The British College
KATHMANDU

Coursework Submission Coversheet
(individual coursework only)

Faculty of Arts, Environment and Technology

LBU Student Id: 77275366

FOR CHECKING BY THE STUDENT:

Please ensure all information is complete and correct and attach this form securely to the front of your work before posting it in a coursework collection box.

Award name: Bsc Hons in Computing

Module code: 36493

Module name: Advance Database System (ADS)

Module run: 2021

Coursework title: Data warehouse Design and Development

Due Date: 7th July 2021

Module leader: (In LBU): Jackie Campbell, Sanela Lazarevski

Module tutor: (In TBC): Dibya Tara Shakya

TURNITIN Checked: YES (NO) ***(please circle)***

Submission date& time: Date: June 6 2021 Time: Before noon

Total Word Count: 10213 including this sheet **Total Number of Pages (including this front sheet):** 93

In submitting this form with your assignment, you make the following declaration:
I declare, that the coursework submitted is my own work and has not (either in whole or part) been submitted towards the award of any other qualification either at LBU or elsewhere. I have fully attributed/referenced all sources of information used during the completion of my assignment, and I am aware that failure to do so constitutes an assessment offence.

Signed: Sundar Tamang Date: June 6 2021

You are strongly advised to retain a second copy of your work in case of any query about the assignment.

FOR COMPLETION BY THE FACULTY:

☐ **This mark is provisional and subject to moderation and approval by the relevant examining board**

Teacher's Feedback

Teacher's Signature: _____ **Date: _____**

Contents

Introduction

No company can survive without a large and accurate store of historical data, ranging from sales inventory to intellectual and personal property record. In the modern days effective and efficient data mining plays a vital role in business, and when it comes to data mining data warehouse is the core part of it. Data warehouse has the ability to look back at early trends and which have the accurate and properly formatted data and trend, Data warehouse play significant role for any company to make a better business decision. Let's say if company suddenly needs to know the sales last 24 then rich historical data provided by the data warehouse can make this possible, not only this data warehouse can also add context to this histogram by listing all the key performance and trends that surround retrospective research. This kind of effective and easy overview can no matched by a legacy database. By enabling the historical overview data warehouse permit company to learn form the past trends and challenges. In essence, the benefit of data warehouse is continuous improvement. Data warehouse can also scale with business any growing company needs more and better data, and for this data warehouse can grow in more robust because of the ability of handling queries. The data are store in cloud or other third party by ensuring data security by using encryption and using specific protection set up.

The main benefit so having data warehouse in company is that it offers a weighty increase in competitive approach by enabling metric based, smarter decision on everything ranging from product release to the inventory level to main sales levels. It almost seems impossible for any business to compete in today's market without the advanced of data warehouse.

Data warehouse

Data warehouse is the data management system, which is specially designed to support and enable the business intelligence (BI) activities, especially analytics. It the process of collecting managing data from different sources to give meaningful business insights. It contains a large amount of data which are derived from a wide range of sources like transaction applications, application log etc. By using this huge amount of data, data analyst can analyze it, which allow organization to make right and valuable decision which improve their decision making. Data warehouse give result base on the data so it is considering as the organization 'single source of truth' (Anon., n.d.).

Designing the data warehouse is essential part of the business development, it makes organization to make better business decision. Once the data ware house is implemented on your organization it can

benefit company in numerous ways. Given below are the some of the notable benefits of data warehouse.

Advantages of data warehouse

- Improve the decision-making process
- Enable organizations to forecast with confidence
- It can enhance the business intelligence
- Enhanced data quality and consistency
- Generate high return on investment
- Business intelligence from multiple sources
- Increased Query and system performance.
- Data warehouse help to get holistic view of current standing and evaluate opportunities and risk.
- Data warehouse simplifies the flow of information through network by connecting all the related and no-related parties.
- Data warehouse keep all the data in one place and save user's time to access the particular set of data, which can make rapid decision on enterprises actions.

Disadvantages of data warehouse

- Bigger the organization more the data, and the extra time to load data warehouse run. Data which are generated by data warehouse require the participant of each department in the organization which bother with extra report work.
- Sometimes the standardization and similarity in the data formats lead to inflexibility and homogenization of data. Which further limit the data in the terms of establishing relation during aggregation and difficult to tune for query speed, and homogenization also cause loss of data.
- Sometimes centralizing data at once place cause lead to different as they hesitate to share their personal data within central repository. Which also raises ownership and security concern for few departments, so organization must sure that analysis of data is given to trusted individuals within the enterprise.
- Sometimes internal source of fueling data warehouse keep pushing problems which are undetected for years.

Characteristics of data warehouse

Integration

Integration refers to the shared of entity to scale the all similar data from various database. Data warehouse is built by just integrating data from multiple sources of data. Moreover, it must have the reliable formats and codes, naming conventions. This integration of data warehouse benefits in the analyzation of data. Reliability of column scaling, encoding structure should be confirmed. This integration of data warehouse handles numerous subject related warehouses.

Subject oriented

Data warehouse is process and it is proposed to handle with specific theme, and this theme can be sales, marketing, distribution etc. Data warehouse focus to demonstrate and analyze of data to make the better decision. It also distributes easy a precise demonstration around the special theme just by eliminating data which is not required to make the decisions.

Time variant

Data is maintained in different time period such as weekly, monthly or annually. Data which are resided in the data warehouse is predictable with the particular time interval and delivers information from historic perspective. Another time variance is that once the data is store in data warehouse can not be modified, alter or updated.

Non-volatile

Data which are stored in data warehouse is permanent. Which means data can not be erased or deleted when new data is inserted.

Data warehouse development methodologies

To design the data warehouse there are two methodologies they are Kimball and Inmon theory. Bill Inmon and Ralph Kimball sated different philosophies, for information collecting, information management, and also for the analytics for the decision support. Both theory approach problem from different viewpoints, techniques of designs and implementation of strategies. Both methodologies have their own advantages and disadvantages, based on our requirements we can use either Kimball

or Inmon to develop the data warehouse, the more important is which one serves user at low redundancy. At first let's discuss about the bill Inmon's and Kimball theory then we will select the best one for our scenario.

Bill Inman's theory

This theory was introduced by the Bill Inmon. Inmon first start with centralized enterprise-wide data warehouse just by the multiple databases to the analytical needs of departments, which are later known by the data marts. Because of this reason Bill Inmon is consider as the top-down approach. The main core of Bill Inmon theory is enterprise data warehouse. The main central repository of the data combined from all the operations system of organization. This theory is the definitive of all the representation of business data, which means all the organizations choose the naming definitions of which data true and their conflicting values and all the other data cleaning operations are done before enters the warehouse. In this model data is stored in normalization form and also warehouse is not directly created, but data is fed into various data marts which are then filtered down to the subsets of particular needs. For example, sales department will have the data only which are used by the sales team. Applications which re retrieve the data will be joined to data marts. The main advantages of having these model is that it is in normalization form so it makes less data redundancy (Vidhya, n.d.).

Disadvantages of Bill Inman's theory

- Implementation of the model can become complex over the time as it involves more table and joins query.
- As the resource required experts of data modeling which is hard to find and often expensive.
- More ETL is require as the data marts are built from the data warehouse.
- Initial set-up and delivery will take more time
- Specialist team are required to successfully manage the environment.

Advantages of Bill Inman's theory

- In Inman's theory it serves as the single source of truth for the enterprises, data marts and for all the data in Datawarehouse is integrated.
- Data update anomalies are avoided because of low redundancy, which make ETL process easier and less to failure.

- In this theory business procedures can be understood easily.
- When the business requirements change, it is easy to update data warehouse.
- It can handle wide range of reporting needs across the enterprise.

Ralph Kimball

This model is introduced by the Ralph Kimball. This model starts with identifying the business process and questions that data warehouse to answer. Then these sets of information are being analyzed and then documented well. Data is collected using ETL process from multiple data sources called data marts and then it is loaded into common area called staging, then it is transformed into OLAP cube. This approach is also known by the bottom-up approach. This idea is when the data is being in normalized into star schema. This approach makes query writing fast and simple, and also get report very quickly. At first data from particular area are process into a star schema, and that Kimball data marts are connected with shared attributes and that form a dimensional data warehouse. The main advantages of having Kimball model is that we can get report very quickly in comparison to the Inmon and writing query is also very simple and fast (Vidhya, n.d.).

Advantages of Ralph Kimball's theory

- The first set up phase of data warehouse is easy and will be delivered quickly.
- Star schema of this theory can be understood easily by business users and easy to use for the reporting.
- It occupies less space in the database which make management system fairly easier.
- Even small team of developers is enough to keep the data warehouse performing effectively.
- It really works well for the department-wise metrics and KPI tracking, as the marts are geared towards department-wise reporting.
- Database engine will perform 'star join; where cartesian product will create and the fact table will be queried for the selective rows.

Disadvantages of Ralph Kimball's theory

- Redundancy of data can cause data anomalies over the time.
- Adding column in fact table can cause performance issues, this is because fact tables are created in very deep, so if new columns are added the size of the table becomes much larger

and will no perform well. Which make dimensional model hard because business requirements change.

- Integration of data in data warehouse can be complex procedures.
- It cannot handle all the enterprises because reporting needs because model is oriented towards business rather than enterprises as whole.

Key difference between Bill Inman's and Ralph Kimball

Ralph Kimball	**Bill Inmon**
Relatively it takes less time to implement	It is quite complex and take more time to implement
This base theory is difficult to maintain	It is easy to maintain in comparison to Inman theory
It cost low in the initial phase because we only need data warehouse and cost remain same for the remaining phase.	Implementing this approach can cost high at initial phase, but the subsequent project development cost low.
Kimball based data architecture can be set up quickly in comparison to Bill Inman approach.	This theory base data warehouse set up require bit more time.
Kimball approach data warehouse require generalist team to implement.	To apply this based for the data warehouse we need a specialist team.
Kimball base data warehouse integration focused on the individual business area.	Inman based data warehouse integration nee enterprise wide data.
Kimball fallow bottom approaches	Inman fallow top-bottom approach

OLTP and OLAP

OLTP

OLTP stands for online transaction processing system. OLTP record data in row and column form, the main focus of OLTP is to record, update and delete the data while transaction. In OLTP queries are short and simpler so it takes less time for processing and it also require a less space. The data of OLTP get update frequently. The example of OLTP data are web application, mobile application, ATM machine where short transactions modify status of our account. OLTP database store data in normalization form (3NF), these OLTP system become source of OLAP (techdifferences, n.d.).

Benefits of using OLTP database system

- Better for daily transactions of an organization.
- It can widen customer base of an organization just by simplifying individual process.

Drawbacks of using OLTP database system

- If it fallows hardware failure then online transactions get affect.
- OLTP permit multiple users to access and modify data at the same time which can created unpredictable situations.

OLAP

OLAP stands for online analytical processing system. It is mainly use for analytical purpose. OLAP database stores historical data which are recorded in the OLTP. Using the OLAP system we can extract the information from a large amount of database and analyze it for decision making. If the transaction is failed in middle it will not harm data integrity as the user use OLAP system to retrieve from large database. User can fire the query again and extract the data for analyzing purpose. In OLAP system transaction are long so it takes more time for processing and requires more space in comparison to OLTP. The table in OLAP database are not normalized, also the data of OLAP system are never deleted. The example of OLAP database is viewing report of financial, budgeting, sales, marketing etc. (techdifferences, n.d.).

Benefits of using OLAP database system

- It can create single platform for all types of business analytical needs, that include budgeting, planning, forecasting and analysis.
- We can easily apply security restrictions over user and apply regulations to protect the sensitive data.
- It can help is consistency of calculations an information

Drawbacks of using OLAP database system

- In OLAP database system implementation and maintenance are dependent on the IT professional, the reason behind this is traditional OLAP need complicated modeling procedures.
- In OLAP, we need co-operation between employee of different department to be effective which might not always possible

Difference between OLAP and OLTP system

Parameters	**OLTP**	**OLAP**
Table	Table in OLTP are in normalized form	Table in OLAP database are not in normalized
Source	OLTP and regular transaction are the source of OLTP	OLTP databases are the source of OLAP database
Query	Inset, update and delete are the normal query of OLTP	Mostly select operations
Method	OLTP use mostly traditional DBMS	OLAP uses data warehouse
Functionality	This is online database modifying system	OLAP is online Databse query management system.
Process	Generally online transactional system. It manages database modification.	OLAP is online analysis retrieving data system.

Characteristic	It is characterized by large number of short online transactions.	It is characterized by large volume of data
Uses	Help to control and run basic needed tasks	Help with planning, problem solving and to make the decision.
Operation	Allow read/write operation	Only read and rarely write
Purpose	Specially designed for real time business operations	Designed for analysis purpose of business measure by category and attributes.

Star Schema

Star schema is one of the most important and simplest schemas. This schema is widely used to develop and build data warehouse and dimensional data marts. Star schema consist of one or more than one fact tables indexing any number of dimensional tables. This schema is necessary for the snowflake schema and it is also efficient to handle basic queries. Star schema form a star shape having fact table at the center surrounded by the dimensional table.

The purpose of star schema is to separate numerical "fact" data form the descriptive or 'dimensional" data. The fact table will have only numerical data like weight, price, speed, quantity, i.e., data which are in numerical format. Dimension table will have data like model, color, names, location employees name, salesperson name etc. i.e., descriptive data along with numerical information.

Fact data is organized into the fact table, and dimensional data is into dimension tables. Fact tables are the center of star schema in data warehouse. They allow to analyze the data as a single unit, and allow business systems to access the data together. Dim table manage and hold the numerical and non-numerical data which coverage through fact tables that make up the data warehouse. Fact table keep the track of numerical information related like they might include numeric values, foreign key that map additional dim tables. Dimension table normally list a surrogate primary key which maps to attributes related to the natural key. Let's say we have dim table with the information related to store "dim_store". You can allocate an ID to each store and its row related non-numerical and other information like size, name, location, category etc. Wherever we list the store ID number on fact table "fact_sales" will map to that specific row of store data on the "dim_store" dim table (Smallcombe, 2020).

Characteristics of star schema

- Each dimension in star schema represent with only one-dimension table.
- Dim tables are connected with fact table using the foreign key
- Dim tables are not connected with each other
- Fact table would contain key and numeric value
- Star schema is easy to understand
- Star schema is supported by BI tools.
- Dim table are not normalized in star schema

Benefits of star schema

- In this model all the data are connect through the fact table and multiple dimension tables are considered as the large table of information which make query simple and easy to perform
- It is easy to pull the busines reports in star schema
- It removes the bottlenecks of highly normalized schema, increase the query speed, and improve the performance of read only command.
- OLPA system can use star schemas to build OLAP cubes.

Challenges of star schema

- Because of denormalized form of star schema, it does not enforce in data integrity, though star schema use countermeasures to stop from anomalies, a simple update and inset can still cause data incongruities.
- Database designers shape and optimize star schemas for particular analytical needs. Because of deformalized data sets, they work with comparatively narrow set of simple queries. Relatively normalized schema allows a far wider variety of more complex analytical queries.

Snowflake Schema

In general snowflake schema is an extension of star schema, the difference is that the dim table in snowflake schema is divided into more table, which create a snowflake pattern. Which means dimension tables are normalized which one dim table into additional dim tables. The main purpose of snowflake schema is to normalize the denormalized data in a star schema. Snowflake schema is multi-dimensional structure. Through this snowflake schema, we normalize dim tables by splitting tables into more tables until that dimension tables are completely normalized. Unlike in the star schema dimension tables of snowflake schema can of their own categories, the main idea behind this

is to normalized dimension tables. Each dim tables can be described by one or more lookup tables. Each lookup tables are separated until the model is fully normalized. This process of normalizing star schema's dimension tables is called the snowflaking (Smallcombe, 2020).

Characteristics of snowflake schema

- Snowflake schema make smaller disk space.
- Easier to implement a dimension is added to the schema
- Because of multiple tables performance of query is reduced.
- The main challenge of snowflake schema is that you need to perform relatively more maintenance efforts because of more lookup tables.

Benefits of snowflake schema

- There are some certain OLAP database tools which are used by data scientist for data analyzing and modeling which are particularly designed to work with snowflake.
- Normalize data normally get denormalized in star schema can reduce the disk space requirements, this is because you are converting long strings of non-numerical data to numerical keys which are vividly less taxing from a storage perspective.

Challenges of snowflake schema

- Snowflake schemas create complexity while normalizing the attributes of a star schema. This complexity requires more complicated join query. In offering a more effective way to store data, snowflake can result in performance declines while browsing complex joins.
- Because of complex join in snowflake schema it may result in slower data processing. The star schema is normally better for cube data processing.
- Snowflake schemas provide greater normalization and low risk of data corruption after insert and update commands, but they do not provide the level of transnational guarantee that offer by traditional and highly normalized database structure. So, when we have to load the data in snowflake, we should be very careful about the quality of information.

Data dictionary

In general data dictionary refers to the collection of definitions, names, and attributes for data models and elements. Data dictionary define purpose and meaning of data elements within the context of a project and offer guidance on interpretation, representation and accepted meaning. It also provides the metadata about the data elements. Metadata of data dictionary can assist in defining the characteristics and scope of data elements along with the rules for their usage and application. Data is very important for many reasons like it prove assist in providing data inconsistencies across a project. It can make a data easier to analyze, not only this it can enforce the use of data standards. The main reason companies are using data dictionaries is to document and share data structure and some other information for all involved all database. Using share dictionary can ensure meaning, quality, and relevance for all data elements for all team members. Without using the data dictionary there is also high risk of losing crucial information in transition (ucmerced, n.d.).

Dim_Flight table data dictionary

Data source	Field name	Data type	Key	Data quality check	Data quality issue	Action note
Flight_2017	**Flight_id**	integer	yes	consistency	N/A	-
Flight_2018		integer	yes	consistency	N/A	-
Flyu_flight						-
Definition	It is unique number					
Note	Flight_id exist in flight_2017, flight_2018 but not is flyu_Flgihts					
Flight_2017	**Flight_number**	varchar	No	consistency	N/A	-
Flight_2018		varchar	No	consistency	N/A	-
Flyu_flight		varcahr	No	consistency	N/A	-
Definition	Flight_number is flight number					
Note	Flight_number consist in all data sources					
Flight_2017	**Cancelled**	Number	No	consistency	N/A	-
Flight_2018		number	No	consistency	N/A	-
Flyu_flight		Number	No	consistency		-
Definition	This consist flight that has been cancelled					

Note	Cancelled column consist in all the table					
Flyu_flight	**diverted**	Number	No	Inconsistency	There is record of diverted when flight is cancelled	Change with appropriate value
Flight_2017		Number	No	Consistency	N/A	-
Flight_2018		Number	No	Consistency	N/A	-
Definition	This consist the flight which has been diverted					
Note	Diverted consist tin all table					
Flight_2017	**Origin airport**	varchar	No	consistency	N/A	
Flight_2018		varchar	No	Inconsistency	Contain non alphabet value	Update with appropriate value
Flyu_flight		varchar	No	Inconsistency	Contain non alphabet value	Update with appropriate value
Definition	Name of the origin airport					
Note	All the data source has origin airports name					
Flight_2017	**destination_airport**	varchar	No	consistency	N/A	-
Flight_2018		Varchar	No	consistency	N/A	-
Flyu_flights		Varchar	No	consistency	N/A	-
Definition	Name of the destination airport					
Note	All the data source has destination airports name					

Dim_complaint table data dictionary

Data source	Field name	Data type	Key	Data quality check	Data quality issue	Action note
Flyu_flights	**Complaint_id**	Number	Yes	consistency	N/A	-
Definition	It contains the id of dim complaint table					
Note	Only one source flyu_flights					
Flyu_flights	**Complaint_type**	Varchar	No	consistency	N/A	-
Definition	It contains the customer type					
Note	Only one source flyu_flights					
Flyu_flights	**description**	Varchar	No	In-consistency	Some of the description are empty	Update with appropriate value
Definition	It consists the description of flight					
Note	Only one source flyu_flights					
Flyu_flights	**Complain_status**	Varchar	No	consistency	N/A	-
Definition	It consists complain status of flight					
Note	Only one source flyu_flights					
Flyu_flights	**Compensation_amount**	Number	No	in consistency	Compensation amount is not in number	Update with appropriate value
Definition	It consists the compensation amount					
Note	Only one source flyu_flights					
Flyu_flights	**Allocated_to**	Allocated_to	No	consistency	N/A	-
Definition	It consists the description of flight					
Note	Only one source flyu_flights					

Dim_customers

Data source	Field name	Data type	Key	Data quality check	Data quality issue	Action note
Flyu_flights	**Customer_id**	Number	Yes	consistency	N/A	-
Definition	It contains the primary id of sutomer					
Note	Only one source flyu_flights					
Flyu_flights	**Customer_type**	Varchar	No	consistency	N/A	-
Definition	It contains the customer type					
Note	Only one source flyu_flights					
Flyu_flights	**description**	Varchar	No	consistency	N/A	-
Definition	It contains the description of customer type					
Note	Only one source flyu_flights					
Flyu_flights	**Customer_zip_code**	Varchar	No	consistency	N/A	-
Definition	It contains the zip code of customer					
Note	Only one source flyu_flights					
Flyu_flights	**Customer_miles**	Number	No	in consistency	Compensation amount is not in number	Update with appropriate value
Definition	It contains the distance of customer miles					
Note	Only one source flyu_flights					

Dim_time

Data source	Field name	Data type	Key	Data quality check	Data quality issue	Action note
Flyu_flights	**Time_id**	Number	Yes	consistency	No data quality issue	-
Definition	It contains the primary key of time id, wihc is automatically generate as a surrogate key					
Note	Only one source flyu_flights					
Flyu_flights, Flight_2017, Flight_2018	**Year**	Number	No	consistency	No data quality issue	-
Definition	It consists the year of all data source					
Note	Year consist in all tables					
Flight_2017,	**Month**	Number	No	consistency	No month data	Update with appropriate data
Flight_2018		Number	No	In-consistency	No data quality issue	-
Flyu_flights		Number	No	In-consistency	No data quality issue	-
Definition						
Note	Day consist in flight_2018 and flyu_flights but not in flight_2017					

Flight_2017,	**Day**	Number	No	In-consistency	No day data	Update with appropriate data
Flight_2018		Number	No	In-consistency	No data quality issue	-
Flyu_flights		Number	No	In-consistency	No data quality issue	-
Definition	It consists the day of all data source					
Note	Only one source flyu_flights					

ETL

ETL stands for Extract, Transform and Load. It is the process in which ETL tool extracts the data from numerous data source systems, transforms data in staging area and then finally load it into the Data warehouse system. It is tempting to think create data warehouse is just simply to extract data from multiple source and load data in the data warehouse, but far from the truth it requires a complex ETL procedure, the ETL procedures require dynamic input from multiple stakeholders including testers, analysts, developers, top executives and is technically challenging. To maintain the value as tool for decision-making, Data warehouse system need to change with business changes. ETL (extraction, tranform, load) is recurrent process like weekly, monthly for a data warehouse and it needed to be automated, agile and should be well documented. Let us understand each of the ETL process in depth.

1. **Extraction**

   This is the fist step of ETL. This is the extraction process, in this step data from multiple source which can be in various formats like XML, SQL, NO SQL, relational databases, into the staging area. It is necessary to extract the data from various source and store it into the stagging are first, not directly into the data warehouse because extracted data is in multiple

formats and it can be corrupted too. So, loading the data directly into the data warehouse may cause problem and it may be very difficult to rollback. Therefore, this is one of the most important steps in ETL procedures.

2. **Transform**

   This transformation process is second step of ETL process. In this stage, some rules or functions are applied on the extracted data to convert it into a single standard format. This transformation may involve following process.

   - Filtering
   - Cleaning
   - Joining
   - Splitting
   - Sorting

3. **Load**

   Third and final step of ETL procedures is loading. In this step transformed data is finally loaded into the data warehouse occasionally data is updated by loading into data warehouse very often and sometimes it is done after longer but regular intervals. Load time is depending upon the requirements and varies from system to system.

Project overview

Designing the data warehouse is essential part of the business development, it makes organization to make better business decision and its future insights. As flyU being the airlines company data warehouse is required to properly manage the data, to make better decisions based on historical data. It keeps the records of flight that leave each airport. Record of departure arrival times, along with actual arrival and departure time from and to each airport. They also kept the record of passenger on each flight and raise complaint against flights if they have any issues. Using all of this data company can plan for the future, make better decision and they can also track the business ongoing progresses. So here the flyu airlines wan to use data warehouse and they want to mainly focus on given below main factors.

- Deliver quality service
- Ensure customer satisfaction
- Grow the company

Here my role is an analyst/developer on data mart project to support the design analysis and collection of information relating of FlyU, I have to concern with company grow. To achieve this requirement, I have made some aim and objectives of the project, which are given below.

Project aim and objectives

As being the data analyst of flyu airlines company my main aim and objective of the project is to analyze the data so that company can make better decision to grow the companies. As per the requirement of company i have to a design and developed the data warehouse so that company can grow their business. Here are some of the project aim and objectives.

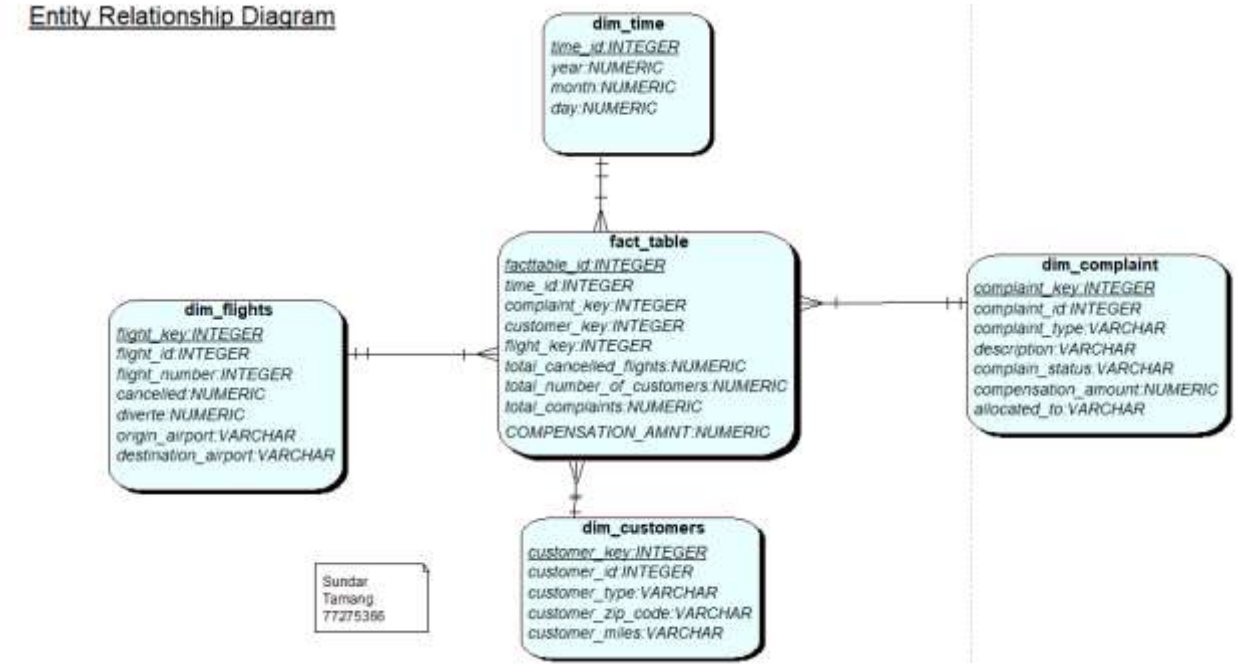
- Design and developed data warehouse from the existing data of flyu flights.
- Carefully analyze the issues and give better decision options to the company
- Keep and track the company progress in well structure.
- Listing and recording customer feedback and complaint.
- Predict the insight future of company using historical data.
- Design and developed report that could support company to make better decision.

To grow the business, we can make sure that customer complaints are addressed, so that we provide quality service to them and which automatically increase the customer number. To support more of this, we have made five reports which are given below.

Reports to be supported by star schema

1. Number of customers per year
2. Airlines with most complain
3. Total compensation amount
4. Number of cancelled flights in each destination
5. Number of flights that are diverted each month

Base on the above reports, if we developed the diagram base on star schema then it will look like this.



Base on the above reports, if we developed the diagram base on snowflake schema then it will look like this.

Considering the requirement of company and many other benefits like easy to pull the business report, increase query speed also improve the performance of read only command and also OLAP can use this schema easily I am going to use the star schema model for this project. Because star schema dim tables do not have another dim table which mean there will be not any complex join query, which make very fast to retrieve data from database. It allows BI tools to deeper across several star schema and which can generate reliable insights for the future of flyu flight to make effective decision and plan for future

Load data from sources

Upload flight 2017 data into oracle apex

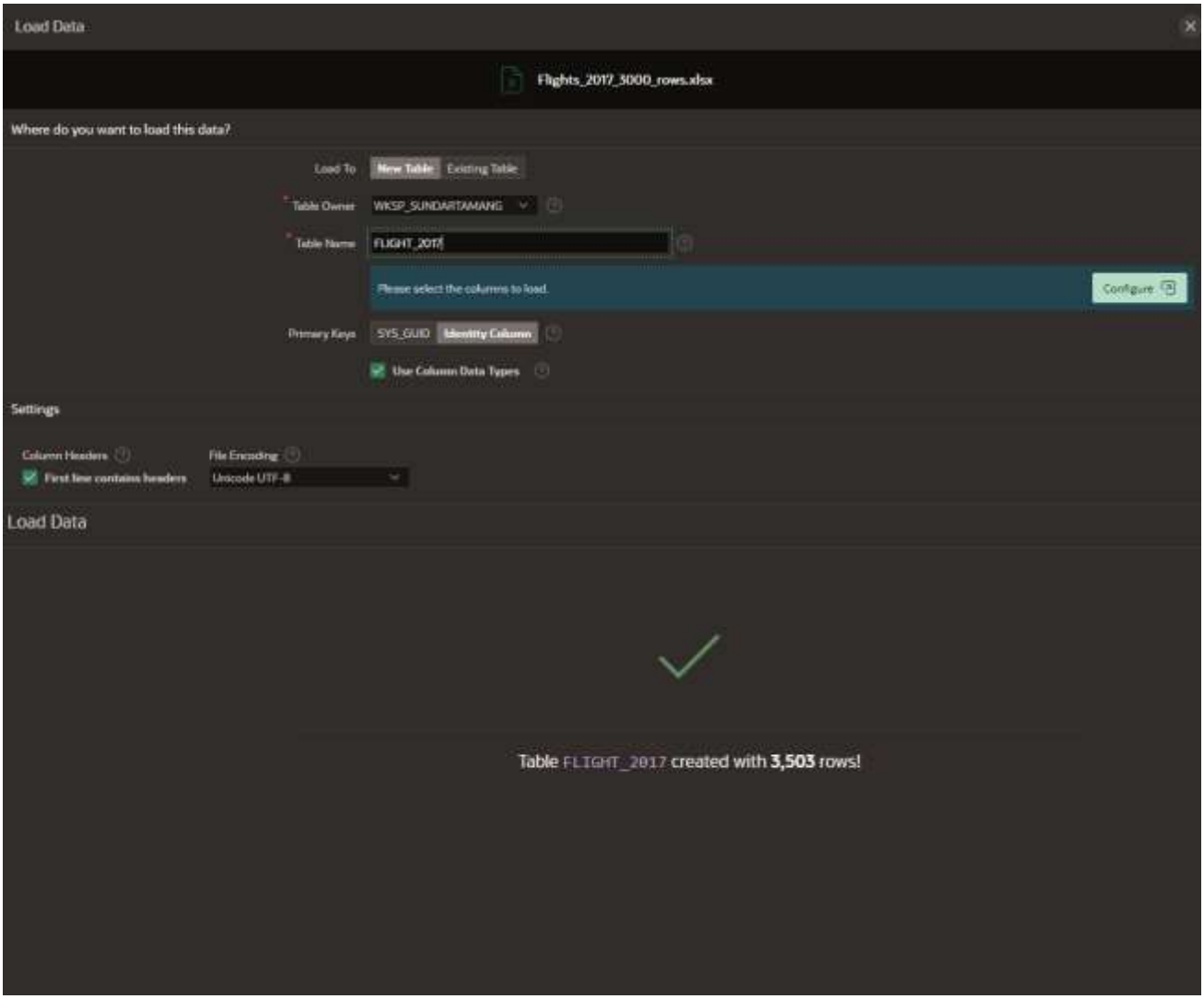


Fig: successfully uploaded flight 2017 data. Table name is flight_2017 in oracle apex.

Upload flight 2018 data into oracle apex

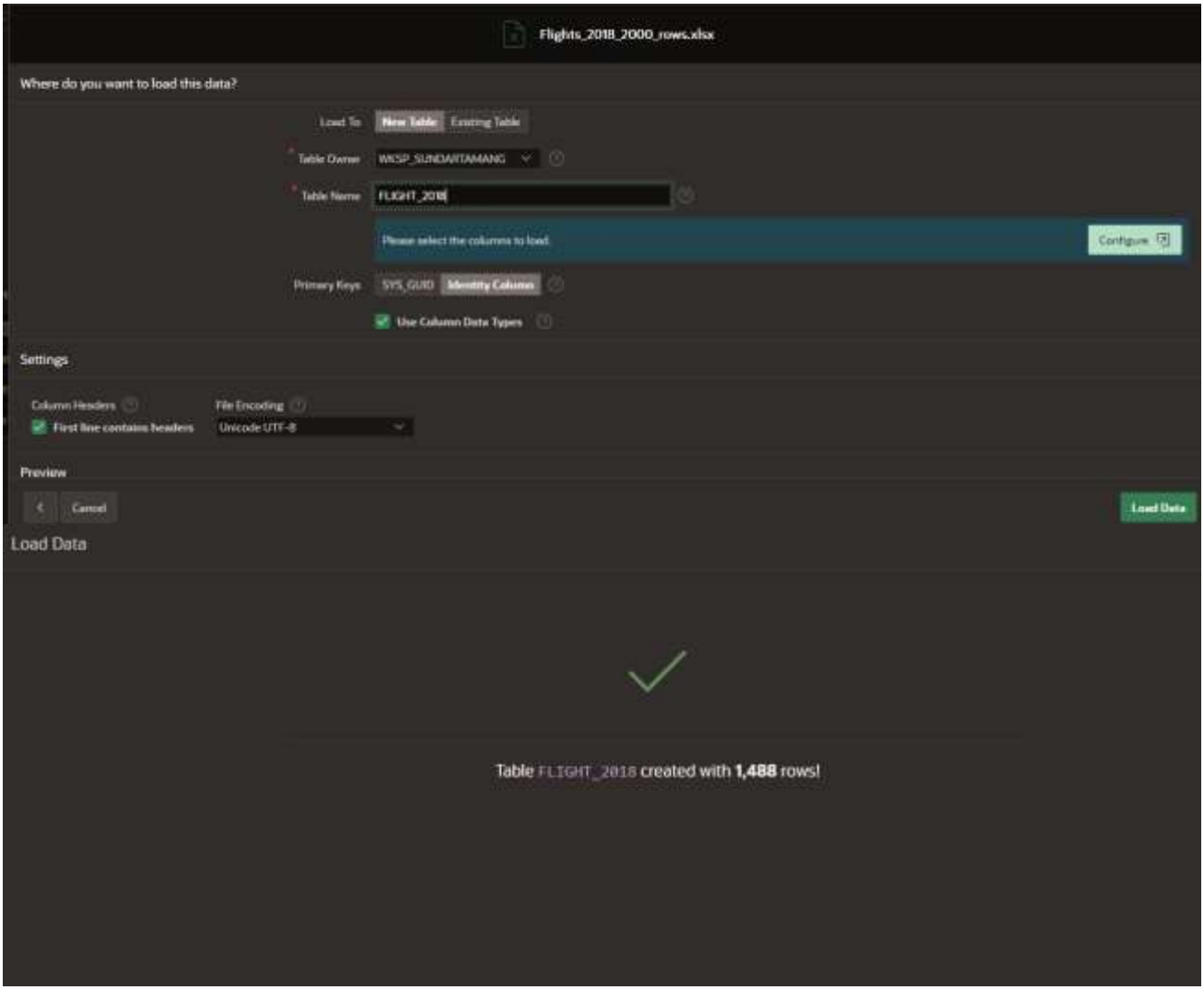


Fig: successfully uploaded flight 2018 data. Table name is flight_2018 in oracle apex.

Run flyu_flight 2020 sql script and insert data in table

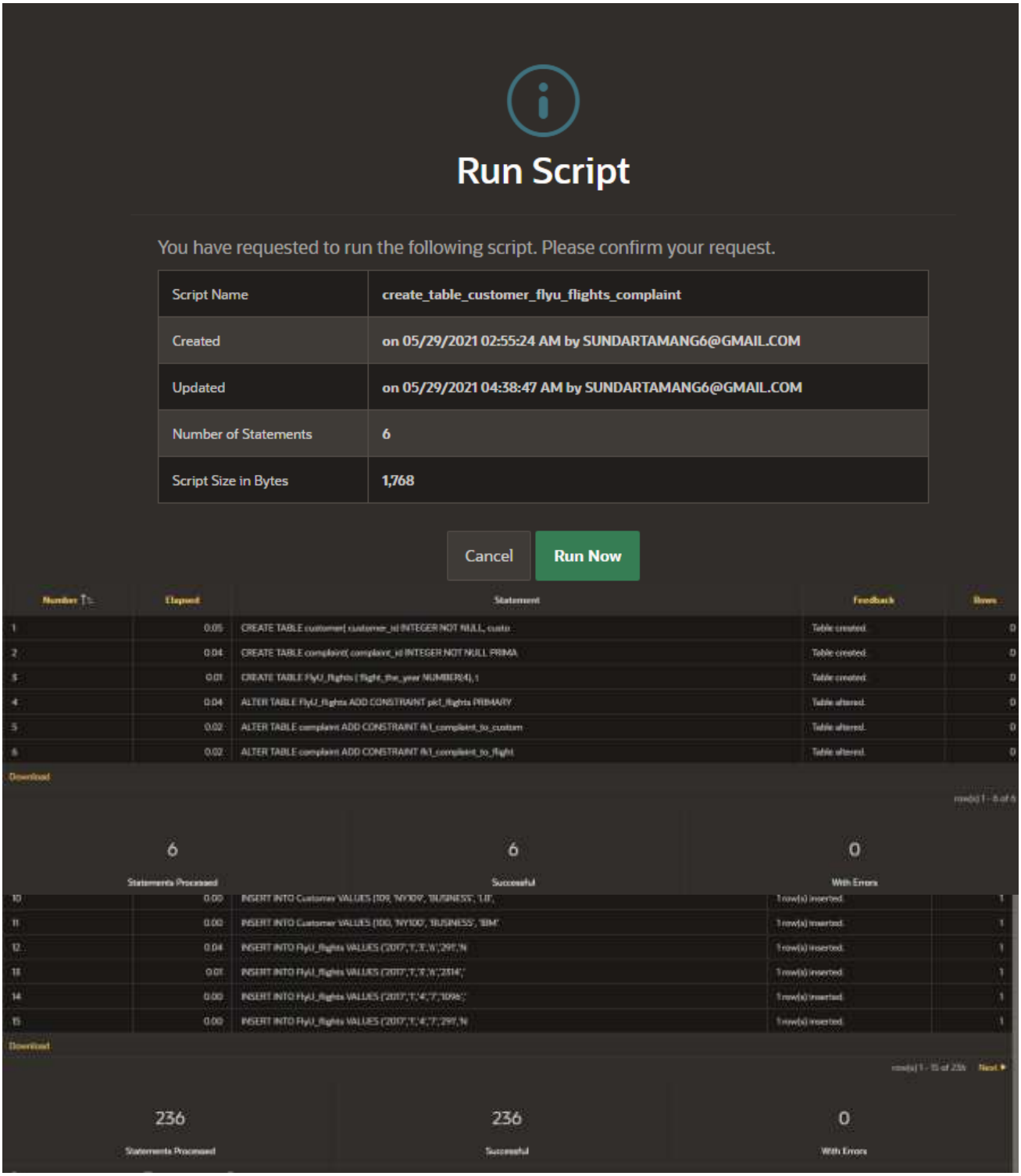


Fig: successfully uploaded data. Table name is flyu_flights, complaint and customer in oracle

apex.

SQL script generate by Qsee

```
-------------------------------------------------------------
-- Table Creation --

-- Each entity on the model is represented by a table that needs to be created within the Database.
-- Within SQL new tables are created using the CREATE TABLE command.
-- When a table is created its name and its attributes are defined.
-- The values of which are derived from those specified on the model.
-- Certain constraints are sometimes also specified, such as identification of primary keys.

-- Create a Database table to represent the "fact_table" entity.
CREATE TABLE fact_table(
        facttable_id    INTEGER NOT NULL,
        time_id INTEGER,
        complaint_id    INTEGER,
        customer_id     INTEGER,
        flight_id       INTEGER,
        total_cancelled_flights NUMERIC(8,2),
        total_number_of_customers       NUMERIC(8,2),
        total_complaints        NUMERIC(8,2),
        fk1_time_id     INTEGER NOT NULL,
        fk2_flight_id   INTEGER NOT NULL,
        fk3_complaint_id        INTEGER NOT NULL,
        fk4_customer_id INTEGER NOT NULL,
        -- Specify the PRIMARY KEY constraint for table "fact_table".
        -- This indicates which attribute(s) uniquely identify each row of data.
        CONSTRAINT      pk_fact_table PRIMARY KEY (facttable_id)
);

-- Create a Database table to represent the "dim_flights" entity.
CREATE TABLE dim_flights(
        flight_id       INTEGER NOT NULL,
        flight_number   INTEGER NOT NULL,
        cancelled       NUMERIC(8,2),
        diverte NUMERIC(8,2),
        origin_airport  VARCHAR(200),
        destination_airport     VARCHAR(200),
        -- Specify the PRIMARY KEY constraint for table "dim_flights".
        -- This indicates which attribute(s) uniquely identify each row of data.
        CONSTRAINT      pk_dim_flights PRIMARY KEY (flight_id)
);

-- Create a Database table to represent the "dim_time" entity.
CREATE TABLE dim_time(
        time_id INTEGER NOT NULL,
        year    DATE NOT NULL,
        month   DATE NOT NULL,
        day     DATE NOT NULL,
        -- Specify the PRIMARY KEY constraint for table "dim_time".
        -- This indicates which attribute(s) uniquely identify each row of data.
        CONSTRAINT      pk_dim_time PRIMARY KEY (time_id)
);

-- Create a Database table to represent the "dim_compalint" entity.
CREATE TABLE dim_compalint(
        complaint_id    INTEGER NOT NULL,
        complaint_type  VARCHAR(200),
        description     VARCHAR(200),
        complain_status VARCHAR(200) NOT NULL,
        compensation_amount     NUMERIC(8,2),
        allocated_to    VARCHAR(200),
        -- Specify the PRIMARY KEY constraint for table "dim_compalint".
        -- This indicates which attribute(s) uniquely identify each row of data.
        CONSTRAINT      pk_dim_compalint PRIMARY KEY (complaint_id)
);

-- Create a Database table to represent the "dim_customers" entity.
CREATE TABLE dim_customers(
        customer_id     INTEGER NOT NULL,
        customer_type   VARCHAR(200) NOT NULL,
        customer_zip_code       VARCHAR(200),
        customer_miles  VARCHAR(200),
        -- Specify the PRIMARY KEY constraint for table "dim_customers".
        -- This indicates which attribute(s) uniquely identify each row of data.
        CONSTRAINT      pk_dim_customers PRIMARY KEY (customer_id)
);
```

```
-- Alter table to add new constraints required to implement the "fact_table_dim_time" relationship

-- This constraint ensures that the foreign key of table "fact_table"
-- correctly references the primary key of table "dim_time"

ALTER TABLE fact_table ADD CONSTRAINT fk1_fact_table_to_dim_time FOREIGN KEY(fk1_time_id) REFERENCES dim_time(time_id) ON DELETE RESTRICT ON UPDATE RESTRICT;

-- Alter table to add new constraints required to implement the "fact_table_dim_flights" relationship

-- This constraint ensures that the foreign key of table "fact_table"
-- correctly references the primary key of table "dim_flights"

ALTER TABLE fact_table ADD CONSTRAINT fk2_fact_table_to_dim_flights FOREIGN KEY(fk2_flight_id) REFERENCES dim_flights(flight_id) ON DELETE RESTRICT ON UPDATE RESTRICT;

-- Alter table to add new constraints required to implement the "fact_table_dim_compalint" relationship

-- This constraint ensures that the foreign key of table "fact_table"
-- correctly references the primary key of table "dim_compalint"

ALTER TABLE fact_table ADD CONSTRAINT fk3_fact_table_to_dim_compalint FOREIGN KEY(fk3_complaint_id) REFERENCES dim_compalint(complaint_id) ON DELETE RESTRICT ON UPDATE

-- Alter table to add new constraints required to implement the "fact_table_dim_customers" relationship

-- This constraint ensures that the foreign key of table "fact_table"
-- correctly references the primary key of table "dim_customers"

ALTER TABLE fact_table ADD CONSTRAINT fk4_fact_table_to_dim_customers FOREIGN KEY(fk4_customer_id) REFERENCES dim_customers(customer_id) ON DELETE RESTRICT ON UPDATE
RESTRICT;
```

Fig: successfully generated SQL script from qsee

Create dim tables with related sequences and trigger

```
CREATE TABLE fact_table(
    facttable_id    INTEGER NOT NULL,
    time_id INTEGER,
    complaint_key   INTEGER,
    customer_key    INTEGER,
    flight_key  INTEGER,
    total_cancelled_flights NUMERIC(8,2),
    total_number_of_customers   NUMERIC(8,2),
    total_complaints    NUMERIC(8,2),
    COMPENSATION_AMNT   NUMERIC(8,2),
    -- Specify the PRIMARY KEY constraint for table "fact_table".
    -- This indicates which attribute(s) uniquely identify each row of data.
    CONSTRAINT  pk_fact_table PRIMARY KEY (facttable_id)
);

-- Create a Database table to represent the "dim_flights" entity.
CREATE TABLE dim_flights(
    flight_key  INTEGER NOT NULL,
    flight_id   INTEGER NOT NULL UNIQUE,
    flight_number   INTEGER NOT NULL,
    cancelled   NUMERIC(8,2),
    diverte NUMERIC(8,2),
    origin_airport  VARCHAR(200),
    destination_airport VARCHAR(200),
    -- Specify the PRIMARY KEY constraint for table "dim_flights".
    -- This indicates which attribute(s) uniquely identify each row of data.
    CONSTRAINT  pk_dim_flights PRIMARY KEY (flight_key)
);

-- Create a Database table to represent the "dim_time" entity.
CREATE TABLE dim_time(
    time_id INTEGER NOT NULL,
    year    NUMERIC(8,2) NOT NULL,
    month   NUMERIC(8,2) NOT NULL,
    day NUMERIC(8,2) NOT NULL,
    -- Specify the PRIMARY KEY constraint for table "dim_time".
    -- This indicates which attribute(s) uniquely identify each row of data.
    CONSTRAINT  pk_dim_time PRIMARY KEY (time_id)
);
```

```
-- Create a Database table to represent the "dim_complaint" entity.
CREATE TABLE dim_complaint(
    complaint_key   INTEGER NOT NULL,
    complaint_id    INTEGER NOT NULL UNIQUE,
    complaint_type  VARCHAR(200),
    description VARCHAR(200),
    complain_status VARCHAR(200) NOT NULL,
    compensation_amount NUMERIC(8,2),
    allocated_to    VARCHAR(200),
    -- Specify the PRIMARY KEY constraint for table "dim_complaint".
    -- This indicates which attribute(s) uniquely identify each row of data.
    CONSTRAINT  pk_dim_complaint PRIMARY KEY (complaint_key)
);

-- Create a Database table to represent the "dim_customers" entity.
CREATE TABLE dim_customers(
    customer_key    INTEGER NOT NULL,
    customer_id INTEGER NOT NULL UNIQUE,
    customer_type   VARCHAR(200) NOT NULL,
    customer_zip_code   VARCHAR(200),
    customer_miles  VARCHAR(200),
    -- Specify the PRIMARY KEY constraint for table "dim_customers".
    -- This indicates which attribute(s) uniquely identify each row of data.
    CONSTRAINT  pk_dim_customers PRIMARY KEY (customer_key)
);


ALTER TABLE fact_table ADD CONSTRAINT fk1_fact_table_to_dim_time FOREIGN KEY(time_id) REFERENCES dim_time(time_id) ON DELETE CASCADE;

ALTER TABLE fact_table ADD CONSTRAINT fk2_fact_table_to_dim_flights FOREIGN KEY(flight_key) REFERENCES dim_flights(flight_key) ON DELETE CASCADE;


ALTER TABLE fact_table ADD CONSTRAINT fk3_fact_table_to_dim_complaint FOREIGN KEY(complaint_key) REFERENCES dim_complaint(complaint_key) ON DELETE
CASCADE;

ALTER TABLE fact_table ADD CONSTRAINT fk4_fact_table_to_dim_customers FOREIGN KEY(customer_key) REFERENCES dim_customers(customer_key) ON DELETE CASCADE;


-- Sequences
create sequence seq_fact_table start with 1 increment by 1;
create sequence seq_dim_flight start with 1 increment by 1;
create sequence seq_dim_time start with 1 increment by 1;
create sequence seq_dim_complaint start with 1 increment by 1;
create sequence seq_dim_customer start with 1 increment by 1;



--Triger for fact_table
CREATE OR REPLACE TRIGGER tri_fact_table
BEFORE INSERT ON fact_table
FOR EACH ROW
BEGIN
    :new.facttable_id := seq_fact_table.NEXTVAL;
END;
/

--Triger for dim_flights
CREATE OR REPLACE TRIGGER tri_dim_flights
BEFORE INSERT ON dim_flights
FOR EACH ROW
BEGIN
    :new.flight_key := seq_dim_flight.NEXTVAL;
END;
/

--Triger for dim_time
CREATE OR REPLACE TRIGGER tri_dim_time
BEFORE INSERT ON dim_time
FOR EACH ROW
BEGIN
    :new.time_id := seq_dim_time.NEXTVAL;
END;
/
```

```
--Triger for dim_compalint
CREATE OR REPLACE TRIGGER tri_dim_compalint
BEFORE INSERT ON dim_complaint
FOR EACH ROW
BEGIN
   :new.complaint_key := seq_dim_complaint.NEXTVAL;
END;
/

--Triger for dim_customer
CREATE OR REPLACE TRIGGER tri_dim_customers
BEFORE INSERT ON dim_customers
FOR EACH ROW
BEGIN
   :new.customer_key := seq_dim_customer.NEXTVAL;
END;
/

```

Fig: script to create fact table, dim tables and with its related sequences and triggers.

Successfully created dim tables, with its related sequences and triggers.

Dim tables are: dim_time, dim_flights, dim_customers, dim_complain

Fact table: fact_table

Sequences are: seq_fact_table, seq_dim_flight, seq_dim_time, seq_dim_complain, seq_dim_customer.

Triggers are: tri_fact_table, tri_dim_flights, tri_dim_customers, tri_dim_complain, tri_dim_time

Note:

I have removed foreign key (fk1_time_id, fk2_flight_id, fk3_complaint_id, fk4_customer_id) from fact_table and made time_id, compliant_key, customer_key and flight_key is foreign key in fact_table.

```
CREATE TABLE fact_table(
    facttable_id    INTEGER NOT NULL,
    time_id INTEGER,
    complaint_key   INTEGER,
    customer_key    INTEGER,
    flight_key  INTEGER,
    total_cancelled_flights NUMERIC(8,2),
    total_number_of_customers   NUMERIC(8,2),
    total_complaints    NUMERIC(8,2),
    COMPENSATION_AMNT   NUMERIC(8,2),
    -- Specify the PRIMARY KEY constraint for table "fact_table".
    -- This indicates which attribute(s) uniquely identify each row of data.
    CONSTRAINT  pk_fact_table PRIMARY KEY (facttable_id)
);

```

```
112  ALTER TABLE fact_table ADD CONSTRAINT fk1_fact_table_to_dim_time FOREIGN KEY(time_id) REFERENCES dim_time(time_id) ON DELETE CASCADE;
113
114  ALTER TABLE fact_table ADD CONSTRAINT fk2_fact_table_to_dim_flights FOREIGN KEY(flight_key) REFERENCES dim_flights(flight_key) ON DELETE CASCADE;
115
116
117  ALTER TABLE fact_table ADD CONSTRAINT fk3_fact_table_to_dim_complaint FOREIGN KEY(complaint_key) REFERENCES dim_complaint(complaint_key) ON DELETE
118  CASCADE;
119
120  ALTER TABLE fact_table ADD CONSTRAINT fk4_fact_table_to_dim_customers FOREIGN KEY(customer_key) REFERENCES dim_customers(customer_key) ON DELETE CASCADE;
121
122
```

Fig: script to change foreign key in table.

Dim tables data structure, constraint and triggers

Dim_flight table

DIM_FLIGHTS

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Add Column | Modify Column | Rename Column | Drop Column | Rename | Copy | Drop | Truncate | Create Lookup Table | Create App

Column Name	Data Type	Nullable	Default	Primary Key
FLIGHT_KEY	NUMBER	No	-	1
FLIGHT_ID	NUMBER	No	-	-
FLIGHT_NUMBER	NUMBER	No	-	-
CANCELLED	NUMBER(8,2)	Yes	-	-
DIVERTE	NUMBER(8,2)	Yes	-	-
ORIGIN_AIRPORT	VARCHAR2(200)	Yes	-	-
DESTINATION_AIRPORT	VARCHAR2(200)	Yes	-	-

Fig: structure of dim_flight table.

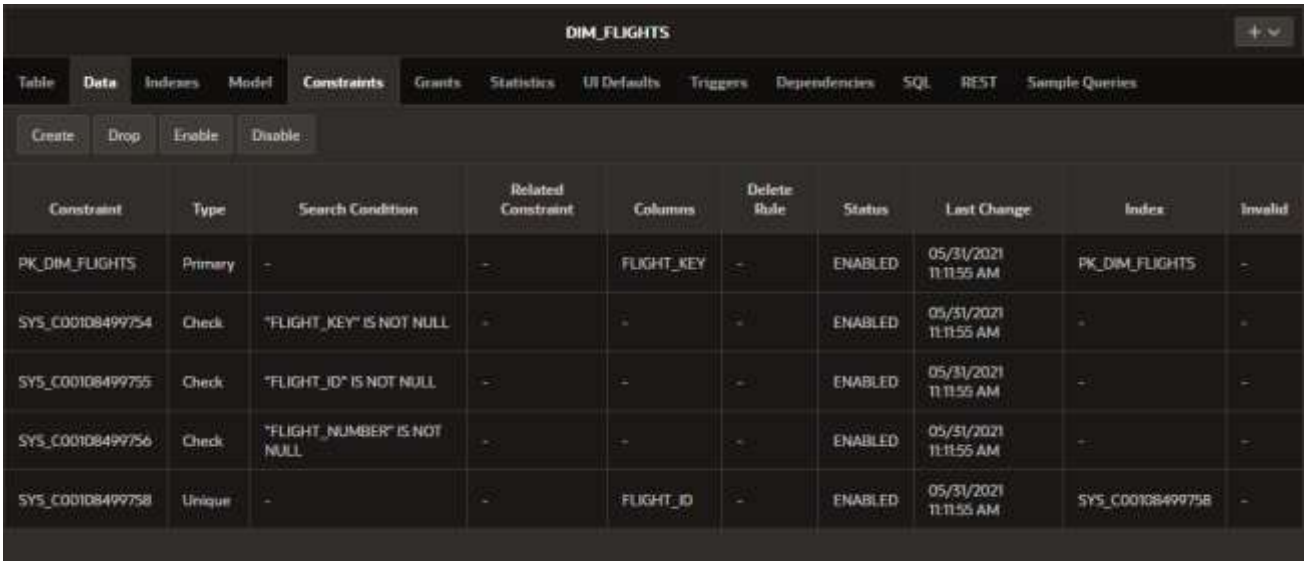


Fig: constraint of dim_flight table.

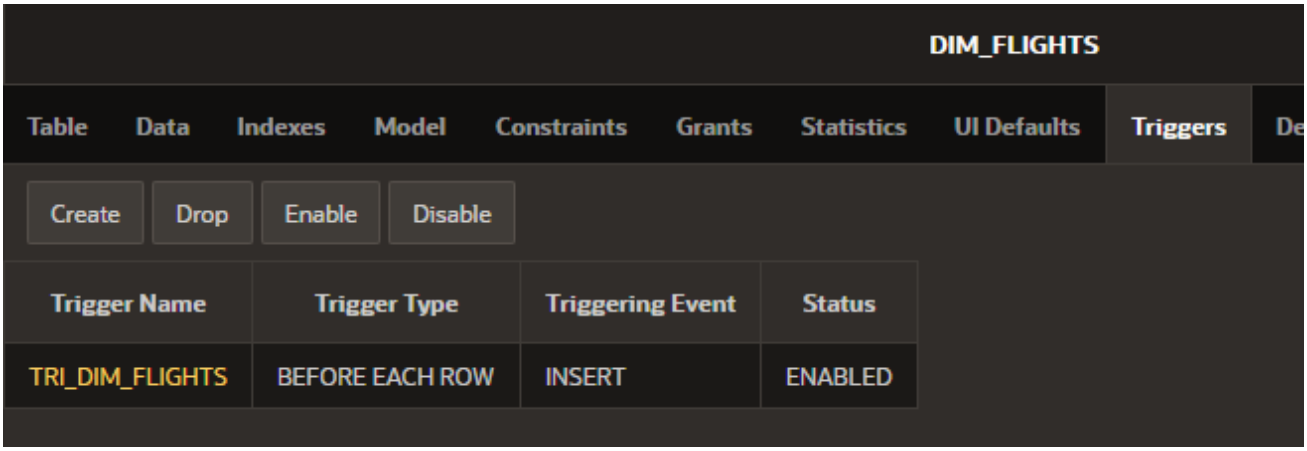


Fig: trigger of dim_flight table.

Dim_complaint table

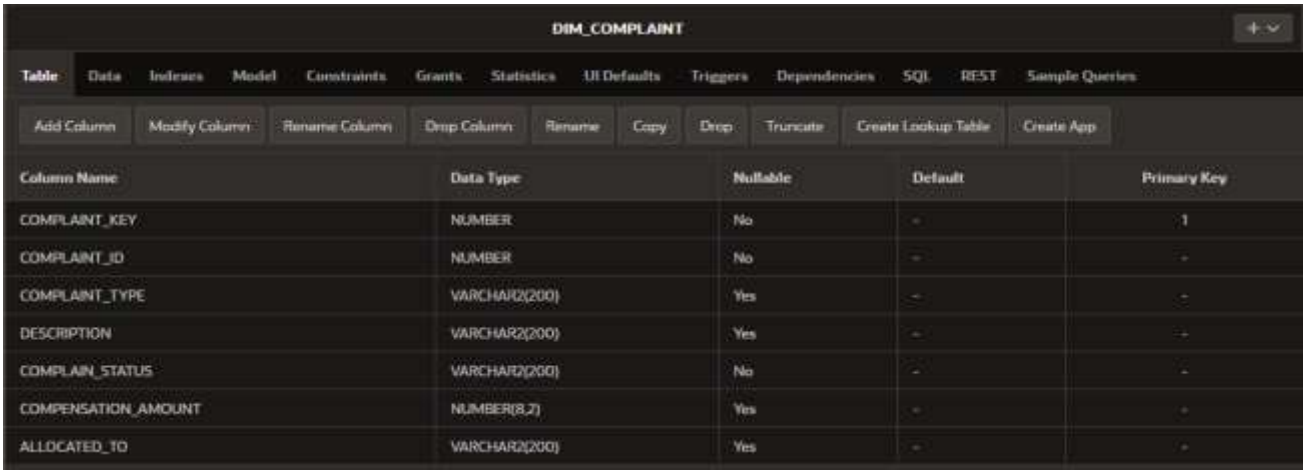


Fig: structure of dim_complaint table.

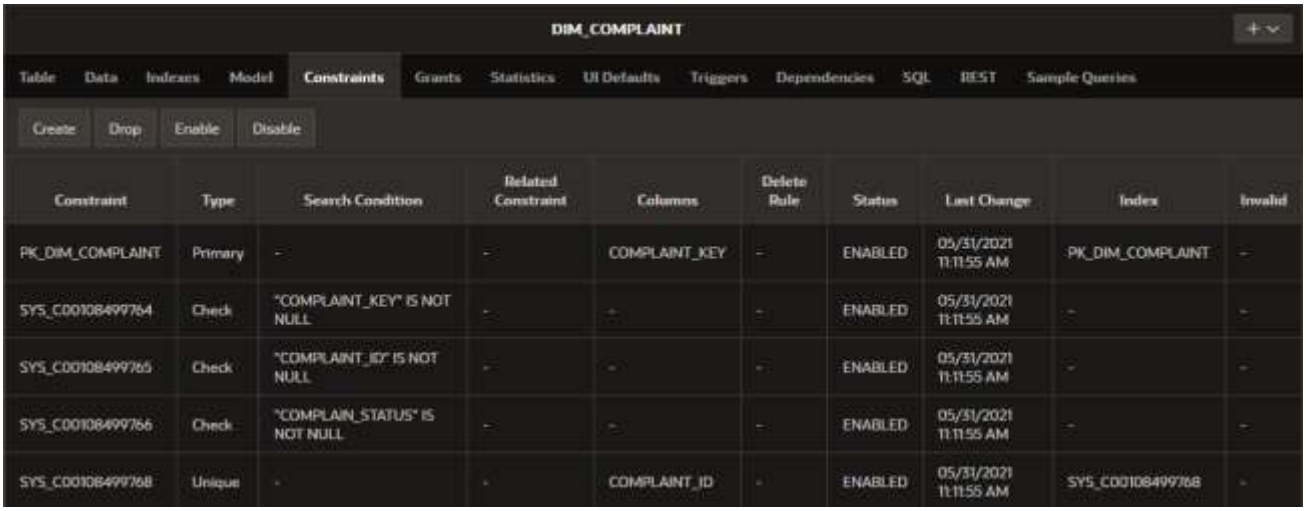


Fig: constraint of dim_complaint table.

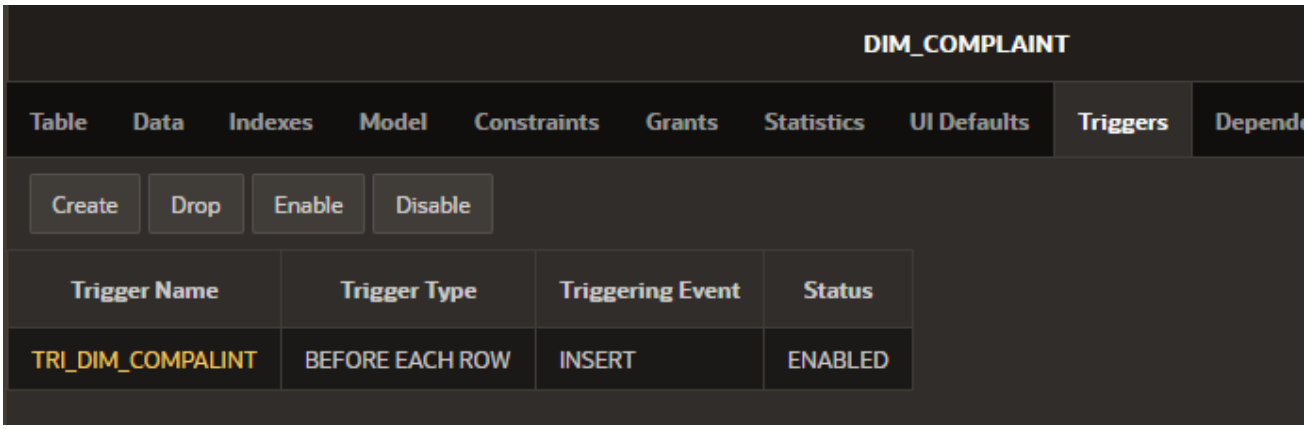


Fig: trigger of dim_complaint table.

Dim_customer table

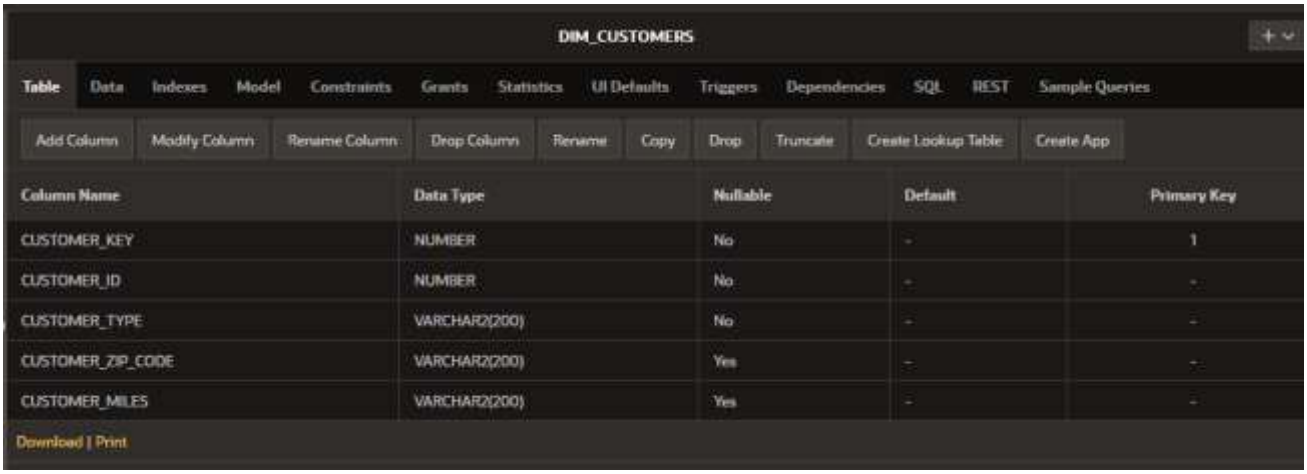


Fig: structure of dim_customer table.

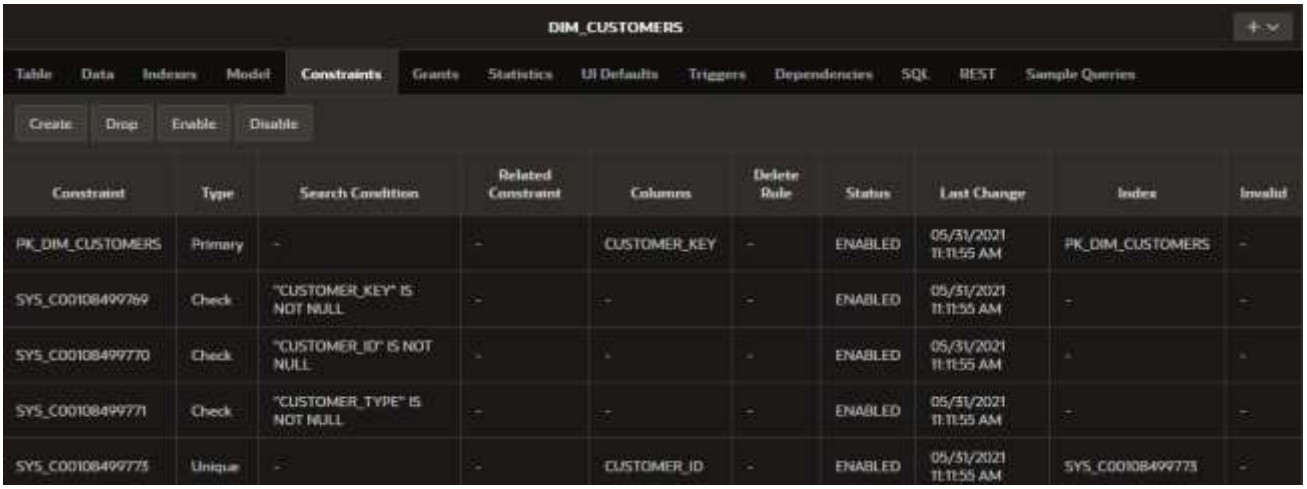


Fig: constraint of dim_customer table.

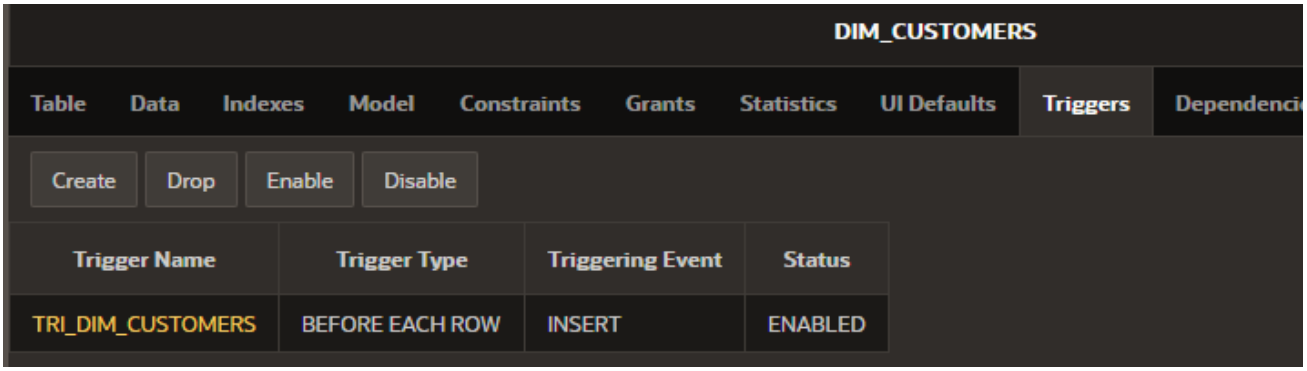


Fig: trigger of dim_customer table.

Dim_time table

DIM_TIME

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Add Column | Modify Column | Rename Column | Drop Column | Rename | Copy | Drop | Truncate | Create Lookup Table | Create App

Column Name	Data Type	Nullable	Default	Primary Key
TIME_ID	NUMBER	No	-	1
YEAR	DATE	No	-	-
MONTH	DATE	No	-	-
DAY	DATE	No	-	-

Download | Print

Fig: structure of dim_time table.

DIM_TIME

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Create | Drop | Enable | Disable

Constraint	Type	Search Condition	Related Constraint	Columns	Delete Rule	Status	Last Change	Index	Invalid
PK_DIM_TIME	Primary	-	-	TIME_ID	-	ENABLED	05/29/2021 06:25:58 AM	PK_DIM_TIME	-
SYS_C00108435539	Check	"TIME_ID" IS NOT NULL	-	-	-	ENABLED	05/29/2021 06:25:58 AM	-	-
SYS_C00108435540	Check	"YEAR" IS NOT NULL	-	-	-	ENABLED	05/29/2021 06:25:58 AM	-	-
SYS_C00108435541	Check	"MONTH" IS NOT NULL	-	-	-	ENABLED	05/29/2021 06:25:58 AM	-	-
SYS_C00108435542	Check	"DAY" IS NOT NULL	-	-	-	ENABLED	05/29/2021 06:25:58 AM	-	-

Fig: constraint of dim_time table.

DIM_TIME

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL | REST | Sample Queries

Create | Drop | Enable | Disable

Trigger Name	Trigger Type	Triggering Event	Status
TRI_DIM_TIME	BEFORE EACH ROW	INSERT	ENABLED

Fig: trigger of dim_time table.

Fact_table table

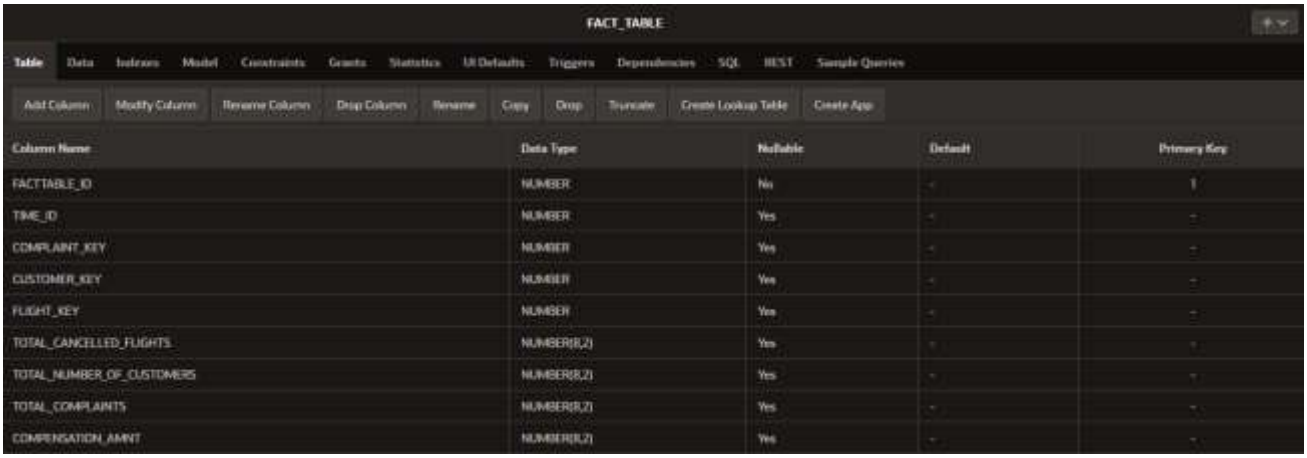


Fig: structure of fact_table table.

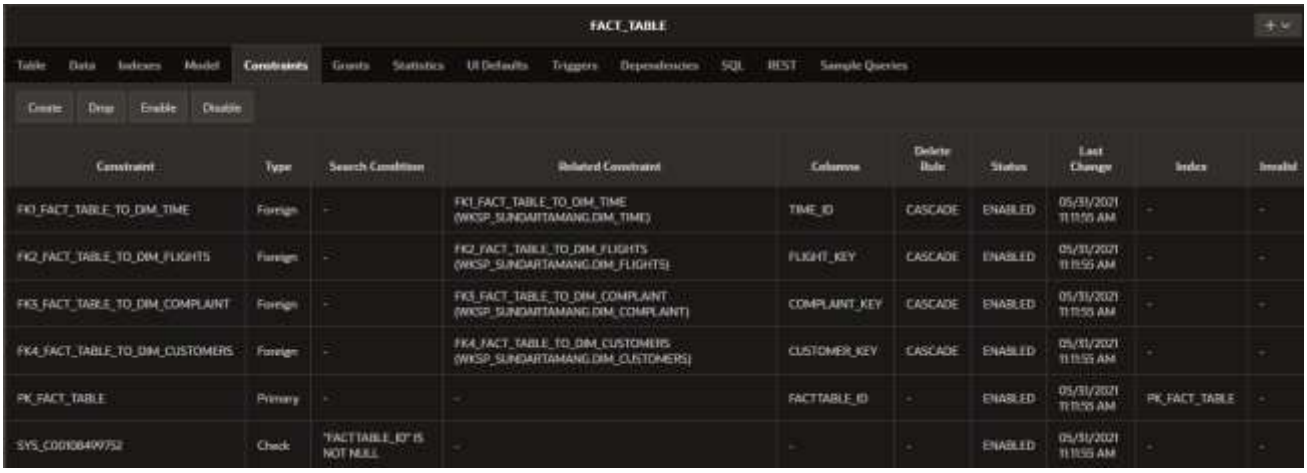


Fig: constraint of fact_table table.

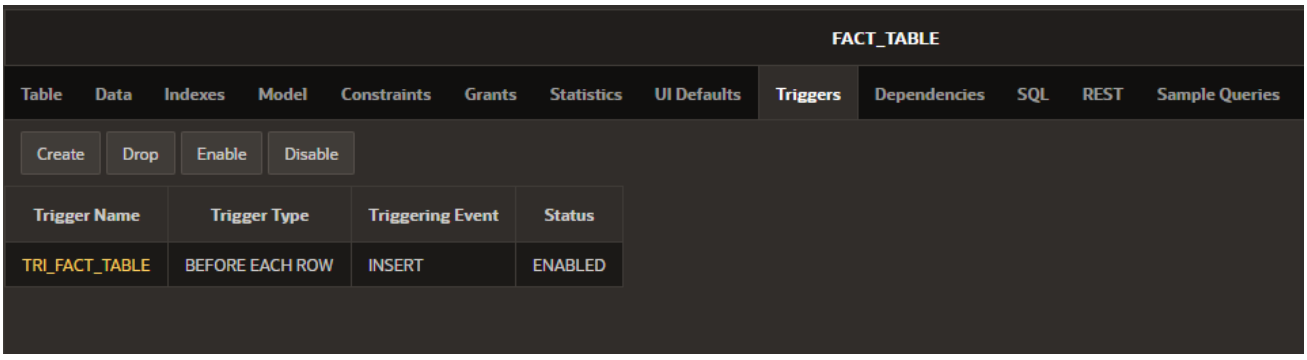


Fig: trigger of fact_table table.

Stagging table creation

```
-- To create stagging table for flights
 create table stage_flight
 (
     flight_key number(5),
     tail_number varchar2(30),
     flight_number varchar(30),
     cancelled number(10),
     diverted number(10),
     origin_airport varchar2(50),
     destination_airport varchar2(50),
     customer_id number(10),
     the_year number(4),
     the_month number(2),
     the_day number(2),
     db_source varchar2(30),
     constraint pk_stagging_flight primary key (flight_key)
 );


-- To create stagging table for customer
create table stage_customer
(
    customer_key number(5),
    customer_id number(10),
    customer_type varchar2(50),
    customer_zip_code varchar2(50),
    business varchar2(30),
    customer_miles varchar(100),
    db_source varchar2(30),
    constraint pk_stagging_customer primary key (customer_key)
);


    -- To create stagging table for complaint
    create table stage_complaint
    (
        complaint_key number(5),
        complaint_id number(10),
        complaint_type varchar2(50),
        complaint_description  varchar2(100),
        complaint_status varchar2(50),
        compensation_amnt varchar2(30),
        allocated_to varchar(100),
        customer_id number(10),
        flight_number number(10),
        the_year number(4),
        the_month number(2),
        the_day number(2),
        db_source varchar2(30),
        constraint pk_stagging_complaint primary key (complaint_key)
    );


-- Sequences
create sequence seq_stageflight start with 1 increment by 1;
create sequence seq_stagecomplaint start with 1 increment by 1;
create sequence seq_stagecustomer start with 1 increment by 1;



--Trigger for stage tables
CREATE OR REPLACE TRIGGER tri_stage_flight
BEFORE INSERT ON stage_flight
FOR EACH ROW
BEGIN
   :new.flight_key := seq_stageflight.NEXTVAL;
END;
/
```

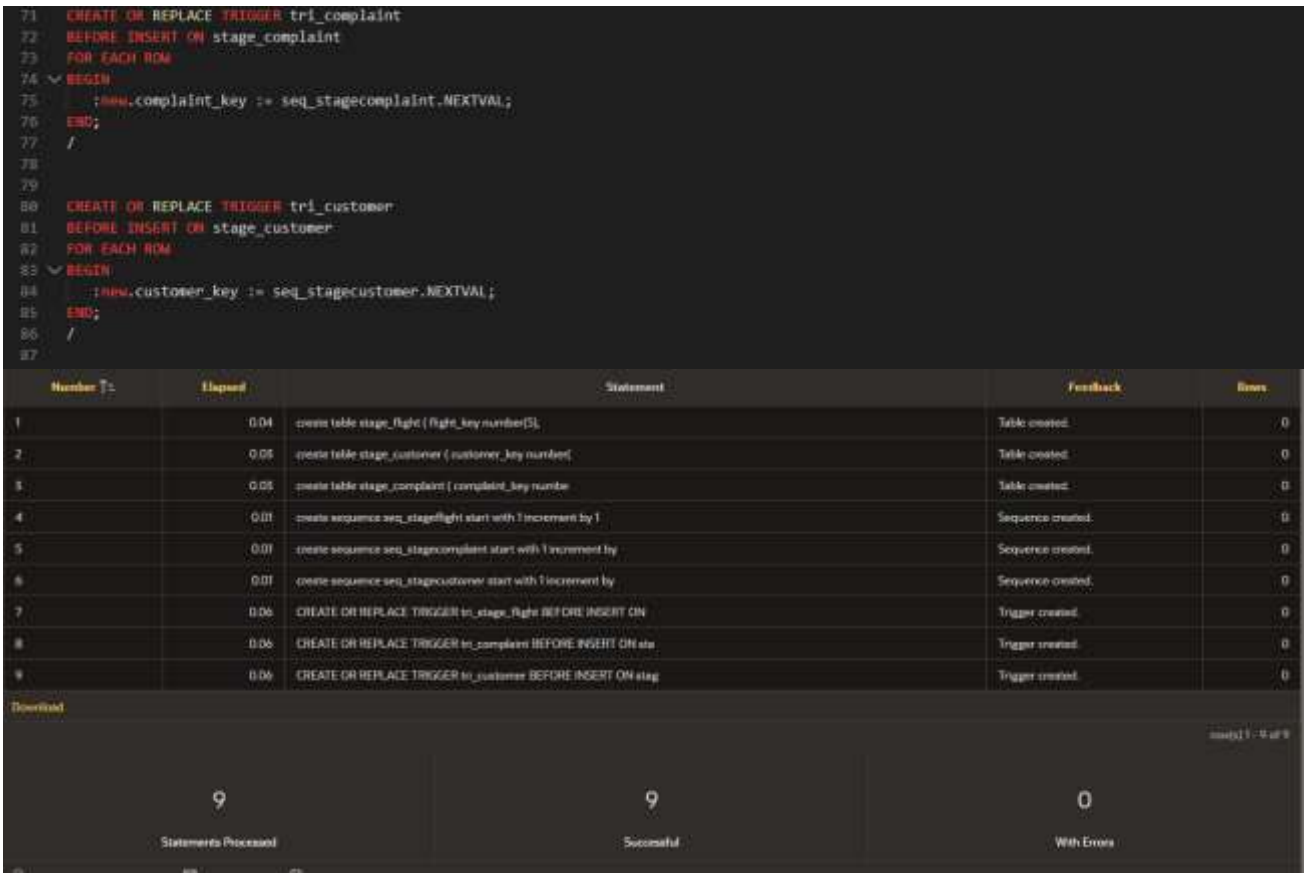


Fig: script to create stagging table with related sequences and triggers

Stagging tables are: stage_flight for flight data, stage_customer for customer data, stage_compliant for complaint data.

Sequences are: seq_stageFlight, seq_stagecomplaint, seq_stagecustomer

Triggers are: tri_stage_flight, tri_stage_customer, tri_stage_complaint.

Merge required data into stagging tables.

```
create or replace procedure pr_stage_flight as
begin

merge into STAGE_FLIGHT d
using FLIGHT_2017 s
on (s.flight_number = d.flight_number and d.DB_SOURCE='FLIGHT_2017')
when matched then
    update  set
        TAIL_NUMBER = s.tail_number,
        CANCELLED   = s.cancelled,
        DIVERTED    = s.diverted,
        ORIGIN_AIRPORT  = s.origin_airport,
        DESTINATION_AIRPORT = s.destination_airport,
        CUSTOMER_ID = s.FK_CUSTOMER_ID,
        THE_YEAR = s.year
when not matched then
    insert (FLIGHT_NUMBER, TAIL_NUMBER, CANCELLED, DIVERTED, ORIGIN_AIRPORT, DESTINATION_AIRPORT, CUSTOMER_ID, THE_YEAR, THE_MONTH, THE_DAY, DB_SOURCE)
    values (s.flight_number, s.tail_number,  s.cancelled, s.diverted, s.origin_airport, s.destination_airport,s.FK_CUSTOMER_ID, s.year, null, null, 'FLI

merge into STAGE_FLIGHT d
using FLYU_FLIGHTS s
on (s.flight_number = d.flight_number and d.DB_SOURCE='FLYU_FLIGHTS')
when matched then
    update  set
        TAIL_NUMBER = s.tail_number,
        CANCELLED   = s.cancelled,
        DIVERTED    = s.diverted,
        ORIGIN_AIRPORT  = s.origin_airport,
        DESTINATION_AIRPORT = s.destination_airport,
        CUSTOMER_ID = s.FK_CUSTOMER_ID,
        THE_YEAR = s.flight_the_year,
        THE_MONTH = s.the_month,
        THE_DAY = s.the_day
when not matched then
    insert (FLIGHT_NUMBER, TAIL_NUMBER, CANCELLED, DIVERTED, ORIGIN_AIRPORT, DESTINATION_AIRPORT, CUSTOMER_ID, THE_YEAR, THE_MONTH, THE_DAY, DB_SOURCE)
    values (s.flight_number, s.tail_number,  s.cancelled, s.diverted, s.origin_airport, s.destination_airport, s.FK_CUSTOMER_ID, s.flight_the_year,
     s.the_month, s.the_day, 'FLYU_FLIGHTS');
end;
/

-- procedure for stage_cutomer table
create or replace procedure pr_stage_customer as
begin
merge into stage_customer d
using customer cu
on (cu.customer_id = d.customer_id)
when matched then
    update  set
        CUSTOMER_TYPE = cu.customer_type,
        CUSTOMER_ZIP_CODE   = cu.customer_zip_code,
        BUSINESS    = cu.business,
        CUSTOMER_MILES  = cu.customer_miles,
        DB_SOURCE = 'customer'
when not matched then
    insert (CUSTOMER_ID, CUSTOMER_TYPE, CUSTOMER_ZIP_CODE, BUSINESS, CUSTOMER_MILES, DB_SOURCE)
    values (cu.customer_id, cu.customer_type, cu.customer_zip_code,  cu.business, cu.customer_miles, 'customer');
end;
/

-- procedure for stage_complaint table
create or replace procedure pr_stage_complaint as
begin
merge into stage_complaint d
using COMPLAINT cmt
on (cmt.COMPLAINT_ID = d.COMPLAINT_ID)
when matched then
    update  set
        COMPLAINT_TYPE = cmt.complaint_type,
        COMPLAINT_DESCRIPTION   = cmt.complaint_description,
        COMPLAINT_STATUS    = cmt.complaint_status,
        COMPENSATION_AMNT   = cmt.compensation_amnt,
        ALLOCATED_TO = cmt.allocated_to,
        CUSTOMER_ID = cmt.CUSTOMER_ID,
        FLIGHT_NUMBER = cmt.FLIGHT_ID_NO,
        THE_YEAR = cmt.THE_YEAR,
        THE_MONTH = cmt.THE_MONTH,
        THE_DAY = cmt.THE_DAY,
        DB_SOURCE = 'COMPLAINT'
when not matched then
```

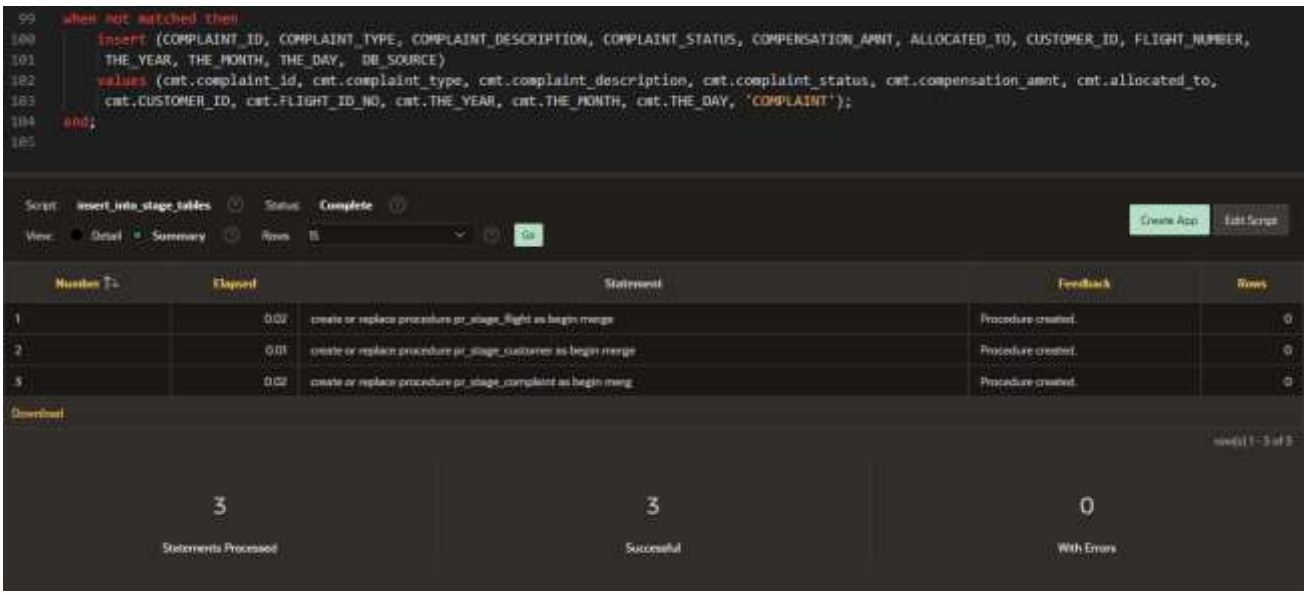


Fig: script to create procedure to insert data into stagging table

Procedures are: pr_stage_flight to insert data from flight_2017, flight_2018 and flyu_flights, pr_stage_customer to insert customer data, pr_stage_complaint to insert complaint data.

Fired created procedures

```
-- flight procedure
begin
pr_stage_flight;
end;
/

-- customer procedure
begin
pr_stage_customer;
end;
/

-- complain procedure
begin
pr_stage_complaint;
end;

```

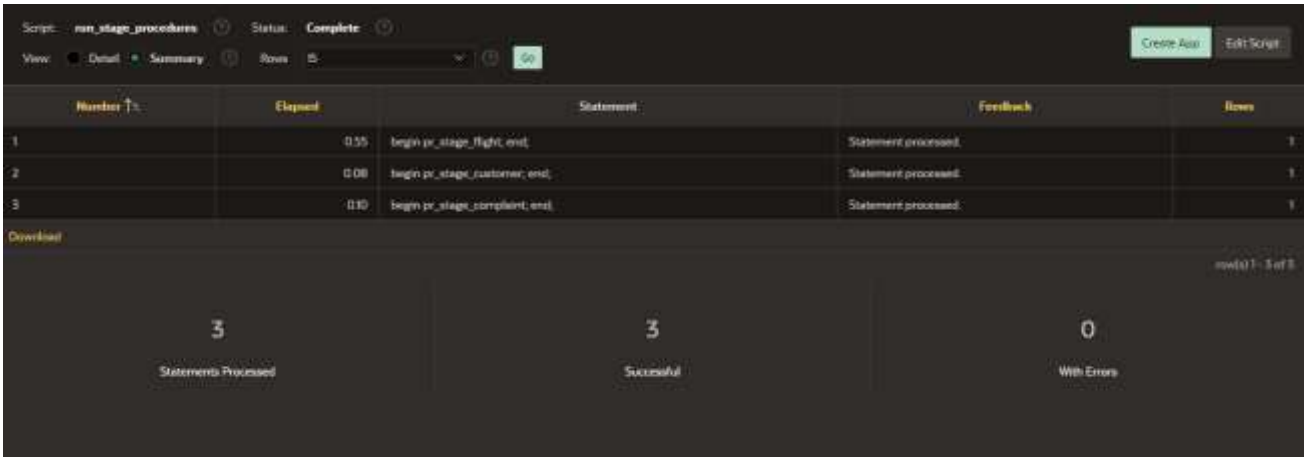


Fig: successfully fired created procedures

We have successfully fired created procedure, which means we have successfully inserted data into stagging tables.

Let's see data into stagging tables

Data of stage_flight table

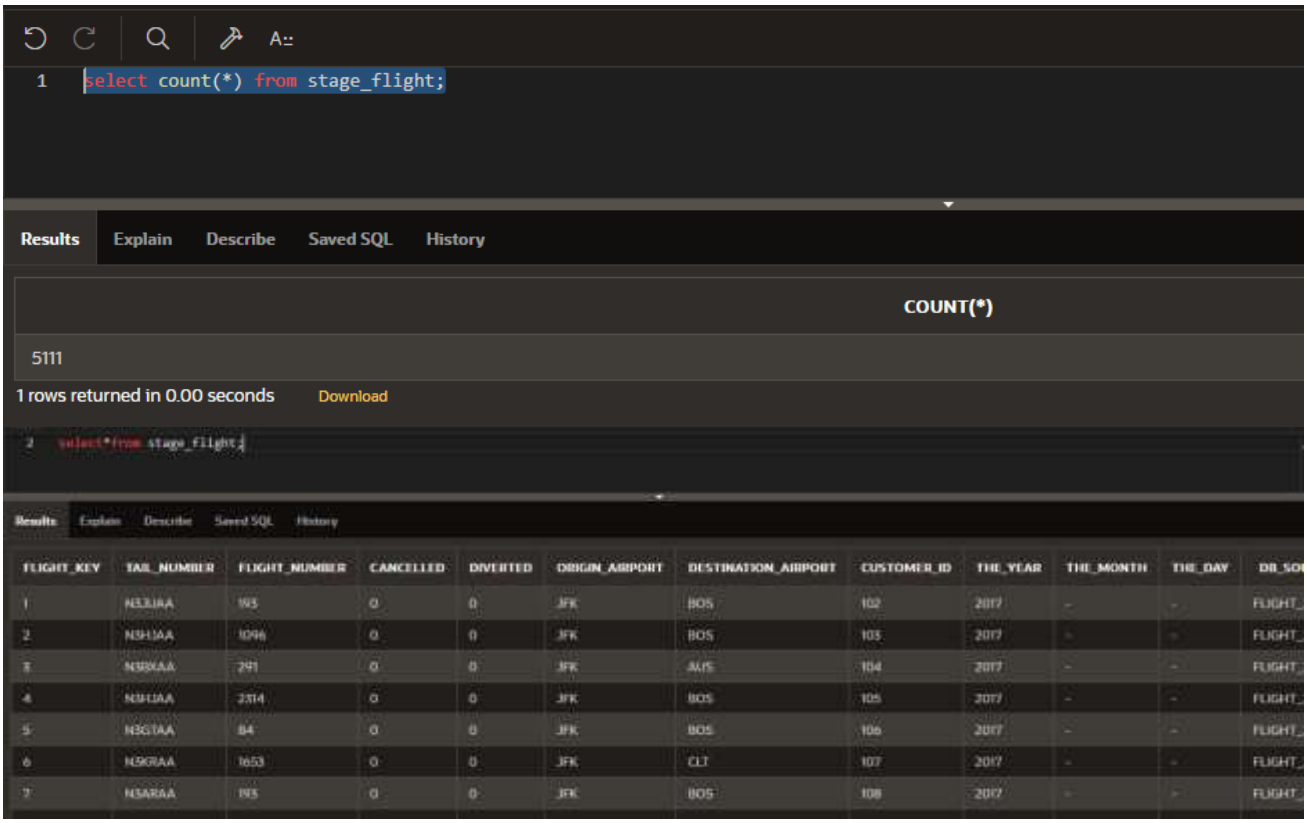


Fig: Data of stage_flight table

We have successfully inserted 5,111 data into stage_flight table from three data sources (flight_2017, flight_2018 and flyu_flights).

Data of stage_complaint table

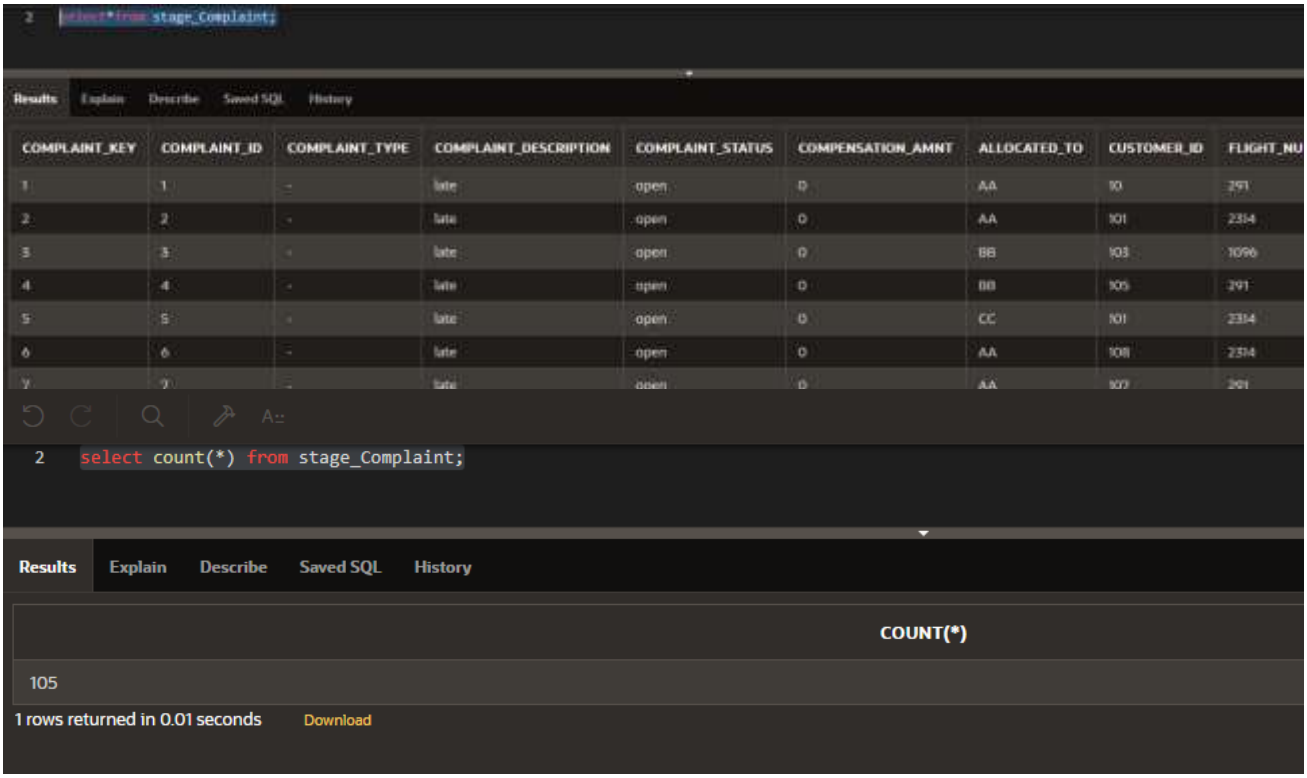
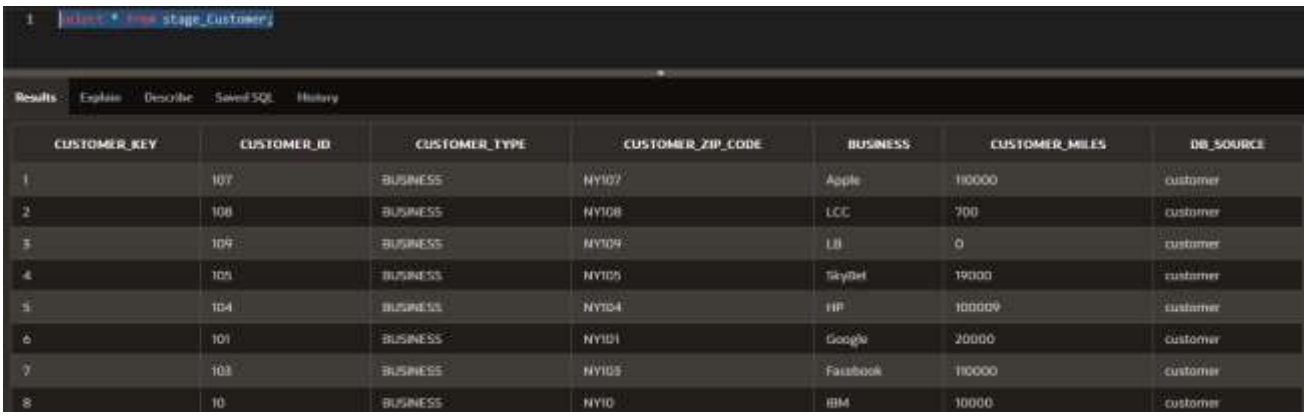


Fig: count all the data of stage_complaint table

We have successfully inserted 105 data into stage_complaint table from complaint data source.

Data of stage_customer table



```
1  select * from stage_Customer;
```

Results | Explain | Describe | Saved SQL | History

CUSTOMER_KEY	CUSTOMER_ID	CUSTOMER_TYPE	CUSTOMER_ZIP_CODE	BUSINESS	CUSTOMER_MILES	DB_SOURCE
1	107	BUSINESS	NY107	Apple	110000	customer
2	108	BUSINESS	NY108	LCC	700	customer
3	109	BUSINESS	NY109	LB	0	customer
4	105	BUSINESS	NY105	Skylliet	19000	customer
5	104	BUSINESS	NY104	HP	100009	customer
6	101	BUSINESS	NY101	Google	20000	customer
7	103	BUSINESS	NY103	Facebook	110000	customer
8	10	BUSINESS	NY10	IBM	10000	customer

Fig: count all the data of stage_customer table

We have successfully inserted 11 data into stage_customer table from customer data source.

From business perspective – rules

- Every plane should have tail number
- If flight is cancelled then there can't be flight diverted, which means if cancelled is 1 then diverted can't be 1.
- Destination and origin airport name should be in alphabet
- If complaint_type is null then there cannot be complaint description.
- Complaint_type should be either A, B or C.
- Complaint_allocated should be either to AA, BB, CC or DD.
- Airport name should be in alphabetic.
- Compensation amount cannot be in negative.
- Customer miles can not be in negative

Good data and bad data

Create good data and bad data tables

```
-- create clean tables
create table clean_flight as select *from stage_flight where 1=0;
create table clean_complaint as select *from stage_complaint where 1=0;
create table clean_customer as select *from stage_customer where 1=0;

-- create bad tables
create table bad_flight as select *from stage_flight where 1=0;
create table bad_complaint as select *from stage_complaint where 1=0;
create table bad_customer as select *from stage_customer where 1=0;


 -- create data_issues table
 create table data_issues(
     issue_id number not null,
     table_name varchar2(100),
     row_id number,
     data_error_code number(10),
     issue_description varchar2(100),
     issue_date DATE,
     issue_status varchar(100),
     status_update_date DATE
 );


 -- create sequences
 create sequence seq_dataIssue start with 1 increment by 1;

 --create triggers
 CREATE OR REPLACE TRIGGER tri_data_issues
 BEFORE INSERT ON data_issues
 FOR EACH ROW
 BEGIN
    :new.issue_id := seq_dataIssue.NEXTVAL;
 END;
 /
```

Fig: script to create table for bad and good tables with require sequences and triggers

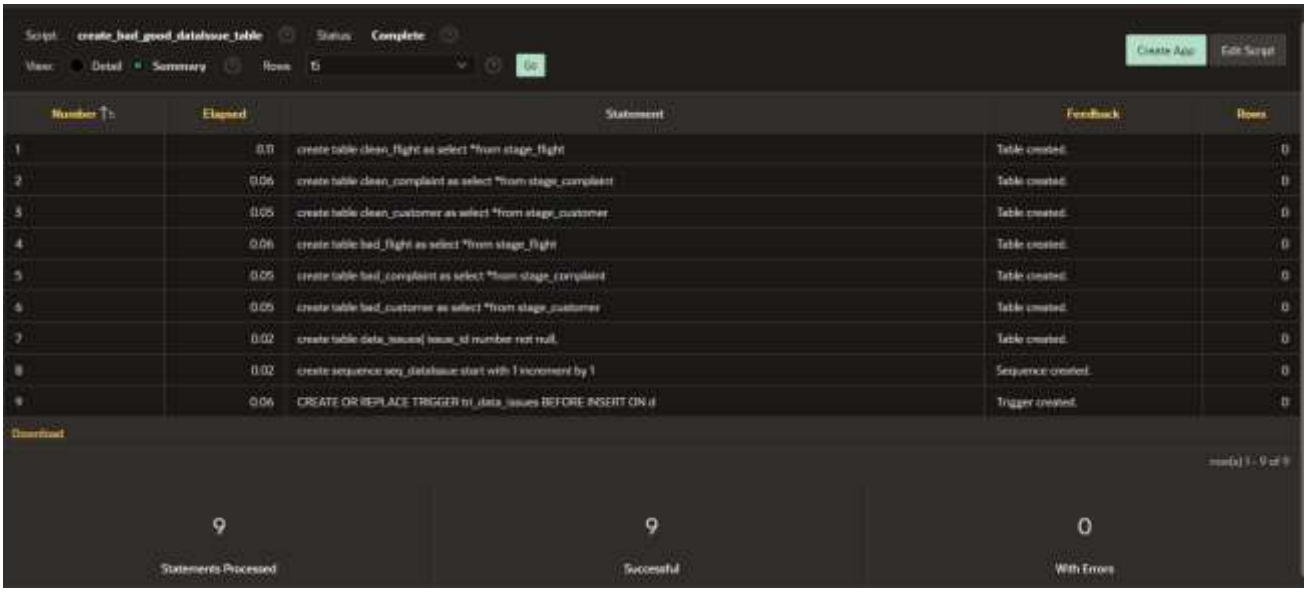


Fig: successfully created table for good, bad data with related sequence and trigger

Successfully created tables for good and bad data with related sequence and trigger.

Bad tables for bad data are: bad_flight, bad_complaint, bad_customer,

Good tables for good data are: clean_flight, clean_customer, clean_complaint

Sequence: seq_dataIssue,

Trigger: tri_data_issue.

Drop script to drop bad and good data tables

```
1   -- drop clean tables
2   drop table clean_flight cascade constraint;
3   drop table clean_complaint cascade constraint;
4   drop table clean_customer cascade constraint;
5
6   -- drop bad tables
7   drop table bad_flight cascade constraint;
8   drop table bad_complaint cascade constraint;
9   drop table bad_customer cascade constraint;
10
11  -- drop sequences
12  drop sequence seq_dataIssue;
```

Fig: script to drop good bad data tables and sequence

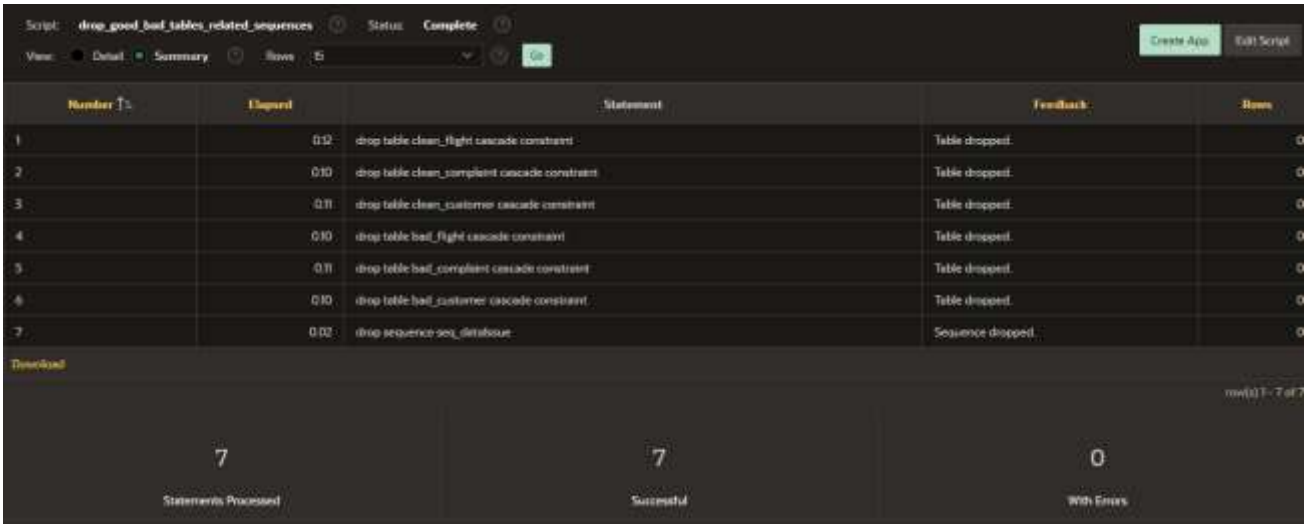


Fig: successfully dropped good, bad table and sequence.

Successfully dropped all the tables with related sequence and trigger

Add bad data to the table

```sql
-- add bad data to flight
update stage_flight
set diverted = 1 where flight_key = 9;

update stage_flight
set diverted = 1 where flight_key = 11;

update stage_flight
set origin_airport = 'J@K' where flight_key = 5110;

update stage_flight
set origin_airport = 'J1K' where flight_key = 4001;

-- add bad data to complaint
update stage_complaint
set COMPENSATION_AMNT = '500w' where complaint_key = 10;

-- add bad data to customer
update stage_customer
set CUSTOMER_MILES = -500 where customer_key = 10;
```

Fig: script to add bad data in the tables.

Update diverted 1 in two rows where flight_key 9 and 11. In thee two row flight is cancelled, which means there is flight when flight is cancelled, so this is against business rule.

We have update airport name so that can be become bad data

In compensation amount we have added w word to make it bad data

And there is customer_miles which is in negative number

Clean Data

Separate good data and bad data of stage_flight table

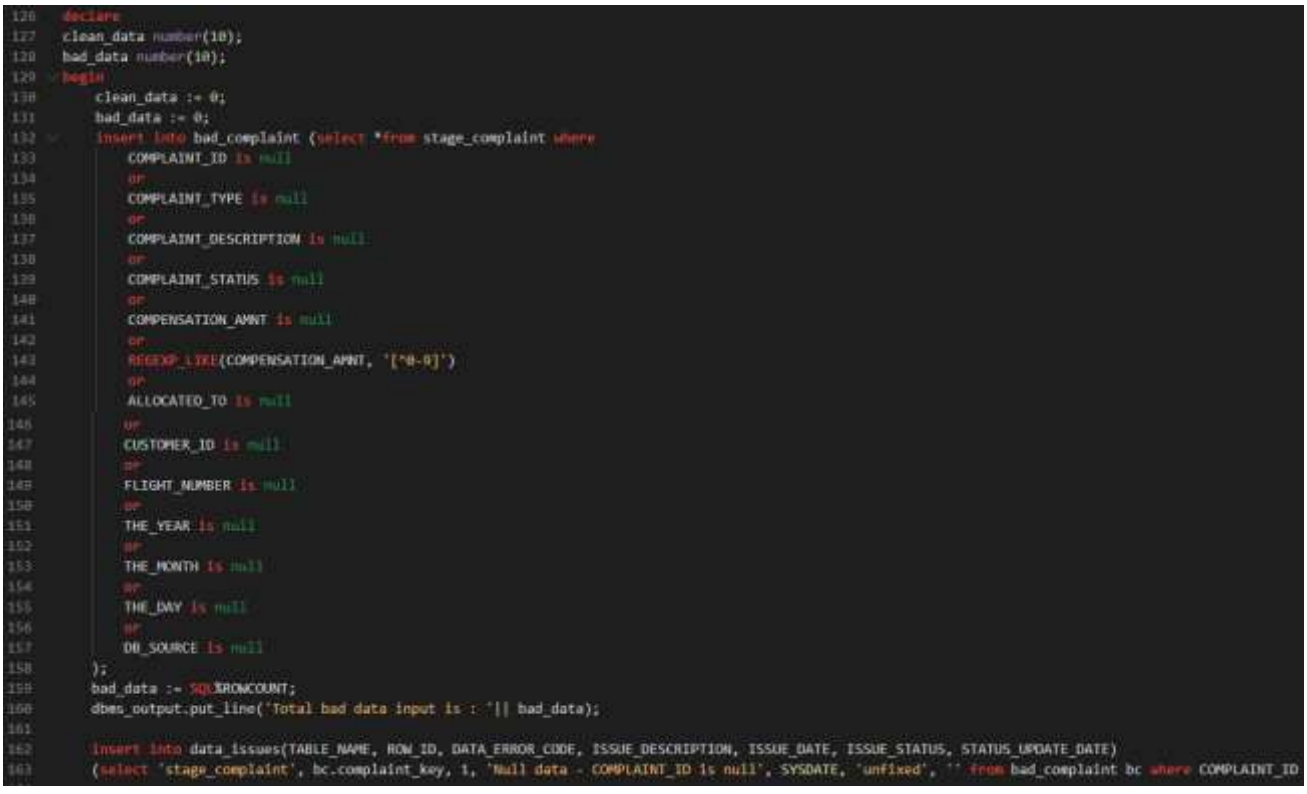
```
-- for flight data
declare
clean_data number(10);
bad_data number(10);
begin
    clean_data := 0;
    bad_data := 0;
    insert into bad_flight (select *from stage_flight where
        (CANCELLED = 1 and DIVERTED = 1)
        or
        TAIL_NUMBER is null
        or
        FLIGHT_NUMBER is null
        or
        CANCELLED is null
        or
        DIVERTED is null
        or
        ORIGIN_AIRPORT is null
        or
        REGEXP_LIKE(ORIGIN_AIRPORT, '[^a-zA-Z]')
        or
        DESTINATION_AIRPORT is null
        or
        REGEXP_LIKE(DESTINATION_AIRPORT, '[^a-zA-Z]')
        or
        CUSTOMER_ID is null
        or
        THE_YEAR is null
        or
        THE_MONTH is null
        or
        THE_DAY is null
        or
        DB_SOURCE is null
    );
    bad_data := SQL%ROWCOUNT;
    dbms_output.put_line('Total bad data input is : '|| bad_data);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 10, 'Null data - tail number is null', SYSDATE, 'unfixed', '' from bad_flight bf where
    tail_number is null );
```

```
    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 20, 'Data entry - flight diverted even flight is cancelled', SYSDATE, 'unfixed', '' from
    bad_flight bf where CANCELLED = 1 and DIVERTED = 1);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 30, 'Null data - flight number is null', SYSDATE, 'unfixed', '' from bad_flight bf where
    FLIGHT_NUMBER is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 40, 'Null data - cancelled is null', SYSDATE, 'unfixed', '' from bad_flight bf where CANCELLED is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 50, 'Null data - diverted is null', SYSDATE, 'unfixed', '' from bad_flight bf where DIVERTED is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 60, 'Null data - origin_airport is null', SYSDATE, 'unfixed', '' from bad_flight bf where  ORIGIN_AIRPORT
     is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 60, 'Non consistency - airport name consist invalid character', SYSDATE, 'unfixed', '' from bad_flight bf
    where REGEXP_LIKE(ORIGIN_AIRPORT, '[^a-zA-Z]'));

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 70, 'Null data - destination_airport is null', SYSDATE, 'unfixed', '' from bad_flight bf where
    DESTINATION_AIRPORT is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 80, 'Non consistency - airport name consist invalid character', SYSDATE, 'unfixed', '' from bad_flight bf
    where REGEXP_LIKE(DESTINATION_AIRPORT, '[^a-zA-Z]'));

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 90, 'Null data - customer_id is null', SYSDATE, 'unfixed', '' from bad_flight bf where CUSTOMER_ID is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 100, 'Null data - year is null', SYSDATE, 'unfixed', '' from bad_flight bf where THE_YEAR is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 110, 'Null data - month is null', SYSDATE, 'unfixed', '' from bad_flight bf where THE_MONTH is null);
    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 111, 'Null data - day is null', SYSDATE, 'unfixed', '' from bad_flight bf where the_day is null);


    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_flight', bf.flight_key, 112, 'Null data - db_Source is null', SYSDATE, 'unfixed', '' from bad_flight bf where THE_DAY is null);

    insert into clean_flight (
        select * from stage_flight where
        (CANCELLED = 1 and DIVERTED = 0 or CANCELLED = 0 and diverted = 0 or cancelled = 0 and diverted = 1)
        AND
        TAIL_NUMBER is not null
        AND
        FLIGHT_NUMBER is not null
        AND
        CANCELLED is not null
        AND
        DIVERTED is not null
        AND
        ORIGIN_AIRPORT is not null
        AND
        not REGEXP_LIKE(ORIGIN_AIRPORT, '[^a-z,A-Z]')
        AND
        DESTINATION_AIRPORT is not null
        AND
        not REGEXP_LIKE(DESTINATION_AIRPORT, '[^a-z,A-Z]')
        AND
        customer_id is not null
        AND
        the_year is not null
        AND
        the_month is not null
        AND
        the_day is not null
        AND
          DB_SOURCE is not null
    );
    clean_data := SQL%ROWCOUNT;
    dbms_output.put_line('Total clean data input is : '|| clean_data);
end;
/
```

Fig: script to insert bad data of flight into bad_flight table and clean data into clean_fligght table

Above SQL code was written to insert good data into clean_flight table and bad data into bad_flight table. There were 3503 bad data and 1608 clean data, and all the bad data is inserted into bad_flight table while clean data is in clean_flight table.

Separate good data and bad data of stage_complaint table

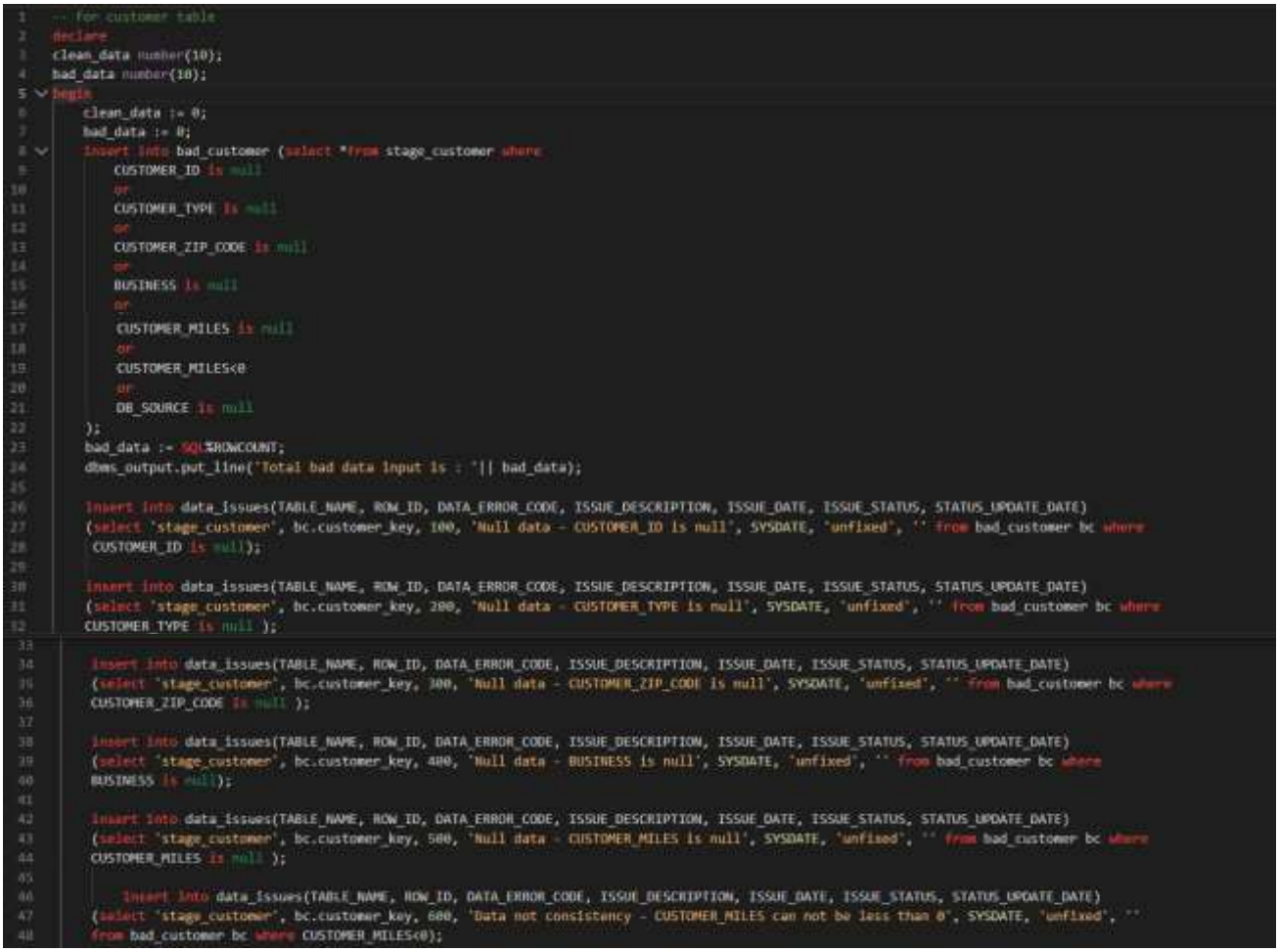


```
    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 2, 'Null data - COMPLAINT_TYPE is null', SYSDATE, 'unfixed', '' from bad_complaint bc where
    COMPLAINT_TYPE is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 3, 'Null data - COMPLAINT_DESCRIPTION is null', SYSDATE, 'unfixed', '' from bad_complaint bc where
    COMPLAINT_DESCRIPTION is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 4, 'Null data - COMPLAINT_STATUS is null', SYSDATE, 'unfixed', '' from bad_complaint bc where
    COMPLAINT_STATUS is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 5, 'Null data - COMPENSATION_AMNT is null', SYSDATE, 'unfixed', '' from bad_complaint bc where
    COMPENSATION_AMNT is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 6, 'Non consistency - COMPENSATION_AMNT is not in string format', SYSDATE, 'unfixed', '' from
    bad_complaint bc where REGEXP_LIKE(COMPENSATION_AMNT, '[^0-9]'));

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 7, 'Null data - ALLOCATED_TO is null', SYSDATE, 'unfixed', '' from bad_complaint bc where
    ALLOCATED_TO is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 8, 'Null data - CUSTOMER_ID is null', SYSDATE, 'unfixed', '' from bad_complaint bc where
    CUSTOMER_ID is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 9, 'Non consistency - FLIGHT_NUMBER is not in string format', SYSDATE, 'unfixed', '' from
    bad_complaint bc where FLIGHT_NUMBER is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 10, 'Null data - year is null', SYSDATE, 'unfixed', '' from bad_complaint bc where THE_YEAR  is null);


    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 11, 'Null data - month is null', SYSDATE, 'unfixed', '' from bad_complaint bc where THE_MONTH is null);


    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 12, 'Null data - day is null', SYSDATE, 'unfixed', '' from bad_complaint bc where THE_DAY is null );


    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_complaint', bc.complaint_key, 13, 'Null data - DB_SOURCE is null', SYSDATE, 'unfixed', '' from bad_complaint bc where DB_SOURCE is
    null);

    insert into clean_complaint(select * from stage_complaint where
        COMPLAINT_ID is not null
        AND
        COMPLAINT_TYPE is not null
        AND
        COMPLAINT_DESCRIPTION is not null
        AND
        COMPLAINT_STATUS is not null
        AND
        COMPENSATION_AMNT is not null
        AND
        not REGEXP_LIKE(COMPENSATION_AMNT, '[^0-9]')
        AND
        ALLOCATED_TO is not null
        AND
        CUSTOMER_ID is not null
        AND
        FLIGHT_NUMBER is not null
        AND
        THE_YEAR is not null
        AND
        THE_MONTH is not null
        AND
        THE_DAY is not null
        AND
        DB_SOURCE is not null
    );
    clean_data := SQL%ROWCOUNT;
    dbms_output.put_line('Total clean data input is : '|| clean_data);
end;
```

Fig: script to insert bad data of complaint into bad_compalint table and clean data into clean_complaint table

Above SQL code was written to insert good data of stage_complaint into clean_complaint table and bad data into bad_complaint table. There are 75 bad data and 30 clean data, and all the bad data is inserted into bad_complaint table while clean data is in clean_complaint table.

Separate good data and bad data of stage_customer table



```
-- for customer table
declare
clean_data number(10);
bad_data number(10);
begin
    clean_data := 0;
    bad_data := 0;
    insert into bad_customer (select *from stage_customer where
        CUSTOMER_ID is null
        or
        CUSTOMER_TYPE is null
        or
        CUSTOMER_ZIP_CODE is null
        or
        BUSINESS is null
        or
        CUSTOMER_MILES is null
        or
        CUSTOMER_MILES<0
        or
        DB_SOURCE is null
    );
    bad_data := SQL%ROWCOUNT;
    dbms_output.put_line('Total bad data input is : '|| bad_data);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_customer', bc.customer_key, 100, 'Null data - CUSTOMER_ID is null', SYSDATE, 'unfixed', '' from bad_customer bc where
     CUSTOMER_ID is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_customer', bc.customer_key, 200, 'Null data - CUSTOMER_TYPE is null', SYSDATE, 'unfixed', '' from bad_customer bc where
    CUSTOMER_TYPE is null );

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_customer', bc.customer_key, 300, 'Null data - CUSTOMER_ZIP_CODE is null', SYSDATE, 'unfixed', '' from bad_customer bc where
    CUSTOMER_ZIP_CODE is null );

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_customer', bc.customer_key, 400, 'Null data - BUSINESS is null', SYSDATE, 'unfixed', '' from bad_customer bc where
    BUSINESS is null);

    insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_customer', bc.customer_key, 500, 'Null data - CUSTOMER_MILES is null', SYSDATE, 'unfixed', '' from bad_customer bc where
    CUSTOMER_MILES is null );

        insert into data_issues(TABLE_NAME, ROW_ID, DATA_ERROR_CODE, ISSUE_DESCRIPTION, ISSUE_DATE, ISSUE_STATUS, STATUS_UPDATE_DATE)
    (select 'stage_customer', bc.customer_key, 600, 'Data not consistency - CUSTOMER_MILES can not be less than 0', SYSDATE, 'unfixed', ''
    from bad_customer bc where CUSTOMER_MILES<0);
```

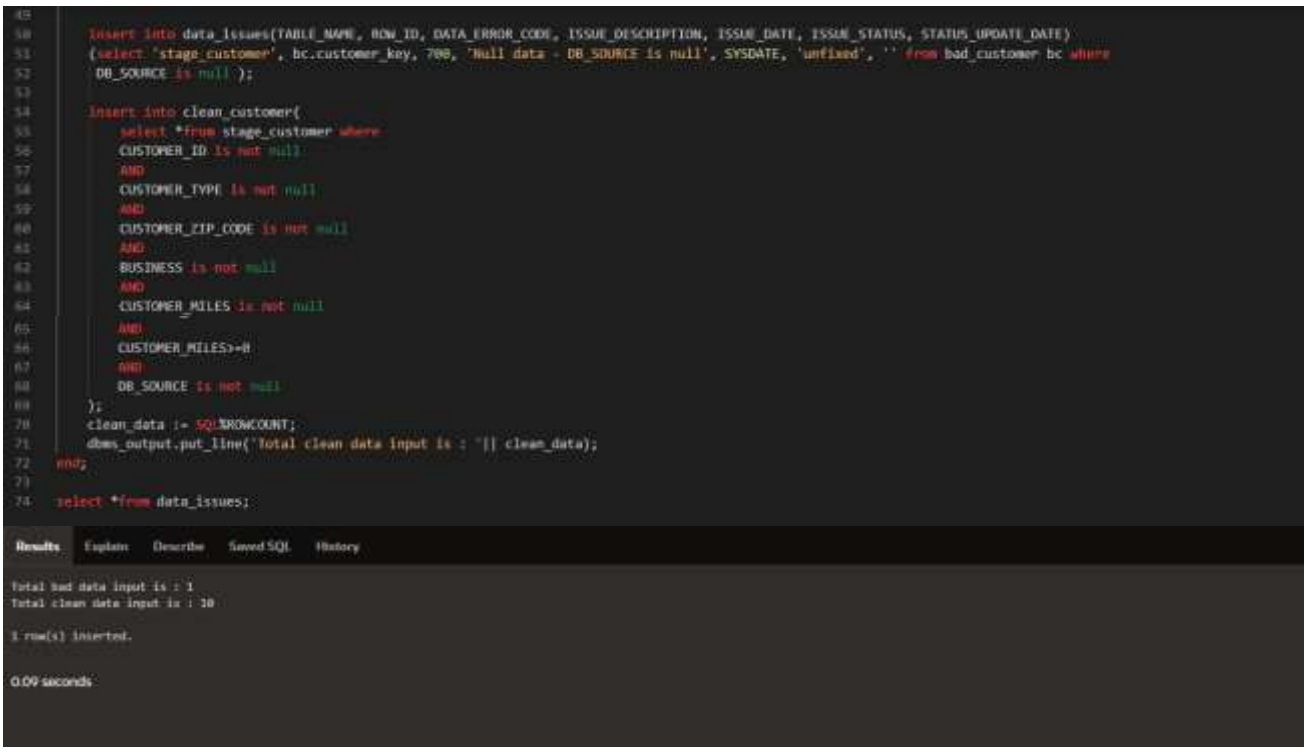


Fig: script to insert bad data of customer into bad_customer table and clean data into clean_customer table

Above SQL code was written to insert good data of stage_customer into clean_customer table and bad data into bad_customer table. There were 1 bad data and 10 clan data, and all the bad data is inserted into bad_customer table while clean data is in clean_customer table.

After successfully executing above query let's see good data in good tables and bad data in bad tables

Good data of flight table which are now migrated into clean_flight table

Fig: total number of good data in clean_flight table

Total number of good data is 1606 which are now migrated into clean_flight table

Good data of complaint table which are now migrated into clean_complaintt table

Fig: total number of good data in clean_complaint table

Total number of good data is 30 which are now migrated into clean_flight table

Good data of customer table which are now migrated into clean_customer table

Fig: total number of good data in clean_customer table

Total number of good data is 10 which are now migrated into clean_flight table

Bad data of flight

Fig: bad data

In total we have 3505 bad data in stage_flight table, and all the bad date is now migrated into

bad_flight table.

Bad data of complaint

COMPLAINT_KEY	COMPLAINT_ID	COMPLAINT_TYPE	COMPLAINT_DESCRIPTION	COMPLAINT_STATUS	COMPENSATION_AMNT	ALLOCATED_TO	CUSTOMER_ID	FLIGHT_NUMBER	THE_YEAR	THE_MONTH	THE_DAY	DB_SOURCE
1	1	-	late	open	0	AA	10	201	2017	1	1	COMPLAINT
2	2	-	late	open	0	AA	101	2314	2017	1	3	COMPLAINT
3	3	-	late	open	0	BB	102	1096	2017	1	4	COMPLAINT
4	4	-	late	open	0	BB	106	291	2017	1	4	COMPLAINT
5	5	-	late	open	0	CC	107	2314	2017	1	4	COMPLAINT
6	6	-	late	open	0	AA	108	2314	2017	1	6	COMPLAINT
7	7	-	late	open	0	AA	107	291	2017	1	7	COMPLAINT
8	8	-	late	open	0	BB	10	1096	2017	2	3	COMPLAINT
9	9	-	late	open	0	BB	106	2314	2017	2	5	COMPLAINT
10	10	-	late	open	1000	CC	107	2314	2017	2	6	COMPLAINT
11	11	-	late	open	0	AA	102	1096	2017	2	8	COMPLAINT
12	12	-	late	open	0	AA	104	1096	2017	2	10	COMPLAINT
13	13	-	late	open	0	BB	107	199	2017	2	10	COMPLAINT
14	14	-	late	open	0	BB	108	1096	2017	2	10	COMPLAINT
15	15	-	late	open	0	CC	109	2314	2017	2	10	COMPLAINT
16	16	-	late	open	0	DD	10	1096	2017	2	11	COMPLAINT
17	17	-	late	open	0	DD	100	2314	2017	2	21	COMPLAINT
18	18	-	late	open	0	AA	100	84	2017	2	22	COMPLAINT
19	19	-	late	open	0	AA	101	2314	2017	2	27	COMPLAINT
20	20	-	late	open	0	BB	10	199	2017	3	1	COMPLAINT
21	21	-	late	open	0	BB	10	199	2017	3	3	COMPLAINT
22	22	-	late	open	0	CC	101	2314	2017	3	3	COMPLAINT
23	23	-	late	open	0	AA	101	291	2017	3	7	COMPLAINT
24	24	-	late	open	0	AA	103	291	2017	3	10	COMPLAINT
25	25	-	late	open	0	BB	103	2314	2017	3	17	COMPLAINT
26	26	-	late	open	0	BB	104	291	2017	3	20	COMPLAINT
27	27	-	late	open	0	CC	105	84	2017	3	21	COMPLAINT
28	28	-	late	open	0	AA	106	2314	2017	3	21	COMPLAINT
29	29	-	late	open	0	AA	107	291	2017	3	24	COMPLAINT
30	30	-	late	open	0	BB	108	2314	2017	3	26	COMPLAINT
31	31	-	late	open	0	BB	109	199	2017	3	28	COMPLAINT
32	32	-	late	open	0	CC	10	84	2017	4	4	COMPLAINT
33	33	-	late	open	0	DD	100	2314	2017	4	4	COMPLAINT
34	34	-	late	open	0	DD	100	1664	2017	4	10	COMPLAINT
35	35	-	late	open	0	AA	101	84	2017	4	12	COMPLAINT
36	36	-	late	open	0	AA	102	291	2017	4	13	COMPLAINT
37	37	-	late	open	0	BB	103	2314	2017	4	15	COMPLAINT
38	38	-	late	open	0	BB	104	1664	2017	4	16	COMPLAINT
39	39	-	late	open	0	CC	106	291	2017	4	20	COMPLAINT
40	40	-	late	open	0	AA	107	2314	2017	4	20	COMPLAINT
41	41	-	late	open	0	AA	108	291	2017	4	22	COMPLAINT
42	42	-	late	open	0	BB	109	2314	2017	4	24	COMPLAINT
43	43	-	late	open	0	BB	10	1664	2017	4	26	COMPLAINT
44	44	-	late	open	0	CC	100	143	2017	5	7	COMPLAINT
45	45	-	late	open	0	AA	10	67	2017	5	10	COMPLAINT
46	46	-	late	open	0	AA	101	143	2017	5	21	COMPLAINT
47	47	-	late	open	0	BB	101	67	2017	5	22	COMPLAINT
48	48	-	late	open	0	BB	101	199	2017	5	31	COMPLAINT
49	49	-	late	open	0	CC	101	291	2017	5	31	COMPLAINT
50	50	-	late	open	0	DD	107	291	2017	6	12	COMPLAINT
51	51	-	late	open	0	DD	107	1443	2017	6	15	COMPLAINT
52	52	-	late	open	0	AA	104	67	2017	6	20	COMPLAINT
53	53	-	late	open	0	AA	109	291	2017	6	20	COMPLAINT
54	54	-	late	open	0	BB	109	67	2017	6	21	COMPLAINT
55	55	-	late	open	0	BB	109	291	2017	6	23	COMPLAINT
56	56	-	late	open	0	CC	109	199	2017	6	28	COMPLAINT
57	57	-	late	open	0	AA	101	291	2017	7	2	COMPLAINT

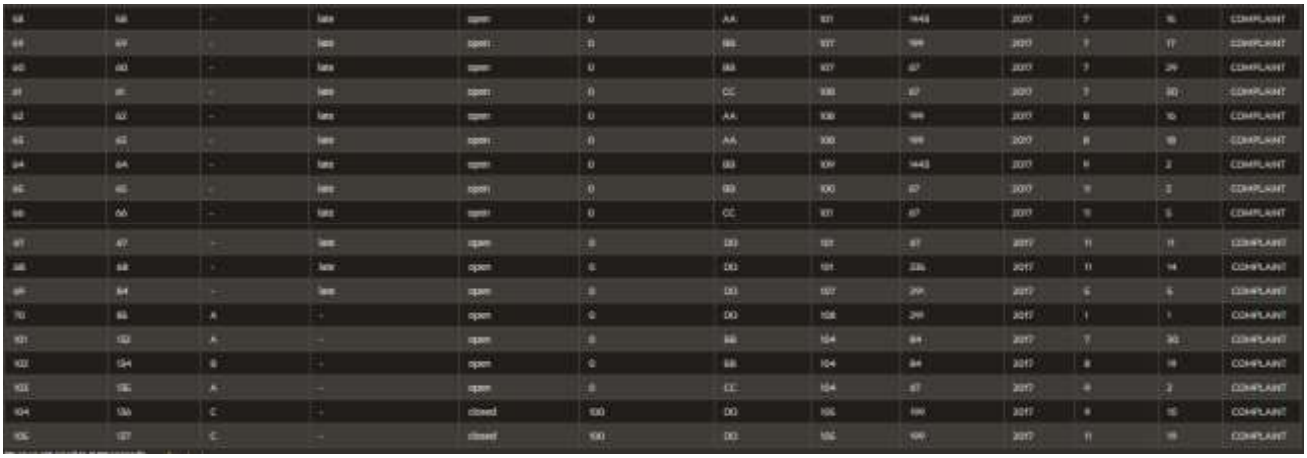


Fig: bad data

In total we have 75 bad data in stage_complaint table, and all the bad date is now migrated into bad_complaint table.

Bad data of customer

Fig: bad data

In total we have 1 bad data in stage_customer able, and all the bad date is now migrated into bad_customer table.

Data_issues table

Data_issues table have all the information of bad data.

In total we have issues with 10610 data. Which means all together we have 10610 bad data in all tables. These are the bad data from stage_flight, stage_complaint and stage_customer tables. Now we have to fix issues of these data, which mean we will convert bad data into good data.

Clean bad data from bad tables

Let's create procedure to clean data of bad tables

```
-- clean bad_flight data
create or replace procedure clean_bad_flight as
begin
    update bad_flight set TAIL_NUMBER = 'unknown' where TAIL_NUMBER is null;
    update bad_flight set FLIGHT_NUMBER = 'unknown' where FLIGHT_NUMBER is null;
    update bad_flight set CANCELLED = 0 where CANCELLED is null;
    update bad_flight set DIVERTED = 0 where DIVERTED is null;
    update bad_flight set ORIGIN_AIRPORT = 'unknown' where ORIGIN_AIRPORT is null;
    update bad_flight set DESTINATION_AIRPORT = 'unknown' where DESTINATION_AIRPORT is null;
    update bad_flight set DB_SOURCE = 'unknown' where DB_SOURCE is null;
    update bad_flight set DIVERTED = 0 where CANCELLED = 1;
    update bad_flight set ORIGIN_AIRPORT = 'JFK' where REGEXP_LIKE(ORIGIN_AIRPORT, '[^J(.*)K$]');
    update bad_flight set DB_SOURCE = 'unknown' where DB_SOURCE is null;
    update bad_flight set CUSTOMER_ID = 0 where CUSTOMER_ID is null;
    update bad_flight set THE_YEAR = 2017 where THE_YEAR is null;
    update bad_flight set THE_MONTH = 1 where THE_MONTH is null;
    update bad_flight set THE_DAY = 1 where THE_DAY is null;
end;
/
```

```
-- clean bad_complaintt data
create or replace procedure clean_bad_complaint as
begin
    update bad_complaint set COMPLAINT_ID = 0 where COMPLAINT_ID is null;
    update bad_complaint set COMPLAINT_TYPE = 'unknown' where COMPLAINT_TYPE is null;
    update bad_complaint set COMPLAINT_DESCRIPTION = 'unknown' where COMPLAINT_DESCRIPTION is null;
    update bad_complaint set COMPLAINT_STATUS = 'unknown' where COMPLAINT_STATUS is null;
    update bad_complaint set COMPENSATION_AMNT = 0 where COMPENSATION_AMNT is null;
    update bad_complaint set ALLOCATED_TO = 'unknown' where ALLOCATED_TO is null;
    update bad_complaint set DB_SOURCE = 'unknown' where DB_SOURCE is null;
    update bad_complaint set COMPENSATION_AMNT = 5000 where complaint_key = 10;
    update bad_complaint set CUSTOMER_ID = 0 where CUSTOMER_ID is null;
    update bad_complaint set FLIGHT_NUMBER = 0 where FLIGHT_NUMBER is null;
    update bad_complaint set THE_YEAR = 2017 where THE_YEAR is null;
    update bad_complaint set THE_MONTH = 1 where THE_MONTH is null;
    update bad_complaint set THE_DAY = 1 where THE_DAY is null;

end;
/

-- clean customer data
create or replace procedure clean_bad_customer as
begin
    update bad_customer set CUSTOMER_TYPE = 'unknown' where CUSTOMER_TYPE is null;
    update bad_customer set CUSTOMER_ZIP_CODE = 'unknown' where CUSTOMER_ZIP_CODE is null;
    update bad_customer set BUSINESS = 'unknown' where BUSINESS is null;
    update bad_customer set CUSTOMER_MILES = 0 where CUSTOMER_MILES is null;
    update bad_customer set CUSTOMER_MILES = 1000 where CUSTOMER_ID=106;
end;

```

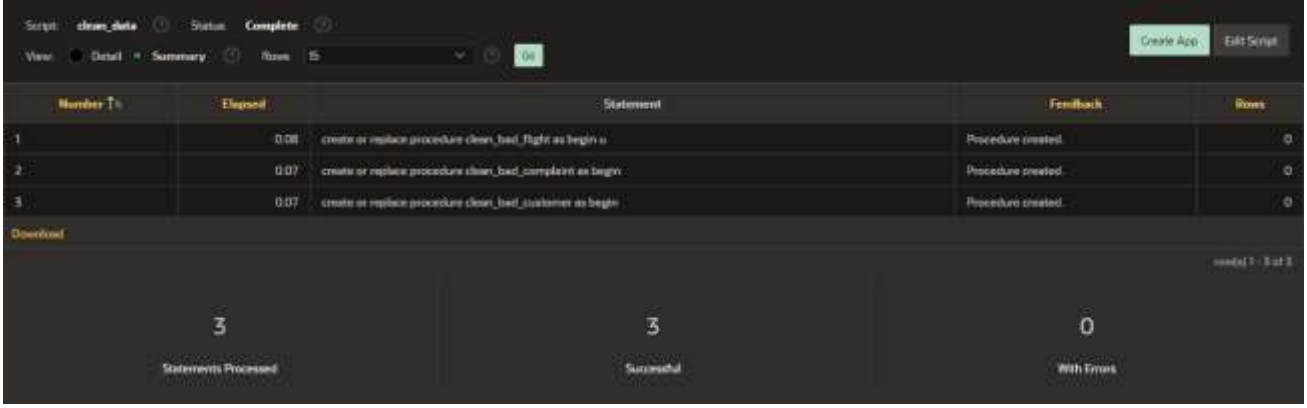


Fig: successfully created procedure to clean data

We have successfully created procedure to convert bad data into good data now the next step is to migrate that converted good data from bad table to good table.

Fire procedures

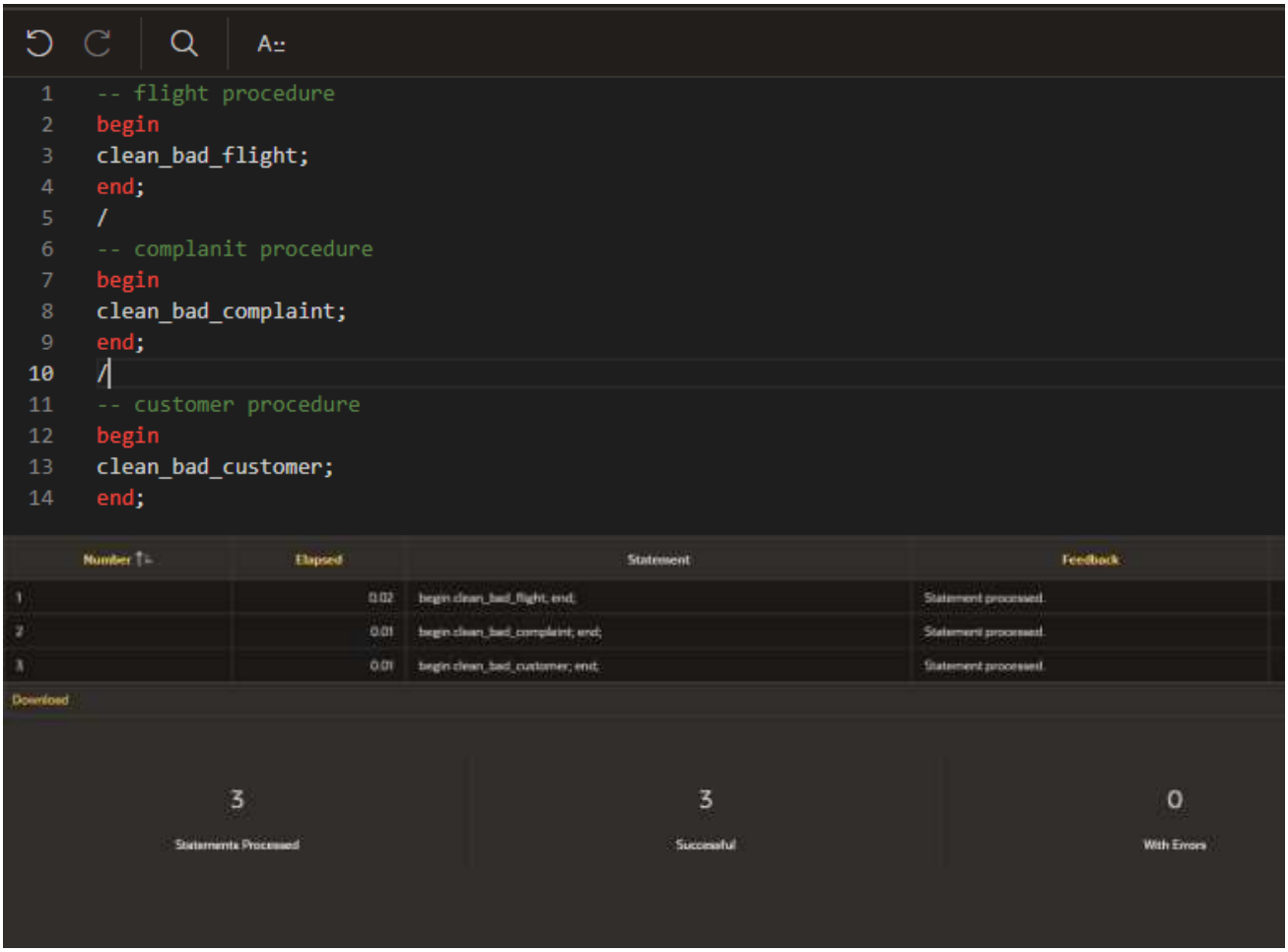


Fig: fire clean_data procedure

Successfully fired the procedure to convert bad data.

Migrate good data from bad table to good table

We have successfully updated the bad data from bad_flight, now let's create procedure to migrate converted good data from bad_flight

```
create or replace procedure migrate_converted_flight_data as
begin
    merge into CLEAN_FLIGHT d
    using BAD_FLIGHT s
    on (s.flight_key = d.flight_key)
    when matched then
    update set
        TAIL_NUMBER = s.tail_number,
        FLIGHT_NUMBER = s.flight_number,
        CANCELLED  = s.cancelled,
        DIVERTED  = s.diverted,
        ORIGIN_AIRPORT = s.origin_airport,
        DESTINATION_AIRPORT = s.destination_airport,
        CUSTOMER_ID = s.CUSTOMER_ID,
        THE_YEAR = s.the_year,
        DB_SOURCE  = s.db_source
    when not matched then
        insert values (s.flight_key, s.tail_number, s.flight_number,  s.cancelled, s.diverted, s.origin_airport, s.destination_airport,s.CUSTOMER_ID, s.the_year,
        s.the_month, s.the_day,
        s.db_source);
end;
 /

 -- for complaint data
 create or replace procedure migrate_converted_complaint_data as
 begin
     merge into CLEAN_COMPLAINT d
     using BAD_COMPLAINT cmt
     on (cmt.complaint_key = d.complaint_key)
     when matched then
     update set
         COMPLAINT_ID = cmt.complaint_id,
         COMPLAINT_TYPE = cmt.complaint_type,
         COMPLAINT_DESCRIPTION   = cmt.complaint_description,
         COMPLAINT_STATUS    = cmt.complaint_status,
         COMPENSATION_AMNT   = cmt.compensation_amnt,
         ALLOCATED_TO = cmt.allocated_to,
         CUSTOMER_ID = cmt.CUSTOMER_ID,
         FLIGHT_NUMBER = cmt.FLIGHT_NUMBER,
         THE_YEAR = cmt.THE_YEAR,
         THE_MONTH = cmt.THE_MONTH,
         THE_DAY = cmt.THE_DAY,
         THE_DAY = cmt.THE_DAY,
         DB_SOURCE = cmt.db_source
     when not matched then
     insert values (cmt.complaint_key, cmt.complaint_id, cmt.complaint_type, cmt.complaint_description, cmt.complaint_status, cmt.compensation_amnt,
     cmt.allocated_to, cmt.CUSTOMER_ID, cmt.FLIGHT_NUMBER, cmt.THE_YEAR, cmt.THE_MONTH, cmt.THE_DAY, cmt.db_source);
 end;
 /

 -- for customer data
 create or replace procedure migrate_converted_customer_data as
 begin
     merge into CLEAN_CUSTOMER d
     using BAD_CUSTOMER cu
     on (cu.customer_key = d.customer_key)
     when matched then
     update  set
         CUSTOMER_ID = cu.customer_id,
         CUSTOMER_TYPE = cu.customer_type,
         CUSTOMER_ZIP_CODE    = cu.customer_zip_code,
         BUSINESS     = cu.business,
         CUSTOMER_MILES  = cu.customer_miles,
         DB_SOURCE = cu.db_source
     when not matched then
         insert values (cu.customer_key, cu.customer_id, cu.customer_type, cu.customer_zip_code,  cu.business, cu.customer_miles, cu.db_source);
 end;
```

Script migrate_clean_data_into_good_data Status Complete

Views Detail Summary Rows 15 Go Create App Edit Script

Number	Elapsed	Statement	Feedback	Rows
1	0.02	create or replace procedure migrate_converted_flight_data as	Procedure created.	0
2	0.01	create or replace procedure migrate_converted_complaint_data	Procedure created.	0
3	0.01	create or replace procedure migrate_converted_customer_data	Procedure created.	0

Download

row(s) 1 - 3 of 3

3	3	0
Statements Processed	Successful	With Errors

Fig: script to create procedure to migrate data

Successfully created procedure to migrate data from bad table to good tables.

Procedures are:

migrate_converted_flight_data to migrate good data from bad_fligh table.

migrate_converted_complaint_data to migrate good data from bad_complaint table. migrate_converted_customer_data to migrate good data from bad_customer table.

Let's fire procedures

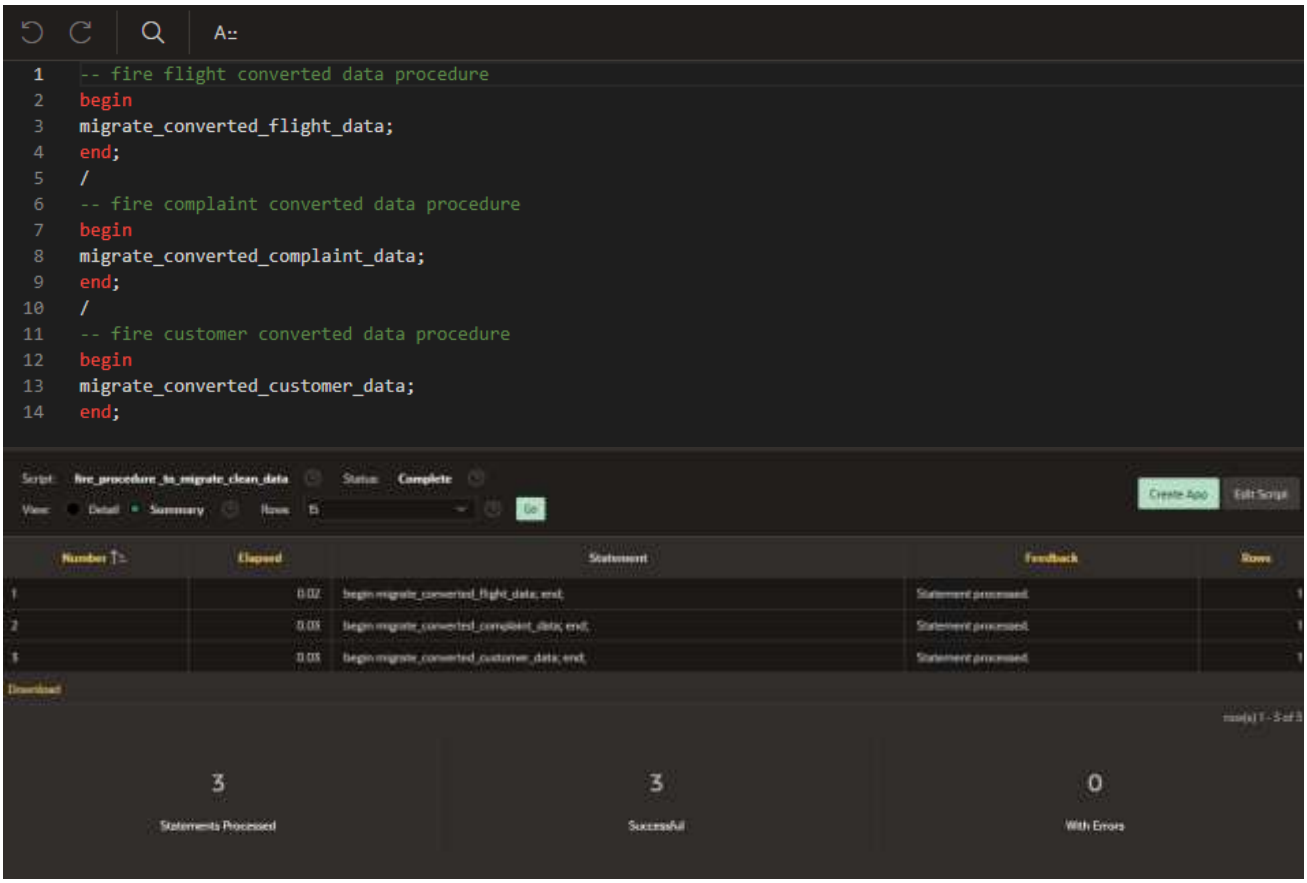


Fig: script to create procedure to migrate data

Successfully fired procedure which means we have successfully migrated good data from bad tables.

Clean data table

Let's count total number of data in good data table. If total number if good data table is equal to total number of data of stagging table then our migration is successful otherwise it's not.

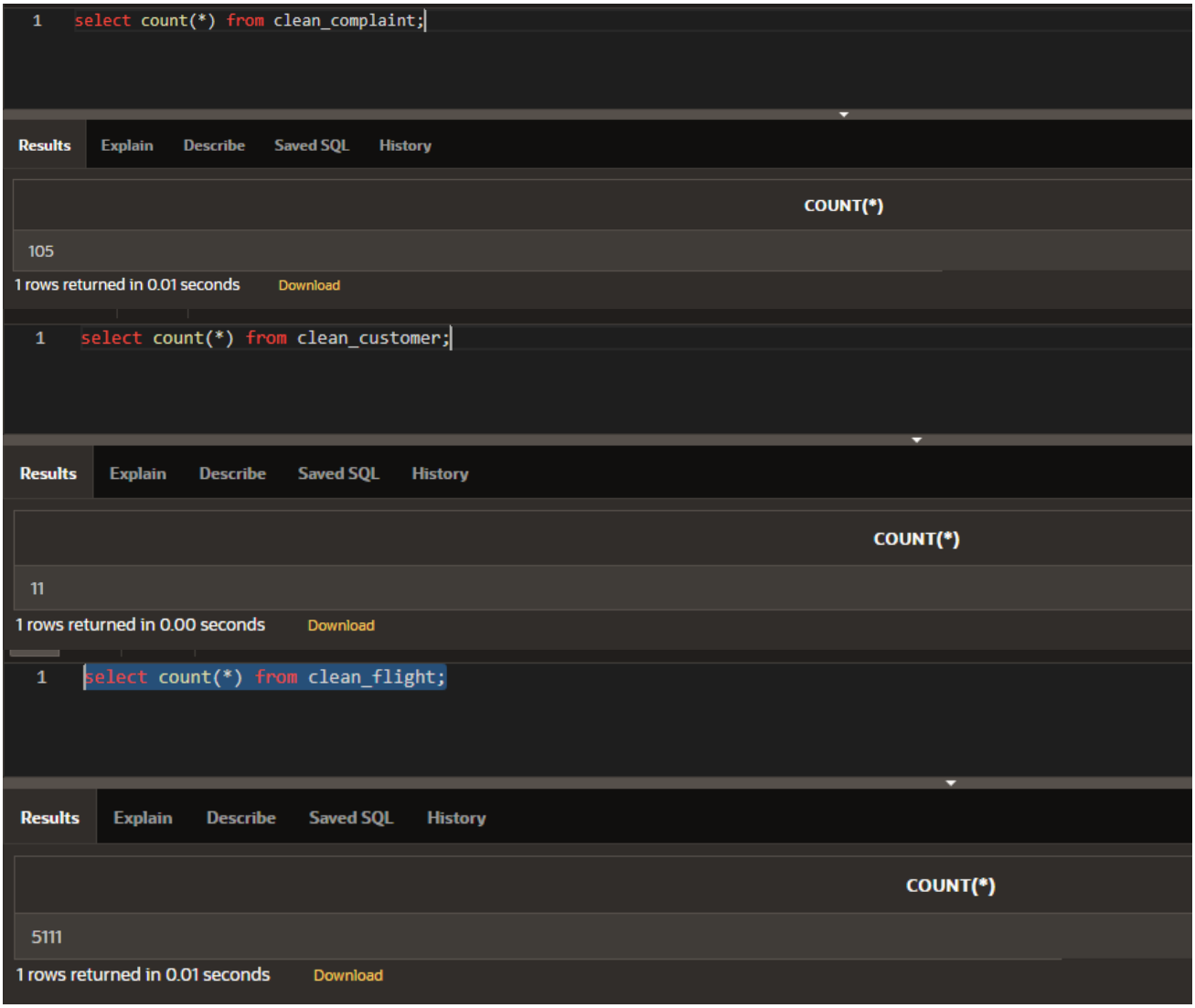


Fig: count data in clean table

Clean table's data is equal to total number data of stagging table. Which means there isn't any data loss.

Confirmation in clean table

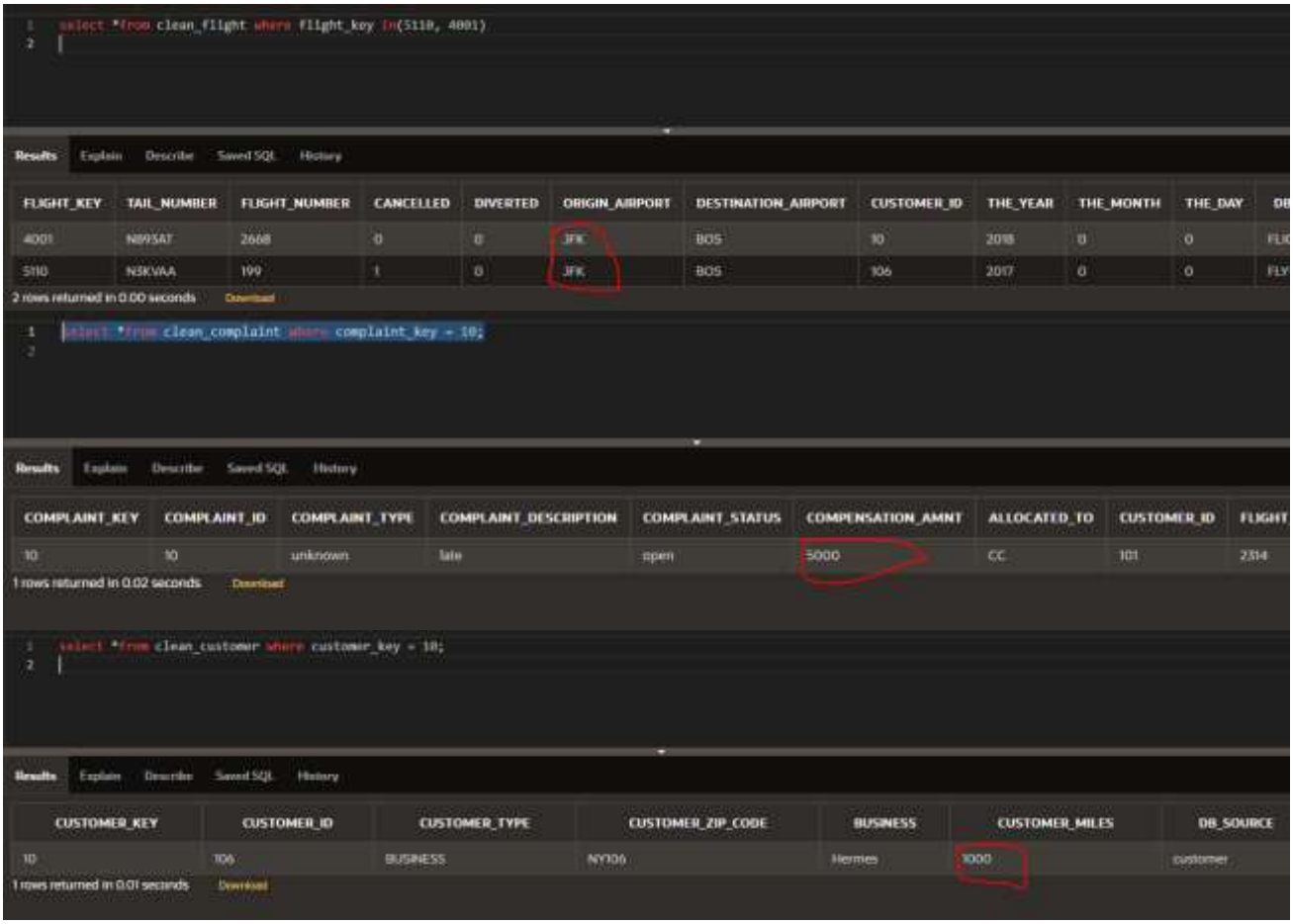


Fig: migrated clean data from bad tables

As seen in the image above we have successfully converted and migrated data from bad tables to good tables.

Converted bad data of bad_flight

```
1  select *from bad_flight;
```

Results Explain Describe Saved SQL History

FLIGHT_KEY	TAIL_NUMBER	FLIGHT_NUMBER	CANCELLED	DIVERTED	ORIGIN_AIRPORT	DESTINATION_AIRPORT	CUSTOMER_ID	THE_YEAR	THE_MONTH	THE_DAY	DB_SOURCE
539	N9CAAA	535	0	0	JFK	CLT	102	2017	1	1	FLIGHT_2017
540	N9AAAA	143	0	0	JFK	BOS	103	2017	1	1	FLIGHT_2017
541	N9GJAA	199	0	0	JFK	BOS	104	2017	1	1	FLIGHT_2017
542	N9KAAA	67	0	0	JFK	AUS	105	2017	1	1	FLIGHT_2017
543	N9GJAA	235	0	0	JFK	BOS	106	2017	1	1	FLIGHT_2017
544	N9AGAA	84	0	0	JFK	BOS	107	2017	1	1	FLIGHT_2017
545	N9JDAA	535	0	0	JFK	CLT	108	2017	1	1	FLIGHT_2017
546	N9BKAA	143	0	0	JFK	BOS	109	2017	1	1	FLIGHT_2017
547	N9MAAA	199	0	0	JFK	BOS	10	2017	1	1	FLIGHT_2017
548	N9CVAA	67	0	0	JFK	AUS	100	2017	1	1	FLIGHT_2017

Fig: Converted bad data of flight

As we see in the above, we have successfully converted bad data of flight into good data.

Converted bad data of bad_complaint

COMPLAINT_KEY	COMPLAINT_ID	COMPLAINT_TYPE	COMPLAINT_DESCRIPTION	COMPLAINT_STATUS	COMPENSATION_AMNT	ALLOCATED_TO	CUSTOMER_ID	FLIGHT_NUMBER	THE_YEAR	THE_MONTH	THE_DAY	DB_SOURCE
1	1	unknown	late	open	0	AA	10	291	2017	1	1	COMPLAINT
2	2	unknown	late	open	0	AA	101	2314	2017	1	3	COMPLAINT
3	3	unknown	late	open	0	BB	103	1096	2017	1	4	COMPLAINT
4	4	unknown	late	open	0	BB	105	291	2017	1	4	COMPLAINT
5	5	unknown	late	open	0	CC	107	2314	2017	1	4	COMPLAINT
6	6	unknown	late	open	0	AA	108	2314	2017	1	6	COMPLAINT
7	7	unknown	late	open	0	AA	107	291	2017	1	7	COMPLAINT
8	8	unknown	late	open	0	BB	10	1096	2017	2	3	COMPLAINT
9	9	unknown	late	open	0	BB	106	2314	2017	2	3	COMPLAINT
10	10	unknown	late	open	5000	CC	101	2314	2017	2	3	COMPLAINT
11	11	unknown	late	open	0	AA	102	1096	2017	2	8	COMPLAINT
12	12	unknown	late	open	0	AA	104	1096	2017	2	10	COMPLAINT
13	13	unknown	late	open	0	BB	107	199	2017	2	14	COMPLAINT
14	14	unknown	late	open	0	BB	108	1096	2017	2	14	COMPLAINT
15	15	unknown	late	open	0	CC	109	2314	2017	2	15	COMPLAINT
16	16	unknown	late	open	0	DD	10	1096	2017	2	21	COMPLAINT
17	17	unknown	late	open	0	EE	100	2314	2017	2	21	COMPLAINT
18	18	unknown	late	open	0	AA	103	84	2017	2	22	COMPLAINT
19	19	unknown	late	open	0	AA	101	2314	2017	2	27	COMPLAINT
20	20	unknown	late	open	0	BB	10	199	2017	3	1	COMPLAINT
21	21	unknown	late	open	0	BB	10	199	2017	3	3	COMPLAINT
22	22	unknown	late	open	0	CC	101	2314	2017	3	3	COMPLAINT
23	23	unknown	late	open	0	AA	101	291	2017	3	7	COMPLAINT
24	24	unknown	late	open	0	AA	102	291	2017	3	11	COMPLAINT
25	25	unknown	late	open	0	BB	103	2314	2017	3	17	COMPLAINT
26	26	unknown	late	open	0	BB	104	291	2017	3	20	COMPLAINT
27	27	unknown	late	open	0	CC	105	84	2017	3	21	COMPLAINT
28	28	unknown	late	open	0	AA	106	2314	2017	3	25	COMPLAINT
29	29	unknown	late	open	0	AA	107	291	2017	3	26	COMPLAINT
30	30	unknown	late	open	0	BB	108	2314	2017	3	26	COMPLAINT
31	31	unknown	late	open	0	BB	109	199	2017	3	28	COMPLAINT
32	32	unknown	late	open	0	CC	10	84	2017	4	4	COMPLAINT
33	33	unknown	late	open	0	DD	100	2314	2017	4	4	COMPLAINT

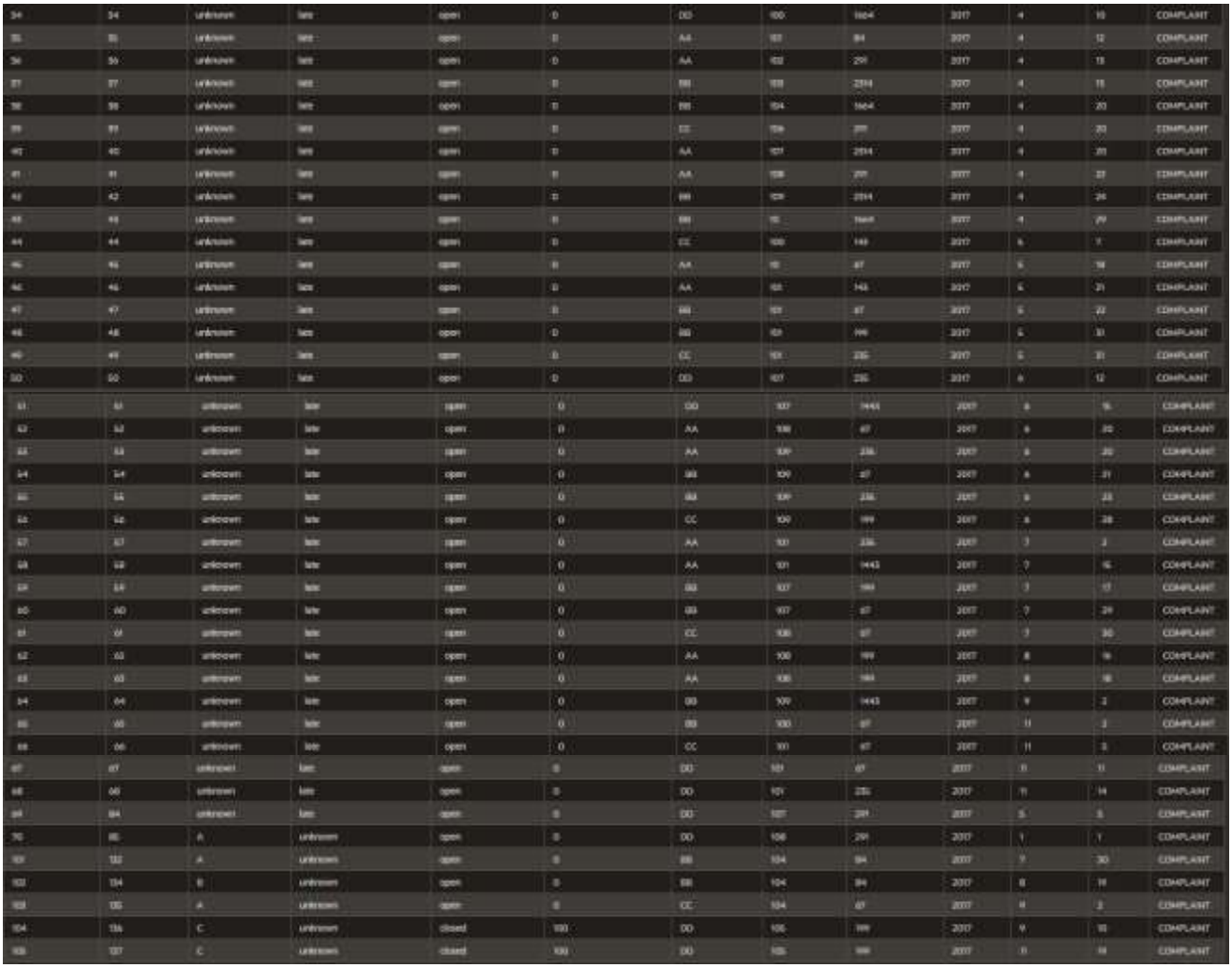
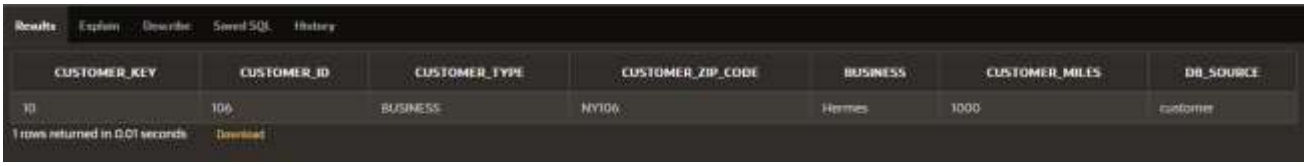


Fig: Converted bad data of complaint

Successfully converted bad data into good data of complaint.

Converted bad data of bad_customer



CUSTOMER_KEY	CUSTOMER_ID	CUSTOMER_TYPE	CUSTOMER_ZIP_CODE	BUSINESS	CUSTOMER_MILES	DB_SOURCE
10	106	BUSINESS	NY106	Hermes	1000	customer

1 rows returned in 0.01 seconds Download

Fig: Converted bad data of customer

Successfully converted bad data into good data of customer.

Update data_issues table

To update data_issues table let's create procedure

```
-- update data issues of flight
create or replace procedure update_data_issue_of_flight as
begin
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 10;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 20;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 30;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 40;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 50;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 60;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 70;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 80;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 90;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 100;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 110;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 111;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 112;
end;
/

-- update data issues of complaint
create or replace procedure update_data_issue_of_complaint as
begin
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 1;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 2;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 3;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 4;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 5;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 6;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 7;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 8;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 9;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 10;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 11;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 12;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 13;
end;
/

-- update data issues of customer
create or replace procedure update_data_issue_of_customer as
begin
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 100;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 200;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 300;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 400;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 500;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 600;
    update data_issues set ISSUE_STATUS = 'fixed', STATUS_UPDATE_DATE = SYSDATE where DATA_ERROR_CODE = 700;
end;

Script update_data_issues_table  Status Complete
View Detail Summary  Rows  Go  Create App  Edit Script

| Number | Elapsed | Statement | Feedback | Rows |
|---|---|---|---|---|
| 1 | 0.07 | create or replace procedure update_data_issue_of_flight as b | Procedure created. | 0 |
| 2 | 0.02 | create or replace procedure update_data_issue_of_complaint a | Procedure created. | 0 |
| 3 | 0.03 | create or replace procedure update_data_issue_of_customer as | Procedure created. | 0 |

Download

3 Statements Processed
3 Successful
0 With Errors
```

Fig: script to create procedure to update data_issues table

As seen in the image above we have successfully created procedure to update data_issues table.

Fire procedure

Fig: successfully fired procedure

Successfully fired procedure, which means we have updated the data_issues table successfully. To confirm it lets checkout data_issues table.

Confirm update

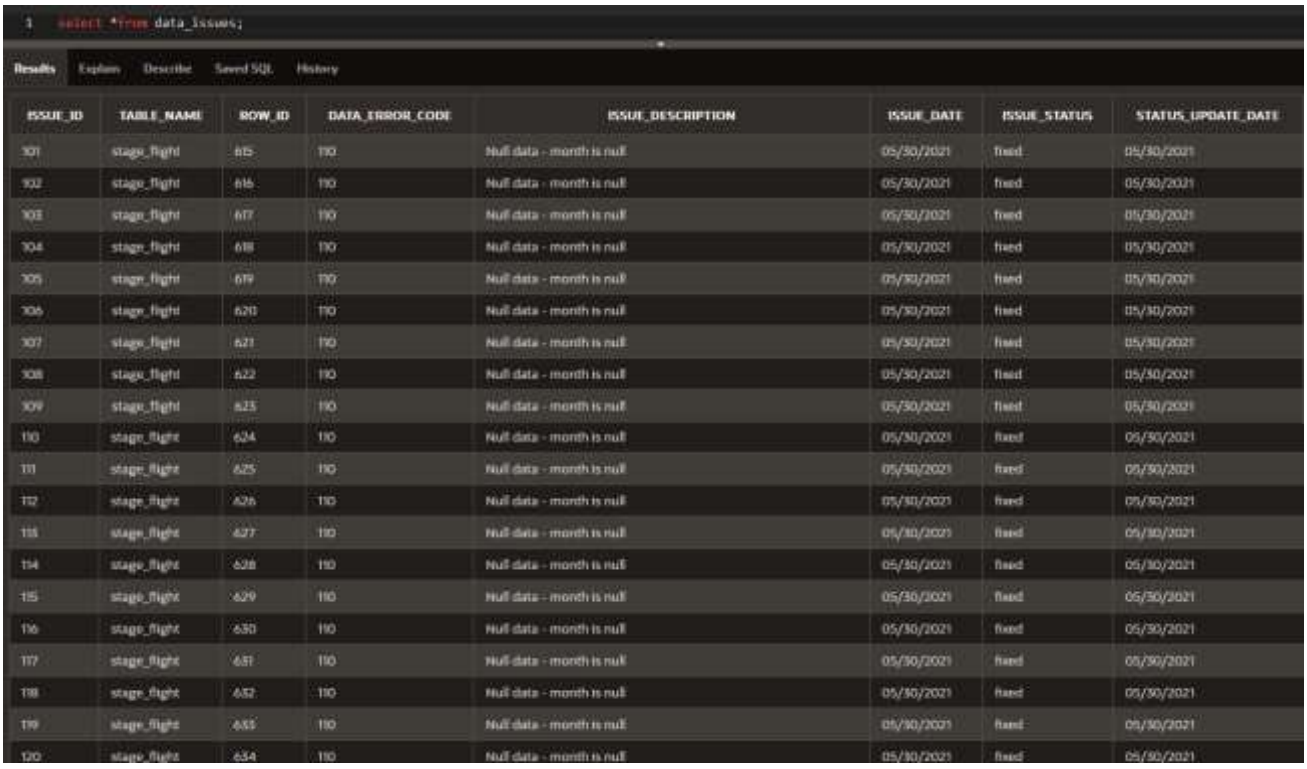


Fig: successfully update data_issues table

Successfully updated the bad data information in data_issues data table. So, in future if anyone wants to know the error description, when it was fixed then they can checkout this table.

Transformation

Create transformation table

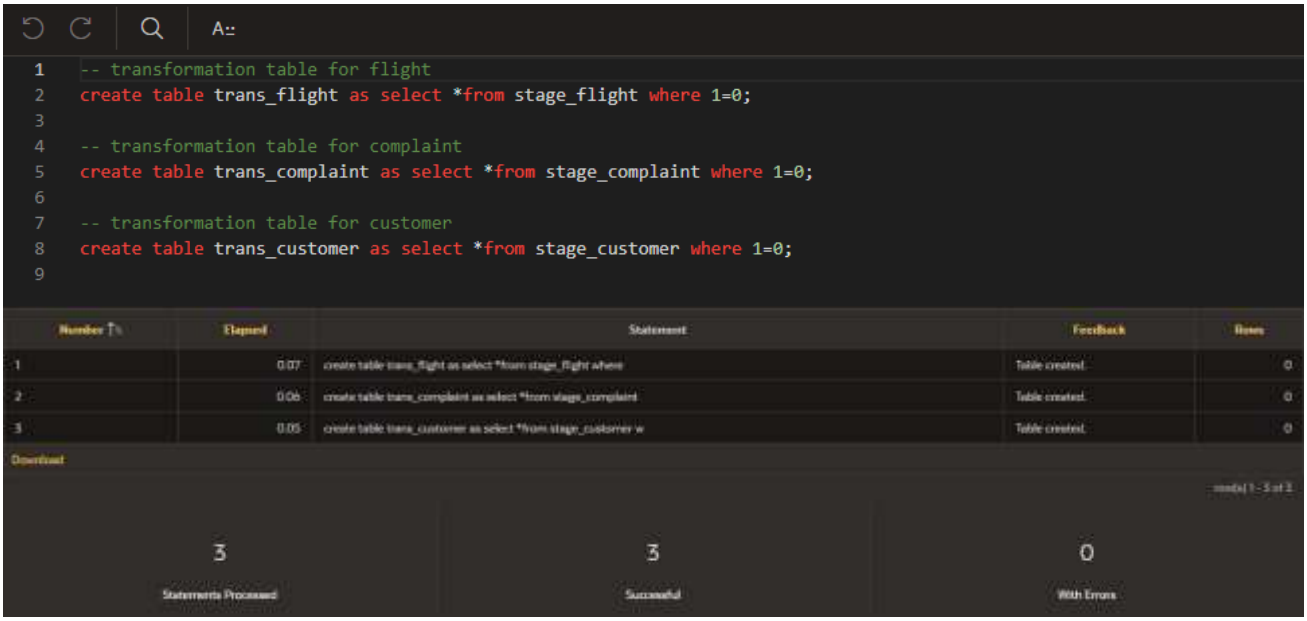


Fig: successfully created transform table

Successfully created transform table

Tables are: trans_flight to transfer data from clean_fight table, trans_complaint to transform data from clean_complaint table, trans_customer to transform from clean_customer table.

Create procedures to migrate data into transform table

```
create or replace procedure trans_clean_flight as
begin
    merge into trans_flight d
    using clean_flight s
    on (s.flight_key = d.flight_key)
    when matched then
    update set
        TAIL_NUMBER = s.tail_number,
        FLIGHT_NUMBER = s.flight_number,
        CANCELLED  = s.cancelled,
        DIVERTED  = s.diverted,
        ORIGIN_AIRPORT = s.origin_airport,
        DESTINATION_AIRPORT = s.destination_airport,
        CUSTOMER_ID = s.CUSTOMER_ID,
        THE_YEAR = s.the_year,
        DB_SOURCE  = s.db_source
    when not matched then
        insert values (s.flight_key, s.tail_number, s.flight_number,  s.cancelled, s.diverted, s.origin_airport, s.destination_airport,s.CUSTOMER_ID,
        s.the_year, s.the_month, s.the_day,
        s.db_source);
end;
/

-- procedure for complaint transformation table
create or replace procedure trans_clean_complaint as
begin
    merge into trans_complaint d
    using clean_complaint cmt
    on (cmt.complaint_key = d.complaint_key)
    when matched then
    update set
        COMPLAINT_ID = cmt.complaint_id,
        COMPLAINT_TYPE = cmt.complaint_type,
        COMPLAINT_DESCRIPTION   = cmt.complaint_description,
        COMPLAINT_STATUS    = cmt.complaint_status,
        COMPENSATION_AMNT   = cmt.compensation_amnt,
        ALLOCATED_TO = cmt.allocated_to,
        CUSTOMER_ID = cmt.CUSTOMER_ID,
        FLIGHT_NUMBER = cmt.FLIGHT_NUMBER,
        THE_YEAR = cmt.THE_YEAR,
        THE_MONTH = cmt.THE_MONTH,
        THE_DAY = cmt.THE_DAY,
        DB_SOURCE = cmt.db_source
    when not matched then
    insert values (cmt.complaint_key, cmt.complaint_id, cmt.complaint_type, cmt.complaint_description, cmt.complaint_status, cmt.compensation_amnt,
    cmt.allocated_to, cmt.CUSTOMER_ID, cmt.FLIGHT_NUMBER, cmt.THE_YEAR, cmt.THE_MONTH, cmt.THE_DAY, cmt.db_source);
end;
/

-- procedure for complaint transformation table
create or replace procedure trans_clean_customer as
begin
    merge into trans_customer d
    using clean_customer cu
    on (cu.customer_key = d.customer_key)
    when matched then
    update  set
        CUSTOMER_ID = cu.customer_id,
        CUSTOMER_TYPE = cu.customer_type,
        CUSTOMER_ZIP_CODE   = cu.customer_zip_code,
        BUSINESS    = cu.business,
        CUSTOMER_MILES  = cu.customer_miles,
        DB_SOURCE = cu.db_source
    when not matched then
        insert values (cu.customer_key, cu.customer_id, cu.customer_type, cu.customer_zip_code,  cu.business, cu.customer_miles, cu.db_source);
end;
```

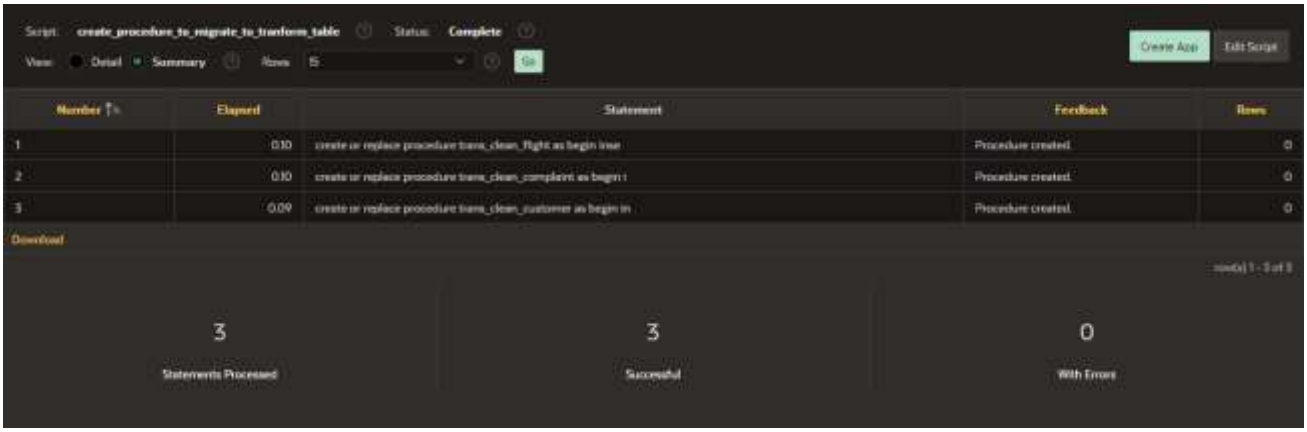


Fig: successfully created procedure

Successfully procedure to migrate data from clean table to transform table.

Fire procedures

```
1   -- flight tans procedure
2   begin
3   trans_clean_flight;
4   end;
5   /
6   -- complaint tans procedure
7   begin
8   trans_clean_complaint;
9   end;
10  /
11  -- customer tans procedure
12  begin
13  trans_clean_customer;
14  end;
```

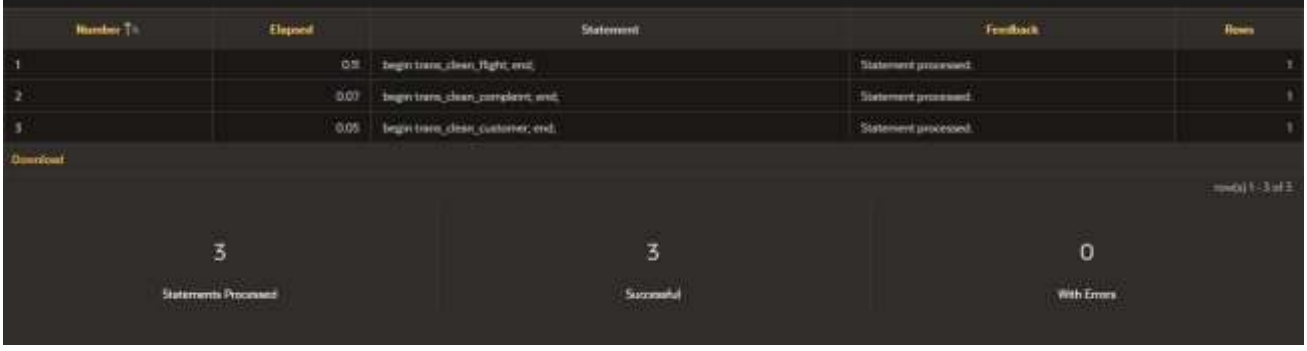


Fig: successfully fired procedure

Successfully fired procedure to migrate data from clean table to transform table. Which means we have successfully migrated data into transform table.

Migrate data from transformation table to dim table

Now I am going to load the data from transformation table to dimension table that we have recently added into tranform table.

Create procedure to migrate into dim tables

```
-- To load into dim flights
create or replace procedure load_into_dimFlights as
begin
    insert into dim_flights(FLIGHT_ID, FLIGHT_NUMBER, CANCELLED, DIVERTE, ORIGIN_AIRPORT, DESTINATION_AIRPORT)
    select  FLIGHT_KEY, FLIGHT_NUMBER, CANCELLED, DIVERTED, ORIGIN_AIRPORT, DESTINATION_AIRPORT from trans_flight;
end;
/

-- To load into dim complaint
create or replace procedure load_into_dimComplaints as
begin
    insert into DIM_COMPALINT(COMPLAINT_ID, COMPLAINT_TYPE, DESCRIPTION, COMPLAIN_STATUS, COMPENSATION_AMOUNT, ALLOCATED_TO)
    select COMPLAINT_ID, COMPLAINT_TYPE, COMPLAINT_DESCRIPTION, COMPLAINT_STATUS, COMPENSATION_AMNT, ALLOCATED_TO from trans_complaint;
end;
/

 -- To load into dim flights
 create or replace procedure load_into_dimCustomers as
 begin
    insert into dim_customers(CUSTOMER_ID, CUSTOMER_TYPE, CUSTOMER_ZIP_CODE, CUSTOMER_MILES)
    select  CUSTOMER_ID, CUSTOMER_TYPE, CUSTOMER_ZIP_CODE, CUSTOMER_MILES from trans_customer;
 end;
 /

 -- To load into dim time
 create or replace procedure load_into_dimTime as
 begin
    insert into dim_time(DAY, MONTH, YEAR)
    select distinct THE_DAY, THE_MONTH, THE_YEAR from trans_flight;
 end;
```

Script: create_procedure_to_migrate_into_dim_tables Status: Complete

View: Detail / Summary Rows: 15 Create App Edit Script

Number	Elapsed	Statement	Feedback	Rows
1	0.01	create or replace procedure load_into_dimFlights as begin	Procedure created.	0
2	0.01	create or replace procedure load_into_dimComplaints as begin	Procedure created.	0
3	0.01	create or replace procedure load_into_dimCustomers as begin	Procedure created.	0
4	0.00	create or replace procedure load_into_dimTime as begin	Procedure created.	0

Download

rows(s) 1 - 4 of 4

4 Statements Processed 4 Successful 0 With Errors

Fig: successfully created procedure

Successfully created procedure to migrate data into dim tables.

Fire procedures

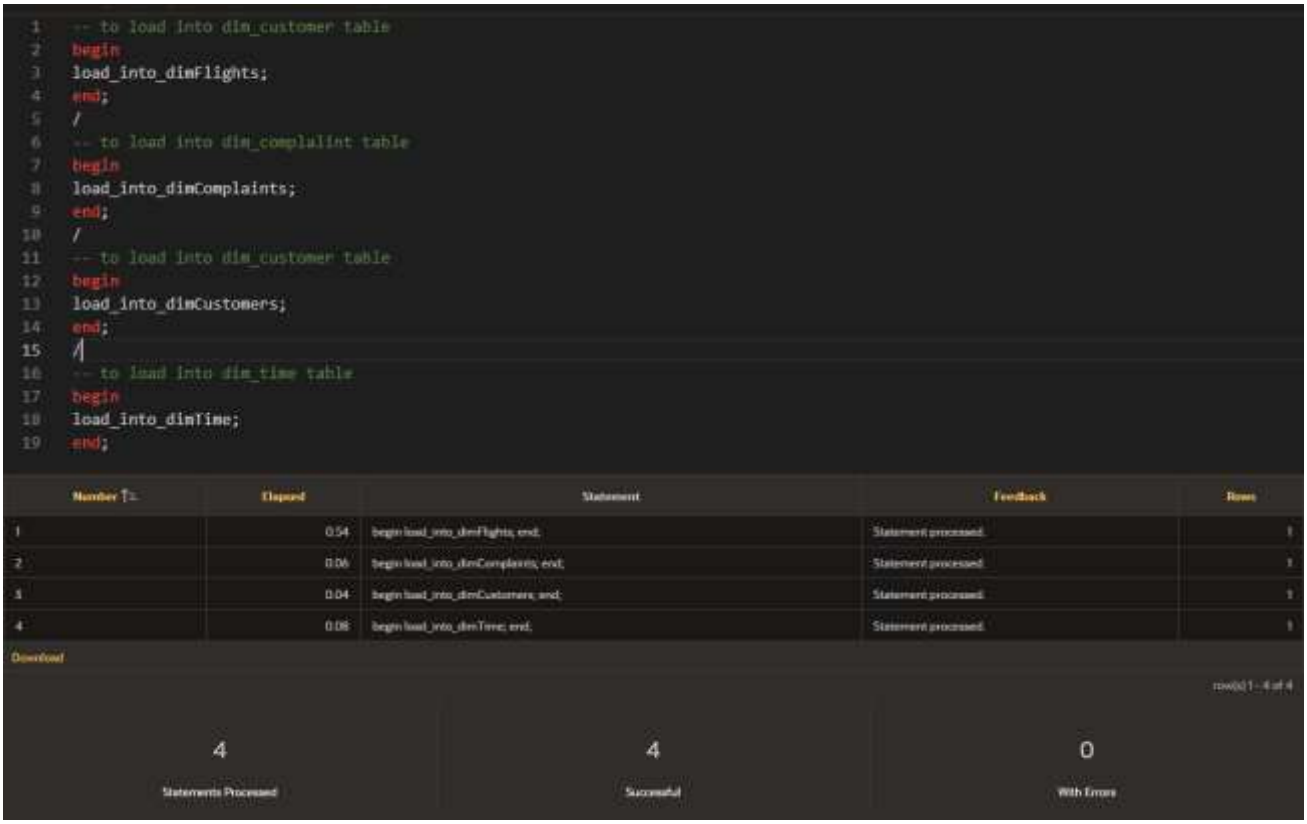
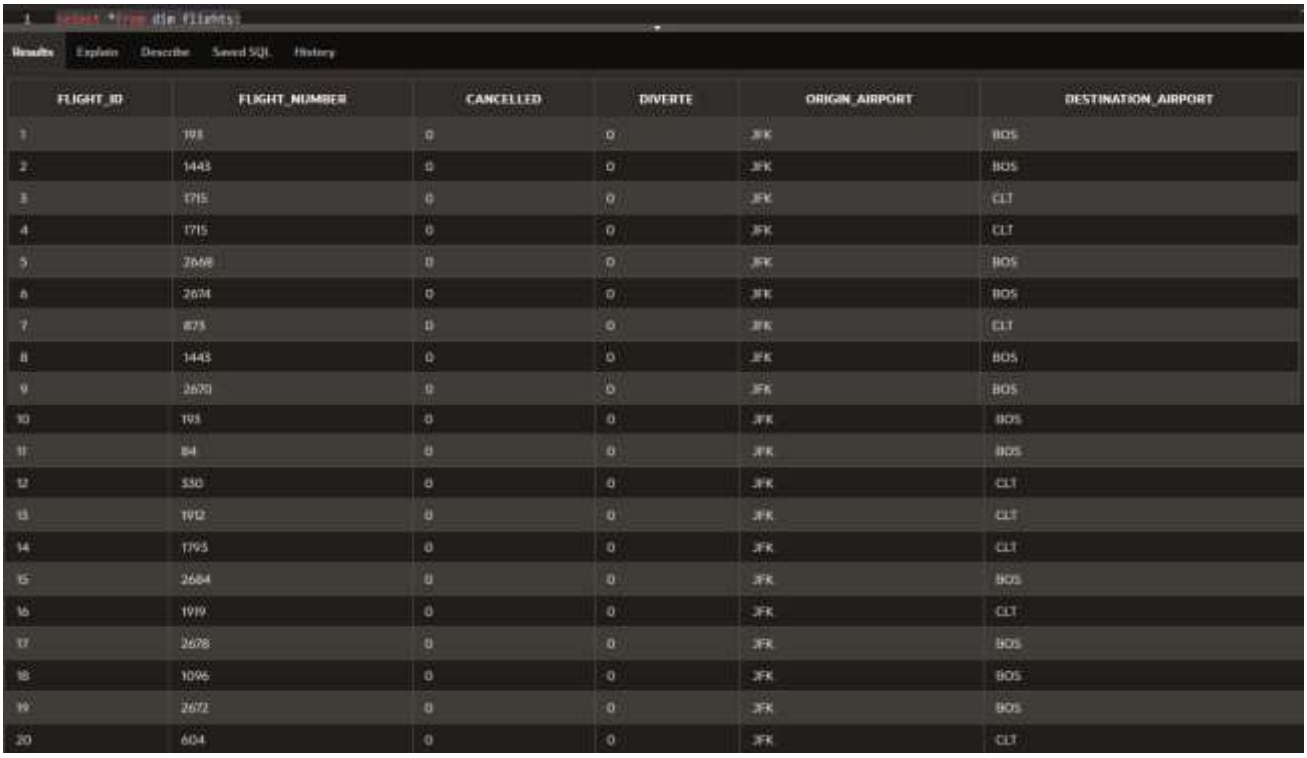


Fig: successfully fired procedure

Successfully fired procedure to migrate data from transform table to dim tables. Which means we have successfully migrated data into dim table

Dim_flight table

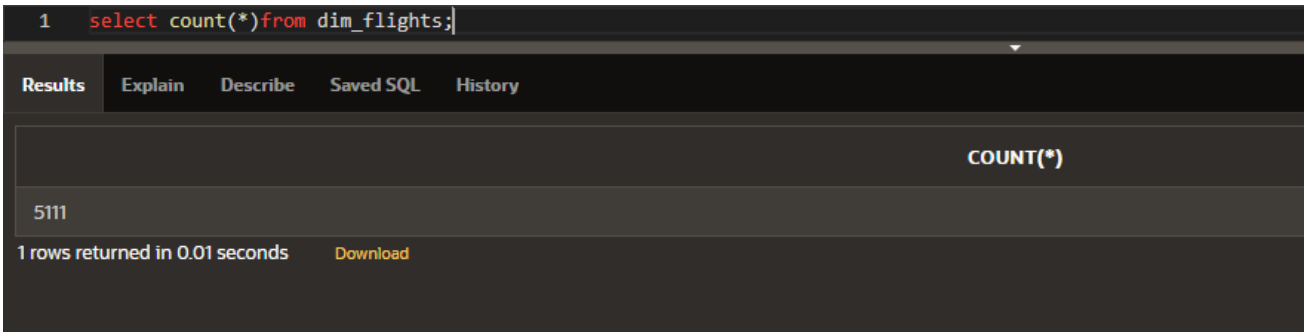
Let's checkout data in dim_flight



FLIGHT_ID	FLIGHT_NUMBER	CANCELLED	DIVERTE	ORIGIN_AIRPORT	DESTINATION_AIRPORT
1	193	0	0	JFK	BOS
2	1443	0	0	JFK	BOS
3	1715	0	0	JFK	CLT
4	1715	0	0	JFK	CLT
5	2668	0	0	JFK	BOS
6	2674	0	0	JFK	BOS
7	875	0	0	JFK	CLT
8	1443	0	0	JFK	BOS
9	2670	0	0	JFK	BOS
10	193	0	0	JFK	BOS
11	84	0	0	JFK	BOS
12	550	0	0	JFK	CLT
13	1912	0	0	JFK	CLT
14	1795	0	0	JFK	CLT
15	2684	0	0	JFK	BOS
16	1919	0	0	JFK	CLT
17	2678	0	0	JFK	BOS
18	1096	0	0	JFK	BOS
19	2672	0	0	JFK	BOS
20	604	0	0	JFK	CLT

Fig: dim_flight data

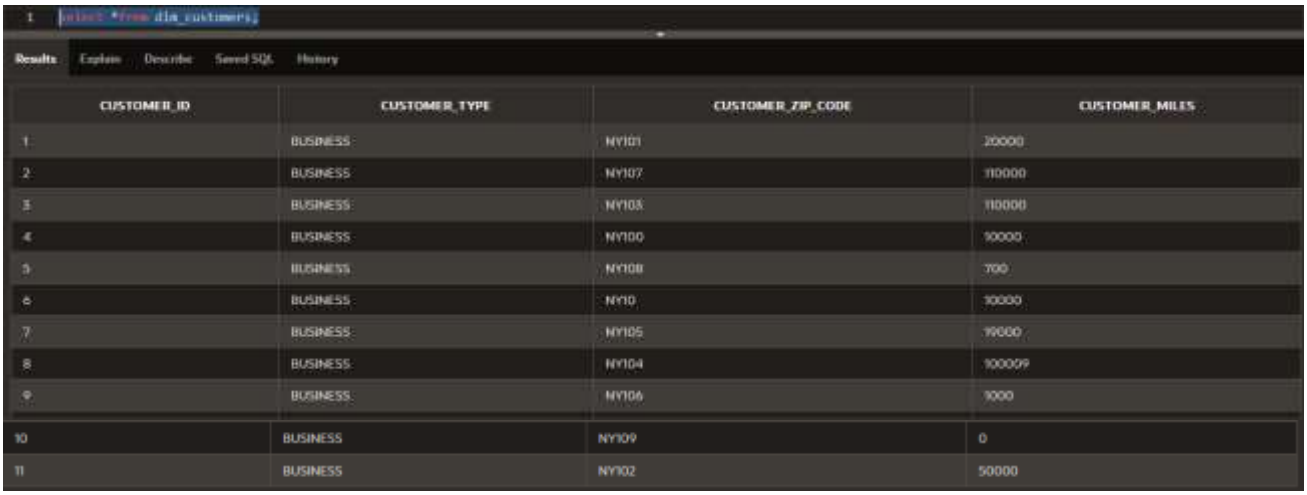
Let's check if there is any data loss or not



No data was loss while comparing dimension table with stagging table

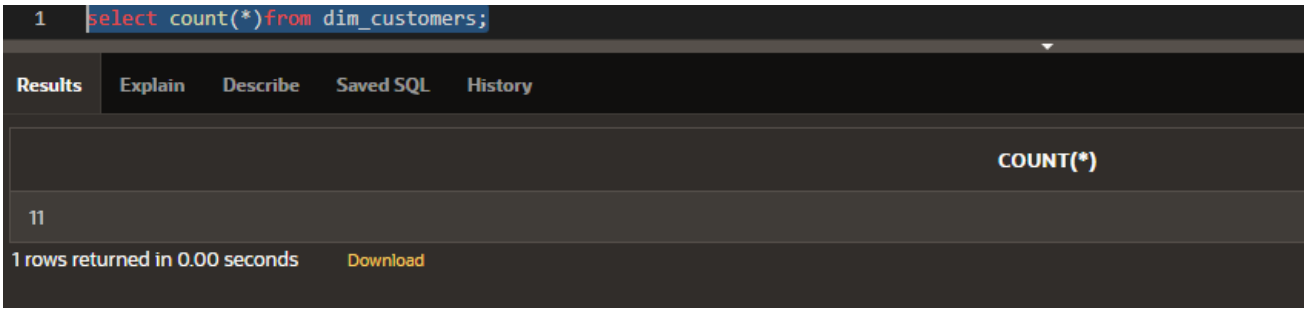
Dim_customers table

Let's checkout data in dim_customers



CUSTOMER_ID	CUSTOMER_TYPE	CUSTOMER_ZIP_CODE	CUSTOMER_MILES
1	BUSINESS	NY101	20000
2	BUSINESS	NY107	110000
3	BUSINESS	NY103	110000
4	BUSINESS	NY100	10000
5	BUSINESS	NY108	700
6	BUSINESS	NY10	10000
7	BUSINESS	NY105	19000
8	BUSINESS	NY104	100009
9	BUSINESS	NY106	1000
10	BUSINESS	NY109	0
11	BUSINESS	NY102	50000

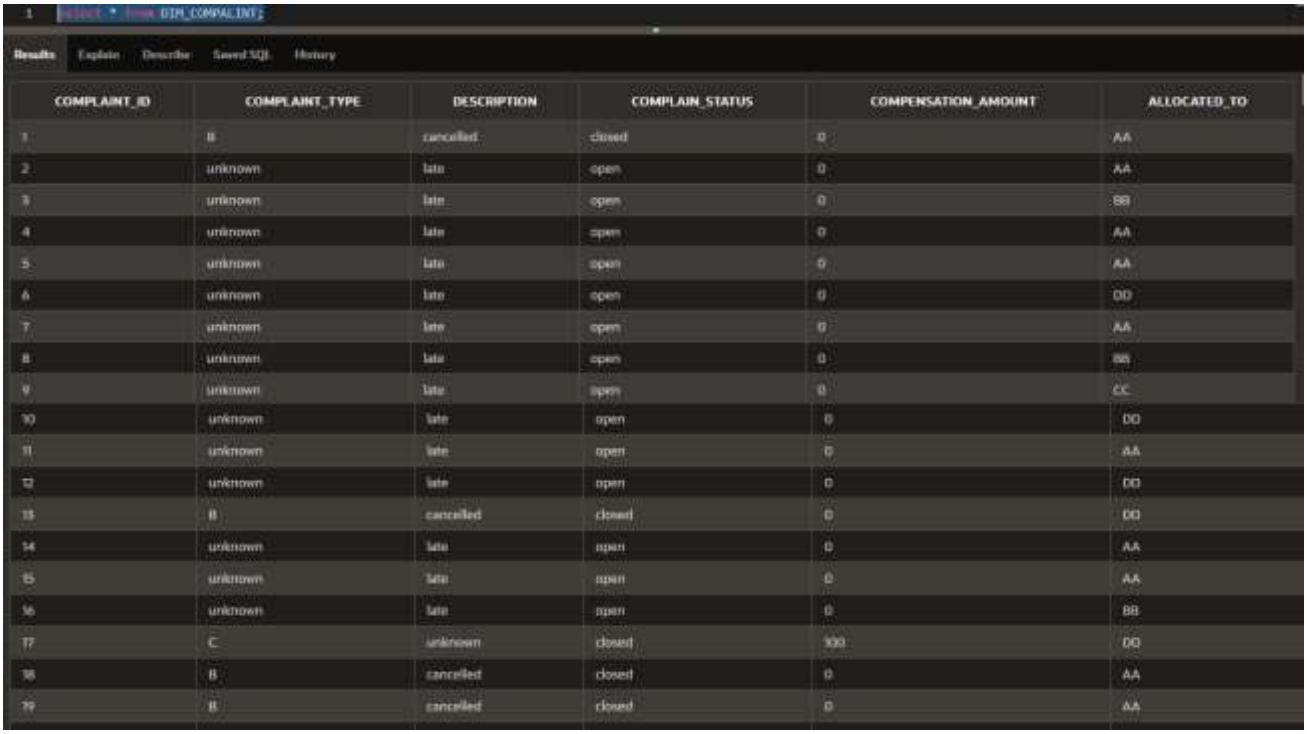
Let's check if there is any data loss or not



No data was loss while comparing dim table with stagging table

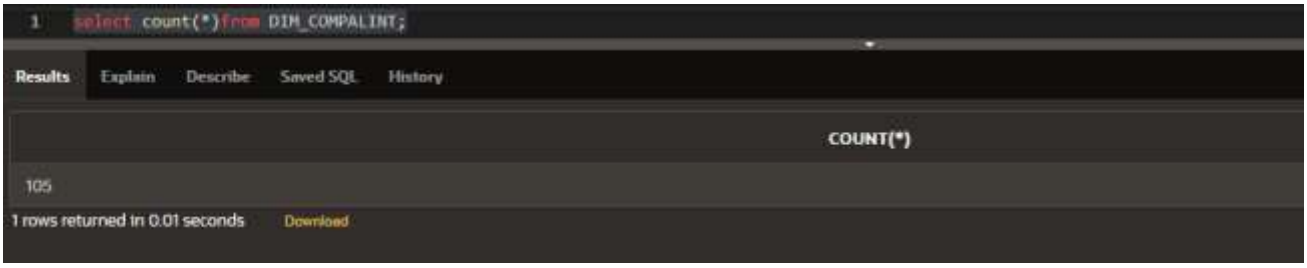
Dim_compilant table

Let's checkout data in dim_customers



COMPLAINT_ID	COMPLAINT_TYPE	DESCRIPTION	COMPLAIN_STATUS	COMPENSATION_AMOUNT	ALLOCATED_TO
1	B	cancelled	closed	0	AA
2	unknown	late	open	0	AA
3	unknown	late	open	0	BB
4	unknown	late	open	0	AA
5	unknown	late	open	0	AA
6	unknown	late	open	0	DD
7	unknown	late	open	0	AA
8	unknown	late	open	0	BB
9	unknown	late	open	0	CC
10	unknown	late	open	0	DD
11	unknown	late	open	0	AA
12	unknown	late	open	0	DD
13	B	cancelled	closed	0	DD
14	unknown	late	open	0	AA
15	unknown	late	open	0	AA
16	unknown	late	open	0	BB
17	C	unknown	closed	100	DD
18	B	cancelled	closed	0	AA
19	B	cancelled	closed	0	AA

Let's check if there is any data loss or not



No data was loss while comparing dimension table with stagging table

Migrate data into fact table

Create procedure to load data into fact_Table

```
create or replace procedure load_into_factTable as
begin
    merge into FACT_TABLE flyu_fact
    using(
        select df.flight_key, dc.complaint_key, dcust.customer_key, dt.time_id,
        sum(df.CANCELLED) as TOTAL_CANCELLED_FLIGHTS,
        count(dc.complaint_id) as TOTAL_COMPLAINTS,
        count(dcust.CUSTOMER_ID) TOTAL_NUMBER_OF_CUSTOMERS,
        sum(dc.COMPENSATION_AMOUNT) as COMPENSATION_AMOUNT
        from trans_flight tf
        join trans_customer tcust on tf.customer_id = tcust.customer_id
        join trans_complaint tc on tc.customer_id = tcust.customer_id and tf.flight_number = tc.flight_number
        join dim_flights df on df.flight_key = tf.flight_key
        join dim_customers dcust on dcust.customer_id = tcust.customer_id
        join dim_complaint dc on dc.complaint_id = tc.complaint_id
        join dim_time dt on dt.year = tf.the_year and dt.month = tf.the_month and dt.day = tf.the_day
        group by df.flight_key, dc.complaint_key, dcust.customer_key, dt.time_id
    ) T

    on (
        flyu_fact.TIME_ID = T.TIME_ID
        and flyu_fact.COMPLAINT_KEY = T.complaint_key
        and flyu_fact.CUSTOMER_KEY = T.customer_key
        and flyu_fact.FLIGHT_KEY = T.flight_key
    )
    when matched then
    update set
        flyu_fact.TOTAL_CANCELLED_FLIGHTS = T.TOTAL_CANCELLED_FLIGHTS,
        flyu_fact.TOTAL_NUMBER_OF_CUSTOMERS = T.TOTAL_NUMBER_OF_CUSTOMERS,
        flyu_fact.TOTAL_COMPLAINTS = T.TOTAL_COMPLAINTS,
        flyu_fact.COMPENSATION_AMNT = T.COMPENSATION_AMOUNT
    when not matched then
    insert (TIME_ID, COMPLAINT_KEY, CUSTOMER_KEY, FLIGHT_KEY, TOTAL_CANCELLED_FLIGHTS, TOTAL_NUMBER_OF_CUSTOMERS, TOTAL_COMPLAINTS, COMPENSATION_AMNT)
    VALUES (T.time_id, T.COMPLAINT_KEY, T.customer_key, T.flight_key,  T.TOTAL_CANCELLED_FLIGHTS, T.TOTAL_NUMBER_OF_CUSTOMERS, T.TOTAL_COMPLAINTS, T.COMPENSATION_AMOUNT);
end;
```

Number	Elapsed	Statement	Feedback	Rows
1	0.05	create or replace procedure load_into_factTable as begin	Procedure created.	0

Download

1 row(s) 1 - 1 of 1

Statements Processed	Successful	With Errors
1	1	0

Fig: Successfully created procedure

Successfully created procedure to load data into fact table.

Fired procedure

Fig: successfully fired procedure

Successfully fired procedure to load data into fact tables. Which means we have successfully migrated data into dim table.

Data in fact table (fact_table)

FACTTABLE_ID	TIME_ID	COMPLAINT_KEY	CUSTOMER_KEY	FLIGHT_KEY	TOTAL_CANCELLED_FLIGHTS	TOTAL_NUMBER_OF_CUSTOMERS	TOTAL_COMPLAINTS	COMPENSATION_AMO
1	284	81	2	241	0	1	1	0
2	284	99	10	1858	0	1	1	0
3	284	105	6	856	0	1	1	0
4	284	7	1	1014	0	1	1	0
5	128	75	9	5080	0	1	1	0
6	284	66	1	1567	0	1	1	0
7	284	9	7	597	0	1	1	0
8	284	16	2	1263	0	1	1	0
9	284	97	7	272	1	1	1	0
10	46	101	8	5068	1	1	1	0
11	284	101	8	651	0	1	1	0
12	284	103	10	414	0	1	1	0
13	284	62	9	82	0	1	1	0
14	284	65	1	2631	0	1	1	0
15	284	50	3	248	0	1	1	0
16	284	31	5	864	0	1	1	0
17	284	65	1	1023	0	1	1	0
18	284	6	2	2288	1	1	1	0
19	128	29	7	5079	0	1	1	0

Fig: Data in fact table

Above shown is the sample of data in fact table.

Reports

OLAP is mainly use for analytical purpose. OLAP database stores historical data which are recorded in the OLTP. Using the OLAP system we can extract the information from a large amount of database and analyze it for decision making and future insights. User can fire the query again and extract the data for analyzing purpose. The example of OLAP database is viewing report of financial, budgeting, sales, marketing et. OLAP database can be linked with the BI tools. In this report we are going to use tableau with our data to create some reports, which could help company to make better decision and insight future.

Report 1: Total number of customers

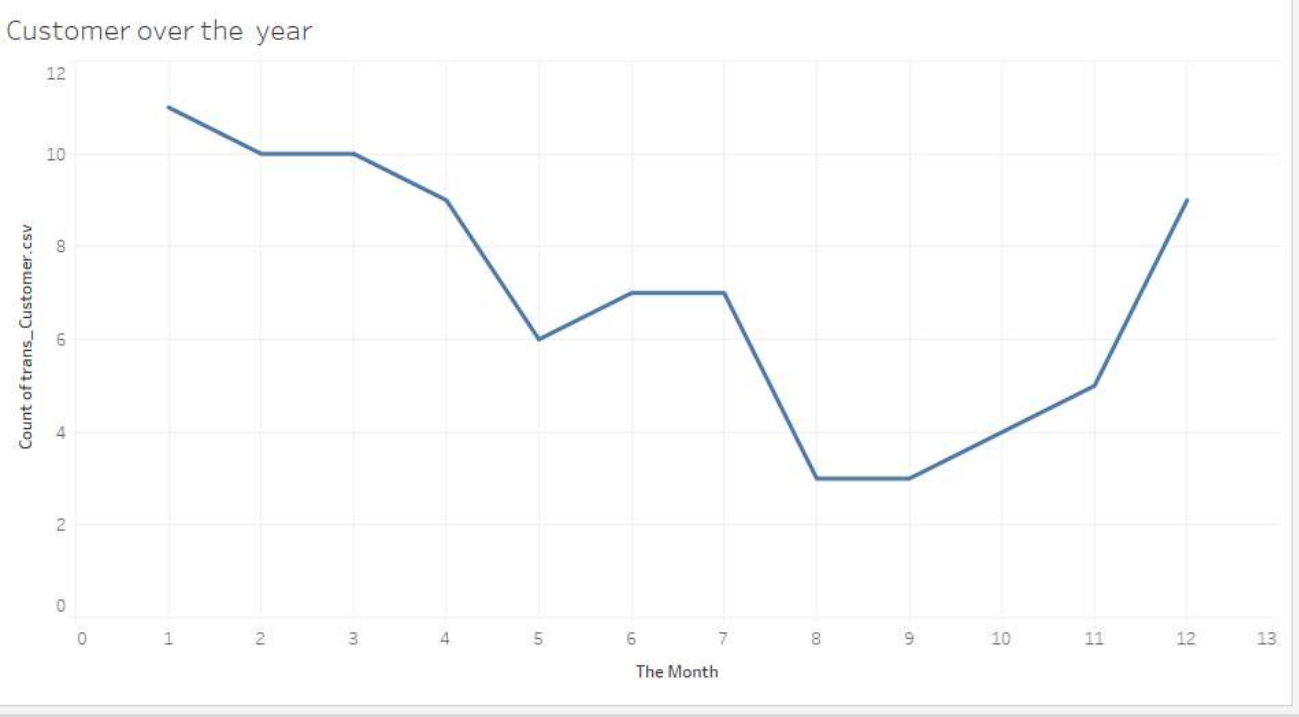


Fig 1: Number of customers over with respect to month

As seen in the above graph the number of customers is high at the first and last month of the year (high in January and December). From august to September there is least number of customers. From the quick analyze customers travelled more during new year time. So, to increase more and more customers we can provide them additional facility like air ticket discount offer for new year, prize for lucky draw winner etc., which can attract more customers. From September to October there is least number of customers, to mitigate this

Report 2: Airlines with most complain

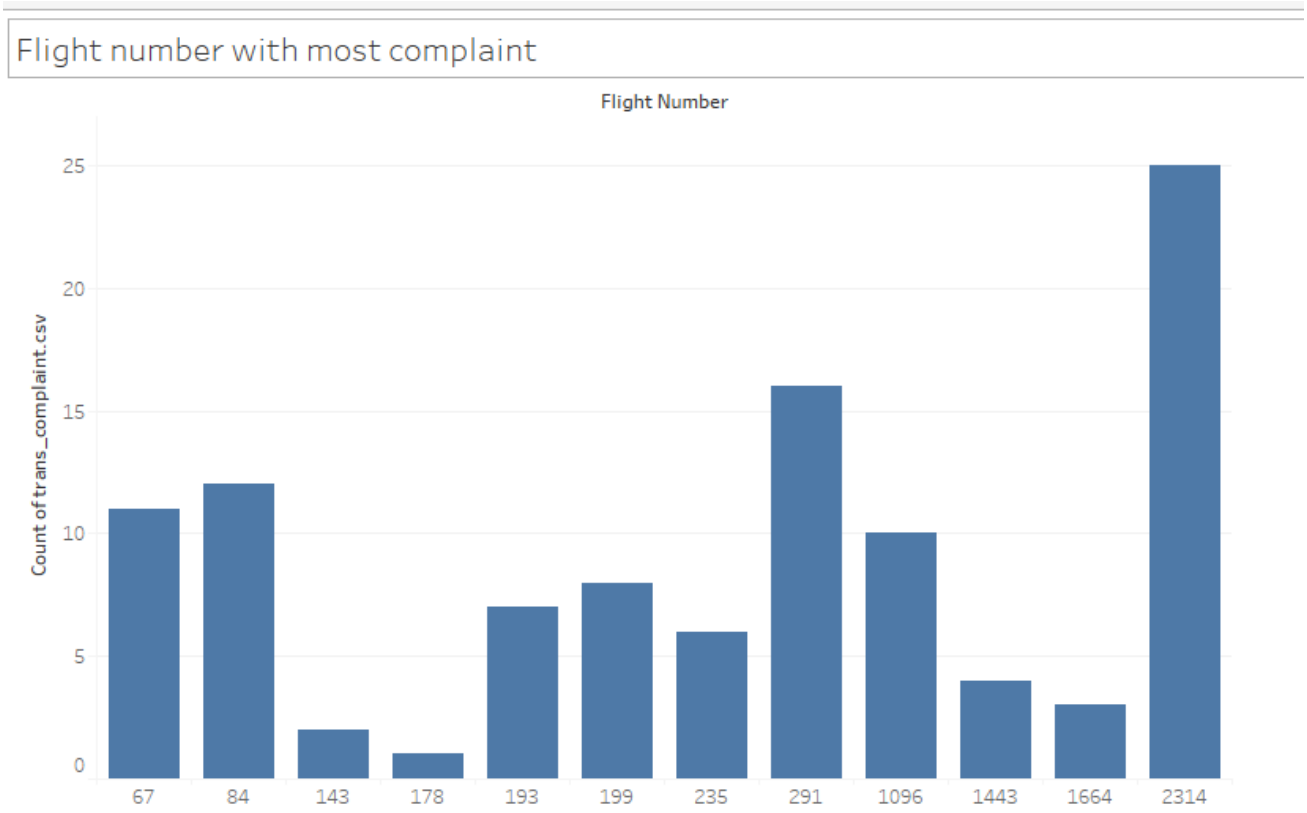


Fig 2: Airlines complaints for flight number

As seen in the above flight number 2314 have the highest number of complaints, flight number 291 have the second highest number of complaints fallowed by flight number 67 and 84 respectively. This reason might have a several reason, it could be because of flight delay, cancelled, couldn't get ticket etc. To overcome from this company needs to focus on using website, mobile app and offline message to provide the information for customer. For example, if the flight was about to cancel then company inform customer through website, mobile and direct message to customers, if customer have booked the ticket then they can book ticket from online which could save their time. Also company needs to focus on hospitality management, food provided by flight, frequent maintenance of plane like 234, 192, 67, 84 etc.

Report 3: Total compensation amount

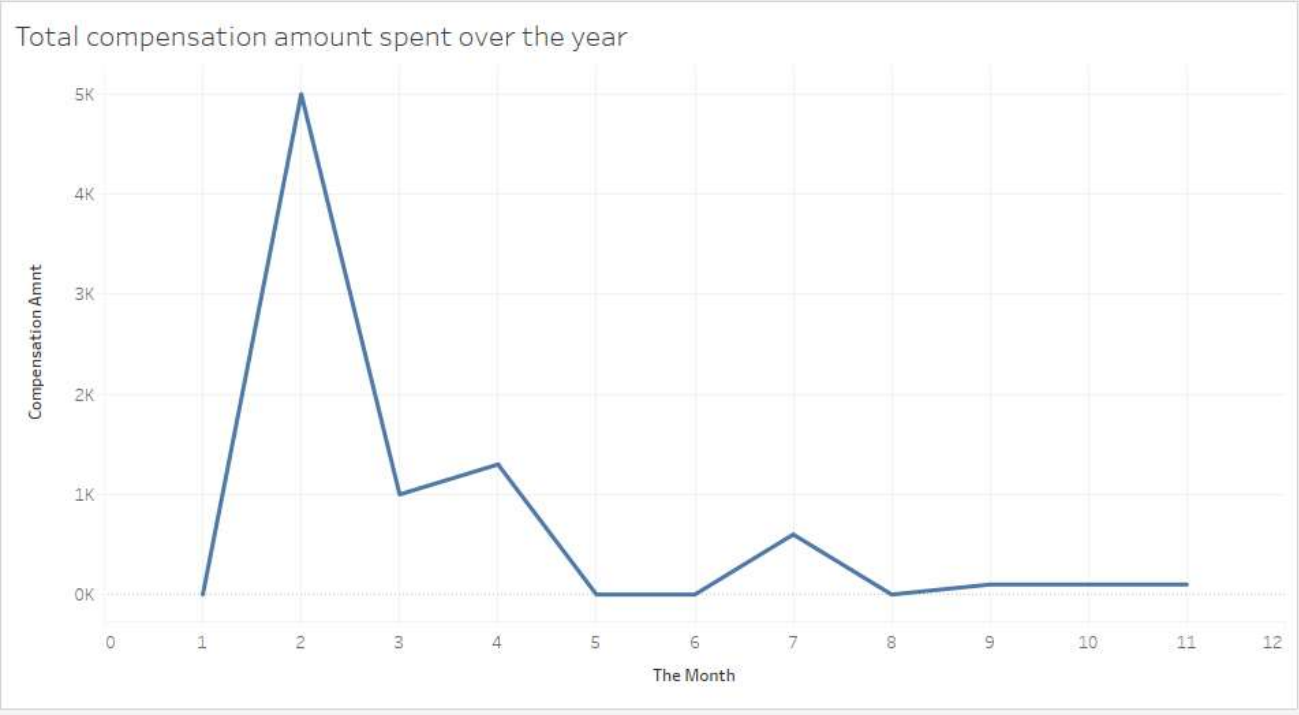


Fig 3: compensation amount spent over the month

From the above line we can clearly see that, the month around 2 to 4 have the highest amount of compensation which is month around march and April. At this time there is high chance of thunderstorm and storm, and this type of bad weather might be the reason of cancelled flight which resulted in high compensation amount during that months of the year. If could mitigate this type of problem then it would help company in huge loss of money. For this company should use less flight during that period, also they can use the flight number which are in better condition for example flight number 178, 143, 1664, 143 could be a better option because they have the less complaint in comparison to other.

Report 4: Number of cancelled flights

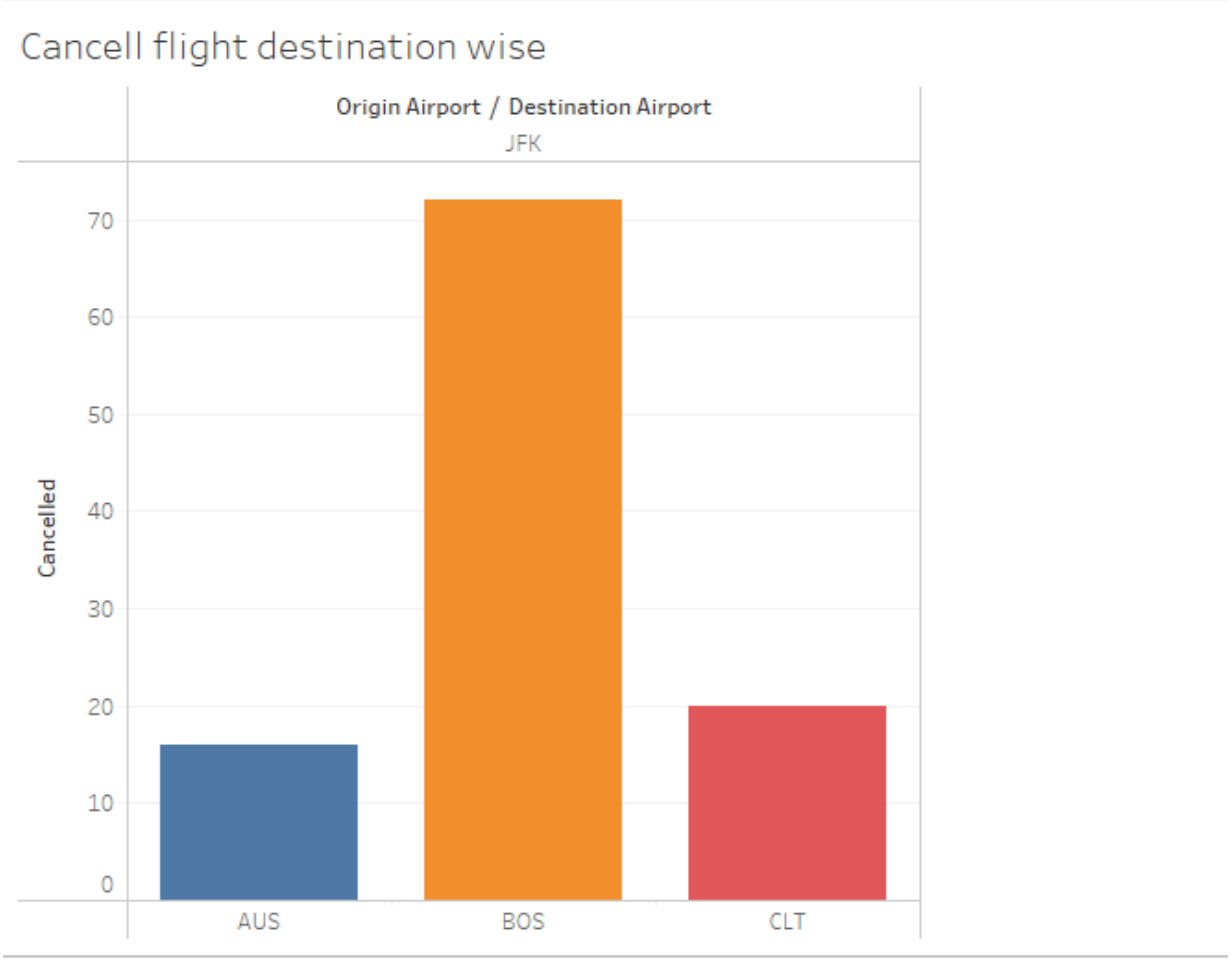


Fig 4: Cancelled flight destination wise

As seen in the above bar graph, flight from JFK to BOS have the highest number of cancelled. Which is fallowed by the JFK to CLT and JFT to AUS respectively. If we did not mitigate flight cancellation then it can effect on the company's finance. Flight cancelation might have several reasons, it could be geographical, airline traffic jam, security issues, lack of passenger, bird strike, computer glitch, plane condition etc. if it is the geographical issues then company needs to use the plane which are really in good condition to avoid from cancellation, if it is the bird issues then company needs use the bird detection radar to detect the bird so that air pilot can react before it strikes with bird. If it is computer glitch issues then company should replace with new one and also hire professional computer operators. If it is the issues with flight then flight number like 83, 143, 144, 1676 could be the better option because they do not have any cancellation yet.

Report 5: Number of flights that are diverted

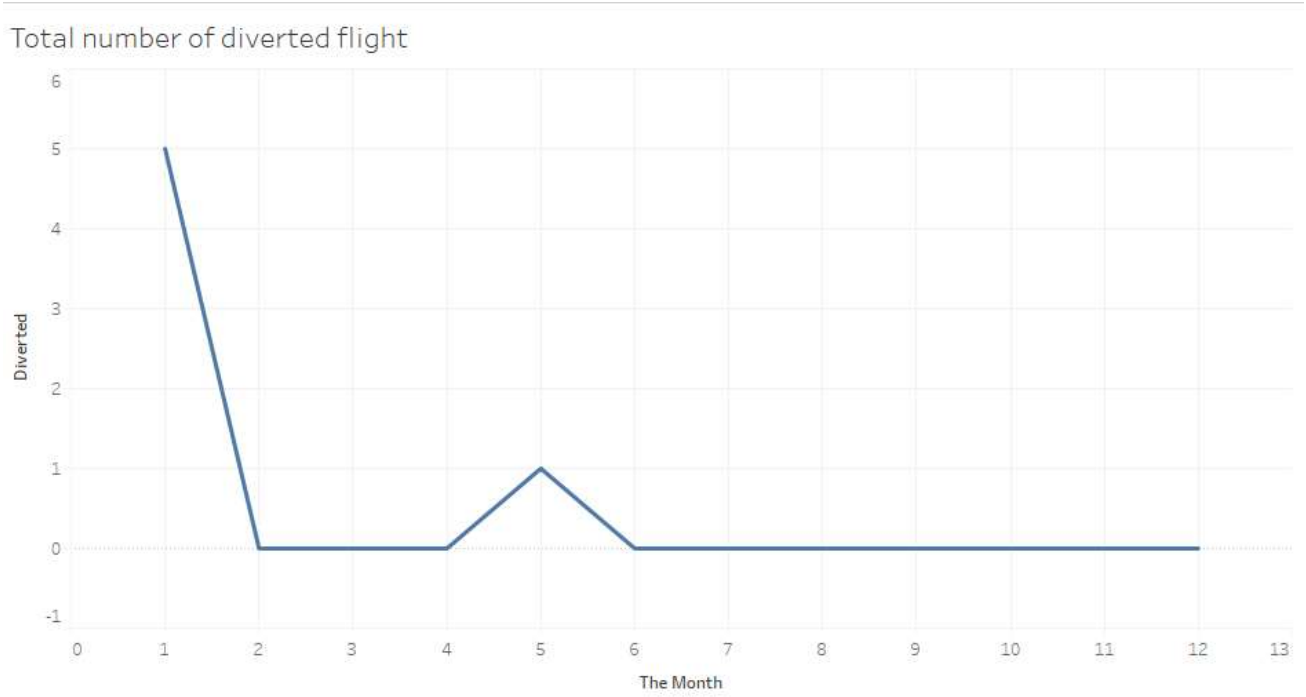


Fig 5: Diverted flight month wise

Diverted cannot satisfy customer, and customer dissatisfaction means loss of customer which could lead to company loss. So, we need to digitate this type of problems. As seen in the above month around 1 is December/January at this time weather is cold and, because of less rain air pollution can cause smog, which can lead to less visibility, and in this period, there is also a chance of radiation fog. Month 4 to 6 is may, June, July and this is rainy and cloudy season and this type of bad weather can divert flight. To overcome from this company needs to make the flight schedule fallowing the weather forecasting, use weather radar that could pilot to see inside the clouds ahead of them, use of wind shear system because it could help to detect hazardous condition ahead making accidental condition caused by the thunderstorms, if all of this is not possible then provide better training for pilot to prepare for the unexpected and also use the flight which are in good condition and better option could be flight number like 67, 143, 1664, 143 could be a better option because they do not have the any diversions yet.

Report 6: customer total miles busines wise

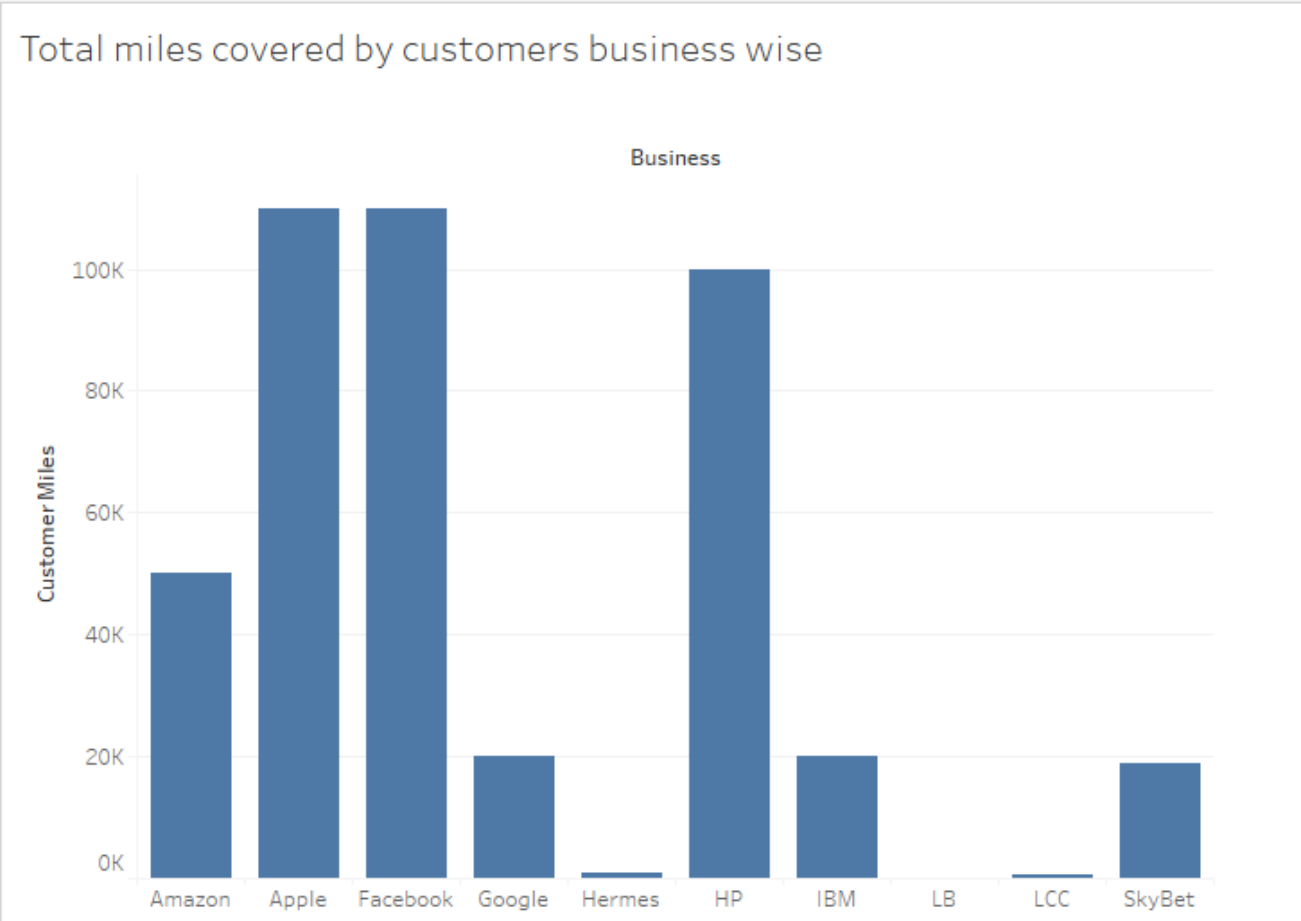


Fig 6: Total miles cover by customer business wise

From the above bar graph, it is clear that company have huge number of customers who are fall under apple and Facebook business. Apple and Facebook have the highest number of customers which is fallowed another two business giants HP and Amazon respectively. So, the company needs to focus more on the customer who are from Apple and Facebook like providing better service quality, better hospitality management etc.

Conclusion

To conclude, at first, we have loaded data into oracle from multiple sources (flight_2018, flight_2017, flyu_flights, customer and complaint table). After that we have migrated data into stagging tables, flight_2018, flight_2017 and flyu_flights data migrated into stage_flight, customer data into stage_customer, complaint data into stage_customer table. After that we have separated good data and bad data from stagging tables. Good data are migrated into clean tables and bad data are migrated into bad tables. After that we have cleaned the bad data and migrated them into clean data tables. Then, we have created transformation table and migrated clean data into transform tables. After all of this extraction, cleaning and tranformation, finally we have loaded data into dimension tables. After the confirmation of no data loss during the entire ETL process we have loaded the data into fact tables too.

Data warehousing approaches with respect to FlyU

As I have explained above, there are two methodologies to design the data warehouse namely Ralph Kimball and Bill Inmon's. Bill Inmon and Ralph Kimball have different philosophies, for information collecting, information management, and also for the analytics for the decision support. Both theory approach problem from different viewpoints, techniques of designs and implementation of strategies. Both methodologies have their own advantages and disadvantages. Based on our requirements we can choose any of these methodologies to design data warehouse base on our project requirements. Here I am going to discuss about the both approaches with respect to flyu flights and choose the best one for our project.

Bill inmon fallow top-down approach. The main core of bill inmon theory is enterprise data warehouse. It starts with centralized enterprise-wide data warehouse just by the multiple databases to the analytical needs of departments, which later known by the data marts. In this model data is stored in normalization form and also the data warehouse is not directly created, but the data is fed into is multiple marts which are then filtered down to the subsets of particular needs. Ralph Kimball fallow bottom-up approach. In this model data is collected using the ETL procedure from the multiple data sources and then loaded into common area called stagging then transformed into OLAP cube. This approach uses dimensional model such as star schema and snowflake schema to maintain the data in data warehouse. This approach makes query writing fast and simple, and also get report very quickly.

Ralph Kimball approach with respect to flyu flights

To design and develop the data warehouse of flyu flights airlines companies I have used this (Ralph Kimball) model approaches. At first, I have identified and collected the requirement, questions that needed to be present in the data warehouse. After that I have made report then make star schema diagram which is given below.

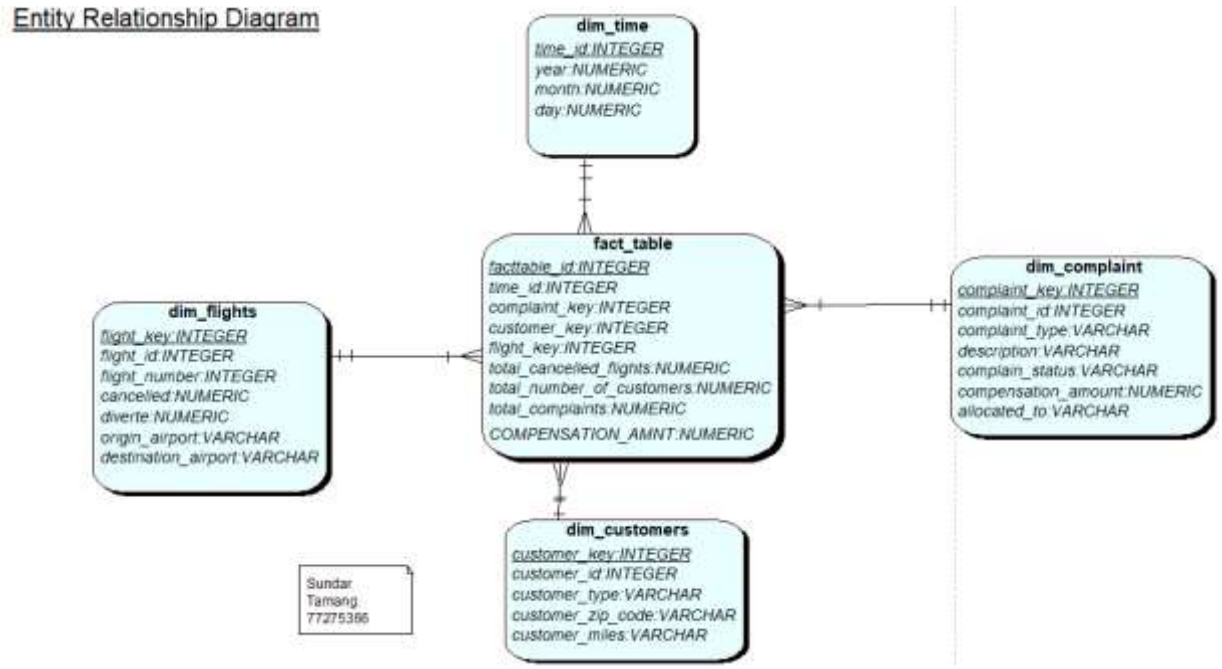


Fig: star schema diagram

Then I have loaded the data from various sources, 'flight_2017' 'flight_2018' which are excels data, 'flyu_flights' which is written in script in SQL. After data is loaded into oracle database, I have created three stagging tables 'stage_flights', 'stage_complaints', 'stage_customer' then migrate data into these tables. This process of Ralph Kimball is known as the extraction. Secondly, three bad, good tables and one data issues table were created to migrate bad data and good data into the respective tables, where in data_issues table I migrated bad data with its information and details. Then I have cleaned all the bad data and transformed them into the good data, after that updated the 'data_issues' table with appropriate information. Then I have created three transform table with respect to 3 good table, and transformed them into the transform table, this is known as the transformation in Kimball model. After data is transformed into transformation table, I have created dim tables using the SQL code which are generated from the QSEE. At last I have loaded the data into dim tables and fact tables.

Some of the advantages of Ralph Kimball with respect to flyu flights

- This approach does not fallow normalization, which make it quick to build which will definitely help to store the ailines company to store data as soon as possible.
- Kimball theory of star schema can be easily understanding and because of this it could simplify query and also easy to analyze which could help flyu companies' employee.
- In star schema dim tables do not have another dim table which mean there will be not any

complex join query, which make very fast to retrieve data from database.

- It allows BI tools to deeper across several star schema and which can generate reliable insights for the future of flyu flight to make effective decision and plan for future.

Some of the flaw of Ralph Kimball with respect to flyu flights

- Because of denormalization data redundancy might occur in the data warehouse.
- It cannot handle all the BI reporting requirements which may arise issue while demonstrating data reports.
- It may require professional skillful manpower because of incorporation of large amount of legacy data.
- It is difficult to make any changes in the business.

Bill Inmon approach with respect to flyu flights

As I have explained above there are two methodologies to design and develop the data warehouse, and Bill inmon approach is one of them, this methodology fallows the top-down approach. It explains as a centralized enterprise-wide data warehouse just by the multiple databases to the analytical needs of departments, which are later known by the data marts. Because of this reason Bill Inmon is known then as the top-down approach. The main central repository of the data combined from all the operations system of organization. These approaches use 3NF which is normalized form to build the data warehouse. Unlike in Kimball approach where data warehouse procedures are done afterwards it is done before data marts and then characterized to the different data marts then it fallow normalization in OLAP cubes. At last data is transferred to the correspond tables. If we fallow normalization model and make schema diagram then it is called snowflake diagram, if we make snowflake diagram of above star schema diagram then somehow it looks like below.

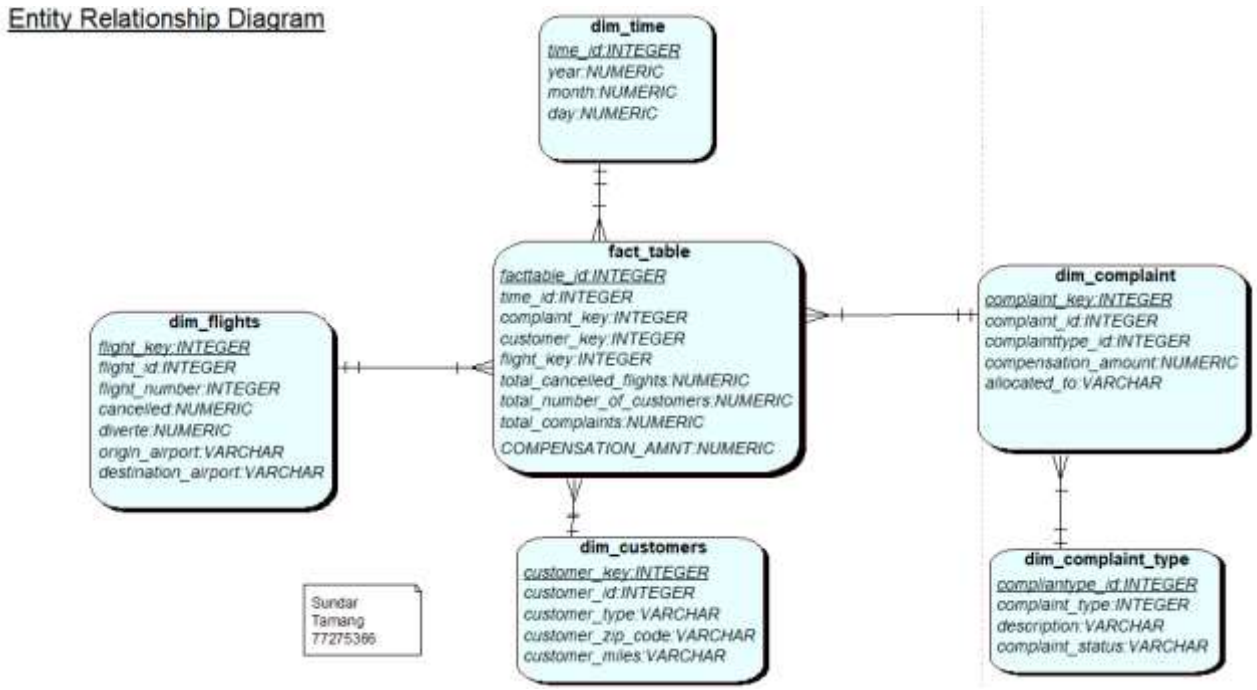


Fig: snowflake schema diagram

Some of the advantages of Bill Inomon model with respect to the flyu flights airlines

- It fallows the 3NF which make ETL data warehouse procedures less susceptible to failure which will definitely help flyu company to bearing from data loss.
- It offers us flexibility, which will make it easier to update the data and any changes in the business requirements which would definitely help flyu flights company to keep the data updated.
- It could simplify the business procedures, as the logical model represents detail objects of business which could help the airlines company to view report of the data in more easy manner.

Some of the flaw of Bill Inmon with respect to flyu flights

- To set up and delivery of data required more time, this time consuming for the flyu airlines company.
- Complexity and difficulty level increase as more tables are added to the data model over the time, which could affect to provide service in future.
- It would require more time for the flyu company for additional ETL operation

- Additional ETL operation is needed since the data marts are created after the data warehouse is create which could lead to required more time for the company.

Conclusion

To conclude, both methodologies have their own advantages and disadvantages, based on our requirements we can use either Kimball or Inmon to develop the data warehouse, the more important is which one serves user at low redundancy, and these design approaches are built and produced as per the needs of users or a business and keeping budget in context. Bill Inmon approach is used for the in-depth output while Ralph Kimball approach give quick results. Base on business need I have chosen Kimball model as it would allow to take quick decision and risk management for the airlines company, it also takes less time, human resources and budget by providing several facilities so it is more suitable for flyu airlines company.

Bibliography

Anon., n.d. *oracle.* [Online] Available at: https://www.oracle.com/database/what-is-a-data-warehouse/ [Accessed 20 4 2021].

skfdjskdfjs, n.d. *sdfsdf.* [Online] [Accessed 2014].

Smallcombe, M., 2020. *xplenty.* [Online] Available at: https://www.xplenty.com/blog/snowflake-schemas-vs-star-schemas-what-are-they-and-how-are-they-different/#:~:text=Star%20schema%20dimension%20tables%20are,to%20simpler%2C%20faster%20SQL%20queries. [Accessed 5 5 2021].

techdifferences, n.d. *techdifferences.* [Online] Available at: https://techdifferences.com/difference-between-oltp-and-olap.html#:~:text=OLTP%20and%20OLAP%20both%20are,is%20an%20analytical%20processing%20system.&text=The%20basic%20difference%20between%20OLTP,online%20database%20query%20answering%20system. [Accessed 20 4 2021].

ucmerced, n.d. *ucmerced.* [Online] Available at: http://library.ucmerced.edu/data-dictionaries [Accessed 7 5 2021].

Vidhya, A., n.d. *medium.* [Online] Available at: https://medium.com/analytics-vidhya/theories-of-kimball-and-inmon-about-data-warehouse-design-c16260fab5e9 [Accessed 5 5 2021].