Contents

Cornateanu Laurentiu
Student ID: 001295276
Course: MSc Big Data and Business Intelligence
Faculty: Engineering & Science
Location: Greenwich Maritime Campus
Lc9754t@gre.ac.uk

Data Warehousing & BI

COMP1848 (2023/24)

Course Leader: Hooman Oroojeni

Tasks allocations among group members:
Group members:
1.Cornateanu Laurentiu

Data Warehousing Fundamentals:

Organisations generate and accumulate large volumes of data on a daily basis in the continually shifting world of modern business. If used correctly, this data has the ability to give significant insights that can drive informed decision-making and strategic planning. The issue, however, is in properly managing, analysing, and extracting relevant information from this huge amount of data.

This is where data warehousing can help. Data warehousing is a thorough method of collecting, storing, and managing data from multiple sources inside an organisation in a centralised repository. The major goal is to develop a unified and optimised data analysis and reporting environment. Organisations can expedite their reporting procedures and acquire a holistic perspective of their operations by integrating data from different sources into a single, integrated repository. A massive amount of data.

This is when data warehouses can help. Data warehousing is a thorough method of collecting, storing, and managing data from multiple sources inside an organisation in a centralised repository. The major goal is to develop a unified and optimised data analysis and reporting environment. Organisations can expedite their reporting procedures and acquire a holistic perspective of their operations by integrating data from different sources into a single, integrated repository.

Key Data Warehousing Components:

ETL (take, Transform, Load): ETL operations are essential in data warehousing because they take data from many sources, transform it into a consistent format, and load it into the data warehouse. This ensures that the data is uniform and ready for analysis.

The data warehouse is the heart of data warehousing, a centralised repository that contains historical and current data from numerous sources. The data warehouse is intended for inquiry and analysis, and it serves as a foundation for business intelligence tools and applications.

Data marts are data warehouse subsets that focus on specific business functions or user groups. They provide more customised research and reporting, adapting to the specific demands of various departments.

Metadata Management: Strong metadata management is essential for effective data warehousing. Metadata, also known as data about data, provides valuable information about the structure, origin, and usage of data within the warehouse, allowing for more efficient data governance and traceability.

Business intelligence (BI) tools allow users to interact with and analyse data stored in a data warehouse. They include reporting, dashboard, and visualisation tools that aid in the transformation of raw data into actionable insights.

Advantages of Data Warehousing:

Improved Decision-Making: Data warehousing enables organisations to make informed decisions based on a unified and comprehensive view of their data.

Data Warehousing improves data quality and accuracy by standardising and centralising data, reducing errors and inconsistencies.

Enhanced Efficiency: The streamlined process of data extraction, transformation, and loading, combined with optimised query performance, leads to enhanced data analysis efficiency.

Data warehousing aids long-term strategic planning by providing historical data trends and insights, allowing organisations to identify patterns and make predictions.

To summarise, data warehousing is an essential component of modern business intelligence, allowing organisations to maximise the value of their data for better decision-making, strategic planning, and overall operational efficiency.

"My data sources are unreliable, but their information is fascinating."
– Ashleigh Brilliant

CREATE TABLES:

CREATE TABLE command in SQL is used to create a new table within a database.

When tables are combined, they can form a star schema, which is the core structure of data marts and data warehouses. Dimension tables include descriptive information, whereas fact tables have quantitative data with cross-references to dimension tables to provide context. This system is intended to supply effective answers to questions in this project case about water quality measures.

A. Dimensional Sensor:

The table being produced is called Dimension Sensor. A dimension table in a data warehouse star schema often stores descriptive information concerning a certain business feature, in this case sensor information.

Columns:

The table contains three columns in parentheses: Sensor ID, Sensor Name, and Sensor Type. These columns show the different attributes or properties of the sensors.

Data types:
INT, VARCHAR (255), and VARCHAR (50) are the data types assigned to columns:
Sensor ID is an integer (integer) and is defined as the primary key of this table. Primary keys are unique identifiers for each record in a table.
Sensor Name is a variable string with a maximum length of 255 characters.
Sensor Type is a variable string with a maximum length of 50 characters.

PRIMARY KEY:
PRIMARY KEY is a constraint applied to the sensor ID column. Ensures that each sensor ID in the table is unique and is the primary identifier for each entry.

Summary:
SQL statement creates a table called Dimension Sensor with three columns (Sensor ID, Sensor Name, and Sensor Type). The Sensor ID column is designated as the primary key, so each sensor has a unique identifier. Other columns store descriptive information about each sensor, such as name and type.

B. Dimension Location:

Dimension The table being generated is called Location. A dimension table in a data warehouse star schema often stores descriptive information about places. Columns:

The following columns are stated in brackets in the table: Location ID, Location Name, Location Type, Latitude, and Longitude. These columns display various aspects or characteristics of the sites.

Data types:
INT, VARCHAR (255), VARCHAR (50), and DECIMAL (10, 6) are the data types assigned to columns:
Location ID is an integer (integer) and is defined as the primary key of this table. Primary keys are unique identifiers for each record in a table.
Location Name is a variable character string with a maximum length of 255 characters.
Location Type is a variable character string with a maximum length of 50 characters.
Latitude and Longitude are decimal numbers with a precision of 10 digits and a scale of 6, standing for the geographic coordinates of the location.

PRIMARY KEY:
PRIMARY KEY is a constraint on the Location ID column. Ensures that each location ID in the table is unique and is the primary identifier for each record.

Summary: creates a Dimension Location table with five columns (Location ID, Location Name, Location Type, Latitude, and Longitude). Location ID column is specified as the primary key, so each location has a unique identifier. Other columns store descriptive information about each location, including name, type, and geographic coordinates.

C. Dimension Time:

Dimension Time is the table that is being built. A dimension table in a data warehouse star schema often houses time-sensitive information. Weather data is more complex and requires more features. It is preferable to keep them in their own dimension. When developing your data warehouse schema, consider the trade-offs between data redundancy, storage, and query performance.

Columns:

The table holds six parenthesized columns: Time ID, Measurement Date, Year, Month, Week, and Day. These columns represent various time-related attributes or properties.

Data types:
INT and DATE are data types assigned to columns:
Time ID is an integer (integer) and is defined as the primary key of this table. Primary keys are unique identifiers for each record in a table.
Measurement Date is a date type that stores information about a specific date.
Year, month, week, and day are integers that store information about the year, month, week, and day of a date, respectively.

PRIMARY KEY:
PRIMARY KEY is a constraint on the Time ID column. This ensures that each time ID in the table is unique and is used as the primary identifier for each record.
In summary, this SQL statement creates a Dimension Timetable with six columns (Time ID, Measurement Date, Year, Month, Week, and Day). The Time ID column is specified as the primary key, so each time entry has a unique identifier. Other columns have information about the measurement date and its components, such as year, month, week, and day. This table can be used as a dimension to analyse data by time.

D. Dimension Measurement Type:
Dimension Measurement Type is the name of the table to be created. In a data warehouse star schema, such a dimension table typically stores information about multiple types of measures.

Columns:
The table holds two columns in parentheses:
Measurement Type ID and Measurement TypeName. These columns show the various attributes or properties associated with measurement types.

Data types:
INT and VARCHAR (50) are data types assigned to columns:
Measurement Type ID is an integer (integer) and is defined as the primary key of this table. Primary keys are unique identifiers for each record in a table.
Measurement Type Name is a string variable with a maximum length of 50 characters. Stores the names of various measurement types.

PRIMARY KEY:
PRIMARY KEY is a constraint on the Measurement Type ID column. Ensures that each Measurement Type ID in the table is unique and is the primary identifier for each record.

UNIQUE Constraint:
UNIQUE is a constraint on the Measurement Type Name column. Ensures that each value in the Measurement Type Name column is unique, thus avoiding duplicate measurement type names in the table.
Essentially, this SQL statement creates a Dimension Measurement Type table with two columns (Measurement Type ID and Measurement Type Name). The Measurement Type ID column is specified as the primary key, so each measurement type record has a unique identifier. The Measurement Type Name column stores the names of the different measurement types, and the UNIQUE constraint ensures that each name is unique in the table. This table can be used as a dimension for classifying tools by type.

E. Fact Water Measurement:
Fact Water Measurement is the name of the table being created. In a data warehouse star schema, a fact table like this typically stores quantitative and numeric measurements.
Columns:
This table contains six columns in parentheses: Fact ID, Sensor ID, Location ID, Time ID, Measurement Type, and Measurement Value. These columns show various attributes or properties related to water measurements.
Data types:
INT, VARCHAR (50), and DECIMAL (12, 6) are the data types assigned to columns:
Fact ID is an integer (integer) and is defined as the primary key of this table. Primary keys are unique identifiers for each record in a table.
Sensor ID, Location ID, and Time ID are integers that represent foreign keys that point to the primary keys in the corresponding dimension tables (Dimension Sensor, Dimension Location, and Dimension Time).

Measurement Type is a string variable with a maximum length of 50 characters that stands for a foreign key that references the Dimension Measurement Type table. Measurement Value is a decimal number with a precision of 12 digits and a scale of 6 that stands for the actual measurement value.

PRIMARY KEY:
PRIMARY KEY is a constraint on the Fact ID column. It ensures that each Fact ID in the table is unique and acts as the primary identifier for each water meter record.

FOREIGN KEY Constraints:
FOREIGN KEY constraints apply to the Sensor ID, Location ID, Time ID, and Measurement Type columns. These constraints ensure referential integrity by proving relationships between fact tables and dimension tables.

fk_sensor, fk_location, fk_time and fk_measurement_type are the constraint names.
The REFERENCES keyword specifies the table and column referenced by each foreign key.

Summary, this SQL statement creates a table called Fact Water Measurement with six columns, where Fact ID is the primary key, and the other columns are foreign keys that point to the dimension tables. This table is intended to store water measurement data, and the outer main constraints are given in the dimensional tables for added contextual information.

1. Your Star Schema Design and BUS plan.

BUS PLAN

Business Intelligence (BI) Bus Plan

The BI bus plan outlines the key dimensions and facts that will be part of data mart and helps define the scope and focus of your BI solution.

Business Questions
What is the current water quality at specific locations?
•How have water quality measurements changed over time?
•Which sensors are most frequently used?
•Are there any trends or anomalies in water quality data?

Dimensions
Define the dimensions (attributes) that are relevant to answering the business questions. These dimensions will be represented in the dimension tables.
Project case, dimensions may include Sensor, Location, Time, and Measurement Type.
Facts
Define the facts (measures) that are critical for the analysis. These facts will be stored in the fact table.
n this case, the primary fact is the MeasurementValue, which provides the quantitative data on water quality.
Attributes
Identify the specific attributes within each dimension that are important for analysis.
Within the Sensor dimension, attributes might include SensorName and SensorType.
Hierarchies

Specify the hierarchies that exist within dimensions. For instance, the Time dimension could have hierarchies like Year > Month > Day.

Aggregations

Determine the aggregations you'll need for efficient reporting. Aggregations allow for quicker retrieval of summary data.

For water quality data, you might want to calculate monthly averages or location-based statistics.
Security and Access Control
Define who has access to which parts of the data mart and implement necessary security measures to protect sensitive data.
Data Quality and Cleansing
Plan for data quality checks and cleansing processes to ensure that the data is accurate and reliable.
Report and Dashboard Requirements
Identify the types of reports and dashboards that will be built on top of the data mart, and the specific KPIs and metrics they will display.
Data Integration and ETL:
Describe the ETL processes that will be used to integrate data from source systems into the data mart.
Performance Optimization
Consider strategies for optimizing query performance, such as indexing, caching, and pre-aggregations.

Data Governance and Documentation

Establish data governance practices and ensure thorough documentation of the data mart's structure and processes.

1.Bus plan serves as a blueprint for data warehouse and guides the development and maintenance and to align with business objectives. It ensures that the datawarehouse is designed and implemented in a way that supports data-driven decision-making and provides valuable insights to stakeholders.

Each row in the BI bus matrix stands for a business process or subject area, and each column stands for a data related part. The presence and appearance of the data element linked with that business process is shown by the intersection of a row and a column.

The BI Bus Matrix's exact design and content will be decided by the specific demands and requirements of your organisation and data analysis operations. It is a tool that aids in the organisation of your knowledge of the linkages between business processes and data pieces.

	Fact Water Measurement	Dimension Sensor	Dimension Location	Dimension Time	Dimension Measurement Type
Water Quality	Measurement Value	Sensor ID	Location ID	Time ID	Measurement Type ID
Sensor Details		Sensor Name			
Location Details			Location Name		
Time Details				Date, Year, Month, Week, Day	
Measurement Type	Measurement Type				

2.FACT TABLE

Fact Table:

Fact Table: Fact Water Measurement
Has quantitative, numerical measurements (water quality measurements).
Primary key (Fact ID) uniquely identifies each record.
Foreign keys connect to dimension tables.

Dimension Tables:

Dimension Sensor:
Descriptive information about sensors.
Primary key (Sensor ID) links to Sensor ID in the fact table.

Dimension Location:

Descriptive information about locations.
Primary key (Location ID) links to Location ID in the fact table.

Dimension Time:

Temporal information about measurements.
Primary key (Time ID) links to Time ID in the fact table.

Dimension Measurement Type:

Descriptive information about measurement types.
Primary key (Measurement Type ID) links to Measurement Type in the fact table.

2. Data Source Selection:

Access the provided Oracle server.
Acquire the water quality data from the Department for Environment Food & Rural Affairs, or use the provided dataset (WaterQuality_CW.zip) .

3. Documentation for your ETL processes to include all scripts**.**

Implement the ETL process to extract data from the source, transform it to fit the data warehouse schema, and load it into the database.
Ensure that the ETL process is automated and can be scheduled to update the data regularly.
Data Understanding and Cleaning:
Review the data quality and assess whether there are any issues with the dataset, as mentioned in the scenario.
Clean and pre-process the data as needed. This may involve handling missing values, outliers, and ensuring data consistency.

4. Queries and Reports**:**

Write SQL queries to answer the questions specified in the project requirements, such as the list of water sensors by type, the number of sensor measurements by type, etc.
Create a report generation mechanism to visualize and present the results

5. User Access and Security**:**
Set up user accounts for the team members to access the database.
Implement security measures to protect the data and ensure that only authorized users can access it.

6. Regular Team Meetings**:**
Schedule regular meetings via MS Teams or face-to-face to discuss progress, challenges, and coordinate work among team members.
Remember to adhere to good database design practices, maintain data integrity, and ensure the data warehouse is capable of handling future data updates and expansions.
Regular communication and collaboration among team members are crucial for the success of this project.

7. Testing and Validation**:**
Thoroughly test the data warehouse and the queries to ensure they produce accurate results.
Validate the results against known benchmarks or expectations.

8. Documentation**:**
Document the entire process, including data source, database design, ETL process, query descriptions, and any issues encountered during the project.
Identify Business Requirements and Queries
Understand the specific needs of the water quality monitoring project. Identify the key queries and reporting requirements that the data mart should support.

2.Design the Star Schema

Design a star schema that aligns with the identified business requirements. The star schema simplifies data access, facilitates efficient querying, and supports multidimensional analysis.

3.Define Dimension Tables

Dimension tables provide context to measurements. For example, the Time, Location, Sensor, and Measurement Type dimensions provide details that allow users to analyze data from different perspectives.

4.Define Fact Table

Rationale: The fact table contains the core measurements. It links to dimension tables through foreign keys, enabling the creation of meaningful relationships for analysis.

In this case, the FactWaterQuality table captures water quality measurements.

5.Indexing for Performance

Indexes on foreign key columns in the fact table improve query performance by speeding up joins. In this case, indexes on TimeID, LocationID, SensorID, and MeasurementTypeID facilitate efficient data retrieval.

6.Data Loading Strategy

Consider efficient data loading strategies, especially for large datasets. Bulk loading methods can significantly improve the loading process.

7.Query Optimization

Optimize queries for efficient execution. Leverage indexes, use appropriate filters, and ensure proper sorting for query performance. The EXPLAIN statement helps analyze query execution plans.

8.Verify and Test

Before deploying the data mart, perform thorough testing. Validate that the schema meets business requirements and those queries execute within acceptable time frames.

9. Documentation

Document the star schema design, including table structures, relationships, and indexing strategies.

Documentation is crucial for future reference, maintenance, and collaboration among team members.

By following these steps & rationales create a well-structured and optimized star schema for the water quality monitoring project. This schema facilitates efficient data analysis, supports reporting requirements, and ensures scalability for future enhancements. Each step is designed to address specific considerations in the data warehousing process, contributing to the overall success of the project.

10. Enhanced Star Schema Design:

Dimension Tables:

Time Dimension (DimTime):
- TimeID (Primary Key)
- Date (Date)
- Year (Integer)
- Month (Integer)
- Week (Integer)
- Day (Integer)

Location Dimension (DimLocation):
- LocationID (Primary Key)
- LocationName (Text)
- LocationType (Text)
- Latitude (Real)
- Longitude (Real)
- Region (Text)
- Country (Text)

Sensor Dimension (DimSensor):
- SensorID (Primary Key)
- SensorName (Text)
- SensorType (Text)
- Manufacturer (Text)
- InstallationDate (Date)

Measurement Type Dimension (DimMeasurementType):
- MeasurementTypeID (Primary Key)
- MeasurementTypeName (Text)
- UnitOfMeasure (Text)

Fact Table:

Water Quality Fact Table (FactWaterQuality):
- FactID (Primary Key)
- TimeID (Foreign Key)
- LocationID (Foreign Key)
- SensorID (Foreign Key)
- MeasurementTypeID (Foreign Key)
- MeasurementValue (Real)
- SampleID (Text)
- QualityFlag (Text)

```
CREATE INDEX idx_time ON FactWaterQuality (TimeID);
CREATE INDEX idx_location ON FactWaterQuality (LocationID);
CREATE INDEX idx_sensor ON FactWaterQuality (SensorID);
CREATE INDEX idx_measurement_type ON FactWaterQuality (MeasurementTypeID);
```

10. ETL Processes:

Extract:

Extract data from the source system (provided by the Environment Agency).

Transform:

Transform and clean data to fit the defined dimensions and fact tables.

Populate the Date, Sensor, Location, and MeasurementType dimension tables.

Generate surrogate keys for each dimension.

Populate the fact table with transformed data.

Load:

Load the data into the data warehouse.

A. SQL Queries:

List of water sensors measured by type by month:

```
SELECT
  SensorType,
  Month,
  COUNT(*) AS MeasurementCount
FROM
  Fact_WaterSensorMeasurement
GROUP BY
  SensorType, Month;
```

B. Number of sensor measurements collected by type of sensor by week:

```
SELECT
  SensorType,
  Week,
  COUNT(*) AS MeasurementCount
FROM
  Fact_WaterSensorMeasurement
GROUP BY
  SensorType, Week;
```

C. Number of measurements made by location by month:

```
SELECT
  LocationName,
  Month,
  COUNT(*) AS MeasurementCount
FROM
  Fact_WaterSensorMeasurement
GROUP BY
  LocationName, Month;
```

D. Average number of measurements covered for PH by year:

```
SELECT
  Year,
  AVG(MeasurementValue) AS AvgPH
FROM
  Fact_WaterSensorMeasurement
WHERE
  MeasurementType = 'PH'
GROUP BY
  Year;
```

E. Average value of Nitrate measurements by locations by year:

```
SELECT
  LocationName,
  Year,
  AVG(MeasurementValue) AS AvgNitrate
FROM
  Fact_WaterSensorMeasurement
WHERE
  MeasurementType = 'Nitrate'
GROUP BY
  LocationName, Year;
```

These queries should provide the necessary information based on the specified requirements. The data warehouse design and ETL processes must be tailored to the specific data available from the Environment Agency.

Data Mart Star Schema Design:
Fact Table:

Fact_WaterQualityMeasurement:
Columns: MeasurementID (Surrogate Key), DateKey, LocationKey, ParameterKey, MeasurementValue, MeasurementUnit.

Dimension Tables:

Dim_Date:
Columns: DateKey, Date, Day, Week, Month, Quarter, Year.

Dim_Location:
Columns: LocationKey, LocationName, LocationType.

Dim_Parameter:
Columns: ParameterKey, ParameterName, ParameterType.

ETL Process:

Extract:
Export data from the Microsoft Access database.

Transform:
Cleanse and transform data as needed (handling missing values, standardizing formats, etc.).
Populate the Dim_Date, Dim_Location, and Dim_Parameter tables by extracting unique values from the source data.
Generate surrogate keys for each dimension table.

Load:
Load the transformed data into the Fact_WaterQualityMeasurement table.
Populate the foreign keys in the fact table using the surrogate keys generated in the dimension tables.

Steps for Exporting Data from Microsoft Access to Oracle:

Export Data:
Use the export functionality in Microsoft Access to export relevant tables or queries to a format compatible with Oracle (e.g., CSV).

Oracle Staging Area:
Create a staging area in Oracle to store the exported data temporarily.

Data Import:
Use Oracle tools (e.g., SQL*Loader, Oracle Data Pump) or other methods to import the exported data into the Oracle staging area.

Cleansing and Transformations:
Write SQL scripts or use Oracle tools to perform data cleansing and necessary transformations in the staging area.

Load into Star Schema:
Load the cleansed and transformed data into the Data Mart star schema tables (Dim_Date, Dim_Location, Dim_Parameter, Fact_WaterQualityMeasurement).

Considerations:

Ensure data types and formats are consistent between Microsoft Access and Oracle.
Handle any data quality issues during the transformation process.
Use appropriate Oracle tools or scripts for efficient data loading and transformation.
Document the ETL process, including scripts and any business rules applied.

Export Data from Microsoft Access:

Export to CSV:
Export your tables or queries from Microsoft Access to CSV files.

Create Oracle Staging Area:

Create Staging Tables in Oracle:

sql

```
CREATE TABLE staging_water_quality (
  Define columns based on your CSV structure
);
```

Import Data into Oracle:

SQL*Loader Example:

Create a control file (e.g., loader.ctl) for SQL*Loader:

```
LOAD DATA
INFILE 'your_data.csv'
INTO TABLE staging_water_quality
FIELDS TERMINATED BY ',' -- Adjust based on your CSV delimiter
(column1, column2, ...);
```

Run SQL*Loader command:

```
sqlldr username/password@your_oracle_database control=loader.ctl
```

Cleansing and Transformation:

Cleanse and Transform in Oracle:

Write SQL scripts to clean and transform data in the staging table:

```
Remove duplicates
DELETE FROM staging_water_quality
WHERE rowid not in (SELECT MIN(rowid) FROM staging_water_quality GROUP BY column1, column2, ...);
```

```
Convert date format
UPDATE staging_water_quality
```

```
    SET date_column = TO_DATE(date_column, 'MM/DD/YYYY');
```

Load into Star Schema:

Load into Star Schema Tables:
Insert data into your Data Mart star schema tables (Dim_Date, Dim_Location, Dim_Parameter, Fact_WaterQualityMeasurement) using appropriate SQL statements.

Insert into Dim_Date

```
INSERT INTO Dim_Date (DateKey, Date, Day, Week, Month, Quarter, Year)
SELECT DISTINCT TO_DATE(date_column, 'MM/DD/YYYY'), ... FROM staging_water_quality;
```

Insert into Fact_WaterQualityMeasurement

```
INSERT INTO Fact_WaterQualityMeasurement (DateKey, LocationKey, ParameterKey, MeasurementValue, MeasurementUnit)
SELECT d.DateKey, l.LocationKey, p.ParameterKey, wqm.value_column, wqm.unit_column
FROM staging_water_quality wqm
JOIN Dim_Date d ON TO_DATE(wqm.date_column, 'MM/DD/YYYY') = d.Date
JOIN Dim_Location l ON wqm.location_column = l.LocationName
JOIN Dim_Parameter p ON wqm.parameter_column = p.ParameterName;
```

design a basic star schema for the water quality data mart. A star schema typically consists of a fact table (containing measurements) and dimension tables (containing information related to dimensions like time, location, and parameter). Here's a simplified example:
Fact Table: Fact_WaterQualityMeasurement

- MeasurementID (Primary Key)
- DateKey (Foreign Key referencing Dim_Date)
- LocationKey (Foreign Key referencing Dim_Location)
- ParameterKey (Foreign Key referencing Dim_Parameter)
- MeasurementValue
- MeasurementUnit

Dimension Tables:
a. Dim_Date

- DateKey (Primary Key)
- Date
- Day
- Week
- Month
- Quarter
- Year

b. Dim_Location

- LocationKey (Primary Key)
- LocationName
- Latitude
- Longitude
- Other relevant location information

c. Dim_Parameter

- ParameterKey (Primary Key)
- ParameterName
- ParameterDefinition
- ParameterNotation
- ParameterUnit

You would then join these tables based on the keys for reporting purposes.

```
SELECT
    d.Date,
```

```
    l.LocationName,
    p.ParameterName,
    f.MeasurementValue,
    f.MeasurementUnit
FROM
    Fact_WaterQualityMeasurement f
JOIN
    Dim_Date d ON f.DateKey = d.DateKey
JOIN
    Dim_Location l ON f.LocationKey = l.LocationKey
JOIN
    Dim_Parameter p ON f.ParameterKey = p.ParameterKey
WHERE
    Add conditions as needed (e.g., date range, specific locations, parameters)
```

This query would give you a report with measurements, including descriptive information from the dimension tables.
Data Quality Checks:
Perform data quality checks to ensure that data integrity is maintained during the load process.

Transform and Load into Data Mart:
Move data from staging tables to the final data mart tables using SQL queries or any ETL tool available in your environment.

Example SQL for Loading into Staging Tables:

Assuming you have already created staging tables in Oracle:

Example for loading data into a staging table
Replace 'your_staging_table' with the actual name of your staging table

Using SQL*Loader for CSV file

```
LOAD DATA
INFILE 'your_exported_data.csv'
INTO TABLE your_staging_table
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
 column1,
 column2,
)
```

Using Oracle Data Pump for Excel file
Replace 'your_exported_data.xls' with the actual name of your Excel file
Replace 'your_staging_table' with the actual name of your staging table

Assuming Excel data is in a sheet named 'Sheet1'
Data will be loaded into the staging table with the same column names

Make sure you have an Oracle directory object created for the file path

```
CREATE DIRECTORY your_directory AS '/path/to/directory';
```

Run the Data Pump job

```
BEGIN
 DBMS_DATAPUMP.create_job(
   job_name       => 'LOAD_STAGING_TABLE_JOB',
   operation      => 'IMPORT',
   job_mode       => 'TABLE',
   remote_link    => null,
   job_queue      => 'DEFAULT_QUEUE',
   parallelism    => 1,
   data_options   => null,
   metadata_options => DBMS_DATAPUMP.KU$_METADATA_QUICK,
   table_exists_action => 'TRUNCATE',
   logfile        => 'LOAD_STAGING_TABLE_LOG.log',
   dumpfile       => 'your_exported_data.xls',
   directory      => 'YOUR_DIRECTORY'
```

```
 );
 DBMS_DATAPUMP.start_job('LOAD_STAGING_TABLE_JOB');
END;
/
```

Data cleansing is a crucial step in the ETL (Extract, Transform, Load) process to ensure the quality and integrity of the data in the Data Warehouse. Below are some common types of errors and techniques for cleansing, along with examples of how SQL can be used for these purposes:

Types of Errors and Cleansing Techniques:

Missing Primary Keys:

Technique: Identify records with missing primary keys and either impute values or exclude the records.

SQL Example:

```
Identify records with missing primary keys
SELECT *
FROM your_table
WHERE primary_key_column IS NULL;
```

Missing Foreign Keys:

Technique: Identify records with missing foreign keys and either impute values, exclude the records, or set to a default value.

```
Identify records with missing foreign keys
SELECT *
FROM your_table
WHERE foreign_key_column IS NULL;
```

Misspellings:

Technique: Identify and correct misspellings in textual data.

```
Find and update records with misspellings
UPDATE your_table
SET column_name = 'correct_spelling'
WHERE column_name = 'mispelled_value';
```

Remove Unnecessary Records/Columns:

Technique: Identify and remove records or columns that are not relevant to the analysis.

SQL Example:

```
Delete unnecessary records
DELETE FROM your_table
WHERE condition;

Drop unnecessary columns
ALTER TABLE your_table
DROP COLUMN unnecessary_column;
```

Impute Missing Values:

Technique: Fill in missing values based on business rules, averages, or other relevant methods.

SQL Example:

Impute missing values with the average

```
UPDATE your_table
SET column_name = (SELECT AVG(column_name) FROM your_table WHERE column_name IS NOT NULL)
```

```
    WHERE column_name IS NULL;
```

Always take backups before performing updates or deletions.
To build the Data Warehouse, you'll need to create and populate the fact and dimension tables for your star schema. Below are examples of SQL queries to create and populate the FACT and TIME tables concurrently using a cursor. After that, I'll provide SQL queries to generate the required statistical information.
Creating and Populating FACT and TIME Tables:

Star schema with a FACT table named WaterQualityFact and a TIME dimension table named TimeDimension:

```
 Create TIME Dimension Table
 CREATE TABLE TimeDimension (
   TimeID INT PRIMARY KEY,
   Year INT,
   Month INT,
   Week INT
);

 Create FACT Table
 CREATE TABLE WaterQualityFact (
   FactID INT PRIMARY KEY,
   SensorID INT, -- Foreign key to Sensor dimension
   LocationID INT, -- Foreign key to Location dimension
   TimeID INT, -- Foreign key to Time dimension
   MeasurementType VARCHAR(50),
   MeasurementValue FLOAT
);

 Populate TIME Dimension Table using a cursor
 DECLARE
   vYear INT;
   vMonth INT;
   vWeek INT;
BEGIN
   FOR r IN (SELECT DISTINCT EXTRACT(YEAR FROM samplesampleDateTime) AS Year,
                             EXTRACT(MONTH FROM samplesampleDateTime) AS Month,
                             TO_CHAR(samplesampleDateTime, 'WW') AS Week
             FROM YourOriginalTable)
   LOOP
      vYear := r.Year;
      vMonth := r.Month;
      vWeek := r.Week;

      INSERT INTO TimeDimension (TimeID, Year, Month, Week)
      VALUES (YourSequence.NEXTVAL, vYear, vMonth, vWeek);
   END LOOP;
END;


Populate FACT Table
INSERT INTO WaterQualityFact (FactID, SensorID, LocationID, TimeID, MeasurementType, MeasurementValue)
SELECT YourSequence.NEXTVAL,
     SensorID, -- Replace with actual Sensor ID based on your schema
     LocationID, -- Replace with actual Location ID based on your schema
     YourSequence.NEXTVAL, -- Use a sequence for TimeID
     DeterminandLabel,
FROM YourOriginalTable;
```

Statistical Information Queries:

Now, you can use SQL queries to provide the required statistical information:

List of water sensors measured by type by month:

```
SELECT t.Year, t.Month, wq.MeasurementType, COUNT(DISTINCT wq.SensorID) AS NumSensors
FROM WaterQualityFact wq
JOIN TimeDimension t ON wq.TimeID = t.TimeID
GROUP BY t.Year, t.Month, wq.MeasurementType;
```

Number of sensor measurements collected by type of sensor by week:

```
SELECT t.Week, wq.MeasurementType, COUNT(wq.SensorID) AS NumMeasurements
FROM WaterQualityFact wq
JOIN TimeDimension t ON wq.TimeID = t.TimeID
GROUP BY t.Week, wq.MeasurementType;
```

Number of measurements made by location by month:

```
SELECT t.Year, t.Month, l.LocationName, COUNT(wq.MeasurementType) AS NumMeasurements
FROM WaterQualityFact wq
JOIN TimeDimension t ON wq.TimeID = t.TimeID
JOIN LocationDimension l ON wq.LocationID = l.LocationID
GROUP BY t.Year, t.Month, l.LocationName;
```

Average number of measurements covered for PH by year:

```
SELECT t.Year, AVG(COUNT(wq.MeasurementType)) AS AvgNumMeasurements
FROM WaterQualityFact wq
JOIN TimeDimension t ON wq.TimeID = t.TimeID
WHERE wq.MeasurementType = 'PH'
GROUP BY t.Year;
```

Average value of Nitrate measurements by locations by year:

```
SELECT t.Year, l.LocationName, AVG(wq.MeasurementValue) AS AvgNitrateValue
FROM WaterQualityFact wq
JOIN TimeDimension t ON wq.TimeID = t.TimeID
JOIN LocationDimension l ON wq.LocationID = l.LocationID
WHERE wq.MeasurementType = 'Nitrate'
GROUP BY t.Year, l.LocationName;
```

connection between Python and Oracle and extract information from your star schema, you can use the cx_Oracle library. Make sure to install the library first by running:

```
pip install cx-Oracle
```

Now, you can use the following Python code as a template. Adjust the connection details, queries, and other parameters according to your specific setup.

```
import cx_Oracle

oracle_username = "your_username"
oracle_password = "your_password"
oracle_connection_string = "your_connection_string"  # e.g., localhost:1521/your_service_name

connection = cx_Oracle.connect(oracle_username, oracle_password, oracle_connection_string)
cursor = connection.cursor()
   SELECT t.Year, t.Month, wq.MeasurementType, COUNT(DISTINCT wq.SensorID) AS NumSensors
   FROM WaterQualityFact wq
   JOIN TimeDimension t ON wq.TimeID = t.TimeID
   GROUP BY t.Year, t.Month, wq.MeasurementType

   cursor.execute(query)

   results = cursor.fetchall()
```

```
    for result in results:
        print(result)

finally:
    cursor.close()
    connection.close()
```

Make sure to replace placeholders like your_username, your_password, and your_connection_string with your actual Oracle database credentials.

This script establishes a connection to Oracle, executes a sample SQL query on your star schema, fetches the results, and prints them. Adapt the query and processing logic based on your specific requirements.

If you encounter any issues, ensure that your Oracle client is correctly installed, and the Oracle database is accessible from your Python environment.

Data Warehouse Implementation Summary.

1. Introduction

The data warehouse implementation project aimed to revolutionize water quality data management, offering a centralized platform for real-time monitoring and advanced analytics. The initiative was driven by the need for more efficient, cost-effective, and accessible methods of handling water quality data across diverse locations and sources.

2. DW Design and Implementation

2.1 Star Schema Design

The star schema was meticulously crafted to optimize query performance and facilitate intuitive data exploration. The central FACT table, "WaterQuality_Fact," serves as the nucleus, surrounded by DIMENSION tables – "Time_Dim," "Sensor_Dim," and "Location_Dim." This design ensures flexibility and scalability, accommodating evolving data needs.

2.2 BUS Plan

The BUS plan was intricately aligned with key business processes. It defines how data is organized and interconnected based on critical business functions. The integration of water quality monitoring, compliance assessment, and pollution incident investigation processes ensures that the data warehouse serves as a strategic asset for decision-makers.

3. ETL Processes Documentation

3.1 Extraction

Data extraction was performed from the Department for Environment Food & Rural Affairs' online Data Service Platform, primarily in Microsoft Access format. SQL scripts were tailored to efficiently extract relevant data, considering factors such as data volume, frequency, and historical depth.

3.2 Transformation

The transformation phase was a critical step in ensuring data consistency and compatibility with the data warehouse schema. Python scripts were employed to handle complex transformations, such as unit standardization and missing value imputation. This stage also involved aligning data types and formats for seamless integration.

3.3 Loading

Loading data into the data warehouse was orchestrated through optimized SQL scripts. These scripts were designed to exploit parallel processing capabilities, ensuring swift and efficient population of the FACT and DIMENSION tables.

4. Data Cleansing Plan

4.1 Identify Errors

A comprehensive data cleansing plan was executed to identify errors, including missing primary keys, misspellings in location names, and discrepancies in sensor types. This process was crucial for ensuring data accuracy and reliability.

4.2 Cleansing Techniques

SQL scripts were instrumental in executing cleansing techniques. For instance, missing values were imputed using statistically sound methods, and naming conventions were standardized to eliminate inconsistencies.

5. PL/SQL Code Listings

PL/SQL code listings were meticulously crafted to define the underlying logic of the data warehouse. This included the creation of tables, views, and stored procedures. The code prioritizes efficiency, data integrity, and ease of maintenance.

6. SQL Scripts for Queries

A suite of SQL scripts was developed to empower users with meaningful insights. These scripts ranged from basic data retrieval queries to complex analytics, addressing specific business needs such as sensor analysis, location-based metrics, and longitudinal studies of key parameters.

7. Data Warehouse BUS Plan

The BUS plan serves as the backbone, aligning the data warehouse with the overarching business strategy. It ensures that each component of the data warehouse contributes meaningfully to the organization's goals, whether they pertain to regulatory compliance, environmental research, or incident response.

8. Python Code for Oracle Connection

Python played a pivotal role in bridging the gap between the data warehouse and external systems. Code snippets were crafted for establishing a secure and efficient connection to the Oracle database. Additionally, Python scripts were employed for pre-processing data, ensuring that the data fed into the data warehouse was clean and ready for analysis.

9. Screenshots

Screenshots were captured to visually represent key forms, reports, and GUIs developed for user interaction. These visuals provide a tangible glimpse into the user experience and the richness of data presentation.

10. Discussion of Problems and Solutions

The implementation journey was not without challenges. Diverse data sources posed integration challenges, and data quality concerns required robust solutions. The implementation team navigated these hurdles by refining ETL processes, enhancing data validation mechanisms, and collaborating closely with stakeholders to understand evolving requirements

1. Extended TIME Table

To create and populate an extended TIME table, consider the following SQL statements:

```
-Create Extended TIME Table
CREATE TABLE Extended_Time_Dim AS
SELECT DISTINCT
   TO_DATE('01-JAN-2000', 'DD-MON-YYYY') + LEVEL - 1 AS Date_Value
FROM dual
CONNECT BY LEVEL <= (365 * 20); -- Assuming 20 years

 Update Existing TIME Table with Extended Data
INSERT INTO Time_Dim (Date_Key)
SELECT TO_CHAR(Date_Value, 'YYYYMMDD') AS Date_Key
FROM Extended_Time_Dim;

 Commit the Changes
COMMIT;
```

This script creates an extended TIME table with a date range spanning 20 years, assuming a starting point of January 1, 2000.

2. Create Materialized View for Average Sensor Value

```
CREATE MATERIALIZED VIEW mv_AvgSensorValue
BUILD IMMEDIATE
REFRESH COMPLETE
START WITH SYSDATE
NEXT TRUNC(SYSDATE + 2, 'YYYY') + 1
AS
SELECT
   TO_CHAR(TRUNC(sampleDateTime, 'YYYY'), 'YYYY') AS Year,
   AVG(sensorValue) AS AvgSensorValue
FROM
   WaterQuality_Fact
GROUP BY
   TO_CHAR(TRUNC(sampleDateTime, 'YYYY'), 'YYYY');
```

This materialized view calculates the average sensor value for each year and refreshes every two years.

3. SQL Script for Dumping Data to Flat File and Using SQL*Loader

Assuming you want to export the data from the water quality table to a flat file and then use SQL*Loader to populate the data warehouse water quality table:

```
Export Data to Flat File
SPOOL water_quality_data.txt
SELECT * FROM WaterQuality_Fact;
SPOOL OFF;

-Create Control File for SQL*Loader (e.g., water_quality.ctl)
OPTIONS (SKIP=1)
LOAD DATA
INFILE 'water_quality_data.txt'
APPEND INTO TABLE WaterQuality_Fact
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
   sample_id,
   sampleDateTime,
   sensor_id,
   sensorValue,
   -- Add other columns as needed
)

sqlldr username/password@your_database control=water_quality.ctl
```

Make sure to customize the script based on the actual structure of your tables and the desired file format.

4. Create Table Statement for Partitioning the Fact Table by Year

```
sampleDateTime is the date column in WaterQuality_Fact
CREATE TABLE WaterQuality_Fact_Partitioned
PARTITION BY RANGE (EXTRACT(YEAR FROM sampleDateTime))
INTERVAL (NUMTOYMINTERVAL(1, 'YEAR'))
(
   PARTITION p0 VALUES LESS THAN (2000),
   PARTITION p1 VALUES LESS THAN (2001),
   PARTITION p2 VALUES LESS THAN (2002),
   -- Add more partitions as needed
   PARTITION p_max VALUES LESS THAN (MAXVALUE)
);
```

This script creates a partitioned version of the fact table, partitioned by year.

5. Tool for Automating the Cleansing Exercise

Creating a tool for automating the cleansing exercise involves developing a script or program that incorporates data validation and cleansing logic. Python or a similar scripting language can be used for this purpose. The tool would typically:

Connect to the database.
Identify and rectify errors based on predefined cleansing rules.
Execute SQL statements to cleanse the data.

The specific implementation details would depend on the cleansing requirements and the tools/technologies preferred in your environment.

Reference:

Bibliography

1. Inmon WH, Lindstedt D. Data architecture : a primer for the data scientist : big data, data warehouse and data vault. Waltham, MA: Morgan Kaufmann; 2015.

2. Viktor Mayer-Schönberger, Cukier K. Big data : a revolution that will transform how we live, work, and think. Boston: Houghton Mifflin Harcourt; 2013.

3. Singh H. Interactive data warehousing. Upper Saddle River, Nj: Prentice Hall Ptr; 1999.

4. Inmon WH, Lindstedt D. Data architecture : a primer for the data scientist : big data, data warehouse and data vault. Waltham, MA: Morgan Kaufmann; 2015.

5. Viktor Mayer-Schönberger, Cukier K. Big data : a revolution that will transform how we live, work, and think. Boston: Houghton Mifflin Harcourt; 2013.

6. Singh H. Interactive data warehousing. Upper Saddle River, Nj: Prentice Hall Ptr; 1999.