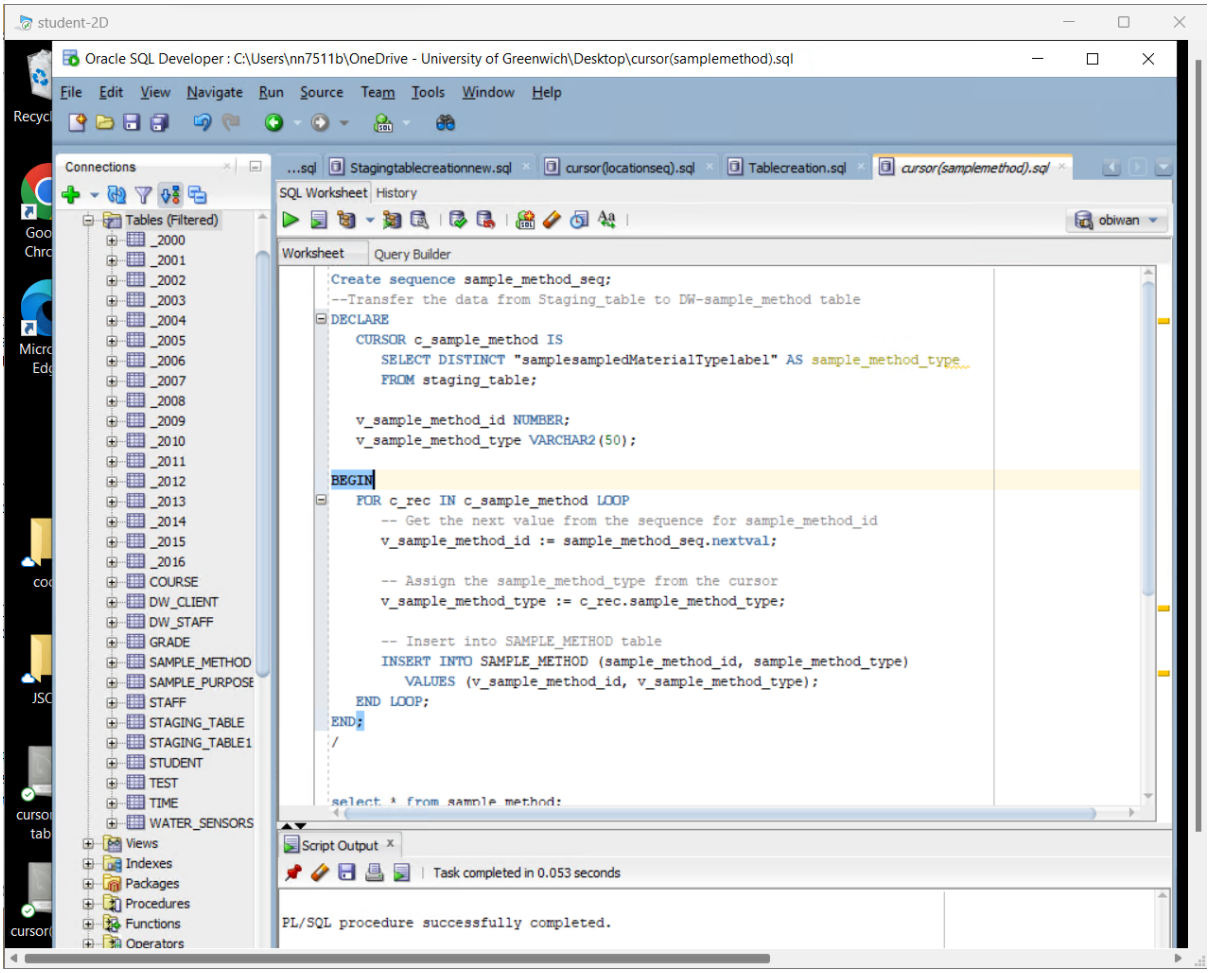


- Verify that data is loaded successfully to the location table
  Select * from location;



- Load data to the sample_method table using cursor

```
Create sequence sample_method_seq;
--Transfer the data from Staging_table to DW-sample_method table
DECLARE
  CURSOR c_sample_method IS
    SELECT DISTINCT "samplesampledMaterialTypelabel" AS sample_method_type
    FROM staging_table;

  v_sample_method_id NUMBER;
  v_sample_method_type VARCHAR2(50);

BEGIN
  FOR c_rec IN c_sample_method LOOP
    -- Get the next value from the sequence for sample_method_id
    v_sample_method_id := sample_method_seq.nextval;

    -- Assign the sample_method_type from the cursor
    v_sample_method_type := c_rec.sample_method_type;

    -- Insert into SAMPLE_METHOD table
    INSERT INTO SAMPLE_METHOD (sample_method_id, sample_method_type)
      VALUES (v_sample_method_id, v_sample_method_type);
  END LOOP;
END;
/
```



- Verify that data is loaded successfully to the sample_method table

```
select * from sample_method;
```

```
select * from sample_method;
```

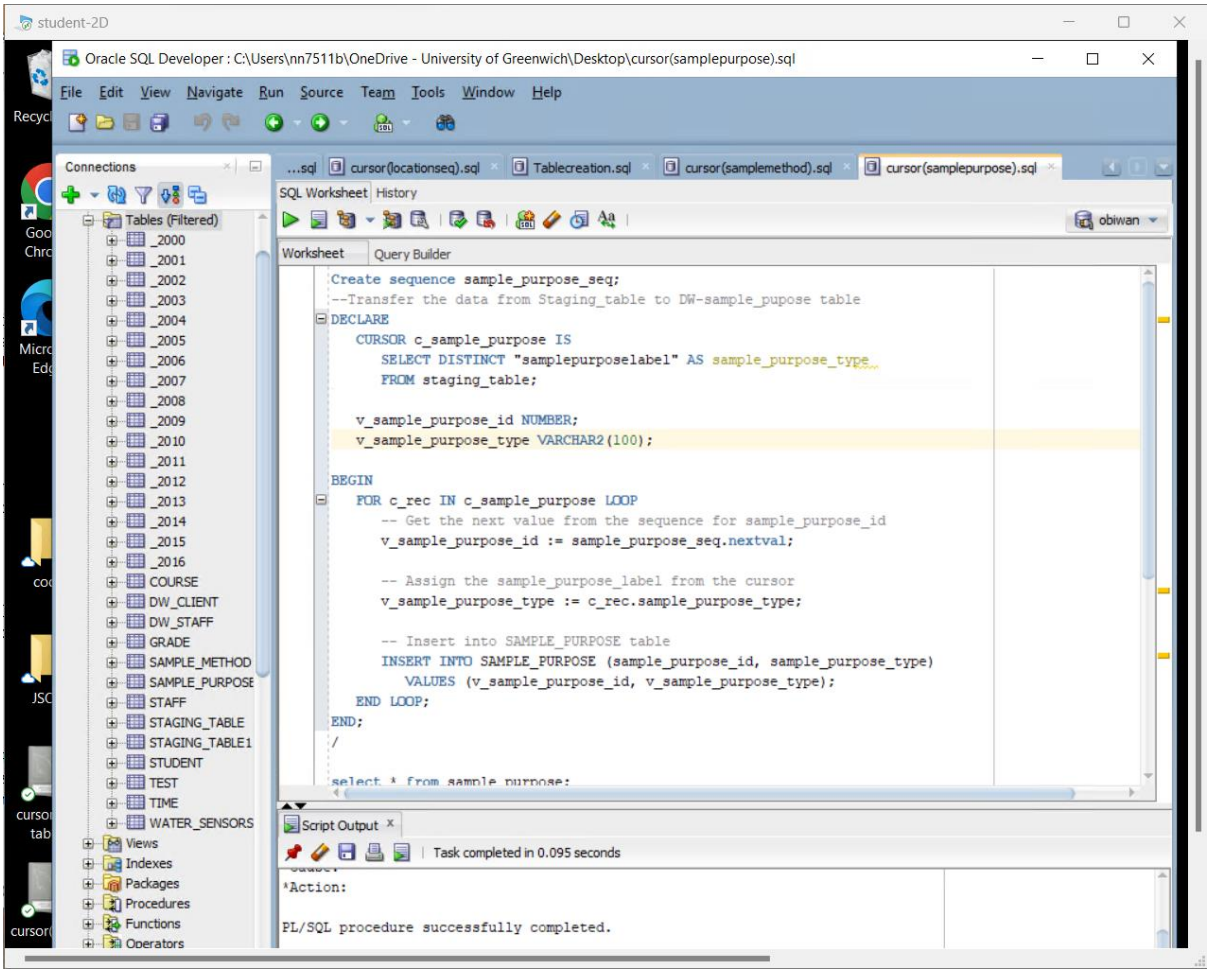
Script Output | Query Result

SQL | All Rows Fetched: 9 in 0.033 seconds

	SAMPLE_METHOD_ID	SAMPLE_METHOD_TYPE
1	21	FINAL SEWAGE EFFLUENT
2	22	ANY SEWAGE
3	23	GROUNDWATER - PURGED/PUMPED/REFILLED
4	24	ANY WATER
5	25	GROUNDWATER
6	26	RIVER / RUNNING SURFACE WATER
7	27	RUNNING SURFACE WATER SEDIMENT - <63UM FRACTION
8	28	UNCODED
9	29	ANY TRADE EFFLUENT

- Load the data to the sample_purpose table

```
Create sequence sample_purpose_seq;
--Transfer the data from Staging_table to DW-sample_pupose table
DECLARE
  CURSOR c_sample_purpose IS
    SELECT DISTINCT "samplepurposelabel" AS sample_purpose_type
    FROM staging_table;

  v_sample_purpose_id NUMBER;
  v_sample_purpose_type VARCHAR2(100);

BEGIN
  FOR c_rec IN c_sample_purpose LOOP
    -- Get the next value from the sequence for sample_purpose_id
    v_sample_purpose_id := sample_purpose_seq.nextval;

    -- Assign the sample_purpose_label from the cursor
    v_sample_purpose_type := c_rec.sample_purpose_type;

    -- Insert into SAMPLE_PURPOSE table
    INSERT INTO SAMPLE_PURPOSE (sample_purpose_id, sample_purpose_type)
      VALUES (v_sample_purpose_id, v_sample_purpose_type);
  END LOOP;
END;
/
```



- Verify that data is loaded successfully to the sample_purpose table
  select * from sample_purpose;

```
select * from sample_purpose;
```

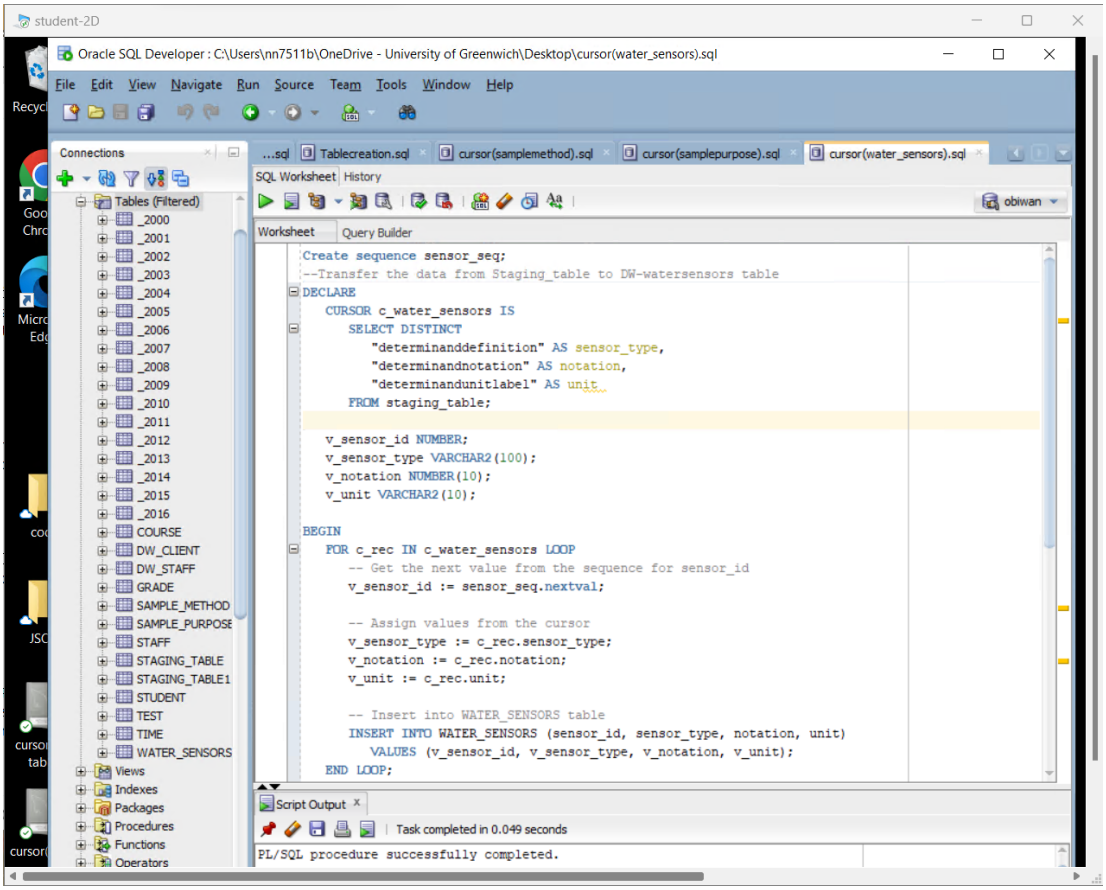
Script Output | Query Result

SQL | All Rows Fetched: 14 in 0.011 seconds

	SAMPLE_PURPOSE_ID	SAMPLE_PURPOSE_TYPE
1	21	UNSPECIFIED AT TIME OF WIMS LOAD
2	22	PLANNED INVESTIGATION (NATIONAL AGENCY POLICY)
3	23	ENVIRONMENTAL MONITORING STATUTORY (EU DIRECTIVES)
4	24	STATUTORY FAILURES (FOLLOW UPS AT NON-DESIGNATED POINTS)
5	25	COMPLIANCE AUDIT (PERMIT)
6	26	MONITORING (UK GOVT POLICY - NOT GQA OR RE)
7	27	MONITORING (NATIONAL AGENCY POLICY)
8	28	PLANNED INVESTIGATION (LOCAL MONITORING)
9	29	UNPLANNED REACTIVE MONITORING FORMAL (POLLUTION INCIDENTS)
10	30	COMPLIANCE FORMAL (PERMIT)
11	31	WATER QUALITY UWWTD MONITORING DATA
12	32	ENVIRONMENTAL MONITORING (GQA & RE ONLY)
13	33	UNPLANNED REACTIVE MONITORING (POLLUTION INCIDENTS)
14	34	WATER QUALITY OPERATOR SELF MONITORING COMPLIANCE DATA

- Load the data to the water_sensors table

```
Create sequence sensor_seq;
--Transfer the data from Staging_table to DW-watersensors table
DECLARE
  CURSOR c_water_sensors IS
    SELECT DISTINCT
      "determinanddefinition" AS sensor_type,
      "determinandnotation" AS notation,
      "determinandunitlabel" AS unit
    FROM staging_table;

  v_sensor_id NUMBER;
  v_sensor_type VARCHAR2(100);
  v_notation NUMBER(10);
  v_unit VARCHAR2(10);

BEGIN
  FOR c_rec IN c_water_sensors LOOP
    -- Get the next value from the sequence for sensor_id
    v_sensor_id := sensor_seq.nextval;

    -- Assign values from the cursor
    v_sensor_type := c_rec.sensor_type;
    v_notation := c_rec.notation;
    v_unit := c_rec.unit;

    -- Insert into WATER_SENSORS table
    INSERT INTO WATER_SENSORS (sensor_id, sensor_type, notation, unit)
      VALUES (v_sensor_id, v_sensor_type, v_notation, v_unit);
  END LOOP;
END;
/
commit;
```



- Verify that data is loaded successfully to the water_sensors table
  select * from water_sensors;

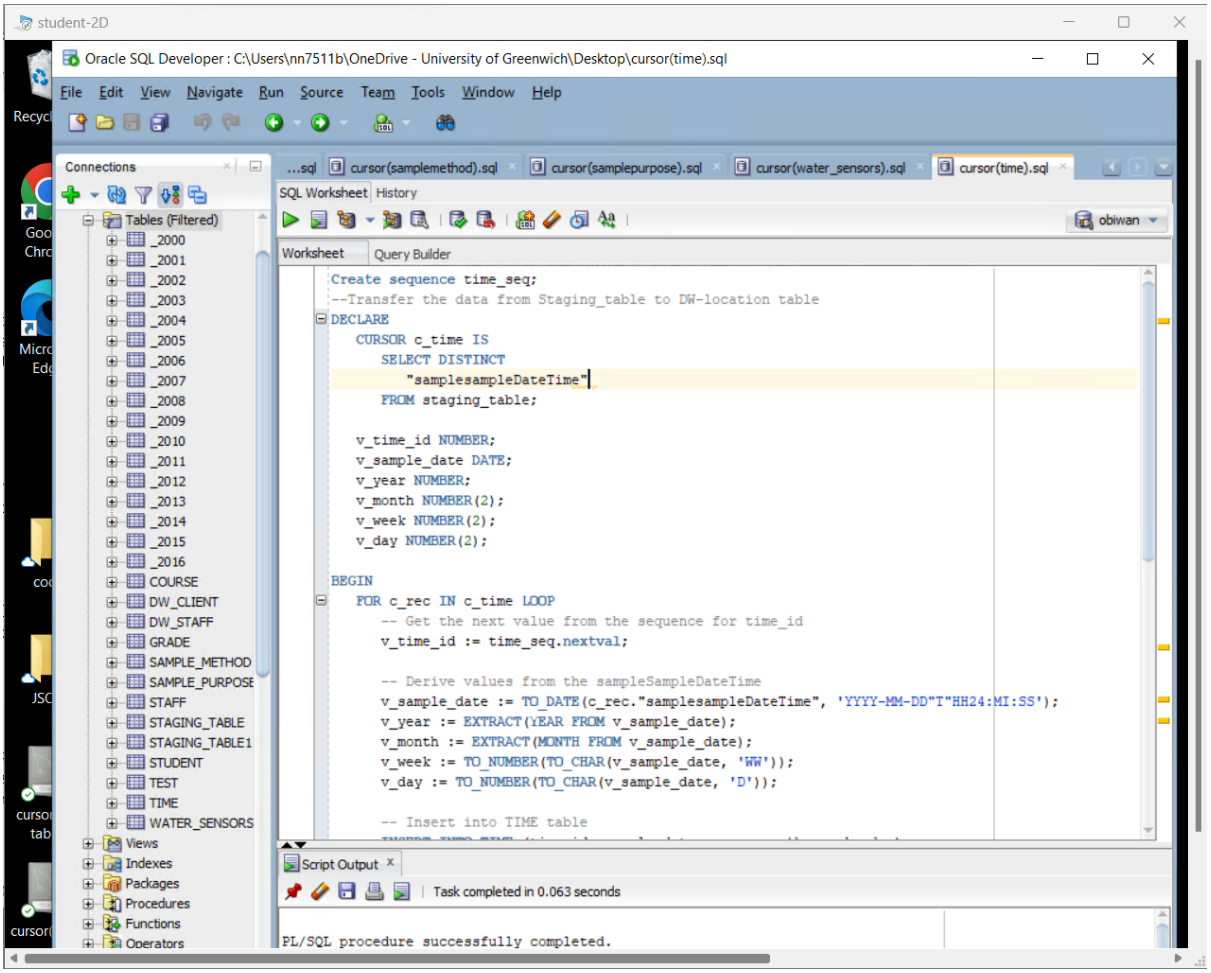
```
select * from water_sensors;
```

Script Output | Query Result | SQL | Fetched 50 rows in 0.012 seconds

	SENSOR_ID	SENSOR_TYPE	NOTATION	UNIT
1	781	Temperature of Water	76	cel
2	782	Nitrite as N	118	mg/l
3	783	Oxygen, Dissolved, % Saturation (Laboratory)	81	%
4	784	Phosphorus, Total as P	348	mg/l
5	785	2-Chlorophenol	9814	ug/l
6	786	Chromium, Dissolved	3409	ug/l
7	787	Bendiocarb	9338	ug/l
8	788	Manganese, Dissolved	6458	ug/l
9	789	Boron	6059	ug/l
10	790	Neburon	9756	ug/l
11	791	cis-Chlordane	577	ug/l
12	792	Beryllium	9696	ug/l
13	793	Tin	6040	ug/l
14	794	Cadmium, Dissolved	106	ug/l
15	795	Chlorotoluron	9348	ug/l
16	796	Phenanthrene	9671	ug/l
17	797	DDT Sum of components	575	ug/l
18	798	ICP-MS Metals screen Semi-quantitative	3401	unitless
19	799	Chromium Hexavalent	4370	ug/l
20	800	Iron	6051	ug/l
21	801	Oxygen, Dissolved (Laboratory) as O2	82	mg/l
22	802	Cadmium	108	ug/l

- Load the data to the Time table

```
Create sequence time_seq;
--Transfer the data from Staging_table to DW-location table
DECLARE
  CURSOR c_time IS
    SELECT DISTINCT
      "samplesampleDateTime"
    FROM staging_table;

  v_time_id NUMBER;
  v_sample_date DATE;
  v_year NUMBER;
  v_month NUMBER(2);
  v_week NUMBER(2);
  v_day NUMBER(2);

BEGIN
  FOR c_rec IN c_time LOOP
    -- Get the next value from the sequence for time_id
    v_time_id := time_seq.nextval;

    -- Derive values from the sampleSampleDateTime
    v_sample_date := TO_DATE(c_rec."samplesampleDateTime", 'YYYY-MM-
DD"T"HH24:MI:SS');
    v_year := EXTRACT(YEAR FROM v_sample_date);
    v_month := EXTRACT(MONTH FROM v_sample_date);
    v_week := TO_NUMBER(TO_CHAR(v_sample_date, 'WW'));
    v_day := TO_NUMBER(TO_CHAR(v_sample_date, 'D'));

    -- Insert into TIME table
    INSERT INTO TIME (time_id, sample_date, year, month, week, day)
      VALUES (v_time_id, v_sample_date, v_year, v_month, v_week, v_day);
  END LOOP;
END;
/

commit;
```



- Verify that data is loaded successfully to the location table
  select * from time;

```
select * from time;
```

Script Output | Query Result

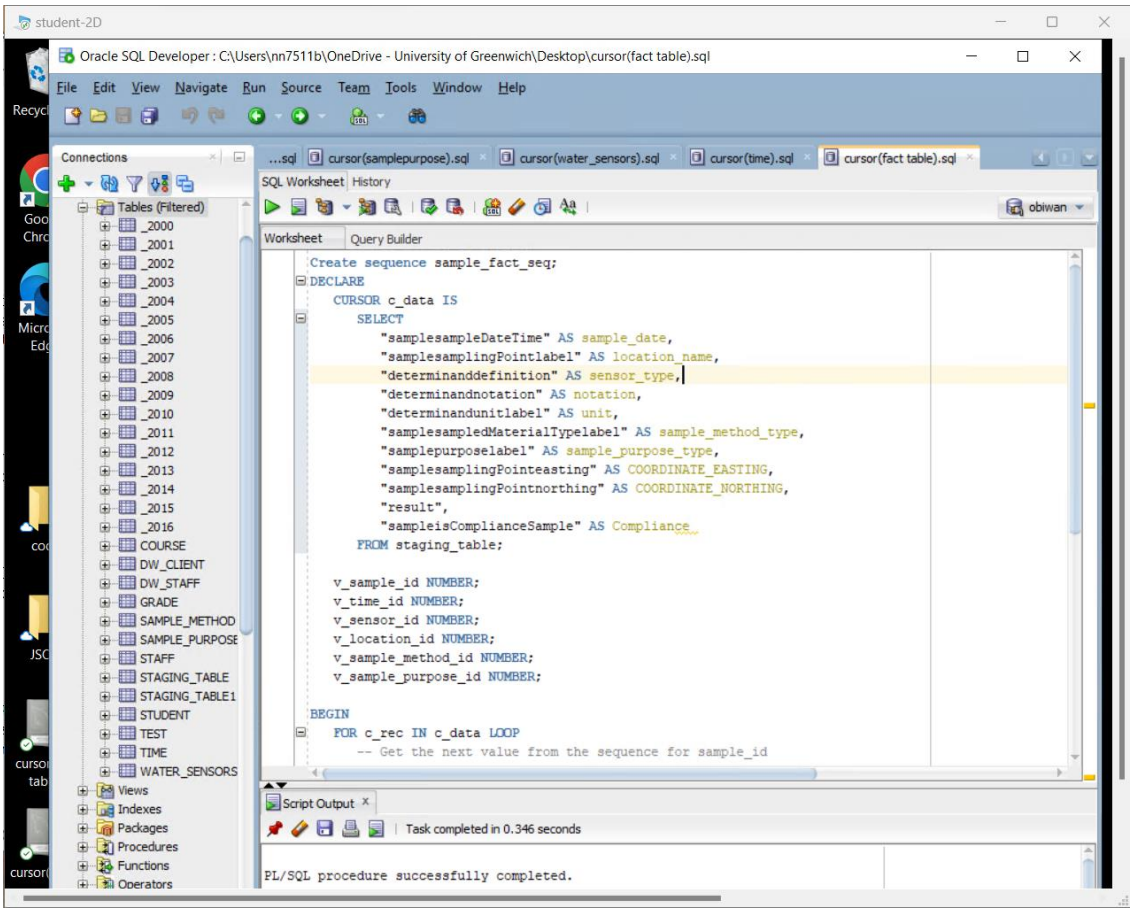
SQL | Fetched 50 rows in 0.013 seconds

	TIME_ID	SAMPLE_DATE	YEAR	MONTH	WEEK	DAY
1	2161	14-DEC-00	2000	12	50	4
2	2162	24-JAN-00	2000	1	4	1
3	2163	22-MAR-00	2000	3	12	3
4	2164	20-JUN-00	2000	6	25	2
5	2165	21-SEP-00	2000	9	38	4
6	2166	24-JAN-00	2000	1	4	1
7	2167	23-MAY-00	2000	5	21	2
8	2168	20-JUN-00	2000	6	25	2
9	2169	11-APR-01	2001	4	15	3
10	2170	28-FEB-01	2001	2	9	3
11	2171	28-JUN-01	2001	6	26	4
12	2172	13-SEP-01	2001	9	37	4
13	2173	28-JUN-01	2001	6	26	4
14	2174	28-AUG-01	2001	8	35	2
15	2175	19-OCT-01	2001	10	42	5
16	2176	22-MAR-01	2001	3	12	4
17	2177	16-MAY-01	2001	5	20	3
18	2178	23-JAN-02	2002	1	4	3
19	2179	22-JUL-02	2002	7	29	1
20	2180	19-AUG-02	2002	8	33	1
21	2181	09-DEC-02	2002	12	49	1

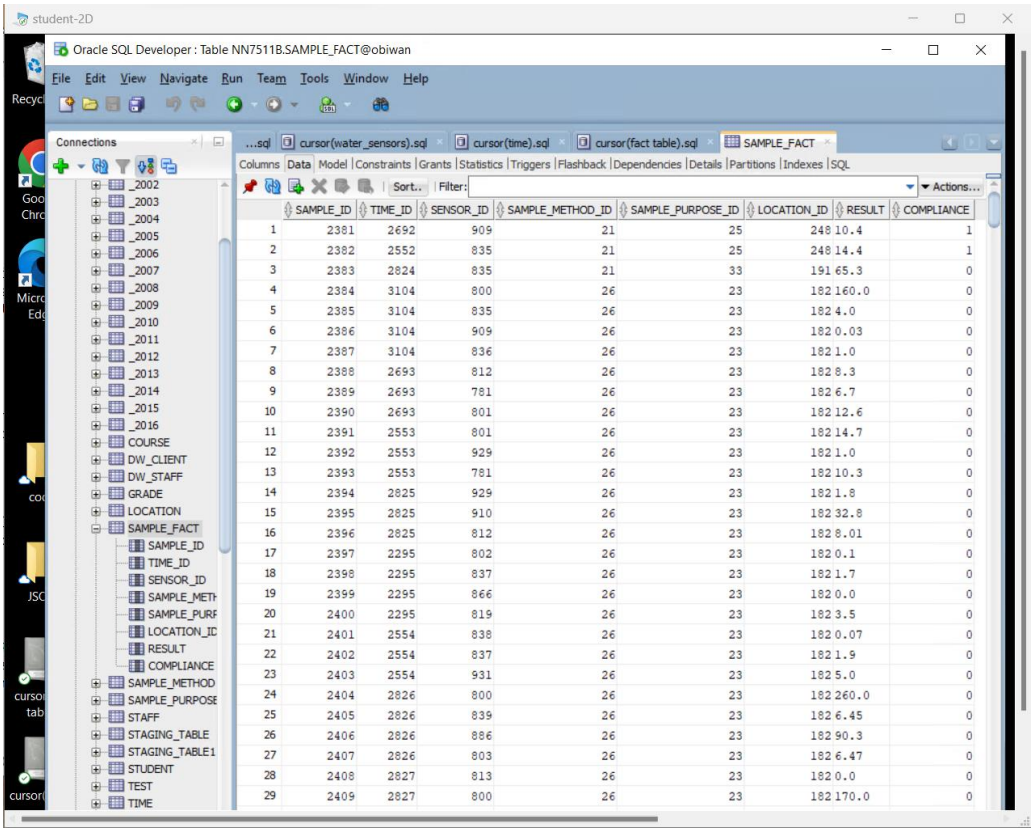
- Load the data to the sample_fact table

```
Create sequence sample_fact_seq;
DECLARE
  CURSOR c_data IS
    SELECT
      "samplesampleDateTime" AS sample_date,
      "samplesamplingPointlabel" AS location_name,
      "determinanddefinition" AS sensor_type,
      "determinandnotation" AS notation,
      "determinandunitlabel" AS unit,
      "samplesampledMaterialTypelabel" AS sample_method_type,
      "samplepurposelabel" AS sample_purpose_type,
      "samplesamplingPointeasting" AS COORDINATE_EASTING,
      "samplesamplingPointnorthing" AS COORDINATE_NORTHING,
      "result",
      "sampleisComplianceSample" AS Compliance
    FROM staging_table;

  v_sample_id NUMBER;
  v_time_id NUMBER;
  v_sensor_id NUMBER;
  v_location_id NUMBER;
  v_sample_method_id NUMBER;
  v_sample_purpose_id NUMBER;

BEGIN
  FOR c_rec IN c_data LOOP
    -- Get the next value from the sequence for sample_id
    v_sample_id := sample_fact_seq.nextval;

    -- Use direct references of foreign key values from dimension tables
    SELECT time_id INTO v_time_id FROM TIME WHERE sample_date =
TO_DATE(c_rec.sample_date, 'YYYY-MM-DD"T"HH24:MI:SS')AND ROWNUM=1;
    SELECT sensor_id INTO v_sensor_id FROM WATER_SENSORS WHERE sensor_type =
c_rec.sensor_type AND ROWNUM=1;
    SELECT location_id INTO v_location_id FROM LOCATION WHERE location_name =
c_rec.location_name AND ROWNUM=1;
    SELECT sample_method_id INTO v_sample_method_id FROM SAMPLE_METHOD WHERE
sample_method_type = c_rec.sample_method_type AND ROWNUM=1;
    SELECT sample_purpose_id INTO v_sample_purpose_id FROM SAMPLE_PURPOSE
WHERE sample_purpose_type = c_rec.sample_purpose_type AND ROWNUM=1;

    -- Insert into SAMPLE_FACT table
    INSERT INTO SAMPLE_FACT (sample_id, time_id, sensor_id, sample_method_id,
sample_purpose_id, location_id, result, compliance)
      VALUES (v_sample_id, v_time_id, v_sensor_id, v_sample_method_id, v_sample_purpose_id,
v_location_id, c_rec."result", c_rec.Compliance);

  END LOOP;

  -- Commit the changes
```

```
    COMMIT;
END;
/
```

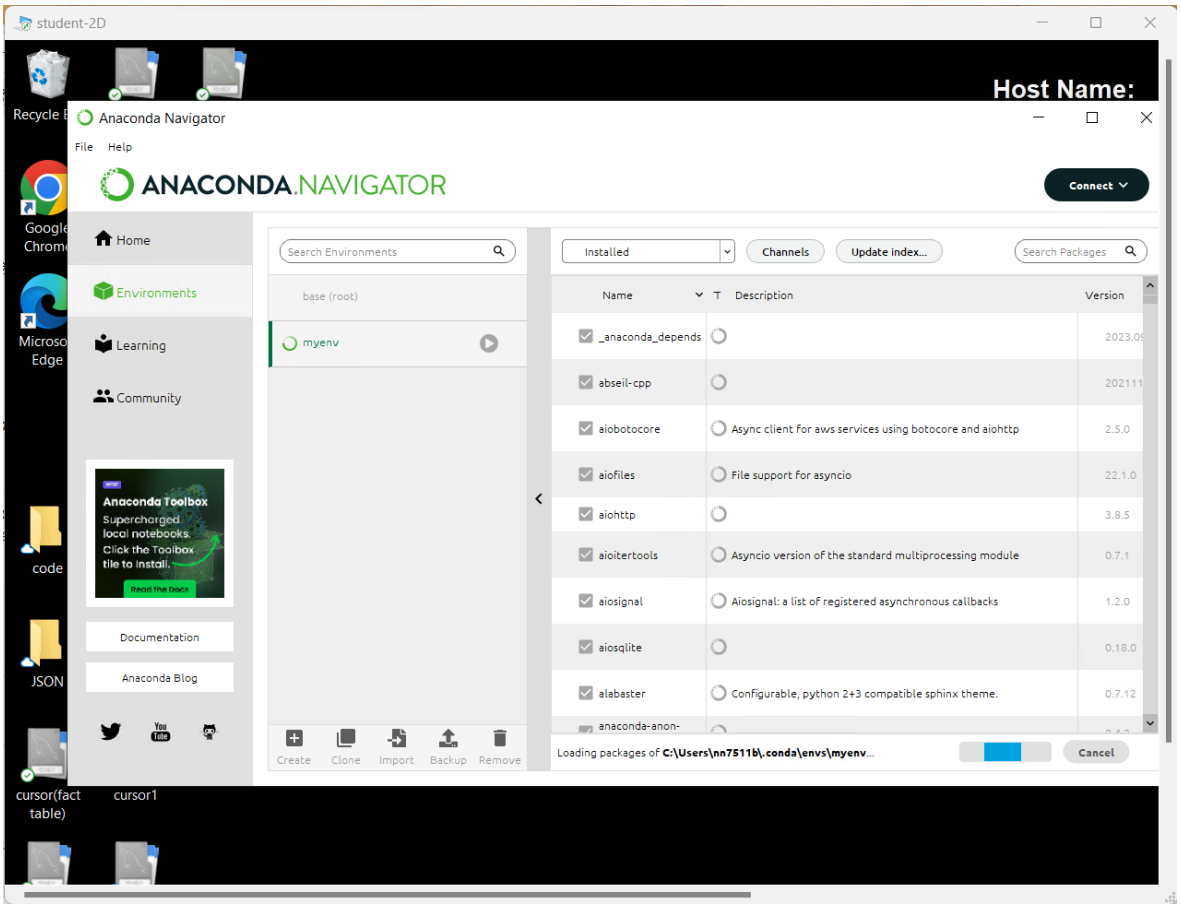


- Verify that data is populated to the sample_fact table successfully
  select * from sample_fact;



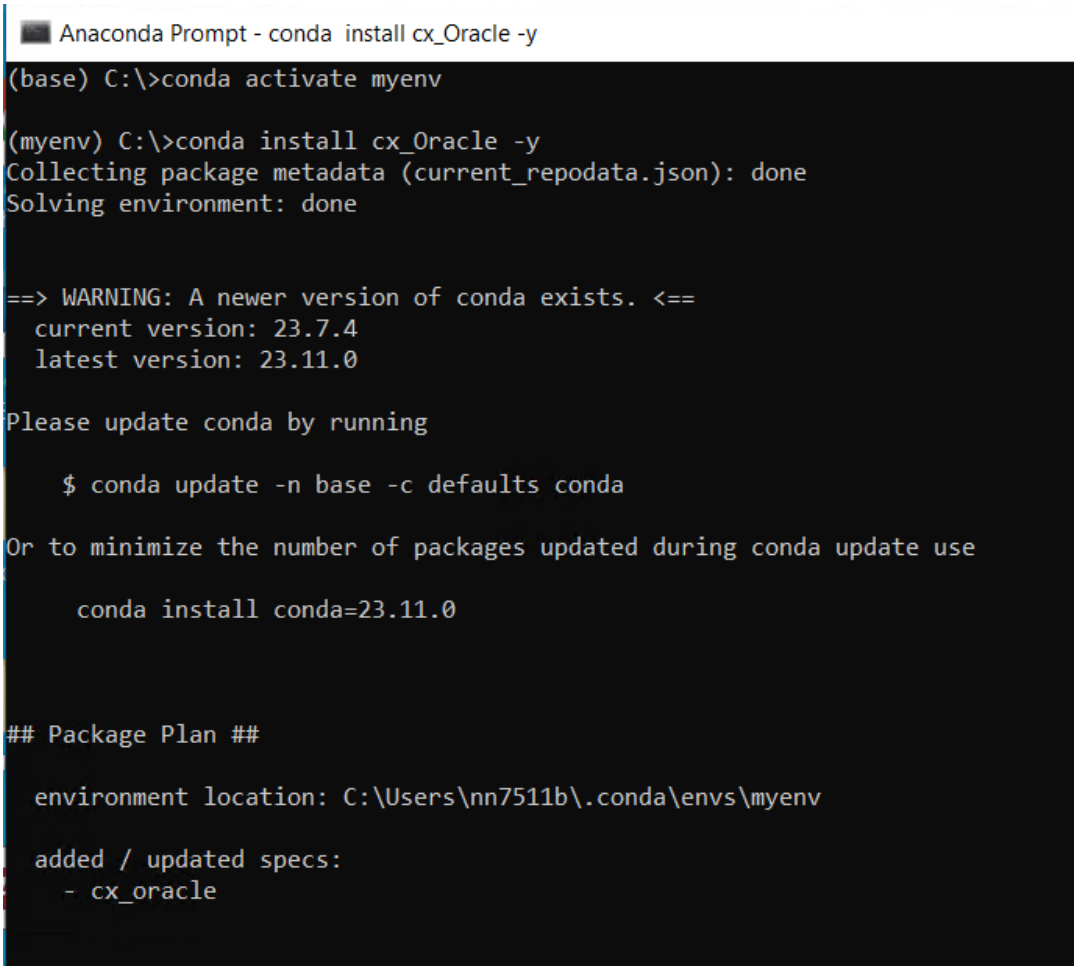
Connection of Oracle and Python

- Open Anaconda Navigator -> Go to Environment-> Create an virtual environment ‘myenv’->Activate your environment



Or you can do this in the anaconda prompt as well

- Open Anaconda prompt
- Create a virtual environment in Anaconda: **conda create --name myenv**
- **Activate your environment: conda activate myenv**
- Install cx_Oracle: **conda install cx_oracle -y**
- Install other useful libraries: **conda install jupyter pandas numpy matplotlib -y**
- Open jupyter notebook (this will open a new tab in your browser): **jupyter notebook**


Anaconda Prompt - conda install cx_Oracle -y
(base) C:\>conda activate myenv
(myenv) C:\>conda install cx_Oracle -y
Collecting package metadata (current_repodata.json): done
Solving environment: done
==> WARNING: A newer version of conda exists. <==
current version: 23.7.4
latest version: 23.11.0
Please update conda by running
$ conda update -n base -c defaults conda
Or to minimize the number of packages updated during conda update use
conda install conda=23.11.0
Package Plan
environment location: C:\Users\nn7511b\.conda\envs\myenv
added / updated specs:
- cx_oracle

SQL Queries

- Display all the records of the fact table

```
cursor = connection.cursor()

with connection.cursor() as cursor:
    cursor.execute('select * from Sample_Fact')
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

I got 2348 lines

	SAMPLE_ID	TIME_ID	SENSOR_ID	SAMPLE_METHOD_ID	SAMPLE_PURPOSE_ID	LOCATION_ID	RESULT	COMPLIANCE
0	2381	2692	909	21	25	248	10.40	1
1	2382	2552	835	21	25	248	14.40	1
2	2383	2824	835	21	33	191	65.30	0
3	2384	3104	800	26	23	182	160.00	0
4	2385	3104	835	26	23	182	4.00	0
...	...	...	...	...	...	...	...	...
2343	4724	2551	781	24	33	247	17.00	0
2344	4725	2689	839	26	33	247	9.28	0
2345	4726	2689	932	26	33	247	879.00	0
2346	4727	2690	839	26	23	247	6.36	0
2347	4728	2691	839	26	28	247	6.19	0

2348 rows × 8 columns

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

```
##The list of water sensors measured by type of it by month
Query1="""
select w.sensor_type, t.month
From WATER_SENSORS W
JOIN SAMPLE_FACT SF ON W.sensor_id=SF.sensor_id
Join Time t on SF.time_id=T.time_id
Group by w.sensor_type, t.month order by T.month"""

with connection.cursor() as cursor:
    cursor.execute(Query1)
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

```
I got 657 lines
```

	SENSOR_TYPE	MONTH
0	2,4,6-Trichlorophenol	1
1	2,6-Dimethylphenol - {2,6-Xylenol}	1
2	2-Methylphenol - {o-Cresol}	1
3	3,4-Dimethylphenol - {3,4-Xylenol}	1
4	4-Methylphenol - {p-Cresol}	1
...	...	...
652	Turbidity	12
653	Visible oil or grease, significant trace Prese...	12
654	Weather Temperature	12
655	Zinc	12
656	pH	12

657 rows × 2 columns

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

```
##The number of sensor measurements collected by type of sensor by week
Query2="""
select w.sensor_type, t.week, count(*) as measurement_count
From WATER_SENSORS W
JOIN SAMPLE_FACT SF ON W.sensor_id=SF.sensor_id
Join Time t on SF.time_id=T.time_id
Group by w.sensor_type, t.week order by T.week
"""

with connection.cursor() as cursor:
    cursor.execute(Query2)
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

```
I got 1293 lines
```

	SENSOR_TYPE	WEEK	MEASUREMENT_COUNT
0	4-Methylphenol - {p-Cresol}	1	1
1	Alkalinity to pH 4.5 as CaCO3	1	1
2	Ammoniacal Nitrogen as N	1	3
3	Cadmium	1	1
4	Chloride	1	1
...	...	...	...
1288	Oxygen, Dissolved as O2	51	1
1289	Phosphorus, Total as P	51	1
1290	Turbidity	51	1
1291	pH	51	1
1292	Phenolic Odour	53	1

1293 rows × 3 columns

- The number of sensor measurements collected by type of sensor by week including sensor_id

```
##The number of sensor measurements collected by type of sensor by week including
sensor_id
Query2="""
select w.sensor_id, w.sensor_type, t.week, count(*) as measurement_count
From WATER_SENSORS W
JOIN SAMPLE_FACT SF ON W.sensor_id=SF.sensor_id
Join Time t on SF.time_id=T.time_id
Group by w.sensor_id, w.sensor_type, t.week order by w.sensor_id, T.week
"""

with connection.cursor() as cursor:
    cursor.execute(Query2)
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

```
I got 1293 lines
```

	SENSOR_ID	SENSOR_TYPE	WEEK	MEASUREMENT_COUNT
0	781	Temperature of Water	1	1
1	781	Temperature of Water	2	3
2	781	Temperature of Water	3	3
3	781	Temperature of Water	4	2
4	781	Temperature of Water	5	2
...	...	...	...	...
1288	952	Copper BLM Bioavailable	1	1
1289	952	Copper BLM Bioavailable	25	1
1290	952	Copper BLM Bioavailable	47	1
1291	953	Chromium Hexavalent, Dissolved - {Cr VI}	1	1
1292	953	Chromium Hexavalent, Dissolved - {Cr VI}	47	1

1293 rows × 4 columns

• The number of measurements made by location by month

```
##The number of measurements made by location by month
Query3="""
select l.location_name, T.month, count(*) as measurement_count
From Location L
JOIN SAMPLE_FACT SF ON L.location_id=SF.location_id
Join Time t on SF.time_id=T.time_id
Group by l.location_name, T.month order by T.month
"""

with connection.cursor() as cursor:
    cursor.execute(Query3)
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

```
I got 417 lines
```

	LOCATION_NAME	MONTH	MEASUREMENT_COUNT
0	APOLLO OFFICE UNITS RADCLIVE RD GAWCOTT	1	4
1	BISHOPTON BECK D/S MILL BR PUMPING STN	1	1
2	BLAKESLEY BK.TRIB.TOVE KINGTHORN MILL	1	2
3	BUCKINGHAM GOLF CLUB TINGEWICK RD.	1	11
4	CASWELL BK.TRIB.TOVE CASWELL	1	3
...	...	...	...
412	SULGRAVE BK.WESTON RD.BR.	12	1
413	SYRESHAM STR.TRIB.OUSE.A43 RD.BR.KINDSHL	12	15
414	TRIB.CLIPSTONE BK.RD.BR.MANOR FARM	12	1
415	TRIB.OUZEL BK.D/S BLUEWATER	12	1
416	WILLOUGHTON BROOK AT BLYBOROUGH	12	2

417 rows × 3 columns

- The number of measurements made by location by month including location_id

```
##The number of measurements made by location by month including location_id
Query3="""
select l.location_id,l.location_name, T.month, count(*) as measurement_count
From Location L
JOIN SAMPLE_FACT SF ON L.location_id=SF.location_id
Join Time t on SF.time_id=T.time_id
Group by l.location_id,l.location_name, T.month order by l.location_id, T.month
"""

with connection.cursor() as cursor:
    cursor.execute(Query3)
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

I got 417 lines

	LOCATION_ID	LOCATION_NAME	MONTH	MEASUREMENT_COUNT
0	182	R.OUSE A422 RD.BR.BRACKLEY	1	37
1	182	R.OUSE A422 RD.BR.BRACKLEY	2	33
2	182	R.OUSE A422 RD.BR.BRACKLEY	3	30
3	182	R.OUSE A422 RD.BR.BRACKLEY	4	23
4	182	R.OUSE A422 RD.BR.BRACKLEY	5	31
...	...	...	...	...
412	261	HINTON STR.TRIB.OUSE A43 RD.BR.BRACKLEY	9	1
413	261	HINTON STR.TRIB.OUSE A43 RD.BR.BRACKLEY	12	1
414	262	SULGRAVE BK.WESTON RD.BR.	1	1
415	262	SULGRAVE BK.WESTON RD.BR.	4	2
416	262	SULGRAVE BK.WESTON RD.BR.	12	1

417 rows × 4 columns

• The average number of measurements covered for PH by year

```
##The average number of measurements covered for PH by year
Query4="""
select t.year,  Round(Avg(sf.result),2) as average_PH_measurement
From WATER_SENSORS W
JOIN SAMPLE_FACT SF ON W.sensor_id=SF.sensor_id
Join Time t on SF.time_id=T.time_id
where w.sensor_type='pH'
Group by t.year
order by t.year
"""

with connection.cursor() as cursor:
    cursor.execute(Query4)
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

```
I got 16 lines
```

	YEAR	AVERAGE_PH_MEASUREMENT
0	2000	8.20
1	2001	8.00
2	2002	8.15
3	2003	8.26
4	2004	7.99
5	2005	8.10
6	2006	8.15
7	2007	7.96
8	2008	8.11
9	2009	8.12
10	2010	7.98
11	2011	8.10
12	2012	7.74
13	2014	7.96
14	2015	8.08
15	2016	7.98

• The average value of Nitrate measurements by locations by year

```
##The average value of Nitrate measurements by location by year
Query5="""
SELECT l.location_id, L.location_name, T.year, AVG(SF.result) AS
average_nitrate_value
FROM WATER_SENSORS W
JOIN SAMPLE_FACT SF ON W.sensor_id = SF.sensor_id
JOIN TIME T ON SF.time_id = T.time_id
JOIN LOCATION L ON SF.location_id = L.location_id
WHERE W.sensor_type = 'Nitrate as N'
GROUP BY l.location_id, L.location_name, T.year
ORDER BY L.location_name, T.year
"""

with connection.cursor() as cursor:
    cursor.execute(Query5)
    df = DataFrame(cursor.fetchall())
    df.columns = [x[0] for x in cursor.description]
    print("I got %d lines " % len(df))

df
```

I got 42 lines

	LOCATION_ID	LOCATION_NAME	YEAR	AVERAGE_NITRATE_VALUE
0	198	BISHOPTON BECK D/S MILL BR PUMPING STN	2014	5.090000
1	188	BUTTERYHAUGH STW	2014	17.800000
2	201	CHIRDON BURN AT TARSET	2015	0.196000
3	213	DUDDO BURN AT WHINNEY HILL FARM	2014	2.595000
4	261	HINTON STR.TRIB.OUSE A43 RD.BR.BRACKLEY	2015	4.180000
5	208	LEAPLISH STW	2015	4.000000
6	241	MARTON WEST BECK BELOW LYTTON STREET	2014	3.790000
7	214	OTTERBURN STW	2014	17.000000
8	187	PADBURY BK.TRIB.OUSE KINGSBRIDGE FORD	2012	8.190000
9	195	PADBURY BK.TRIB.OUSE PRESTON BISSET	2012	1.431111
10	236	PADBURY BK.TRIB.OUSE STRATTON AUDLEY MIL	2012	11.097143
11	221	PRESTON BROOK WINDCROSS	2004	5.680000
12	182	R.OUSE A422 RD.BR.BRACKLEY	2000	6.450000
13	182	R.OUSE A422 RD.BR.BRACKLEY	2003	7.820000
14	182	R.OUSE A422 RD.BR.BRACKLEY	2005	6.610000
15	182	R.OUSE A422 RD.BR.BRACKLEY	2006	5.523333

Conclusion

In conclusion, the development of the water quality monitoring data warehouse, anchored by a robust star schema and aligned with a well-defined BUS plan, marks a pivotal advancement in modernizing environmental data management. From meticulous data extraction and cleansing to strategic loading processes, the warehouse now stands as a reliable repository for real-time water quality insights. This project not only addresses the limitations of traditional monitoring but also showcases the potential for informed decision-making through advanced analytics. As it seamlessly connects Oracle with Python, the data warehouse not only meets coursework objectives but also embodies a technological bridge between environmental science and data-driven solutions, paving the way for more effective water quality management.

References

Inmon, W. H. and Hackathorn, R. D. (2016) Using the Data Warehouse. Wiley.

Kimball, R. and Ross, M. (2002) The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. Wiley.

Redmond, E. and Riddle, J. (2008) Data Warehousing For Dummies. Wiley.

Inmon, W. H. (2005) 'Building the Data Warehouse', Database Trends and Applications, 19(9), pp. 23-26.

Kimball, R. (1996) 'The Data Warehouse Toolkit', Computerworld, 30(48), pp. 109-112.

Wrembel, R. and Koncilia, C. (2007) 'ETL Processes in Data Warehousing: A Survey', Information Systems Frontiers, 9(1), pp. 79-91.

Oracle (2020) 'Introduction to Data Warehousing and Business Intelligence'. Available at: https://docs.oracle.com/en/database/oracle/oracle-database/19/dwhsg/introduction-to-data-warehousing-and-business-intelligence.html (Accessed: 07/12/2023).

Talend (2021) 'What is ETL? Extract, Transform, Load Explained'. Available at: https://www.talend.com/resources/what-is-etl/ (Accessed: 07/12/2023).

Inmon, W. H. and O'Neil, D. (1996) 'Data Warehousing and OLAP', in Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB '96), pp. 146-155.

Kimball, R. and Ross, M. (2003) 'Data Warehouse Bus Architecture', in Proceedings of the 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK '03), pp. 173-188.