Title: Building a data warehouse using Oracle for GreenHomeHelp

Author: Usman Basharat & Yunus Hassan

Date: 15th March 2020

Course: COMP-1434 Data Warehousing

Word Count: 9885

School of Computing and Mathematical Sciences

UNIVERSITY of GREENWICH

Table of Contents

Introduction

According to Matteo Golfarelli, Data Warehousing has huge amounts of electronic data stored in recent years with the urgent need of accomplishing goals that go beyond the routine tasks daily (Golfarelli and Rizzi, 2009). GreenHomeHelp is a care provider that provides urgent cover for care workers. In this specification, we require to design and build a database using the information that has been provided. We have been given a sample data to extract, transform, load, cleanse and implement this information. We are going to be using SQL Developer to perform numerous tasks. Please view the following for a summary of what is to be expected of this report:

1. ETL – exporting the data
2. ETL – staging area to dimensions and fact table
3. Data Cleansing
4. Implementation of Dimensions and Fact
5. Bonus
6. Queries

Snowflake Schema

Referring to Figure 1, you can see the snowflake schema that demonstrates how the design upcoming of this data warehousing. Snowflake is a "is a multi-dimensional" structure. At its core, it has a fact table that has dimensions surrounded. In this case, the fact table, as shown, is the session table. Alongside this shows extended dimensions that are referred as the snowflake shape. This was decided as many unnecessary columns were used that was not needed. Therefore, it was decided to drop all foreign keys tables within these dimensions that have "dw" in-front of the name in reference to this. We have planned this to replace the data that has been referred to this within the dimension only. This makes it easier in reference. As many have stated, snowflake schema has been doomed as complex. We have provided materialized views to make this easier for users to see each reference.

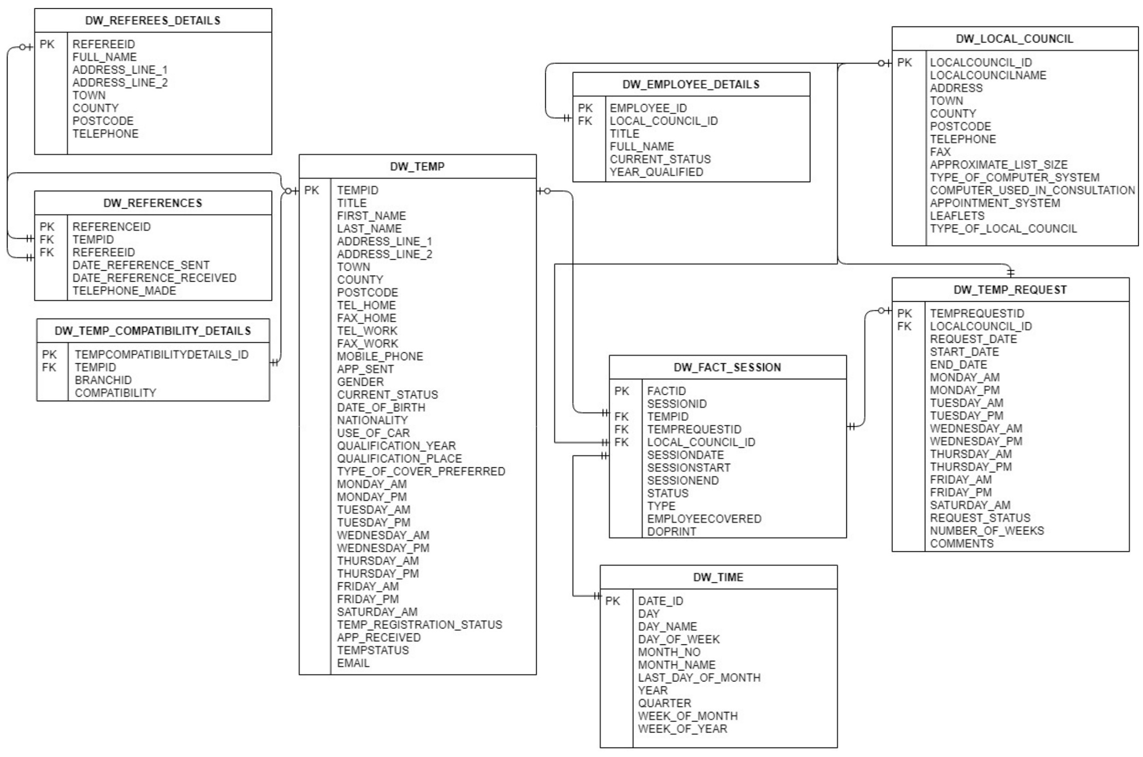


Figure 1 shows the snowflake schema for GreenHomeHelp

A: ETL Processes Screenshots

This section will talk about how the data is extracted from the original data source into the data warehouse ready to be populated. An outline of how ETL was applied are the earlier stages of the implementation of a data warehouse. From exporting the data to then transforming it to the oracle data warehouse and loading it into fact and dimension tables using cursor. Below, it is evident that these steps have been followed for a well-functioning data warehouse that fulfills the user requirements. The cleansing stage also took part as it was fundamental to only cursor clean data into our dimensions and fact table from the staging area. The database containing information based on temporary care workers than the government has set up in order to fill in shifts for those regular workers who decide to take leave. However, there were many errors with the information and the data types but that will be discussed in more details in the part b section. As mentioned earlier, cleansing was vital in this section as it is considered as transforming the data for it to be considered as the final data warehouse with all records finalized and cleansed.

The first step was to export the data from a data source, which in this case was Microsoft access database, into oracle. We followed a step of instruction that will allow us to transfer data from the original data source into oracle but at first, we had issues accomplishing this. However, those issues were later resolved. This will be discussed in the 'issues' section to follow in this report.

Export steps

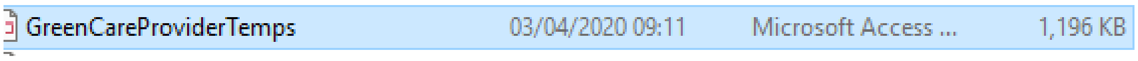
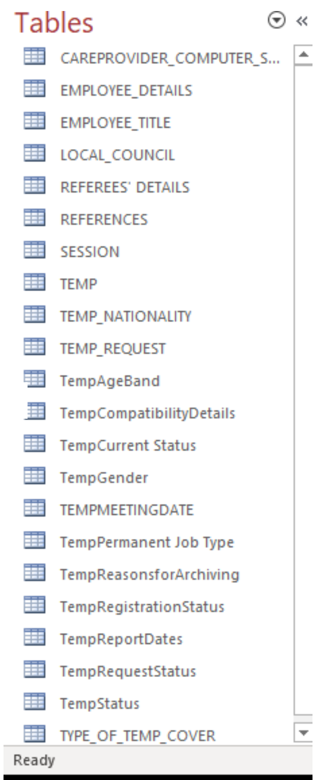

GreenCareProviderTemps 03/04/2020 09:11 Microsoft Access ... 1,196 KB

Figure 2 shows the file for Access

Referring to Figure 2 and Figure 3 shows screenshots of the Microsoft access database which is the original data source that will be used. The aim now is to export this data into oracle in order to populate it into our data warehouse. This will then allow for all cleansing to take part in the staging area before it is then transformed into the dimensions and tables using cursor.

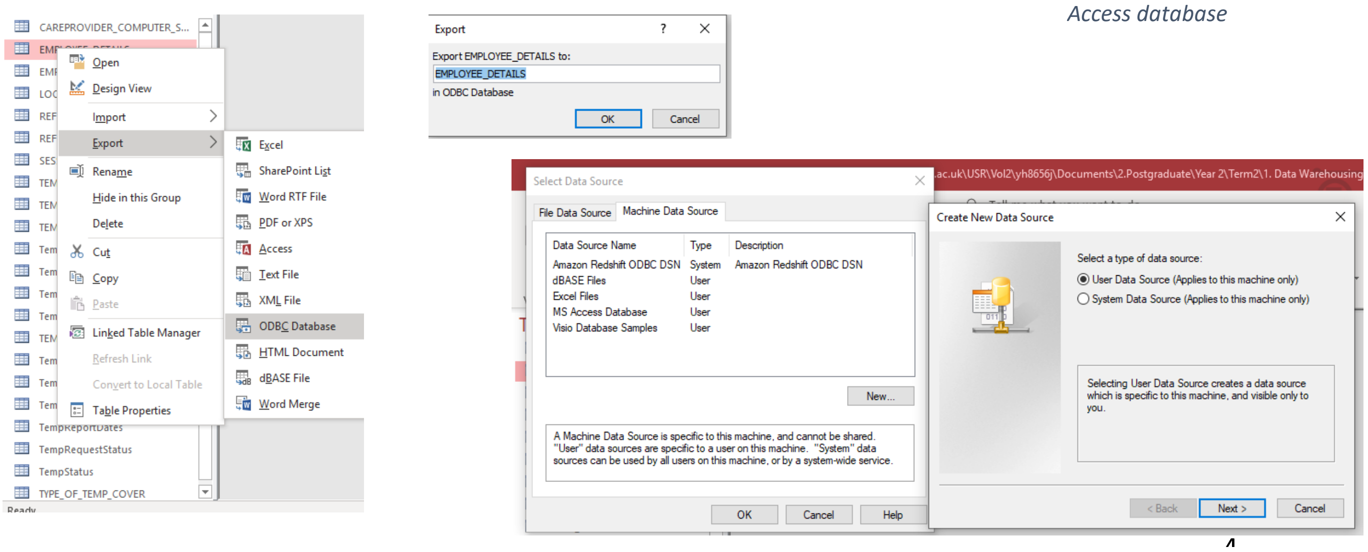
Referring to Figure 4 and 5, this shows the steps of exporting a table from Microsoft access to oracle ODBC. The first three steps include choosing the odbc application that we intend to transfer the data to, naming the table and choosing the user data source.


Tables

CAREPROVIDER_COMPUTER_S...
EMPLOYEE_DETAILS
EMPLOYEE_TITLE
LOCAL_COUNCIL
REFEREES' DETAILS
REFERENCES
SESSION
TEMP
TEMP_NATIONALITY
TEMP_REQUEST
TempAgeBand
TempCompatibilityDetails
TempCurrent Status
TempGender
TEMPMEETINGDATE
TempPermanent Job Type
TempReasonsforArchiving
TempRegistrationStatus
TempReportDates
TempRequestStatus
TempStatus
TYPE_OF_TEMP_COVER

Ready

Figure 4 shows all the tables of Access database



Export

Export EMPLOYEE_DETAILS to:

EMPLOYEE_DETAILS

in ODBC Database

OK Cancel

.ac.uk\USR\Vol2\yh8656j\Documents\2.Postgraduate\Year 2\Term2\1. Data Warehousing

Select Data Source

File Data Source | Machine Data Source

Data Source Name	Type	Description
Amazon Redshift ODBC DSN	System	Amazon Redshift ODBC DSN
dBASE Files	User	
Excel Files	User	
MS Access Database	User	
Visio Database Samples	User	

New...

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

OK Cancel Help

Create New Data Source

Select a type of data source:

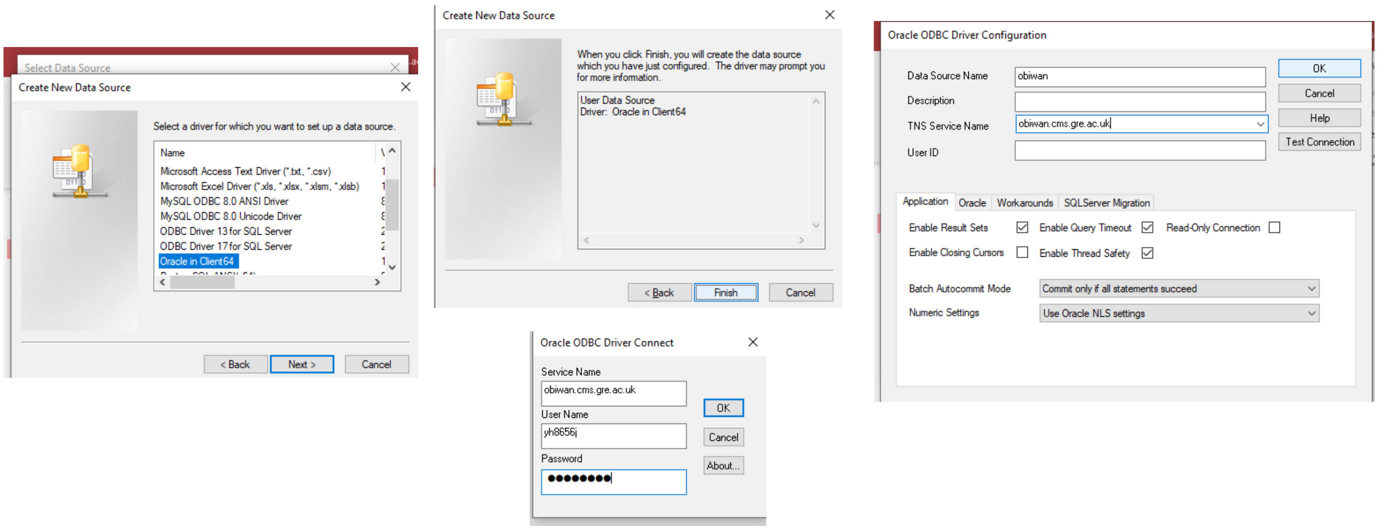
[✓ User Data Source (Applies to this machine only)]
System Data Source (Applies to this machine only)

Selecting User Data Source creates a data source which is specific to this machine, and visible only to you.

< Back Next > Cancel

Figure 3 shows the first three steps of transferring data.

The data source that we are using, as evident in Figure 5 is ‘oracle in client64’. Once we have exported the table to odbc and chose the relevant data source, we are then required to configure the


Create New Data Source

Select a driver for which you want to set up a data source.

Name
Microsoft Access Text Driver (*.txt, *.csv)
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
MySQL ODBC 8.0 ANSI Driver
MySQL ODBC 8.0 Unicode Driver
ODBC Driver 13 for SQL Server
ODBC Driver 17 for SQL Server
Oracle in Client64

Create New Data Source

When you click Finish, you will create the data source which you have just configured. The driver may prompt you for more information.

User Data Source
Driver: Oracle in Client64

Oracle ODBC Driver Configuration

Data Source Name	obiwan
Description	
TNS Service Name	obiwan.cms.gre.ac.uk
User ID	

Enable Result Sets ☑ Enable Query Timeout ☑ Read-Only Connection ☐
Enable Closing Cursors ☐ Enable Thread Safety ☑
Batch Autocommit Mode: Commit only if all statements succeed
Numeric Settings: Use Oracle NLS settings

Oracle ODBC Driver Connect

Service Name	obiwan.cms.gre.ac.uk
User Name	yh8656j
Password	●●●●●●●●

Figure 5 shows the final steps of transferring data

odbc drive by entering ‘obiwan’ as our data source name and ‘obiwan.cms.gre.ac.uk’ as the TNS service name. These are the details we require to log into the database. Finally, we are prompted with a credentials interface which will require for the obiwan oracle details to be filled in order for a successful exportation of the given tables. The exportation is now complete. The table that has been exported has successfully been transferred to the Oracle table. Below, I will show you a different way of importing data to Oracle.

Importing to Oracle ODBC

Now that we have exported the table from Microsoft access, we are now left with the other half of the process which is to import that same table into the oracle odbc database. Figure 6 below shows the necessary steps taken to do this. We first right clicked on the tables to be able to import data and then from there simply locate the table, in this case in Figure 7 you can see our SESSION.xls file.

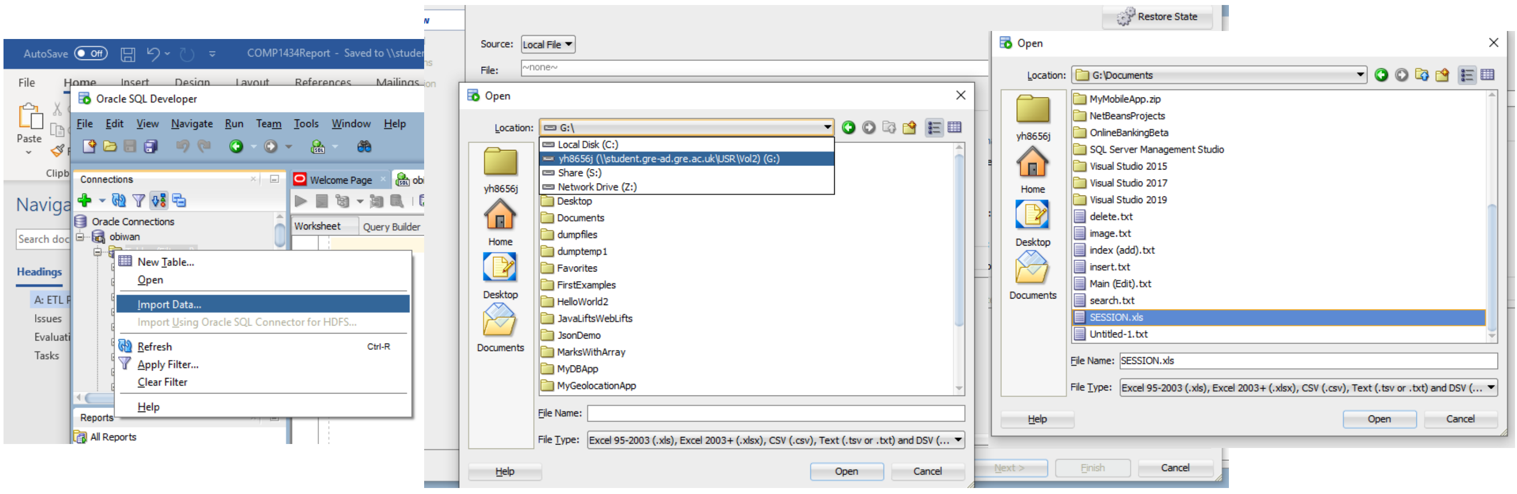
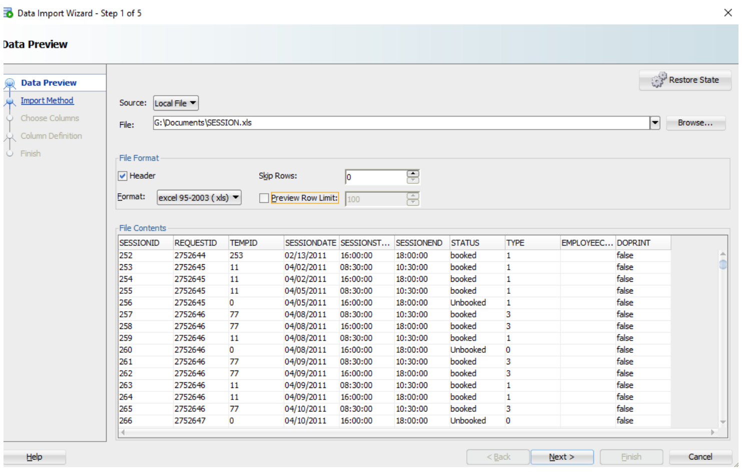


Figure 6 shows the necessary steps of importing



Source: Local File

File: G:\Documents\SESSION.xls

File Format

[✓ Header] Skip Rows: 0

Format: excel 95-2003 (xls) Preview Row Limit: ☐ 100

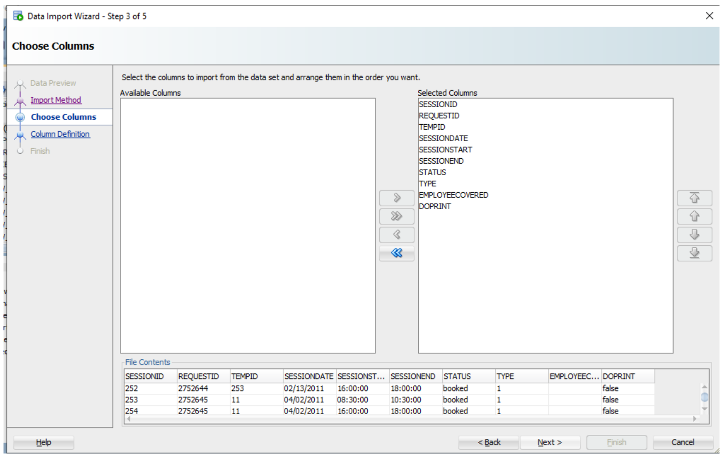
File Contents

SESSIONID	REQUESTID	TEMPID	SESSIONDATE	SESSIONST...	SESSIONEND	STATUS	TYPE	EMPLOYEEC...	DOPRINT
252	2752644	253	02/13/2011	16:00:00	18:00:00	booked	1		false
253	2752645	11	04/02/2011	08:30:00	10:30:00	booked	1		false
254	2752645	11	04/02/2011	16:00:00	18:00:00	booked	1		false
255	2752645	11	04/05/2011	08:30:00	10:30:00	booked	1		false
256	2752645	0	04/05/2011	16:00:00	18:00:00	Unbooked	1		false
257	2752646	77	04/08/2011	08:30:00	10:30:00	booked	3		false
258	2752646	77	04/08/2011	16:00:00	18:00:00	booked	3		false
259	2752646	11	04/08/2011	08:30:00	10:30:00	booked	1		false
260	2752646	0	04/08/2011	16:00:00	18:00:00	Unbooked	0		false
261	2752646	77	04/09/2011	08:30:00	10:30:00	booked	3		false
262	2752646	77	04/09/2011	16:00:00	18:00:00	booked	3		false
263	2752646	11	04/09/2011	08:30:00	10:30:00	booked	1		false
264	2752646	11	04/09/2011	16:00:00	18:00:00	booked	1		false
265	2752646	77	04/10/2011	08:30:00	10:30:00	booked	3		false
266	2752647	0	04/10/2011	16:00:00	18:00:00	Unbooked	0		false

Figure 7 shows the importing steps

Figure 7 shows the data in the session table that we have exported from Microsoft access and are now importing into our oracle database tables. As you can see the data is previewed as it is important that we do not check the box which write "preview Row limit" as we have multiple rows of data

Figure 8 shows the file content which is the sessions data at the bottom but also important shows the columns in the table. These columns were original to the left in the available columns section and as we require them all we then dragged them over to the selected columns section ready to be used in the data warehouse.



Select the columns to import from the data set and arrange them in the order you want.

Available Columns

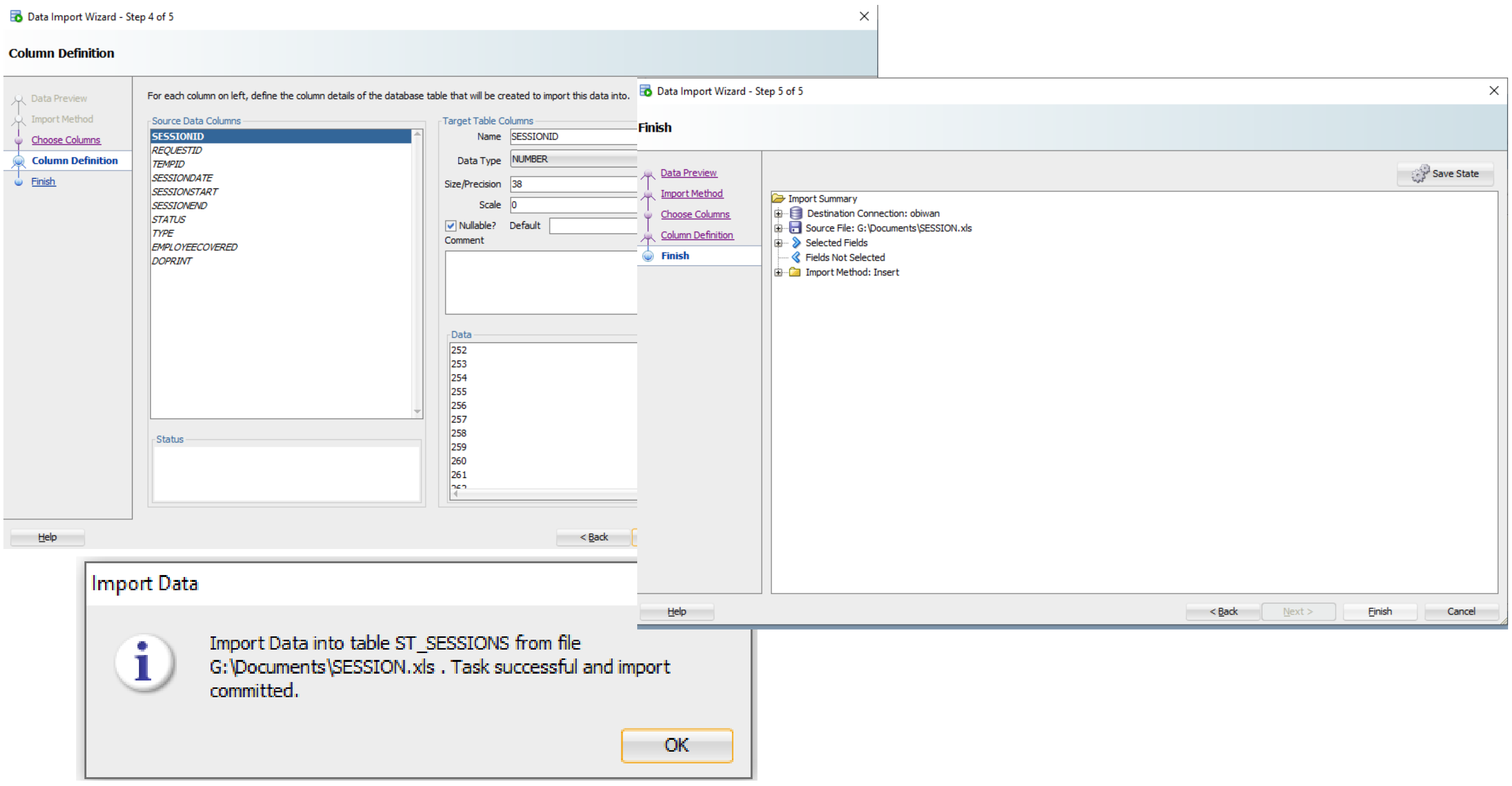
Selected Columns
SESSIONID
REQUESTID
TEMPID
SESSIONDATE
SESSIONSTART
SESSIONEND
STATUS
TYPE
EMPLOYEECOVERED
DOPRINT

File Contents

SESSIONID	REQUESTID	TEMPID	SESSIONDATE	SESSIONST...	SESSIONEND	STATUS	TYPE	EMPLOYEEC...	DOPRINT
252	2752644	253	02/13/2011	16:00:00	18:00:00	booked	1		false
253	2752645	11	04/02/2011	08:30:00	10:30:00	booked	1		false
254	2752645	11	04/02/2011	16:00:00	18:00:00	booked	1		false

Figure 8 shows the final step of importing

Finally, Figure 9 is a representation of a confirmation that will allow us to see the final visualization of our data so that we are curtain we are importing the correct data with the precise column names. The Data import wizard consist of 5 steps to import which we have discussed, and it is a matter of clicking next as long as the data is correct, and the user is satisfied. Once we have done that, we are then prompted with the final screen will allow us to finish the setup before the confirmation dialog then states that the importation of the session table is indeed successful.


Data Import Wizard - Step 4 of 5

Column Definition

For each column on left, define the column details of the database table that will be created to import this data into.

Source Data Columns: SESSIONID, REQUESTID, TEMPID, SESSIONDATE, SESSIONSTART, SESSIONEND, STATUS, TYPE, EMPLOYEECOVERED, DOPRINT

Target Table Columns
Name: SESSIONID
Data Type: NUMBER
Size/Precision: 38
Scale: 0
Nullable? ✓ Default:
Comment:

Data: 252, 253, 254, 255, 256, 257, 258, 259, 260, 261

Data Import Wizard - Step 5 of 5

Finish

Import Summary
- Destination Connection: obiwan
- Source File: G:\Documents\SESSION.xls
- Selected Fields
- Fields Not Selected
- Import Method: Insert

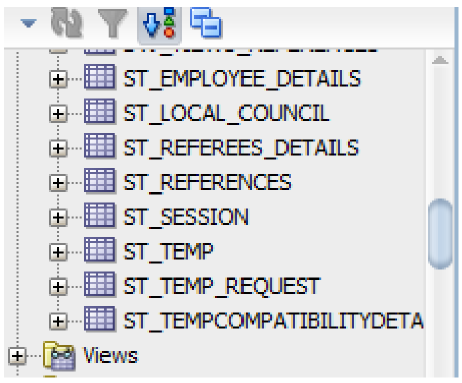
Import Data

Import Data into table ST_SESSIONS from file G:\Documents\SESSION.xls . Task successful and import committed.

Figure 9 shows you the last step of importing

Staging Area

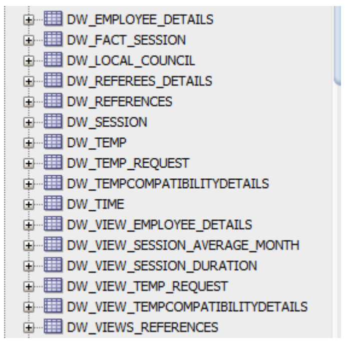
Once we have successfully exported the data from Microsoft Access database into Oracle, we are now ready to utilize the data and make the necessary amendments. In order to do that, we must first initialize a staging area where all the original data will be held. This will allow for data cleansing to take place and then for the transfer of that cleansed data into our data warehouse tables and dimensions. The image in Figure 11 shows the staging area has been named so that each table has an 'ST' Infront of them so it is clear that it is the original data ready to be cleansed.



ST_EMPLOYEE_DETAILS
ST_LOCAL_COUNCIL
ST_REFEREES_DETAILS
ST_REFERENCES
ST_SESSION
ST_TEMP
ST_TEMP_REQUEST
ST_TEMPCOMPATIBILITYDETA
Views

Figure 11 shows all of the table that are staging tables

We decide in order to differentiate between the original data and then new fact and dimension tables, we will put an ST in front of the staging area and a DW in the cleansed tables. Once all the data is cleansed, we then created the new tables as showing in Figure 10.



DW_EMPLOYEE_DETAILS
DW_FACT_SESSION
DW_LOCAL_COUNCIL
DW_REFEREES_DETAILS
DW_REFERENCES
DW_SESSION
DW_TEMP
DW_TEMP_REQUEST
DW_TEMPCOMPATIBILITYDETAILS
DW_TIME
DW_VIEW_EMPLOYEE_DETAILS
DW_VIEW_SESSION_AVERAGE_MONTH
DW_VIEW_SESSION_DURATION
DW_VIEW_TEMP_REQUEST
DW_VIEW_TEMPCOMPATIBILITYDETAILS
DW_VIEWS_REFERENCES

Figure 10 shows all the cleansed tables.

As you may realize, there are more tables in Figure 10 and that is because it includes extra tables such as time dimensions and different materialized views. The cleansing section looks more at how the data was cleansed in terms of incorrect and inconsistent data and datatypes. Once all information was then correct it was then transformed into the new tables and dimensions by using the cursor technique which will be explained further. Within the tables there have been implementations of extra bonus features such as partitioning, extended timetable, materialized views and dumping the temp table into a flat file to then be populated. All of this will be further discussed in the bonus section also with steps on how this was achievable.

Using Cursor

Thus far, we spoke about the different steps of ETL which include exporting data and importing data and transforming the data by cleansing the data. This section looks at a method used to load the data into another table by using the cursor method. We saw how easy it was to transfer data from one data source into oracle ODBC now we will look at how we transfer data from the staging area into the tables and dimension that have been populated for the final version of the cleansed data and tables with the correct data types. As you can see going back to Figure 10 and Figure 11, it shows the 'ST' tables in the staging area that are there to be cleansed and then populated into the newly created tables in DW. Below is an example of the code we used for session to load data into the dw_session dimension table using cursor.

```
declare
Cursor c_client is
Select  SESSION_ID, REQUEST_ID, TEMP_ID, SESSION_DATE, SESSION_START,
SESSION_END, STATUS_S, SESSION_TYPE, EMPLOYEE_COVERED,
DO_PRINT
from SESSION_S ;

begin
```

```
 for c_rec in c_client loop
 insert into DW_SESSION values(
c_rec.SESSION_ID, c_rec.REQUEST_ID, c_rec.TEMP_ID, c_rec.SESSION_DATE,
c_rec.SESSION_START, c_rec.SESSION_END, c_rec.STATUS_S,
c_rec.SESSION_TYPE, c_rec.EMPLOYEE_COVERED, c_rec.DO_PRINT);
end loop;
end;
```

This cursor load technique was used on all tables that were remaining and we all populated the same way. However, that later changed as we decided to attempt the bonus points of dumping our data from the temp table into a flat file then using that data file to populate our data into the new dw_temp dimension table. Both ways are effective for transferring details and an important aspect of ETL. From gaining access to the data sources original content to then transferring it into the oracle odbc application and then transforming the data and cleansing it to finally loading that data into the new dimensions and fact tables. This section clearly demonstrates the use of ETL and cleansing when it comes to preparing a data warehouse. The cleansing section to follow, will explain in more detail the steps taken to clean the data and what we looked out for in terms of errors, inconsistent data and data types.

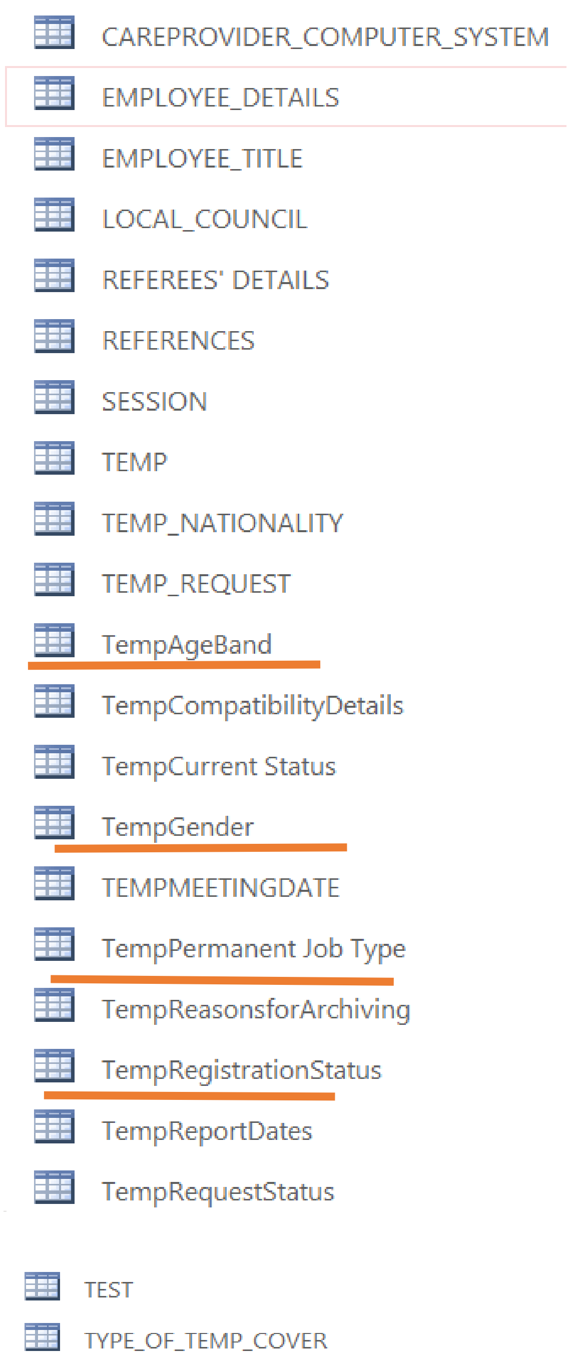
B: Data Cleansing

In this section, I will be discussing the cleansing exercise that identifies errors that I will search for during this stage. I will be searching for misspelt words, mismatching primary keys and foreign keys and inconsistent data. This will be identified in all dimensions that have been verified.

As mentioned previously, a long list of inconsistent tables was given. We decided, with my colleague, to drop all the foreign key links to replace the data that it has been referred to. This decision was taken carefully as it was only dropped if small data is held within the table itself. For example, Title table can be replaced as it only holds one column with it such as Mr, Mrs, and Dr. You can refer yourself to Figure 1 where this demonstrates the layout of this whole database. However, I will explain in detail which tables were dropped and for what reason. Please refer yourself to Figure 13 where this demonstrates a list of all the tables that were dealt with. You can compare the difference with the schema in Figure 1 and the list of what has been cleansed. Within this section, I will also be comparing the original data with the cleansed data and explain what has been cleaned, dropped, replaced data.

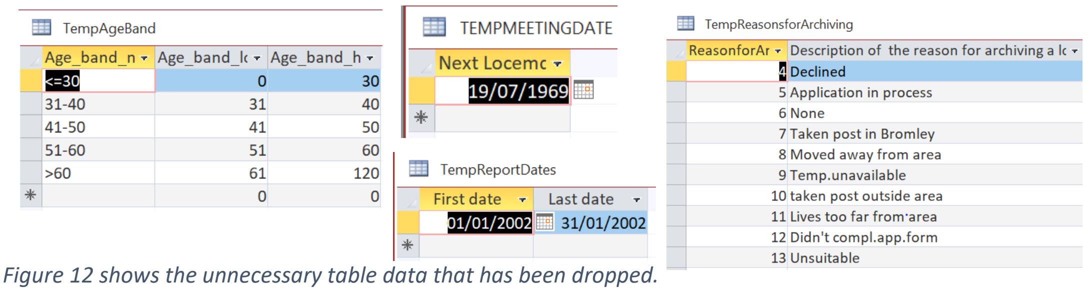
I would also like to point out that all cleansing was done before cursor. Therefore, this was an option as discussed with my partner and the outcome was that the cleansing was to be done before transforming.

Firstly, I would like to point out all the unnecessary tables that were decided to drop. These tables had unnecessary information that did not relate to any of the tables. You can see Figure 12 for more information that contained within these and can understand why these tables were dropped. TempAgeBand shows the first table where this has unwarranted information that does not link with any information. TempMeetingDate has only one date of information which is 19th July 1969. This is unrelated to any context of GreenCareProvider. In addition to this is TempReportDates has January 2002 length. However, this information is not sufficient to keep this information. Hence the reason why this has been dropped. TempReasonsforArchiving was a table that did not get any reference within the temp table. Therefore, as there was no reference to this table, it was dropped too. You can see this the data within Figure 12



- CAREPROVIDER_COMPUTER_SYSTEM
- EMPLOYEE_DETAILS
- EMPLOYEE_TITLE
- LOCAL_COUNCIL
- REFEREES' DETAILS
- REFERENCES
- SESSION
- TEMP
- TEMP_NATIONALITY
- TEMP_REQUEST
- TempAgeBand
- TempCompatibilityDetails
- TempCurrent Status
- TempGender
- TEMPMEETINGDATE
- TempPermanent Job Type
- TempReasonsforArchiving
- TempRegistrationStatus
- TempReportDates
- TempRequestStatus
- TEST
- TYPE_OF_TEMP_COVER

Figure 13 shows all of the original tables within GreenCareProvider



TempAgeBand

Age_band_n	Age_band_lc	Age_band_h
<=30	0	30
31-40	31	40
41-50	41	50
51-60	51	60
>60	61	120
	0	0

TEMPMEETINGDATE

Next Locemc
19/07/1969

TempReportDates

First date	Last date
01/01/2002	31/01/2002

TempReasonsforArchiving

ReasonforAr	Description of the reason for archiving a lc
4	Declined
5	Application in process
6	None
7	Taken post in Bromley
8	Moved away from area
9	Temp.unavailable
10	taken post outside area
11	Lives too far from area
12	Didn't compl.app.form
13	Unsuitable

Figure 12 shows the unnecessary table data that has been dropped.

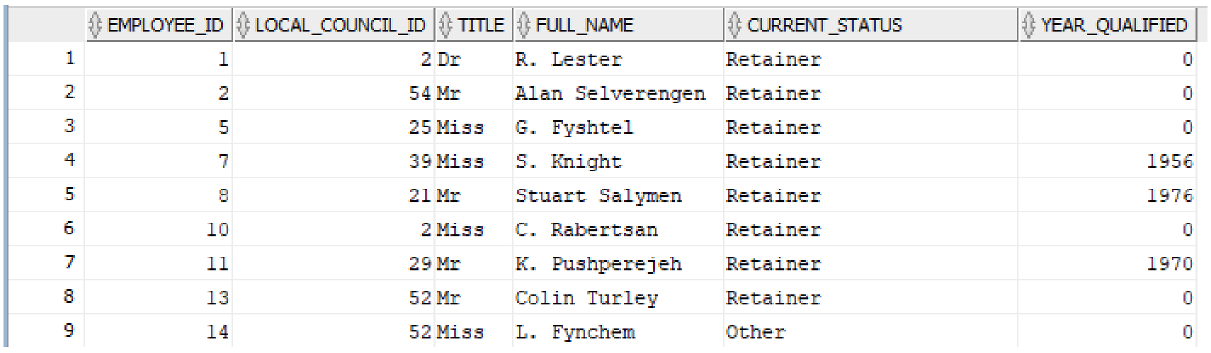
Employee Details

As you may notice, comparing Figure 1 and Figure 13, you may notice many tables that have been dropped. Please note that I will explain the query used within this table and include an example of this. Referring to Figure 14, title column has been replaced by using the query below. This query updates and replaces all the no.1 with Dr. Each reference updated according to employee_title. For example, 2 would be Mr. This has been the same process for each table that has been replaced each of its data.

UPDATE EMPLOYEE_DETAILS SET TITLE = REPLACE(TITLE,'1','Dr');

Another column that has been replaced is current_status. This query is the same process as how title data has been replaced. Each number is in reference to the foreign key that it is in reference to the relevant table. For example, 3 is in reference to Assistant.

UPDATE EMPLOYEE_DETAILS SET CURRENT_STATUS_ID = REPLACE(CURRENT_STATUS_ID,'3','Assistant');



	EMPLOYEE_ID	LOCAL_COUNCIL_ID	TITLE	FULL_NAME	CURRENT_STATUS	YEAR_QUALIFIED
1	1	2	Dr	R. Lester	Retainer	0
2	2	54	Mr	Alan Selverengen	Retainer	0
3	5	25	Miss	G. Fyshtel	Retainer	0
4	7	39	Miss	S. Knight	Retainer	1956
5	8	21	Mr	Stuart Salymen	Retainer	1976
6	10	2	Miss	C. Rabertsan	Retainer	0
7	11	29	Mr	K. Pushperejeh	Retainer	1970
8	13	52	Mr	Colin Turley	Retainer	0
9	14	52	Miss	L. Fynchem	Other	0

Figure 14 shows the data that has been cleansed.

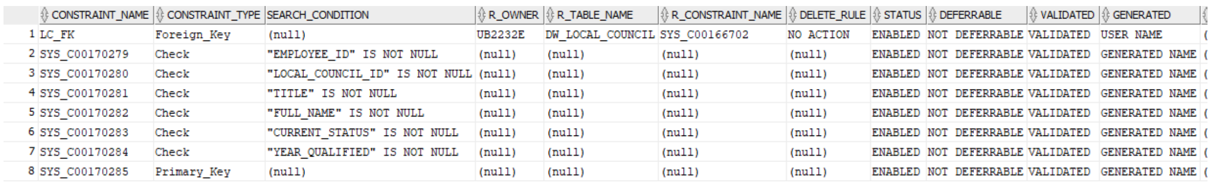
UPDATE EMPLOYEE_DETAILS SET YEAR_QUALIFIED = '0' WHERE YEAR_QUALIFIED IS NULL;

Referring to Figure 14, you can also notice that year_qualified column has 0's in them. This is the default data that has nothing in them. Therefore, instead of having nothing in them, the above query was used to replace all of this. You can also note that first_name and last name has been inserted together. This makes it easier to view the full name of the employee. In addition, this gets rid of any unnecessary empty columns that have no first name. You can see the query below which was used to delete any unnecessary data. You may also notice that the last three columns have been dropped. This is due to all three columns had nothing inside these columns. Therefore, it was necessary for cleansing and efficient purposes to drop any columns that have nothing in them.

DELETE FROM EMPLOYEE_DETAILS WHERE FIRST_NAME IS NULL and LAST_NAME IS NULL;

You may also notice small changes that make a big difference too. I made sure that each employee's first name has a dot with only the first letter. For example, Mr T. Scofield. This is due to different people have long names and they short these names for efficient purposes. As I did this for all of the employees, I manually changed the employees that this is not relevant. I also made sure that each employee has a capital name for each employee. This was also done for last name too.

UPDATE EMPLOYEE_DETAILS SET FIRST_NAME = concat(FIRST_NAME, '.');
UPDATE EMPLOYEE_DETAILS SET FIRST_NAME = initcap(FIRST_NAME);



	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE	VALIDATED	GENERATED	
1	LC_FK	Foreign_Key	(null)	UB2232E	DW_LOCAL_COUNCIL	SYS_C00166702	NO ACTION	ENABLED	NOT DEFERRABLE	VALIDATED	USER NAME	(
2	SYS_C00170279	Check	"EMPLOYEE_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME	(
3	SYS_C00170280	Check	"LOCAL_COUNCIL_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME	(
4	SYS_C00170281	Check	"TITLE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME	(
5	SYS_C00170282	Check	"FULL_NAME" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME	(
6	SYS_C00170283	Check	"CURRENT_STATUS" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME	(
7	SYS_C00170284	Check	"YEAR_QUALIFIED" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME	(
8	SYS_C00170285	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME	(

Figure 15 shows all the constraints with employee details

Referring to Figure 15, this shows all of the constraints that are with employee details. You may notice that employee details have local council as a foreign key. This was key to make sure that this was added as all of the foreign keys that have been inserted much match the primary key with the relevant table. Otherwise, it would show an error that this cannot insert a foreign key. You may notice also that primary key was also added as this is a routine for all tables that have been inserted. Lastly, it was important to make sure that all columns must check that it is not null. Otherwise, they must follow the routine of the default data. You can see this change with all of the queries below:

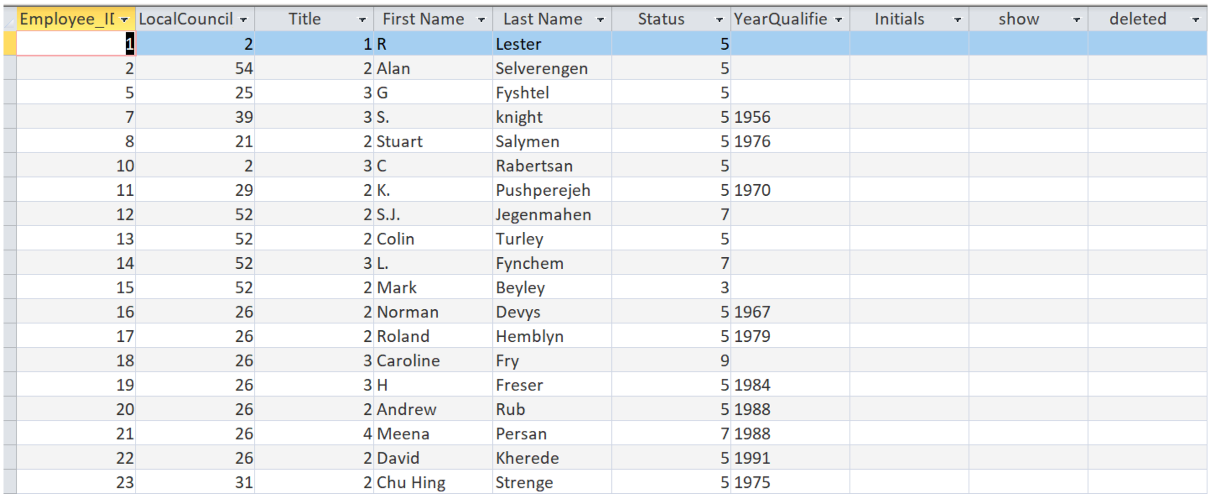
ALTER TABLE DW_EMPLOYEE_DETAILS MODIFY TITLE DEFAULT 'M.';

ALTER TABLE DW_EMPLOYEE_DETAILS MODIFY FULL_NAME DEFAULT 'N/A';

ALTER TABLE DW_EMPLOYEE_DETAILS MODIFY CURRENT_STATUS DEFAULT 'Not Known';

ALTER TABLE DW_EMPLOYEE_DETAILS MODIFY YEAR_QUALIFIED DEFAULT '0';

Referring to Figure 16, this shows the data previously. You can notice the big difference between the two tables of how the data was shown before I cleansed this table. You can see many of the data was inconsistent.



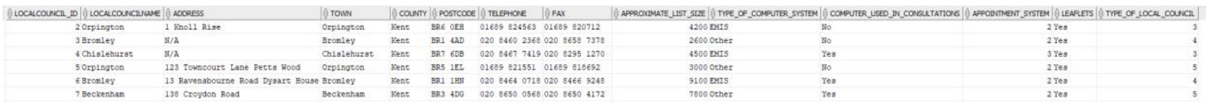
Employee_ID	LocalCouncil	Title	First Name	Last Name	Status	YearQualifie	Initials	show	deleted
1	2	1	R	Lester	5				
2	54	2	Alan	Selverengen	5				
5	25	3	G	Fyshtel	5				
7	39	3	S.	knight	5	1956			
8	21	2	Stuart	Salymen	5	1976			
10	2	3	C	Rabertsan	5				
11	29	2	K.	Pushperejeh	5	1970			
12	52	2	S.J.	Jegenmahen	7				
13	52	2	Colin	Turley	5				
14	52	3	L.	Fynchem	7				
15	52	2	Mark	Beyley	3				
16	26	2	Norman	Devys	5	1967			
17	26	2	Roland	Hemblyn	5	1979			
18	26	3	Caroline	Fry	9				
19	26	3	H	Freser	5	1984			
20	26	2	Andrew	Rub	5	1988			
21	26	4	Meena	Persan	7	1988			
22	26	2	David	Kherede	5	1991			
23	31	2	Chu Hing	Strenge	5	1975			

Figure 16 shows the original data within employee details.

Local Council

Referring to Figure 17, this shows the cleansed data that has been cleaned. Consistency is key, following from employee details, I did the same approach. Any foreign key links that this had with another table had to be replaced. This table only had one foreign key that were replaced with the data that that was type of computer systems. As stated previously, the foreign key data represents the data within the table. Below you can see that 4 represents Other. This was the same for each foreign key that was in the table.

UPDATE LOCAL_COUNCIL SET TYPE_OF_COMPUTER_SYSTEM = REPLACE(TYPE_OF_COMPUTER_SYSTEM,'4','Other');



LOCALCOUNCIL_ID	LOCALCOUNCILNAME	ADDRESS	TOWN	COUNTY	POSTCODE	TELEPHONE	FAX	APPROXIMATE_LIST_SIZE	TYPE_OF_COMPUTER_SYSTEM	COMPUTER_USED_IN_CONSULTATIONS	APPOINTMENT_SYSTEM	LEAFLETS	TYPE_OF_LOCAL_COUNCIL
2	Orpington	1 Knoll Rise	Orpington	Kent	BR6 0EH	01689 824563	01689 820712	4200	EMIS	No	2	Yes	3
3	Bromley	N/A	Bromley	Kent	BR1 4AD	020 8460 2368	020 8658 7378	2600	Other	No	2	Yes	4
4	Chislehurst	N/A	Chislehurst	Kent	BR7 6DB	020 8467 7419	020 8295 1270	4500	EMIS	Yes	3	Yes	3
5	Orpington	123 Towncourt Lane Petts Wood	Orpington	Kent	BR5 1EL	01689 821551	01689 818692	3000	Other	No	2	Yes	5
6	Bromley	13 Ravensbourne Road Dysart House	Bromley	Kent	BR1 1HN	020 8464 0718	020 8466 9248	9100	EMIS	Yes	2	Yes	4
7	Beckenham	138 Croydon Road	Beckenham	Kent	BR3 4DG	020 8650 0568	020 8650 4172	7800	Other	Yes	2	Yes	5

Figure 17 shows the cleansed data for local council table

For efficiency purposes, I had to get rid of any empty rows. I only noticed that there was only 1 row using a simple select statement to do this. Therefore, it was only right to delete this using the id that has been selected.

DELETE FROM local_council WHERE LOCALCOUNCIL_ID = '57';

I also noticed that many of the data that got transferred from stated 1 and 0. This was in reference to YES and NO for the answer. This was an error that had been made during the ETL stage. This was noticed when comparing the data. The following queries set this straight by replacing the 1s and 0s within leaflet and computer consultation.

UPDATE LOCAL_COUNCIL SET COMPUTER_USED_IN_CONSULTATIONS = REPLACE(COMPUTER_USED_IN_CONSULTATIONS,'1','Yes');

UPDATE LOCAL_COUNCIL SET LEAFLETS = REPLACE(LEAFLETS,'2',No);

Once that these queries were inserted to clean these data, I had to fix the address that was a mess. This included the full address for the local council. I noticed that many of the data had been swapped around and this was not in the correct place. I used the following query below to update these addresses. However, the issue with this was it was too slow to do. Therefore, as it 63 records to do, I recorded only some of the update statements. Within the SQL scripts, I did not want to record all update statements as it would have been too long to do so. Below only shows you an example of how I did one of them.

UPDATE LOCAL_COUNCIL SET COUNTY = 'Kent' WHERE localcouncil_id = '60';

UPDATE LOCAL_COUNCIL SET TOWN = 'Beckenham' WHERE localcouncil_id = '60';

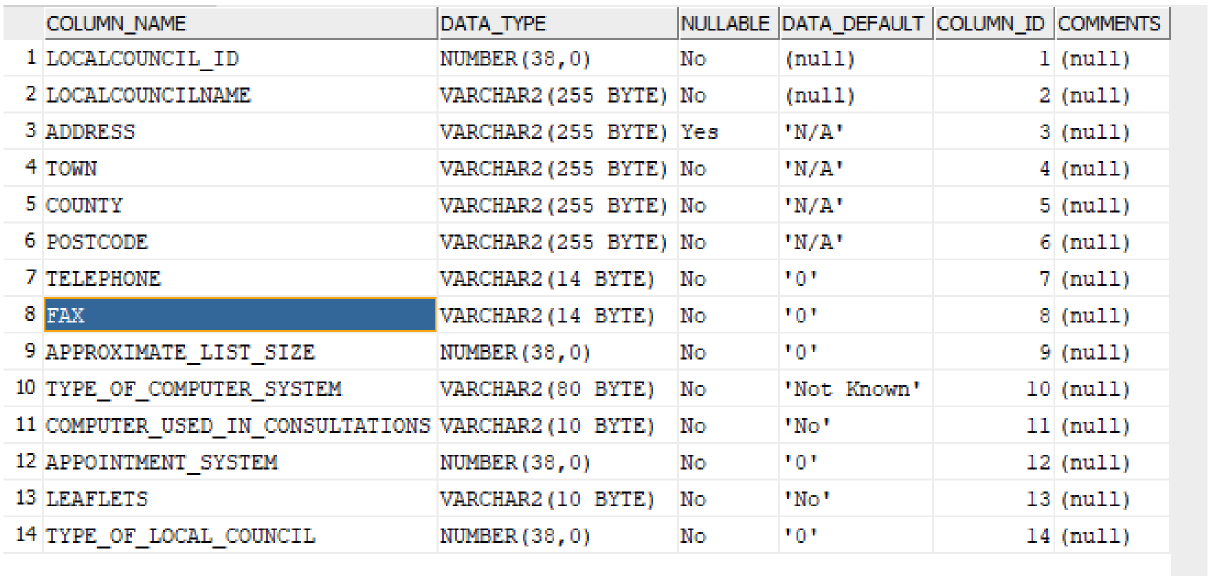
UPDATE LOCAL_COUNCIL SET ADDRESS_LINE_2 = '' WHERE localcouncil_id = '60';

As for efficiency, I had to replace the data with default data. This is a value that has been predesignated. Therefore, a user would recognize that this data is default. Below are some of the queries where these empty rows have replaced the data with default data. Referring to Figure 18 where this shows all of the default data using the query below to make this happen.

UPDATE DW_LOCAL_COUNCIL SET APPROXIMATE_LIST_SIZE = '0000' WHERE APPROXIMATE_LIST_SIZE IS NULL;

UPDATE DW_LOCAL_COUNCIL SET ADDRESS = 'N/A' WHERE ADDRESS IS NULL;

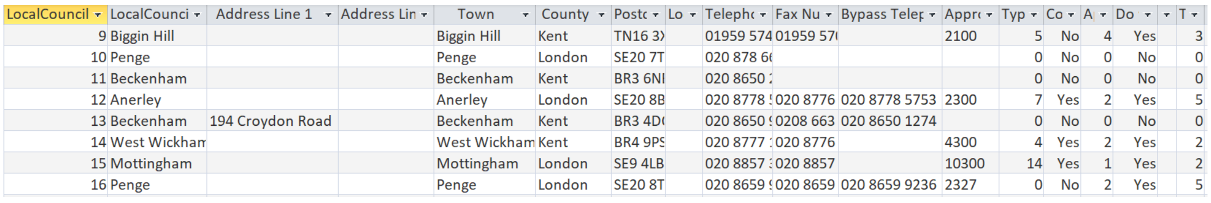
UPDATE DW_LOCAL_COUNCIL SET FAX = `0` WHERE FAX IS NULL;



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	LOCALCOUNCIL_ID	NUMBER(38,0)	No	(null)	1	(null)
2	LOCALCOUNCILNAME	VARCHAR2(255 BYTE)	No	(null)	2	(null)
3	ADDRESS	VARCHAR2(255 BYTE)	Yes	'N/A'	3	(null)
4	TOWN	VARCHAR2(255 BYTE)	No	'N/A'	4	(null)
5	COUNTY	VARCHAR2(255 BYTE)	No	'N/A'	5	(null)
6	POSTCODE	VARCHAR2(255 BYTE)	No	'N/A'	6	(null)
7	TELEPHONE	VARCHAR2(14 BYTE)	No	'0'	7	(null)
8	FAX	VARCHAR2(14 BYTE)	No	'0'	8	(null)
9	APPROXIMATE_LIST_SIZE	NUMBER(38,0)	No	'0'	9	(null)
10	TYPE_OF_COMPUTER_SYSTEM	VARCHAR2(80 BYTE)	No	'Not Known'	10	(null)
11	COMPUTER_USED_IN_CONSULTATIONS	VARCHAR2(10 BYTE)	No	'No'	11	(null)
12	APPOINTMENT_SYSTEM	NUMBER(38,0)	No	'0'	12	(null)
13	LEAFLETS	VARCHAR2(10 BYTE)	No	'No'	13	(null)
14	TYPE_OF_LOCAL_COUNCIL	NUMBER(38,0)	No	'0'	14	(null)

Figure 18 shows the default data for local council

Referring to Figure 19, this shows us the original data that has been portrayed. You may notice that two columns have been dropped. Locality and Date of Last Update were both dropped as they were no data that linked with these columns. Therefore, it was no use and choice but to drop these and they were not in use. However, you may notice the difference between the data of Figure 9 and Figure 7.



LocalCouncil	LocalCounci	Address Line 1	Address Lin	Town	County	Postc	Lo	Telephc	Fax Nu	Bypass Telep	Appro	Typ	Co	A	Do		T
9	Biggin Hill			Biggin Hill	Kent	TN16 3X		01959 574	01959 57		2100	5	No	4	Yes		3
10	Penge			Penge	London	SE20 7T		020 878 6				0	No	0	No		0
11	Beckenham			Beckenham	Kent	BR3 6NI		020 8650				0	No	0	No		0
12	Anerley			Anerley	London	SE20 8B		020 8778	020 8776	020 8778 5753	2300	7	Yes	2	Yes		5
13	Beckenham	194 Croydon Road		Beckenham	Kent	BR3 4D		020 8650	0208 663	020 8650 1274		0	No	0	No		0
14	West Wickham			West Wickham	Kent	BR4 9PS		020 8777	020 8776		4300	4	Yes	2	Yes		2
15	Mottingham			Mottingham	London	SE9 4LB		020 8857	020 8857		10300	14	Yes	1	Yes		2
16	Penge			Penge	London	SE20 8T		020 8659	020 8659	020 8659 9236	2327	0	No	2	Yes		5

Figure 19 shows the original data within Access for local council

Referees' Details

As previously mentioned, the address had many missing data all over the columns. Therefore, they were too many update statements to record this many. Therefore, the below is an example of an example of how one of them was completed. Referring to Figure 20, it shows that the correct address is within the correct place. This was rearranged in the correct place.

UPDATE ST_REFEREES_DETAILS SET ADDRESS_LINE_2 = '' WHERE REFEREEID = '3';

UPDATE ST_REFEREES_DETAILS SET TOWN = 'Upper Norwood' WHERE REFEREEID = '3';

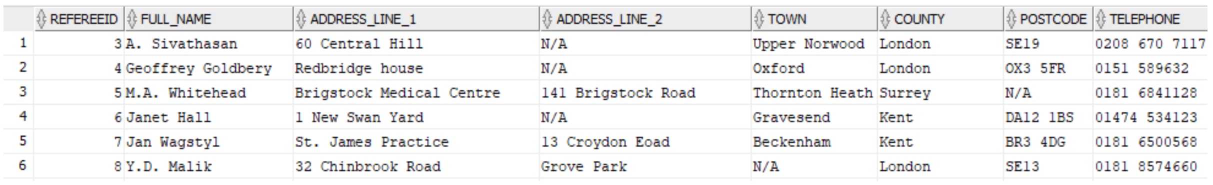
UPDATE ST_REFEREES_DETAILS SET COUNTY = 'London' WHERE REFEREEID = '3';

Within the address, they were apostrophes within the wrong section. This needed to be replaced. Therefore, the query below all apostrophes within postcode. This is the same for commas that were found within the address line too. Both were replaced by null.

UPDATE ST_REFEREES_DETAILS SET POSTCODE = REPLACE(POSTCODE,'`','');

UPDATE ST_REFEREES_DETAILS SET ADDRESS_LINE_1 = REPLACE(ADDRESS_LINE_1,',','');

UPDATE ST_REFEREES_DETAILS SET ADDRESS_LINE_2 = REPLACE(ADDRESS_LINE_2,',','');



	REFEREEID	FULL_NAME	ADDRESS_LINE_1	ADDRESS_LINE_2	TOWN	COUNTY	POSTCODE	TELEPHONE
1	3	A. Sivathasan	60 Central Hill	N/A	Upper Norwood	London	SE19	0208 670 7117
2	4	Geoffrey Goldbery	Redbridge house	N/A	Oxford	London	OX3 5FR	0151 589632
3	5	M.A. Whitehead	Brigstock Medical Centre	141 Brigstock Road	Thornton Heath	Surrey	N/A	0181 6841128
4	6	Janet Hall	1 New Swan Yard	N/A	Gravesend	Kent	DA12 1BS	01474 534123
5	7	Jan Wagstyl	St. James Practice	13 Croydon Eoad	Beckenham	Kent	BR3 4DG	0181 6500568
6	8	Y.D. Malik	32 Chinbrook Road	Grove Park	N/A	London	SE13	0181 8574660

Figure 20 shows the cleansed data for referees' details

As for previous errors, I found empty rows that had nothing in them. I did this by using a simple select statement to find this. By doing this, I deleted those that had nothing within all of those errors. I simply used the query below of the three that had been found.

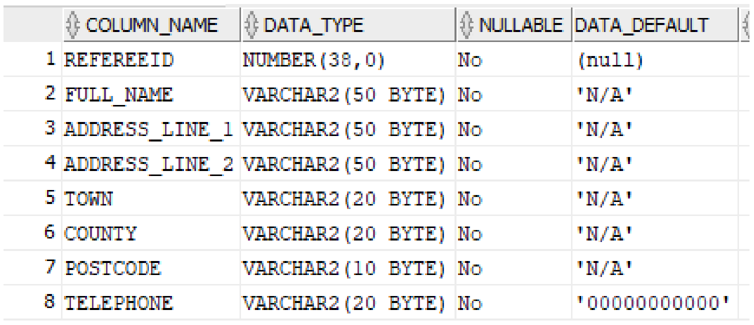
DELETE FROM ST_REFEREES_DETAILS WHERE refereeid = '131';

DELETE FROM ST_REFEREES_DETAILS WHERE refereeid = '233';

DELETE FROM ST_REFEREES_DETAILS WHERE refereeid = '243';

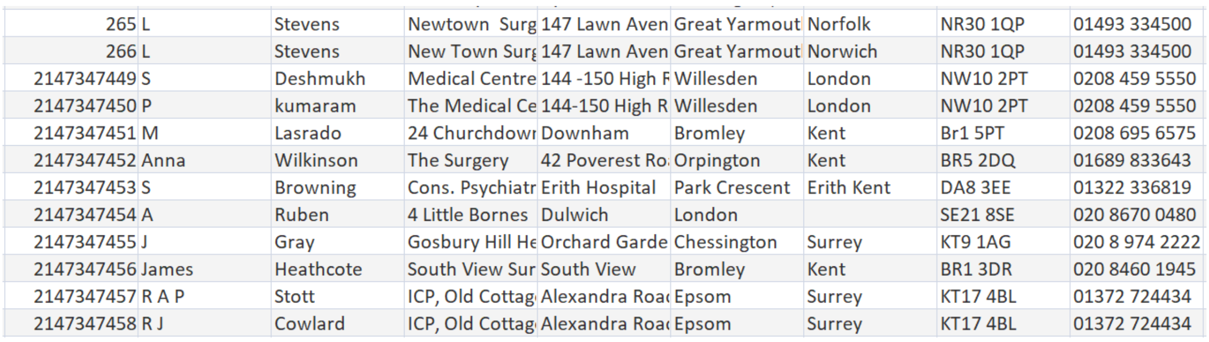
I also put default data within each of the columns to make sure that this is efficient. I used a simple update statement as for those other tables. Referring to Figure 21,

UPDATE DW_REFEREES_DETAILS SET ADDRESS_LINE_2 = 'N/A' WHERE ADDRESS_LINE_2 IS NULL;



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT
1	REFEREEID	NUMBER(38,0)	No	(null)
2	FULL_NAME	VARCHAR2(50 BYTE)	No	'N/A'
3	ADDRESS_LINE_1	VARCHAR2(50 BYTE)	No	'N/A'
4	ADDRESS_LINE_2	VARCHAR2(50 BYTE)	No	'N/A'
5	TOWN	VARCHAR2(20 BYTE)	No	'N/A'
6	COUNTY	VARCHAR2(20 BYTE)	No	'N/A'
7	POSTCODE	VARCHAR2(10 BYTE)	No	'N/A'
8	TELEPHONE	VARCHAR2(20 BYTE)	No	'00000000000'

Figure 21 shows the default data for referees' details



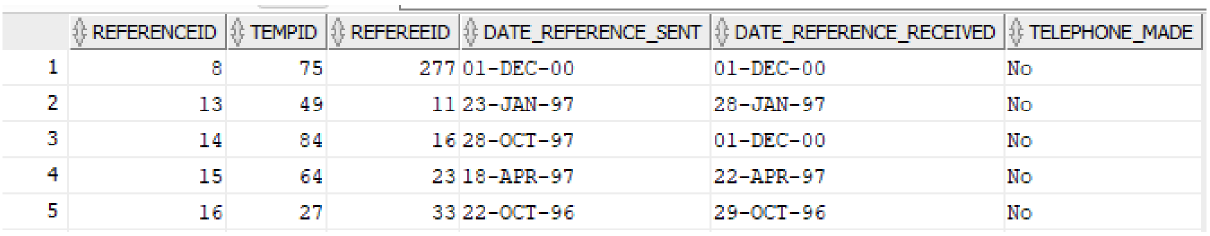
265	L	Stevens	Newtown Surg	147 Lawn Aven	Great Yarmout	Norfolk	NR30 1QP	01493 334500
266	L	Stevens	New Town Surg	147 Lawn Aven	Great Yarmout	Norwich	NR30 1QP	01493 334500
2147347449	S	Deshmukh	Medical Centre	144 -150 High F	Willesden	London	NW10 2PT	0208 459 5550
2147347450	P	kumaram	The Medical Ce	144-150 High R	Willesden	London	NW10 2PT	0208 459 5550
2147347451	M	Lasrado	24 Churchdowr	Downham	Bromley	Kent	Br1 5PT	0208 695 6575
2147347452	Anna	Wilkinson	The Surgery	42 Poverest Ro	Orpington	Kent	BR5 2DQ	01689 833643
2147347453	S	Browning	Cons. Psychiatr	Erith Hospital	Park Crescent	Erith Kent	DA8 3EE	01322 336819
2147347454	A	Ruben	4 Little Bornes	Dulwich	London		SE21 8SE	020 8670 0480
2147347455	J	Gray	Gosbury Hill He	Orchard Garde	Chessington	Surrey	KT9 1AG	020 8 974 2222
2147347456	James	Heathcote	South View Sur	South View	Bromley	Kent	BR1 3DR	020 8460 1945
2147347457	R A P	Stott	ICP, Old Cottag	Alexandra Roa	Epsom	Surrey	KT17 4BL	01372 724434
2147347458	R J	Cowlard	ICP, Old Cottag	Alexandra Roa	Epsom	Surrey	KT17 4BL	01372 724434

Figure 22 shows the original data within referees' details

Referring to Figure 22, this shows the original data within Referees' Details. You can see within some of the rows that the county, town are all switched. You can see that this is not efficient and as the queries above sorted this. You may notice that the primary keys are not efficient. This was updated as a primary key and updated within the references section where this is used as a foreign key.

References

Referring to Figure 23, this shows the original data for this table. As for this table, there was not much to change. As mentioned previously, I changed the correlated foreign key within this section. All I needed to do was match both in order to get it correct. As you may notice, it is inappropriate for inconsistent primary keys not to match. There was not much to update in this table.



	REFERENCEID	TEMPID	REFEREEID	DATE_REFERENCE_SENT	DATE_REFERENCE_RECEIVED	TELEPHONE_MADE
1	8	75	277	01-DEC-00	01-DEC-00	No
2	13	49	11	23-JAN-97	28-JAN-97	No
3	14	84	16	28-OCT-97	01-DEC-00	No
4	15	64	23	18-APR-97	22-APR-97	No
5	16	27	33	22-OCT-96	29-OCT-96	No

Figure 23 shows the original data for references

As mentioned, very little cleansing was needed for this. However, efficiency is needed to replace the default data. I used a default date of 1st December 2000. You may notice that throughout this whole table, this is a default date that need to be noticed. I must also add that the default data for telephone is No. This was not needed as none of them were empty. However, this must be mentioned to clarify this.

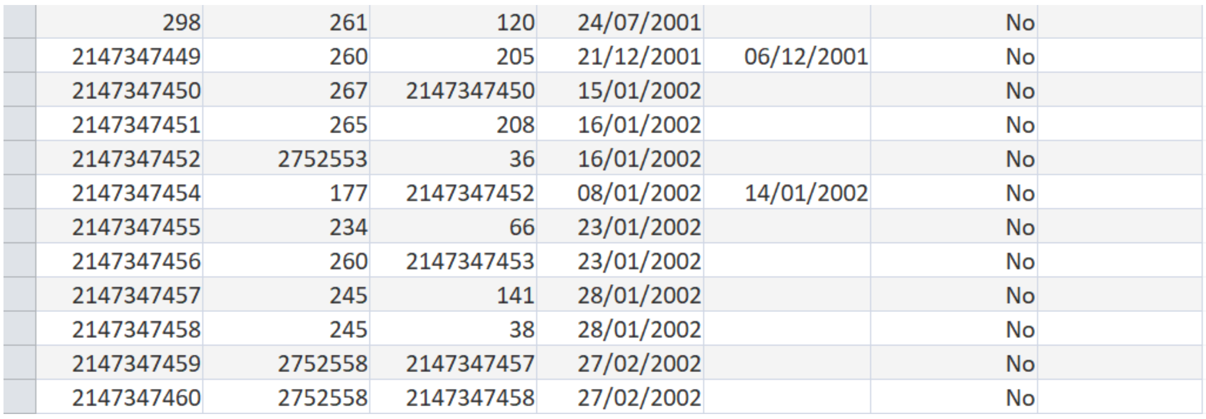
UPDATE DW_REFERENCES SET DATE_REFERENCE_SENT = '01-DEC-00' WHERE DATE_REFERENCE_SENT IS NULL;

UPDATE DW_REFERENCES SET DATE_REFERENCE_RECEIVED = '01-DEC-00' WHERE DATE_REFERENCE_RECEIVED IS NULL;

This is the same for efficiency purposes. When this data got transferred, it replaced the Yes with 1. Therefore, it needed to be replaced back to make that users understand this information that the telephone has been replaced.

UPDATE DW_REFERENCES SET TELEPHONE_MADE = REPLACE(TELEPHONE_MADE,'1','Yes');

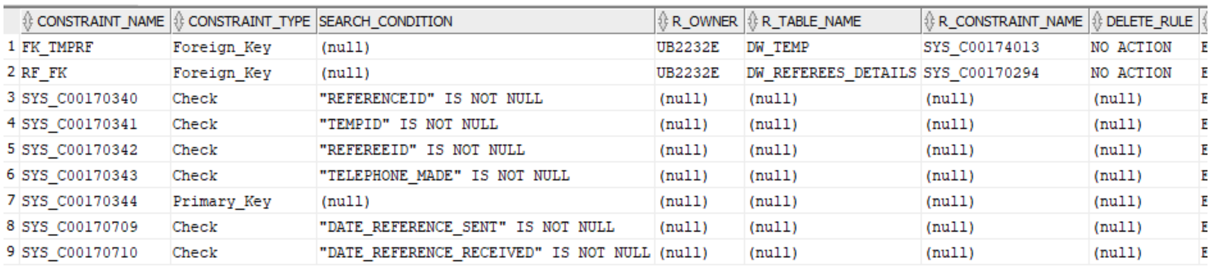
Referring to Figure 24, this shows the original data. As you can see, there are many missing data. You may notice that many of the data is missing. As previously mentioned, this was replaced with the default data. In addition, you can see, the inconsistent primary keys that were changed and made relevant.



	298	261	120	24/07/2001		No	
	2147347449	260	205	21/12/2001	06/12/2001	No	
	2147347450	267	2147347450	15/01/2002		No	
	2147347451	265	208	16/01/2002		No	
	2147347452	2752553	36	16/01/2002		No	
	2147347454	177	2147347452	08/01/2002	14/01/2002	No	
	2147347455	234	66	23/01/2002		No	
	2147347456	260	2147347453	23/01/2002		No	
	2147347457	245	141	28/01/2002		No	
	2147347458	245	38	28/01/2002		No	
	2147347459	2752558	2147347457	27/02/2002		No	
	2147347460	2752558	2147347458	27/02/2002		No	

Figure 24 shows the original data within references

Referring to Figure 25, this shows all constraints within References. Therefore, all these constraints that are foreign keys within other tables are shown below. In addition, other tables shows that it cannot be empty. In other words, default data must be entered in replace of this.

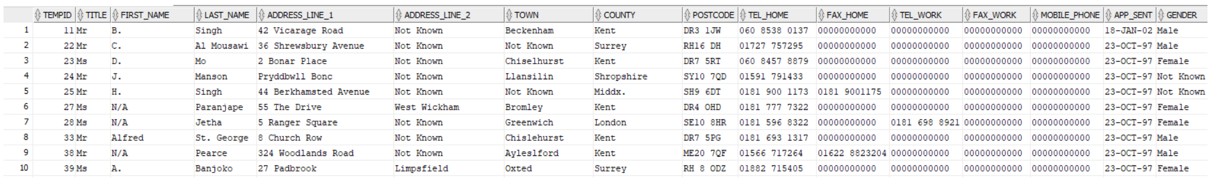


	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	
1	FK_TMPRF	Foreign_Key	(null)	UB2232E	DW_TEMP	SYS_C00174013	NO ACTION	E
2	RF_FK	Foreign_Key	(null)	UB2232E	DW_REFEREES_DETAILS	SYS_C00170294	NO ACTION	E
3	SYS_C00170340	Check	"REFERENCEID" IS NOT NULL	(null)	(null)	(null)	(null)	E
4	SYS_C00170341	Check	"TEMPID" IS NOT NULL	(null)	(null)	(null)	(null)	E
5	SYS_C00170342	Check	"REFEREEID" IS NOT NULL	(null)	(null)	(null)	(null)	E
6	SYS_C00170343	Check	"TELEPHONE_MADE" IS NOT NULL	(null)	(null)	(null)	(null)	E
7	SYS_C00170344	Primary_Key	(null)	(null)	(null)	(null)	(null)	E
8	SYS_C00170709	Check	"DATE_REFERENCE_SENT" IS NOT NULL	(null)	(null)	(null)	(null)	E
9	SYS_C00170710	Check	"DATE_REFERENCE_RECEIVED" IS NOT NULL	(null)	(null)	(null)	(null)	E

Figure 25 shows the constraints within references

Temp

As for challenges, this was the highest out of all tables that have been cleansed. This was a challenge due to 38 columns that this needed to be cleaned. Therefore, you can see with Figure 26, you can see only half of the columns that have been cleansed. However, I will be going through all the queries that have been used on each column throughout this table. As mentioned previously, the address had many of the data being placed in wrong columns. Therefore, I changed this using the queries without recording them. This is due to this being very long and repetitive.



	TEMPID	TITLE	FIRST_NAME	LAST_NAME	ADDRESS_LINE_1	ADDRESS_LINE_2	TOWN	COUNTY	POSTCODE	TEL_HOME	FAX_HOME	TEL_WORK	FAX_WORK	MOBILE_PHONE	APP_SENT	GENDER
1	11	Mr	B.	Singh	42 Vicarage Road	Not Known	Beckenham	Kent	DR3 1JW	060 8538 0137	00000000000	00000000000	00000000000	00000000000	18-JAN-02	Male
2	22	Mr	C.	Al Mousawi	36 Shrewsbury Avenue	Not Known	Not Known	Surrey	RH16 DH	01727 757295	00000000000	00000000000	00000000000	00000000000	23-OCT-97	Male
3	23	Ms	D.	Mo	2 Bonar Place	Not Known	Chiselhurst	Kent	DR7 5RT	060 8457 8879	00000000000	00000000000	00000000000	00000000000	23-OCT-97	Female
4	24	Mr	J.	Manson	Pryddbwll Bonc	Not Known	Llansilin	Shropshire	SY10 7QD	01591 791433	00000000000	00000000000	00000000000	00000000000	23-OCT-97	Not Known
5	25	Mr	H.	Singh	44 Berkhamsted Avenue	Not Known	Not Known	Middx.	SH9 6DT	0181 900 1173	0181 9001175	00000000000	00000000000	00000000000	23-OCT-97	Not Known
6	27	Ms	N/A	Paranjape	55 The Drive	West Wickham	Bromley	Kent	DR4 OHD	0181 777 7322	00000000000	00000000000	00000000000	00000000000	23-OCT-97	Female
7	28	Ms	N/A	Jetha	5 Ranger Square	Not Known	Greenwich	London	SE10 8HR	0181 596 8322	00000000000	0181 698 8921	00000000000	00000000000	23-OCT-97	Female
8	33	Mr	Alfred	St. George	8 Church Row	Not Known	Chislehurst	Kent	DR7 5PG	0181 693 1317	00000000000	00000000000	00000000000	00000000000	23-OCT-97	Male
9	38	Mr	N/A	Pearce	324 Woodlands Road	Not Known	Aylesiford	Kent	ME20 7QF	01566 717264	01622 8823204	00000000000	00000000000	00000000000	23-OCT-97	Male
10	39	Ms	A.	Banjoko	27 Padbrook	Limpsfield	Oxted	Surrey	RH 8 ODZ	01882 715405	00000000000	00000000000	00000000000	00000000000	23-OCT-97	Female

Figure 26 shows the cleansed data for temp table

Firstly, I replaced all tables that were going to be replaced. I used the same queries that I have been using for the other tables for efficient purposes. Each example that has been given below is an example for a different column. As said previously, the foreign key is in correspondent to the primary key within the relevant table. It is key to make sure that both match in order to execute each table.

UPDATE ST_TEMP SET NATIONALITY = REPLACE(NATIONALITY,'1','British');

UPDATE ST_TEMP SET USE_OF_A_CAR = REPLACE(USE_OF_A_CAR,'0', 'No');

UPDATE DW_TEMP SET CURRENT_STATUS = REPLACE(CURRENT_STATUS,'0','Not Known');

UPDATE DW_TEMP SET TYPE_OF_COVER_PREFERRED = REPLACE(TYPE_OF_COVER_PREFERRED,'0','Not Known');

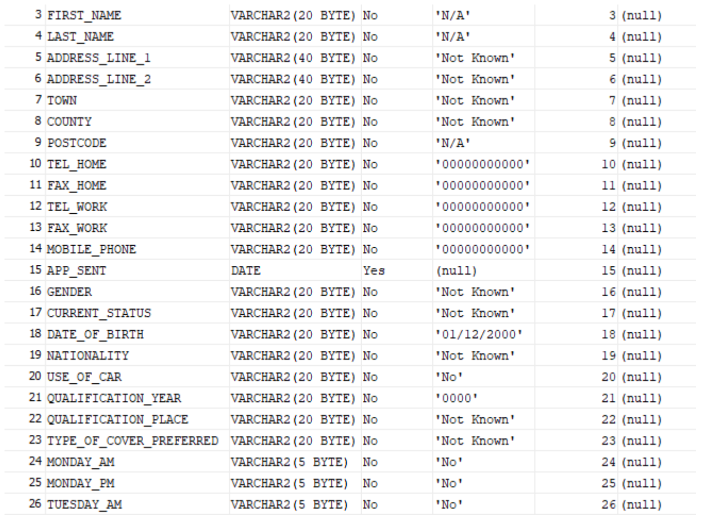
UPDATE DW_TEMP SET TEMP_REGISTRATION_STATUS = REPLACE(TEMP_REGISTRATION_STATUS,'0','Not Known');

UPDATE DW_TEMP SET TEMPSTATUS = REPLACE(TEMPSTATUS,'1.0','Live');

I also made sure that for all 38 columns, it was necessary to replace the default data for all. I made sure that each of the columns and relevant of the previous tables. Referring to Figure 27 and the query below, you can see both examples of how this was complete.

UPDATE ST_TEMP SET MONDAY_AM = REPLACE(MONDAY_AM,'1', 'Yes');

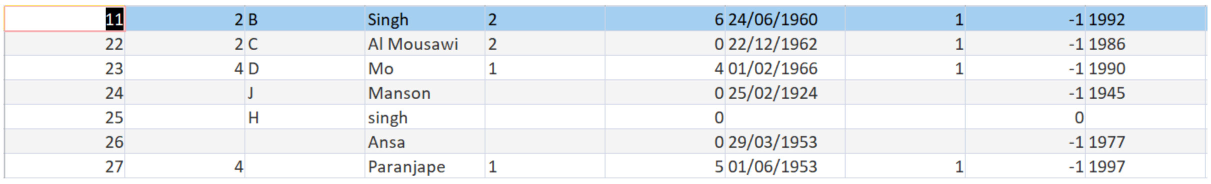
UPDATE ST_TEMP SET ADDRESS_LINE_2 = 'Not Known' WHERE ADDRESS_LINE_2 IS NULL;



3	FIRST_NAME	VARCHAR2 (20 BYTE)	No	'N/A'	3	(null)
4	LAST_NAME	VARCHAR2 (20 BYTE)	No	'N/A'	4	(null)
5	ADDRESS_LINE_1	VARCHAR2 (40 BYTE)	No	'Not Known'	5	(null)
6	ADDRESS_LINE_2	VARCHAR2 (40 BYTE)	No	'Not Known'	6	(null)
7	TOWN	VARCHAR2 (20 BYTE)	No	'Not Known'	7	(null)
8	COUNTY	VARCHAR2 (20 BYTE)	No	'Not Known'	8	(null)
9	POSTCODE	VARCHAR2 (20 BYTE)	No	'N/A'	9	(null)
10	TEL_HOME	VARCHAR2 (20 BYTE)	No	'00000000000'	10	(null)
11	FAX_HOME	VARCHAR2 (20 BYTE)	No	'00000000000'	11	(null)
12	TEL_WORK	VARCHAR2 (20 BYTE)	No	'00000000000'	12	(null)
13	FAX_WORK	VARCHAR2 (20 BYTE)	No	'00000000000'	13	(null)
14	MOBILE_PHONE	VARCHAR2 (20 BYTE)	No	'00000000000'	14	(null)
15	APP_SENT	DATE	Yes	(null)	15	(null)
16	GENDER	VARCHAR2 (20 BYTE)	No	'Not Known'	16	(null)
17	CURRENT_STATUS	VARCHAR2 (20 BYTE)	No	'Not Known'	17	(null)
18	DATE_OF_BIRTH	VARCHAR2 (20 BYTE)	No	'01/12/2000'	18	(null)
19	NATIONALITY	VARCHAR2 (20 BYTE)	No	'Not Known'	19	(null)
20	USE_OF_CAR	VARCHAR2 (20 BYTE)	No	'No'	20	(null)
21	QUALIFICATION_YEAR	VARCHAR2 (20 BYTE)	No	'0000'	21	(null)
22	QUALIFICATION_PLACE	VARCHAR2 (20 BYTE)	No	'Not Known'	22	(null)
23	TYPE_OF_COVER_PREFERRED	VARCHAR2 (20 BYTE)	No	'Not Known'	23	(null)
24	MONDAY_AM	VARCHAR2 (5 BYTE)	No	'No'	24	(null)
25	MONDAY_PM	VARCHAR2 (5 BYTE)	No	'No'	25	(null)
26	TUESDAY_AM	VARCHAR2 (5 BYTE)	No	'No'	26	(null)

Figure 27 shows all default data within temp

Referring to Figure 28, you can see that the original data. You may notice that they are many different empty rows that does not correspond to the original data that has been shown with the queries. This is important to notice the change of what has been changed too.



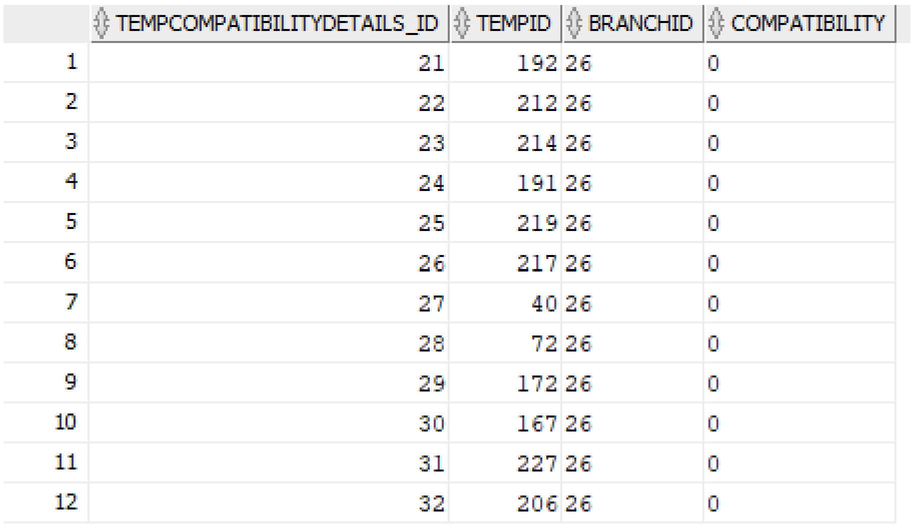
11	2	B	Singh	2	6	24/06/1960	1	-1	1992
22	2	C	Al Mousawi	2	0	22/12/1962	1	-1	1986
23	4	D	Mo	1	4	01/02/1966	1	-1	1990
24		J	Manson		0	25/02/1924		-1	1945
25		H	singh		0			0	
26			Ansa		0	29/03/1953		-1	1977
27	4		Paranjape	1	5	01/06/1953	1	-1	1997

Figure 28 shows the original data within temp

Temp Compatibility Details

I kept this table only due to the foreign key that this was made. The only data that was changed throughout this table was to add the ID to make sure this can be referenced back. I also made sure that the constraint of this has been declared too. The query below has been showed to demonstrate how this was done. As you can see, the sequence was created with the next value. This was created and automatically added within temp compatibility details. Referring to Figure 29, you can see that this has been completed and transferred successfully.

```
Declare
Cursor c_client is
SELECT TEMPID, BRANCHID, COMPATIBILITY from ST_TEMPCOMPATIBILITYDETAILS;
Begin
for c_rec in c_client loop
insert into DW_TEMPCOMPATIBILITYDETAILS
values(DW_TEMPCOMPATIBILITYDETAILS_SEQ.nextval, c_rec.TEMPID, c_rec.BRANCHID,
c_rec.COMPATIBILITY);
```



	TEMPCOMPATIBILITYDETAILS_ID	TEMPID	BRANCHID	COMPATIBILITY
1	21	192	26	0
2	22	212	26	0
3	23	214	26	0
4	24	191	26	0
5	25	219	26	0
6	26	217	26	0
7	27	40	26	0
8	28	72	26	0
9	29	172	26	0
10	30	167	26	0
11	31	227	26	0
12	32	206	26	0

Figure 29 shows the original data for temp compatibility details

Temp Request

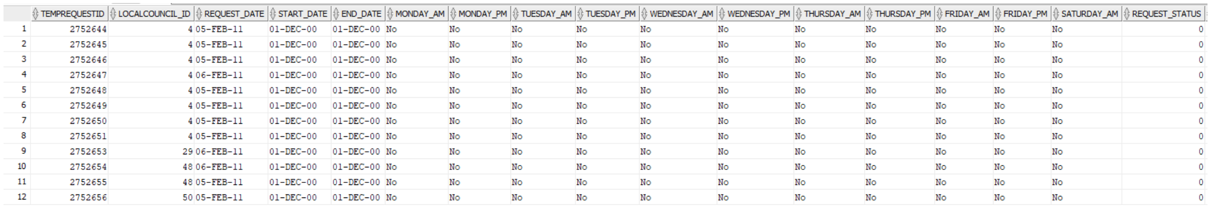
Referring to Figure 30, you may notice that there is not much to change. You may notice that not much difference was made on this table. I updated the queries that were empty using the update statement below for the relevant columns. I also included those that have default data within the relevant columns to notify administrators of these changes that have been made.

UPDATE ST_TEMP_REQUEST SET START_DATE = '01-DEC-00' WHERE START_DATE IS NULL;

UPDATE ST_TEMP_REQUEST SET COMMENTS = ‘No Comment' WHERE END_DATE IS NULL;

The last change that I noticed is that I saw many zeros within the table. I felt that this looked unprofessional and not understandable from a user’s point of view. Therefore, I changed this using the replace update statement below which updated all zeros to no within seconds. This was changed due to the transmission of data. When the data was transferred, the data within Monday_AM changed to all zeros. Therefore, this needed to be changed back for efficiency purposes.

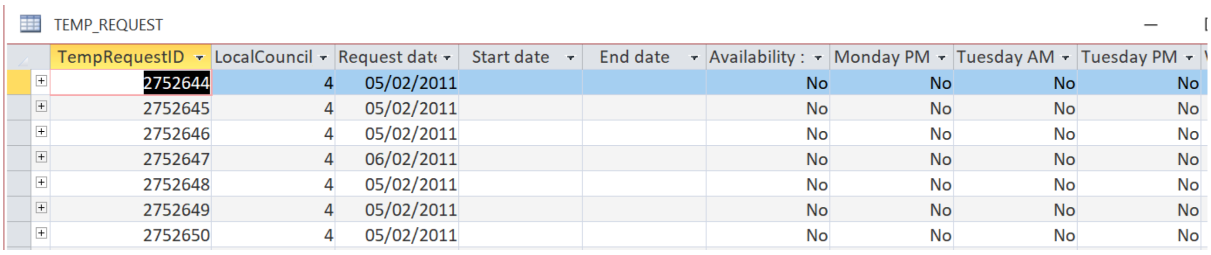
UPDATE DW_TEMP_REQUEST SET WEDNESDAY_AM = REPLACE(WEDNESDAY_AM,'0', 'No');



	TEMPREQUESTID	LOCALCOUNCIL_ID	REQUEST_DATE	START_DATE	END_DATE	MONDAY_AM	MONDAY_PM	TUESDAY_AM	TUESDAY_PM	WEDNESDAY_AM	WEDNESDAY_PM	THURSDAY_AM	THURSDAY_PM	FRIDAY_AM	FRIDAY_PM	SATURDAY_AM	REQUEST_STATUS
1	2752644	4	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
2	2752645	4	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
3	2752646	4	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
4	2752647	4	06-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
5	2752648	4	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
6	2752649	4	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
7	2752650	4	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
8	2752651	4	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
9	2752653	29	06-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
10	2752654	48	06-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
11	2752655	48	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0
12	2752656	50	05-FEB-11	01-DEC-00	01-DEC-00	No	No	No	No	No	No	No	No	No	No	No	0

Figure 30 shows the data for temp request.

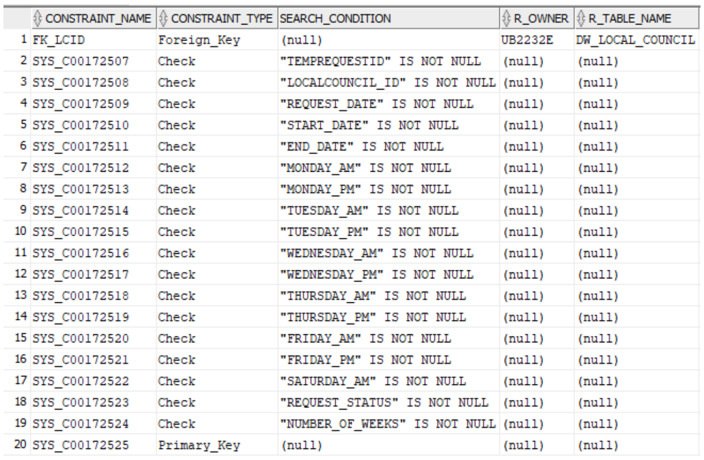
Referring to Figure 31, this demonstrates the original data that has shown. You may notice the difference between this and Figure 30. You notice that the original data, many of the statements contained empty data which got replaced by default date as you may notice.



TEMP_REQUEST

TempRequestID	LocalCouncil	Request dat	Start date	End date	Availability :	Monday PM	Tuesday AM	Tuesday PM
2752644	4	05/02/2011			No	No	No	No
2752645	4	05/02/2011			No	No	No	No
2752646	4	05/02/2011			No	No	No	No
2752647	4	06/02/2011			No	No	No	No
2752648	4	05/02/2011			No	No	No	No
2752649	4	05/02/2011			No	No	No	No
2752650	4	05/02/2011			No	No	No	No

Figure 31 shows the original data for Temp Request.



	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME
1	FK_LCID	Foreign_Key	(null)	UB2232E	DW_LOCAL_COUNCIL
2	SYS_C00172507	Check	"TEMPREQUESTID" IS NOT NULL	(null)	(null)
3	SYS_C00172508	Check	"LOCALCOUNCIL_ID" IS NOT NULL	(null)	(null)
4	SYS_C00172509	Check	"REQUEST_DATE" IS NOT NULL	(null)	(null)
5	SYS_C00172510	Check	"START_DATE" IS NOT NULL	(null)	(null)
6	SYS_C00172511	Check	"END_DATE" IS NOT NULL	(null)	(null)
7	SYS_C00172512	Check	"MONDAY_AM" IS NOT NULL	(null)	(null)
8	SYS_C00172513	Check	"MONDAY_PM" IS NOT NULL	(null)	(null)
9	SYS_C00172514	Check	"TUESDAY_AM" IS NOT NULL	(null)	(null)
10	SYS_C00172515	Check	"TUESDAY_PM" IS NOT NULL	(null)	(null)
11	SYS_C00172516	Check	"WEDNESDAY_AM" IS NOT NULL	(null)	(null)
12	SYS_C00172517	Check	"WEDNESDAY_PM" IS NOT NULL	(null)	(null)
13	SYS_C00172518	Check	"THURSDAY_AM" IS NOT NULL	(null)	(null)
14	SYS_C00172519	Check	"THURSDAY_PM" IS NOT NULL	(null)	(null)
15	SYS_C00172520	Check	"FRIDAY_AM" IS NOT NULL	(null)	(null)
16	SYS_C00172521	Check	"FRIDAY_PM" IS NOT NULL	(null)	(null)
17	SYS_C00172522	Check	"SATURDAY_AM" IS NOT NULL	(null)	(null)
18	SYS_C00172523	Check	"REQUEST_STATUS" IS NOT NULL	(null)	(null)
19	SYS_C00172524	Check	"NUMBER_OF_WEEKS" IS NOT NULL	(null)	(null)
20	SYS_C00172525	Primary_Key	(null)	(null)	(null)

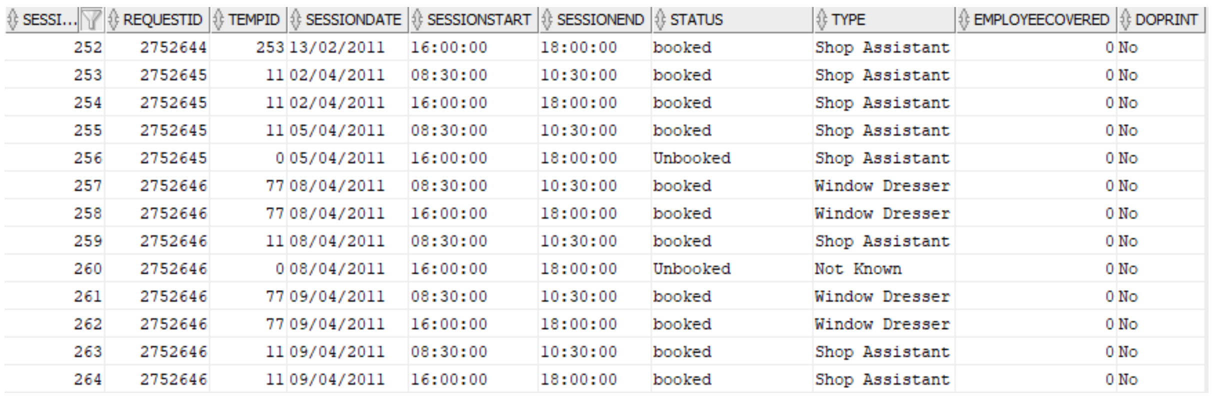
Referring to Figure 32, this shows all of the conditions that these checks within temp request. You may notice that one of the conditions this does check is the foreign key, and all the not nulls within other columns too. This is important to not enter any inconsistent data

Figure 32 shows all the constraints of temp request

Session

Referring to Figure 33, this shows the original staging data before being transferred as a fact table. A few issues came into place when I came across this table. Two columns were converted to date when transmission of data from Access to Oracle. This was noticed when cleaning the data. This got fixed by changing the data type from Date/Time to Short Text. I later converted the data again from Access to Oracle. This was the result by showing the data once this was changed. I also changed the Type column to its relevant foreign key using the update statement below.

UPDATE DW_SESSION SET TYPE_OF_COVER_PREFERRED = REPLACE(TYPE,'1','Shop Assistant');



SESSI...	REQUESTID	TEMPID	SESSIONDATE	SESSIONSTART	SESSIONEND	STATUS	TYPE	EMPLOYEECOVERED	DOPRINT
252	2752644	253	13/02/2011	16:00:00	18:00:00	booked	Shop Assistant	0	No
253	2752645	11	02/04/2011	08:30:00	10:30:00	booked	Shop Assistant	0	No
254	2752645	11	02/04/2011	16:00:00	18:00:00	booked	Shop Assistant	0	No
255	2752645	11	05/04/2011	08:30:00	10:30:00	booked	Shop Assistant	0	No
256	2752645	0	05/04/2011	16:00:00	18:00:00	Unbooked	Shop Assistant	0	No
257	2752646	77	08/04/2011	08:30:00	10:30:00	booked	Window Dresser	0	No
258	2752646	77	08/04/2011	16:00:00	18:00:00	booked	Window Dresser	0	No
259	2752646	11	08/04/2011	08:30:00	10:30:00	booked	Shop Assistant	0	No
260	2752646	0	08/04/2011	16:00:00	18:00:00	Unbooked	Not Known	0	No
261	2752646	77	09/04/2011	08:30:00	10:30:00	booked	Window Dresser	0	No
262	2752646	77	09/04/2011	16:00:00	18:00:00	booked	Window Dresser	0	No
263	2752646	11	09/04/2011	08:30:00	10:30:00	booked	Shop Assistant	0	No
264	2752646	11	09/04/2011	16:00:00	18:00:00	booked	Shop Assistant	0	No

Figure 33 shows the cleaned data for session table.

You may notice that throughout each of the tables, I have kept the consistency key with each table. This is important to make sure that all tables have the same consistency and efficiency. If not, this could eventually affect performance if not done correctly. As you may realise that the consistency of structure throughout each table is key. This is paramount as each table has different tables of consistency, pattern, cleaning to do so. As previously mentioned, I would also like to mention that the staging area for all these tables have been done at the staging phase. Therefore, once the data was cleansed, it was transferred using the cursor. This was the way that myself and my colleague tackled this difficult task. I would also like to point out that the Fact Table and other tables will be mentioned within the next section.

I would also like to point out that for the session table, this was the staging area. Therefore, as you may see all default data and constraints being mentioned for the other tables. This would not be necessary as this is not the final dimension being added. This would be explained further when this is enhanced to a fact table later on in this report.

B: Implementation of Dimensions & Fact

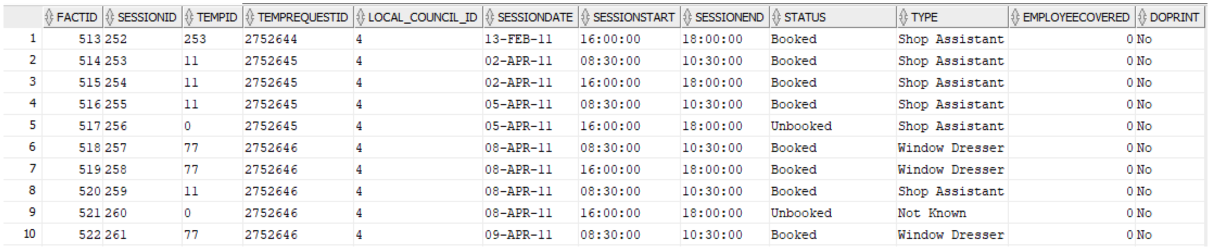
Within this section, I will go through and validate the schema within Figure 1 to confirm the implementation of the schema. As shown in Figure 1, I used the Snowflake Schema to do this. I do realise the complexity of this and I have made it easier by including Materialized Views. I will explain within this section of how, and what contains within the fact table and materialized view. The Dimensions will be previously touched upon too. This would be the same layout as previously explained.

Fact Session

Referring to Figure 34, this shows the cleansed data from the session table by using different id. I did this by doing this a different way. You may notice that the transfer from this was used by the DW_Session table. This was purely done to make it consistent with all the other tables. This was made the fact table later in the stage of this. Therefore, you may see this. However, as soon as this was successful, it was later dropped. As you may notice, you can see that this was done differently to using a cursor as was suggested. However, I found that this was an easier way to see by simply using a select statement and inserting this as a combination of using both. I did struggle with this later by many errors. However, this will be mentioned within the issues and evaluation section of the report. You may notice that I used many of the INNER JOIN statements. This is a combination of using all three to join all ID's together. I will explain the reasoning behind this within the evaluation stage also.

```
INSERT INTO DW_FACT_SESSION( FACTID, SESSIONID, TEMPREQUESTID, TEMPID,
LOCAL_COUNCIL_ID,  SESSIONDATE, SESSIONSTART, SESSIONEND, STATUS, TYPE,
EMPLOYEECOVERED, DOPRINT)

select  DW_FACT_TABLE_SEQ.nextval, s.sessionid, r.temprequestid, t.tempid, c.localcouncil_id,
To_Date(s.sessiondate, 'DD-MM-YYYY'),  s.sessionstart, s.sessionend, s.status, s.type,
s.employeecovered, s.doprint from dw_session s
inner join dw_temp t on (s.tempid = t.tempid)
inner join DW_TEMP_REQUEST r on (s.TEMPREQUESTID = r.TEMPREQUESTID)
inner join DW_LOCAL_COUNCIL c on (r.LOCALCOUNCIL_ID = c.LOCALCOUNCIL_ID);
```



	FACTID	SESSIONID	TEMPID	TEMPREQUESTID	LOCAL_COUNCIL_ID	SESSIONDATE	SESSIONSTART	SESSIONEND	STATUS	TYPE	EMPLOYEECOVERED	DOPRINT
1	513	252	253	2752644	4	13-FEB-11	16:00:00	18:00:00	Booked	Shop Assistant	0	No
2	514	253	11	2752645	4	02-APR-11	08:30:00	10:30:00	Booked	Shop Assistant	0	No
3	515	254	11	2752645	4	02-APR-11	16:00:00	18:00:00	Booked	Shop Assistant	0	No
4	516	255	11	2752645	4	05-APR-11	08:30:00	10:30:00	Booked	Shop Assistant	0	No
5	517	256	0	2752645	4	05-APR-11	16:00:00	18:00:00	Unbooked	Shop Assistant	0	No
6	518	257	77	2752646	4	08-APR-11	08:30:00	10:30:00	Booked	Window Dresser	0	No
7	519	258	77	2752646	4	08-APR-11	16:00:00	18:00:00	Booked	Window Dresser	0	No
8	520	259	11	2752646	4	08-APR-11	08:30:00	10:30:00	Booked	Shop Assistant	0	No
9	521	260	0	2752646	4	08-APR-11	16:00:00	18:00:00	Unbooked	Not Known	0	No
10	522	261	77	2752646	4	09-APR-11	08:30:00	10:30:00	Booked	Window Dresser	0	No

Figure 34 shows the sample data for the Fact Table.

Dimensions

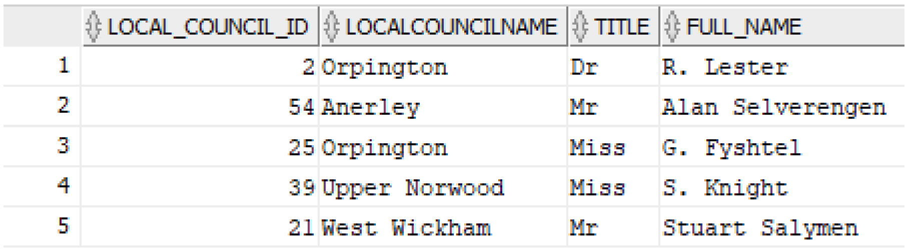
I would like to mention back to Figure 1 where this shows the layout that has been executed. I would like to mention that the session was made the fact table as it had many foreign keys already linked with each other. I felt that this was necessary to add one more to make it complete. Although, it does not have the most data, it is the important data that the Government is trying gain information across. I would also like to point out that the necessary of keeping the layout for each is necessary. This bring out restrictions on each of the tables where consistency and efficiency is key. I would like to point out that all of the points of each table have been finalized and mentioned elsewhere. Hence, I can only summaries of what has been done.

Materialized View

We adopted a style of using a Snowflake Schema. I realised that the severity of using a Materialized View (MV) is necessary. Therefore, I will be four MVs that have a necessary link with each other. This would clarify and make it easier for the users to view. I will also mention the code that has been used to create these necessary views for each of them.

DW_View_Employee_Details

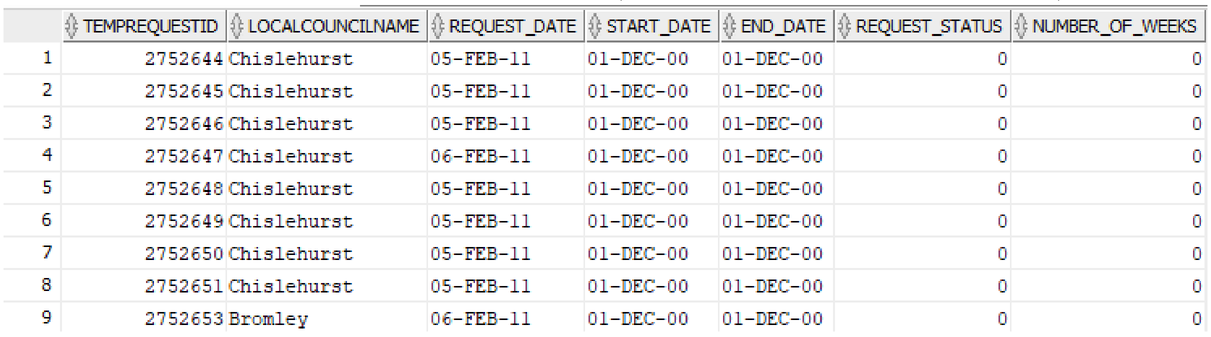
```
DROP MATERIALIZED VIEW VIEW_EMPLOYEE_DETAILS;
Create Materialized view DW_VIEW_EMPLOYEE_DETAILS
Build deferred
Refresh on demand
Enable query rewrite
As select s.LOCAL_COUNCIL_ID, c.LOCALCOUNCILNAME, TITLE, FULL_NAME from
DW_LOCAL_COUNCIL c, DW_EMPLOYEE_DETAILS s where  c.LOCALCOUNCIL_ID =
s.LOCAL_COUNCIL_ID;
EXECUTE DBMS_MVIEW.REFRESH('DW_VIEW_EMPLOYEE_DETAILS', 'C');
```



	LOCAL_COUNCIL_ID	LOCALCOUNCILNAME	TITLE	FULL_NAME
1	2	Orpington	Dr	R. Lester
2	54	Anerley	Mr	Alan Selverengen
3	25	Orpington	Miss	G. Fyshtel
4	39	Upper Norwood	Miss	S. Knight
5	21	West Wickham	Mr	Stuart Salymen

DW_View_Temp_Request

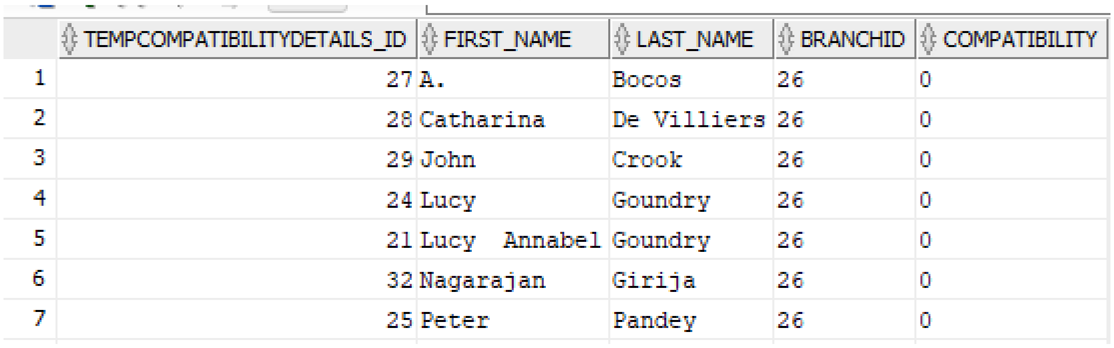
```
DROP MATERIALIZED VIEW DW_VIEW_TEMP_REQUEST;
Create Materialized view DW_VIEW_TEMP_REQUEST
Build deferred
Refresh on demand
Enable query rewrite
As select tr.TEMPREQUESTID, LOCALCOUNCILNAME,
tr.REQUEST_DATE, tr.START_DATE, tr.END_DATE, tr.REQUEST_STATUS,
tr.NUMBER_OF_WEEKS
from DW_TEMP_REQUEST tr, DW_LOCAL_COUNCIL c
where  tr.LOCALCOUNCIL_ID = c.LOCALCOUNCIL_ID;
EXECUTE DBMS_MVIEW.REFRESH('DW_VIEW_TEMP_REQUEST', 'C');
```



	TEMPREQUESTID	LOCALCOUNCILNAME	REQUEST_DATE	START_DATE	END_DATE	REQUEST_STATUS	NUMBER_OF_WEEKS
1	2752644	Chislehurst	05-FEB-11	01-DEC-00	01-DEC-00	0	0
2	2752645	Chislehurst	05-FEB-11	01-DEC-00	01-DEC-00	0	0
3	2752646	Chislehurst	05-FEB-11	01-DEC-00	01-DEC-00	0	0
4	2752647	Chislehurst	06-FEB-11	01-DEC-00	01-DEC-00	0	0
5	2752648	Chislehurst	05-FEB-11	01-DEC-00	01-DEC-00	0	0
6	2752649	Chislehurst	05-FEB-11	01-DEC-00	01-DEC-00	0	0
7	2752650	Chislehurst	05-FEB-11	01-DEC-00	01-DEC-00	0	0
8	2752651	Chislehurst	05-FEB-11	01-DEC-00	01-DEC-00	0	0
9	2752653	Bromley	06-FEB-11	01-DEC-00	01-DEC-00	0	0

DW_View_TempCompabitilityDetails

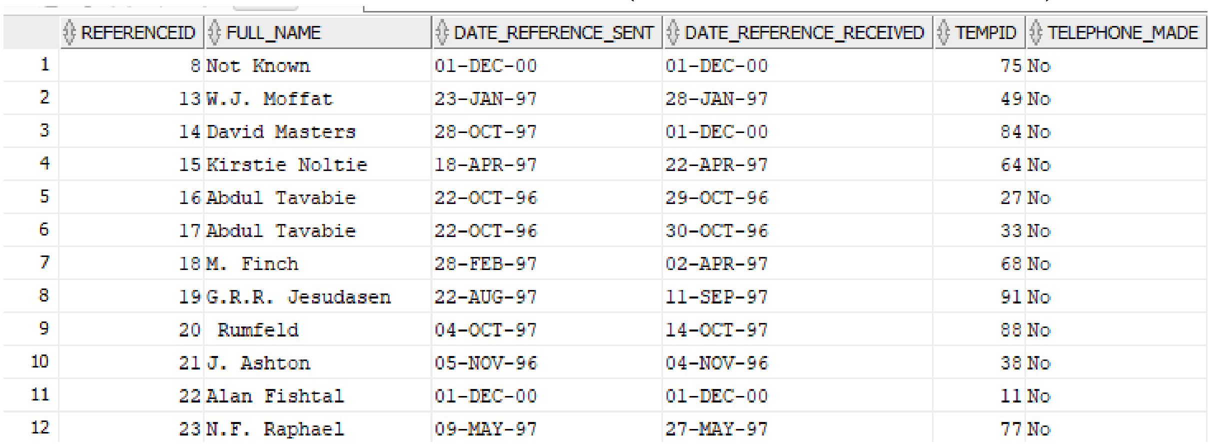
```
DROP MATERIALIZED VIEW DW_VIEW_TEMPCOMPATIBILITYDETAILS;
Create Materialized view DW_VIEW_TEMPCOMPATIBILITYDETAILS
Build deferred
Refresh on demand
Enable query rewrite
As select tc.TEMPCOMPATIBILITYDETAILS_ID, FIRST_NAME, LAST_NAME,
tc.BRANCHID, tc.COMPATIBILITY from DW_TEMPCOMPATIBILITYDETAILS tc, DW_TEMP
t where  tc.TEMPID = t.TEMPID;
EXECUTE DBMS_MVIEW.REFRESH('DW_VIEW_TEMPCOMPATIBILITYDETAILS', 'C');
```



	TEMPCOMPATIBILITYDETAILS_ID	FIRST_NAME	LAST_NAME	BRANCHID	COMPATIBILITY
1	27	A.	Bocos	26	0
2	28	Catharina	De Villiers	26	0
3	29	John	Crook	26	0
4	24	Lucy	Goundry	26	0
5	21	Lucy Annabel	Goundry	26	0
6	32	Nagarajan	Girija	26	0
7	25	Peter	Pandey	26	0

DW_View_References

```
DROP MATERIALIZED VIEW DW_REFERENCES_VIEW;
Create Materialized view DW_VIEWS_REFERENCES
Build deferred
Refresh on demand
Enable query rewrite
As select r.referenceid, FULL_NAME, r.date_reference_sent, r.date_reference_received, r.tempid,
r.telephone_made
from DW_REFERENCES r, DW_REFEREES_DETAILS d
where  r.REFEREEID = d.REFEREEID;
EXECUTE DBMS_MVIEW.REFRESH('DW_VIEWS_REFERENCES', 'C');
```



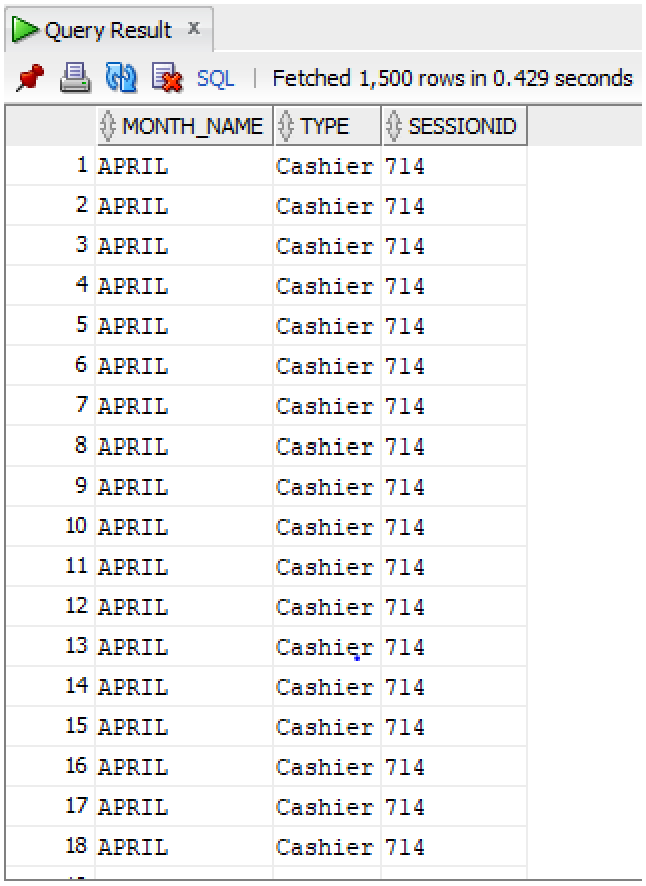
	REFERENCEID	FULL_NAME	DATE_REFERENCE_SENT	DATE_REFERENCE_RECEIVED	TEMPID	TELEPHONE_MADE
1	8	Not Known	01-DEC-00	01-DEC-00	75	No
2	13	W.J. Moffat	23-JAN-97	28-JAN-97	49	No
3	14	David Masters	28-OCT-97	01-DEC-00	84	No
4	15	Kirstie Noltie	18-APR-97	22-APR-97	64	No
5	16	Abdul Tavabie	22-OCT-96	29-OCT-96	27	No
6	17	Abdul Tavabie	22-OCT-96	30-OCT-96	33	No
7	18	M. Finch	28-FEB-97	02-APR-97	68	No
8	19	G.R.R. Jesudasen	22-AUG-97	11-SEP-97	91	No
9	20	Rumfeld	04-OCT-97	14-OCT-97	88	No
10	21	J. Ashton	05-NOV-96	04-NOV-96	38	No
11	22	Alan Fishtal	01-DEC-00	01-DEC-00	11	No
12	23	N.F. Raphael	09-MAY-97	27-MAY-97	77	No

Queries

Government have asked GreenHomeHelp to provide the information to them consistently. Therefore, to do this, queries have been made for each of the following to make it easier. You can see here the requested information, the query and a screenshot of the result of what has been shown of query once executed. Please also note that some of the screenshots cannot provide all the information. This is only the case as many rows show up as a result. Therefore, the example shown of the screenshot shows an example of what has been executed

The number of sessions filled by type of temp cover by month

SELECT t.MONTH_NAME, s.TYPE, s.SESSIONID FROM DW_FACT_SESSION s, DW_TIME t where s.TYPE != 'Not Known'order by t.MONTH_NAME, s.TYPE;

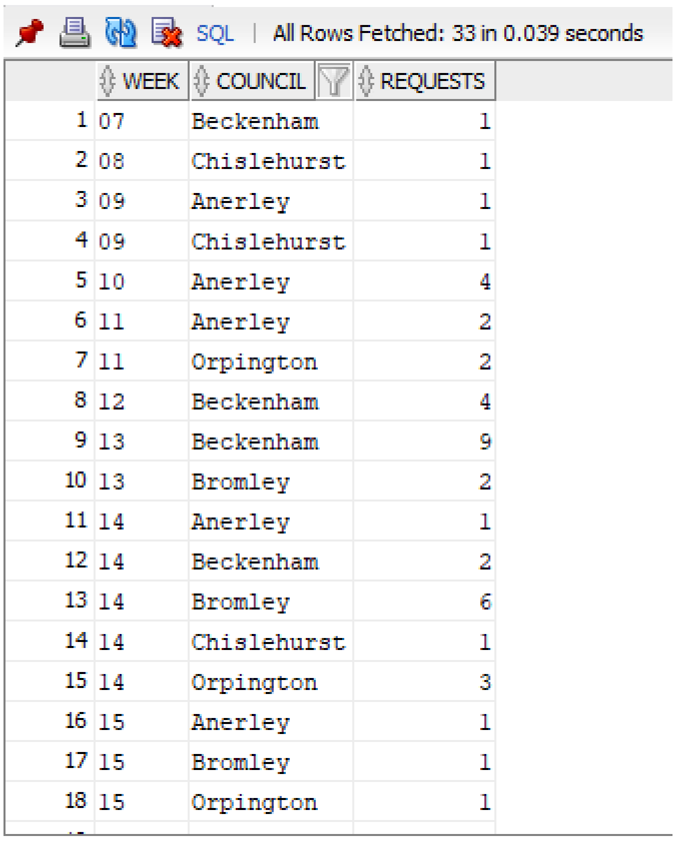


Fetched 1,500 rows in 0.429 seconds

	MONTH_NAME	TYPE	SESSIONID
1	APRIL	Cashier	714
2	APRIL	Cashier	714
3	APRIL	Cashier	714
4	APRIL	Cashier	714
5	APRIL	Cashier	714
6	APRIL	Cashier	714
7	APRIL	Cashier	714
8	APRIL	Cashier	714
9	APRIL	Cashier	714
10	APRIL	Cashier	714
11	APRIL	Cashier	714
12	APRIL	Cashier	714
13	APRIL	Cashier	714
14	APRIL	Cashier	714
15	APRIL	Cashier	714
16	APRIL	Cashier	714
17	APRIL	Cashier	714
18	APRIL	Cashier	714

The number of temp care worker requests made by council for each week

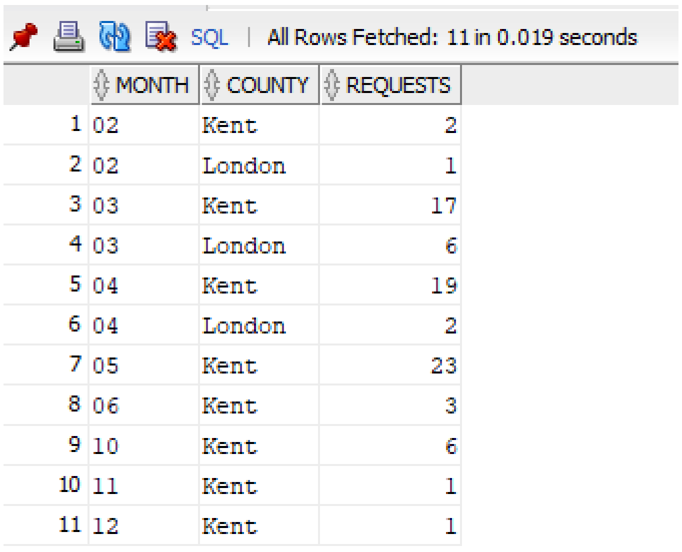
select t.WEEK_OF_YEAR as Week, c.LOCALCOUNCILNAME as Council, count(s.TEMPREQUESTID) As Requests from DW_FACT_SESSION s , DW_TIME t, DW_LOCAL_COUNCIL c, DW_TEMP_REQUEST r where s.TEMPREQUESTID = r.TEMPREQUESTID AND c.LOCALCOUNCIL_ID = r.LOCALCOUNCIL_ID AND s.SESSIONID = t.DATE_ID group by t.WEEK_OF_YEAR, c.LOCALCOUNCILNAME order by t.WEEK_OF_YEAR, c.LOCALCOUNCILNAME;



	WEEK	COUNCIL	REQUESTS
1	07	Beckenham	1
2	08	Chislehurst	1
3	09	Anerley	1
4	09	Chislehurst	1
5	10	Anerley	4
6	11	Anerley	2
7	11	Orpington	2
8	12	Beckenham	4
9	13	Beckenham	9
10	13	Bromley	2
11	14	Anerley	1
12	14	Beckenham	2
13	14	Bromley	6
14	14	Chislehurst	1
15	14	Orpington	3
16	15	Anerley	1
17	15	Bromley	1
18	15	Orpington	1

The number of temp care worker requests filled by county for each month

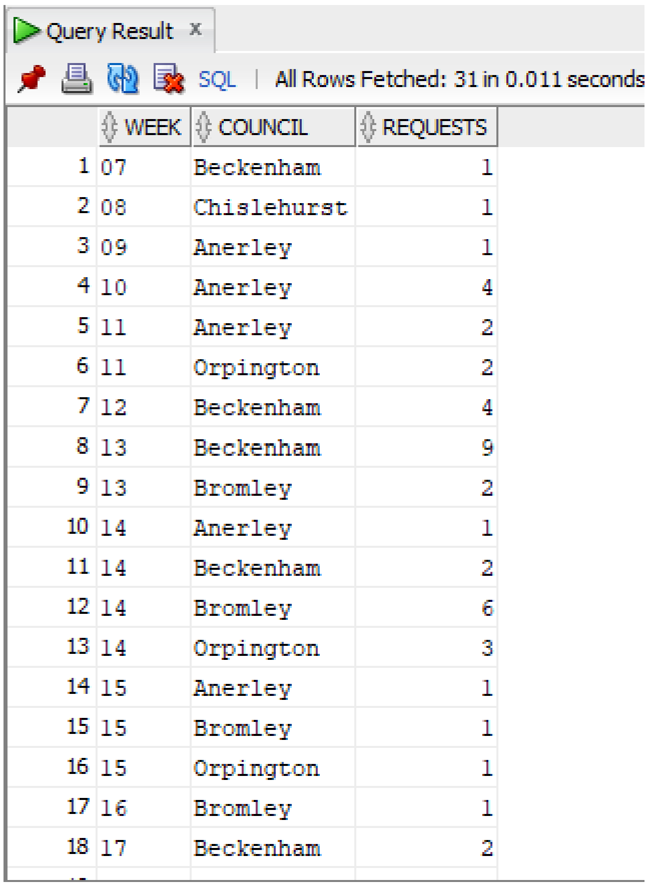
select t.MONTH_NO as month, c.county as County, count(s.TEMPREQUESTID) As Requests from DW_FACT_SESSION s , DW_TIME t, DW_LOCAL_COUNCIL c, DW_TEMP_REQUEST r where s.TEMPREQUESTID = r.TEMPREQUESTID AND c.LOCALCOUNCIL_ID = r.LOCALCOUNCIL_ID AND s.sessionid = t.date_id AND s.type != 'Not Known' group by t.MONTH_NO, c.county order by t.MONTH_NO, c.county;



	MONTH	COUNTY	REQUESTS
1	02	Kent	2
2	02	London	1
3	03	Kent	17
4	03	London	6
5	04	Kent	19
6	04	London	2
7	05	Kent	23
8	06	Kent	3
9	10	Kent	6
10	11	Kent	1
11	12	Kent	1

The number temp care worker requests filled by council each week

select t.week_of_year as Week, c.LOCALCOUNCILNAME as Council, count(s.TEMPREQUESTID) As Requests from DW_FACT_SESSION s , DW_TIME t, DW_LOCAL_COUNCIL c, DW_TEMP_REQUEST r where s.TEMPREQUESTID = r.TEMPREQUESTID AND c.LOCALCOUNCIL_ID = r.LOCALCOUNCIL_ID AND s.sessionid = t.date_id AND s.type != 'Not Known' group by t.week_of_year, c.LOCALCOUNCILNAME order by t.week_of_year, c.LOCALCOUNCILNAME;



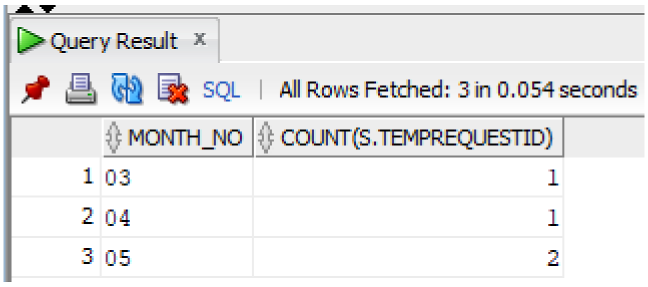
Query Result

SQL | All Rows Fetched: 31 in 0.011 seconds

	WEEK	COUNCIL	REQUESTS
1	07	Beckenham	1
2	08	Chislehurst	1
3	09	Anerley	1
4	10	Anerley	4
5	11	Anerley	2
6	11	Orpington	2
7	12	Beckenham	4
8	13	Beckenham	9
9	13	Bromley	2
10	14	Anerley	1
11	14	Beckenham	2
12	14	Bromley	6
13	14	Orpington	3
14	15	Anerley	1
15	15	Bromley	1
16	15	Orpington	1
17	16	Bromley	1
18	17	Beckenham	2

The number of temp care worker requests which were cancelled each month

select t.month_no, count(s.TEMPREQUESTID) from DW_FACT_SESSION s , dw_time t, DW_TEMP_REQUEST r where s.TEMPREQUESTID = r.TEMPREQUESTID AND s.sessionid = t.date_id AND s.status = 'Temp Cancelled' group by t.month_no order by t.month_no;



Query Result

SQL | All Rows Fetched: 3 in 0.054 seconds

	MONTH_NO	COUNT(S.TEMPREQUESTID)
1	03	1
2	04	1
3	05	2

Bonus

Please view the following bonus task as an attempt and complete of the following. We will put the code of the bonus task and a sample of the result of what is shown. You can see this in each of the following tasks that has been named. As you see, we only completed four bonus tasks.

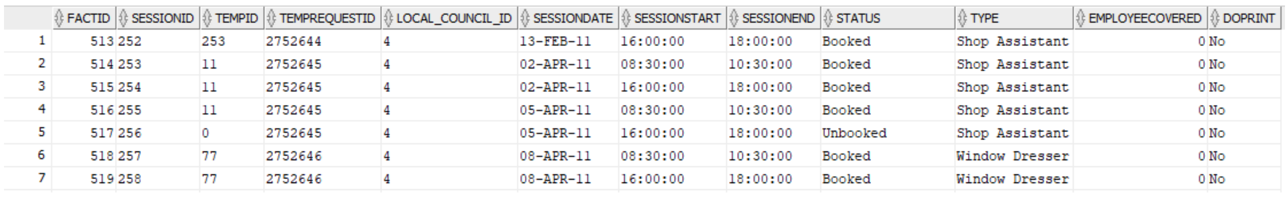
An extended TIME dimension should be created and populated

As you can see the following code creates the table for time. To do this, I added all these dimensions to make sure they are different variants that users can. You can see that al of these are only referred to dates only. However, I feel that this can be improved and extended to hour, minute and second of when a data was entered. This can be discussed further in the report. You can also see some constraints and not nulls for each of the columns to make sure that data that has been transferred is not null. This is important when creating a table for efficiency purposes. You can also see Figure 1 to demonstrate this in the schema.

```
create table DW_TIME (
DATE_ID date NOT NULL,
DAY     number(4) NOT NULL,
DAY_NAME VARCHAR2(9) NOT NULL,
DAY_OF_WEEK VARCHAR2(6),
MONTH_NO  NUMBER(4),
MONTH_NAME VARCHAR2(12),
LAST_DAY_OF_MONTH number(10),
YEAR   number(6),
QUARTER VARCHAR2(6),
WEEK_OF_MONTH  NUMBER(4),
WEEK_OF_YEAR  VARCHAR2(6),
PRIMARY KEY (DATE_ID),
CONSTRAINT SESSION_S_DAY CHECK(DAY<=31),
CONSTRAINT SESSION_S_DAY_NAME CHECK(DAY_NAME IN('MONDAY', 'TUESDAY',
'WEDNESDAY', 'THURSDAY', 'FRIDAY', 'SATURDAY', 'SUNDAY')),
CONSTRAINT SESSION_S_WEEK CHECK(WEEK_OF_YEAR<=56),
CONSTRAINT SESSION_S_MONTH_NO CHECK(MONTH_NO<=12),
CONSTRAINT SESSION_S_MONTH_NAME CHECK(MONTH_NAME IN('JANUARY',
'FEBRUARY', 'MARCH', 'APRIL', 'MAY', 'JUNE', 'JULY' , 'AUGUST', 'SEPTEMBER',
'OCTOBER', 'NOVEMBER', 'DECEMBER')));
```

As you can see with the above code that it has been to create the code. Here also shows transferring values using a cursor. This was declared using dates from session table. You can refer yourself to Figure 35 to show a sample from the results of using the query below. The reason behind the use of this is purely because this was defined as the fact table from the schema. Therefore, it would be useful to have a time dimension for the main table. You may notice that this has been used from the st_session i.e. staging session data. Therefore, this was cleaned beforehand and has been updated to make sure it has the most consistent and up to date data.

```
declare
Cursor c_client is
SELECT SESSIONID, REQUESTID, SESSION_DATE, STATUS FROM ST_SESSION;
begin
for c_rec in c_client loop
insert into DW_TIME values(dw_time_seq1.nextval,
to_char(c_rec.SESSION_DATE,'DD'),
to_char(c_rec.SESSION_DATE,'DAY'),
to_char(c_rec.SESSION_DATE,'D'),
to_char(c_rec.SESSION_DATE,'MM'),
to_char(c_rec.SESSION_DATE,'MONTH'),
to_char(LAST_DAY(c_rec.SESSION_DATE)),
to_char(c_rec.SESSION_DATE,'yyyy'),
to_char(c_rec.SESSION_DATE,'Q'),
to_char(c_rec.SESSION_DATE,'W'),
to_char(c_rec.SESSION_DATE,'WW'));
end loop;
end;
```



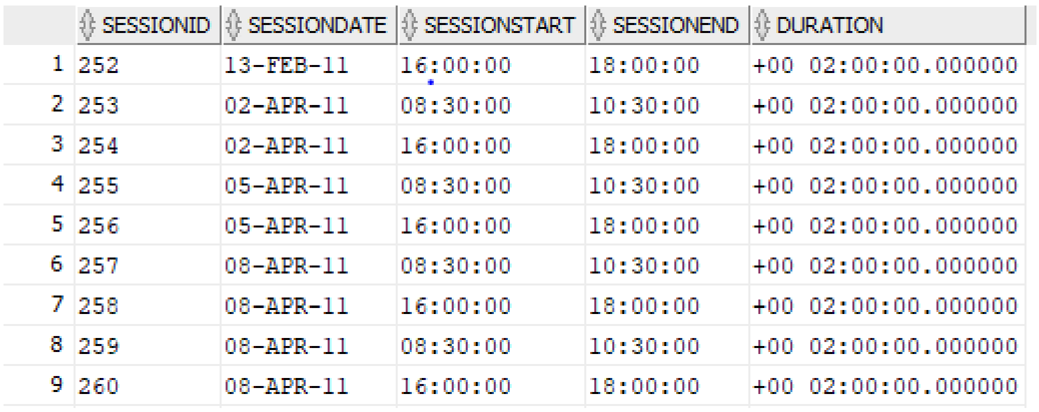
	FACTID	SESSIONID	TEMPID	TEMPREQUESTID	LOCAL_COUNCIL_ID	SESSIONDATE	SESSIONSTART	SESSIONEND	STATUS	TYPE	EMPLOYEECOVERED	DOPRINT
1	513	252	253	2752644	4	13-FEB-11	16:00:00	18:00:00	Booked	Shop Assistant	0	No
2	514	253	11	2752645	4	02-APR-11	08:30:00	10:30:00	Booked	Shop Assistant	0	No
3	515	254	11	2752645	4	02-APR-11	16:00:00	18:00:00	Booked	Shop Assistant	0	No
4	516	255	11	2752645	4	05-APR-11	08:30:00	10:30:00	Booked	Shop Assistant	0	No
5	517	256	0	2752645	4	05-APR-11	16:00:00	18:00:00	Unbooked	Shop Assistant	0	No
6	518	257	77	2752646	4	08-APR-11	08:30:00	10:30:00	Booked	Window Dresser	0	No
7	519	258	77	2752646	4	08-APR-11	16:00:00	18:00:00	Booked	Window Dresser	0	No

Figure 35 shows a sample data from the time dimension

Write the SQL to create a materialized view to show: The Average, length of session cover, by month

Please note that this was created separately due to the failure of grouping. This would be discussed in further detail later on in this report as a discussion of what went wrong. Please refer yourself to Figure 36, this shows the length of each session and the average of each month. As stated previously, you can only see a sample of the data However, this was shown as a timestamp by casting the format shown. Refer yourself to Figure 37, this shows you the sample of average by month. This was not specific as to which one to put per month. However, it makes more sense as an assumption to make duration, and average per month.

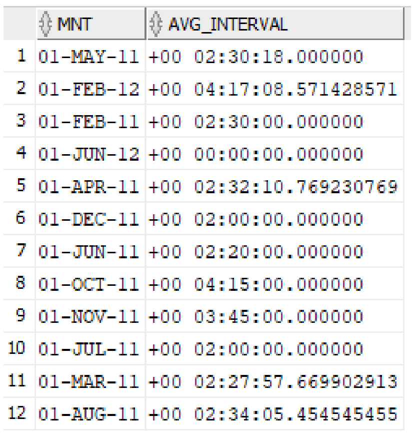
```
DROP MATERIALIZED VIEW DW_VIEW_SESSION_DURATION;
Create Materialized view DW_VIEW_SESSION_DURATION
Build deferred
Refresh on demand
disable query rewrite
As select s.SESSIONID, s.SESSIONDATE, s.SESSIONSTART,s.SESSIONEND,
(TO_TIMESTAMP(s.SESSIONEND, 'hh24:mi:ss') - TO_TIMESTAMP(s.SESSIONSTART,
'hh24:mi:ss')) duration from DW_FACT_SESSION s;
EXECUTE DBMS_MVIEW.REFRESH('DW_VIEW_SESSION_DURATION', 'C');
```



	SESSIONID	SESSIONDATE	SESSIONSTART	SESSIONEND	DURATION
1	252	13-FEB-11	16:00:00	18:00:00	+00 02:00:00.000000
2	253	02-APR-11	08:30:00	10:30:00	+00 02:00:00.000000
3	254	02-APR-11	16:00:00	18:00:00	+00 02:00:00.000000
4	255	05-APR-11	08:30:00	10:30:00	+00 02:00:00.000000
5	256	05-APR-11	16:00:00	18:00:00	+00 02:00:00.000000
6	257	08-APR-11	08:30:00	10:30:00	+00 02:00:00.000000
7	258	08-APR-11	16:00:00	18:00:00	+00 02:00:00.000000
8	259	08-APR-11	08:30:00	10:30:00	+00 02:00:00.000000
9	260	08-APR-11	16:00:00	18:00:00	+00 02:00:00.000000

Figure 36 shows sample of duration between sessions

```
DROP MATERIALIZED VIEW DW_VIEW_SESSION_AVERAGE_MONTH;
Create Materialized view DW_VIEW_SESSION_AVERAGE_MONTH
Build deferred
Refresh on demand
disable query rewrite
As select
trunc(to_date(s.SESSIONDATE, 'dd-mon-yy'), 'mm') mnt,
numtodsinterval( avg(to_date(s.SESSIONEND, 'hh24:mi:ss') - to_date(s.SESSIONSTART,
'hh24:mi:ss')), 'day') avg_interval
from DW_FACT_SESSION s  group by trunc(to_date(s.SESSIONDATE, 'dd-mon-yy'), 'mm');
EXECUTE DBMS_MVIEW.REFRESH('DW_VIEW_SESSION_AVERAGE_MONTH', 'C');
```



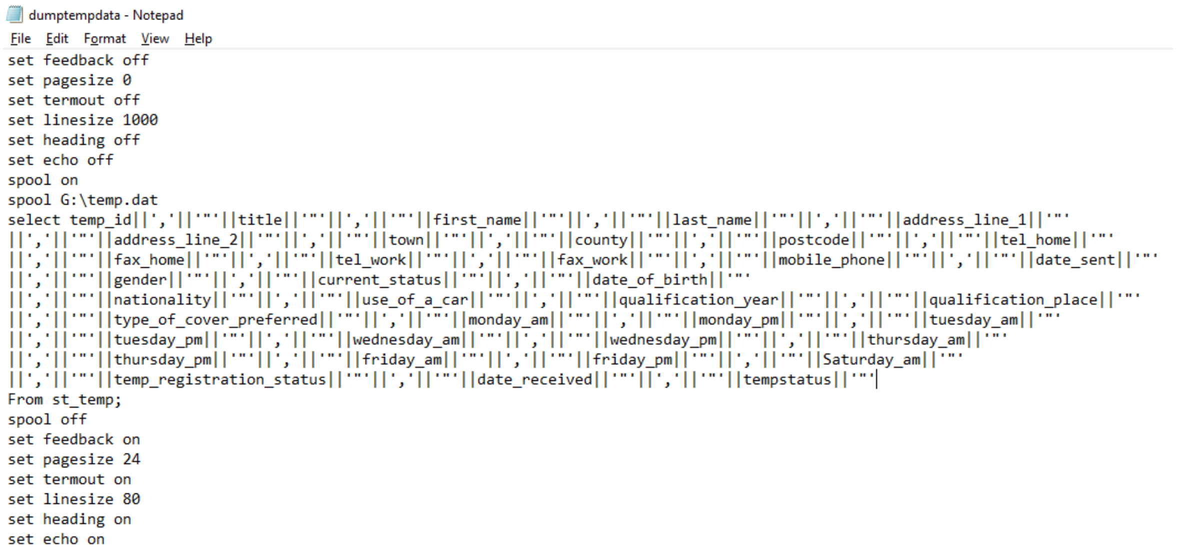
	MNT	AVG_INTERVAL
1	01-MAY-11	+00 02:30:18.000000
2	01-FEB-12	+00 04:17:08.571428571
3	01-FEB-11	+00 02:30:00.000000
4	01-JUN-12	+00 00:00:00.000000
5	01-APR-11	+00 02:32:10.769230769
6	01-DEC-11	+00 02:00:00.000000
7	01-JUN-11	+00 02:20:00.000000
8	01-OCT-11	+00 04:15:00.000000
9	01-NOV-11	+00 03:45:00.000000
10	01-JUL-11	+00 02:00:00.000000
11	01-MAR-11	+00 02:27:57.669902913
12	01-AUG-11	+00 02:34:05.454545455

Figure 37 shows sample of average each month

Write a SQL script to dump the data from the TEMP table into a flat file and use SQL*Loader to populate your data warehouse TEMP table.

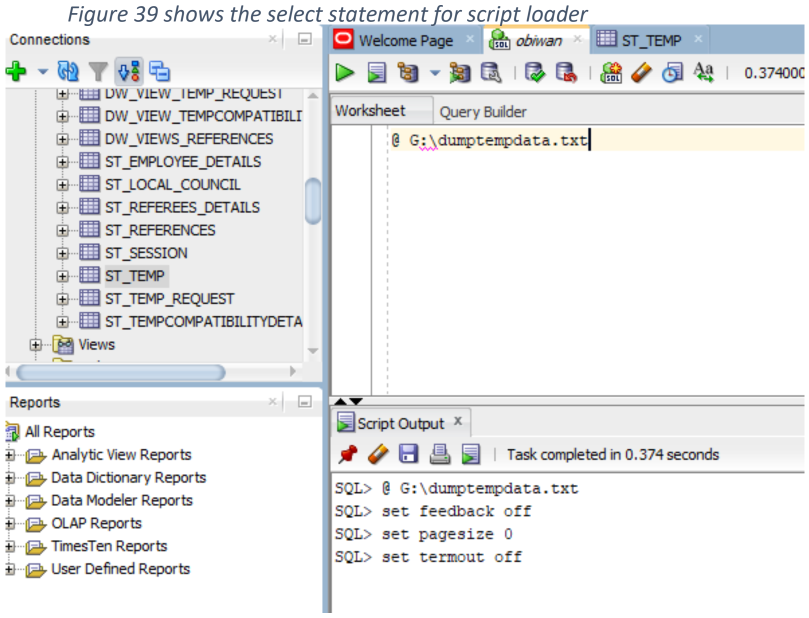
The aim of this bonus task is to find an alternative way of transferring and populating data from the original database into the new data warehouse. As you see, in this report we discuss using cursor to populate our fact and dimensions but this task was based on dumping the data from the original database into a flat file and then using SQL * Loader to populated our data into the data warehouse TEMP table. Below is a step by step guide that shows the steps taken to dump our temp table and transfer the data.

The first step involved us having to dump the data we required into a flat file which is our data file with the extension(.dat). In order to do this, we created a dump file called dumptempdata.txt which included the temp.dat(data file) with all the records of the temp table from the original database. Referring to Figure 39, shows the SQL Script save as dumptempdata that will dump our original data into a flat



```
set feedback off
set pagesize 0
set termout off
set linesize 1000
set heading off
set echo off
spool on
spool G:\temp.dat
select temp_id||','||'"'||title||'"'||','||'"'||first_name||'"'||','||'"'||last_name||'"'||','||'"'||address_line_1||'"'
||','||'"'||address_line_2||'"'||','||'"'||town||'"'||','||'"'||county||'"'||','||'"'||postcode||'"'||','||'"'||tel_home||'"'
||','||'"'||fax_home||'"'||','||'"'||tel_work||'"'||','||'"'||fax_work||'"'||','||'"'||mobile_phone||'"'||','||'"'||date_sent||'"'
||','||'"'||gender||'"'||','||'"'||current_status||'"'||','||'"'||date_of_birth||'"'
||','||'"'||nationality||'"'||','||'"'||use_of_a_car||'"'||','||'"'||qualification_year||'"'||','||'"'||qualification_place||'"'
||','||'"'||type_of_cover_preferred||'"'||','||'"'||monday_am||'"'||','||'"'||monday_pm||'"'||','||'"'||tuesday_am||'"'
||','||'"'||tuesday_pm||'"'||','||'"'||wednesday_am||'"'||','||'"'||wednesday_pm||'"'||','||'"'||thursday_am||'"'
||','||'"'||thursday_pm||'"'||','||'"'||friday_am||'"'||','||'"'||friday_pm||'"'||','||'"'||Saturday_am||'"'
||','||'"'||temp_registration_status||'"'||','||'"'||date_received||'"'||','||'"'||tempstatus||'"'
From st_temp;
spool off
set feedback on
set pagesize 24
set termout on
set linesize 80
set heading on
set echo on
```

Figure 39 shows the select statement for script loader



```
@ G:\dumptempdata.txt
```

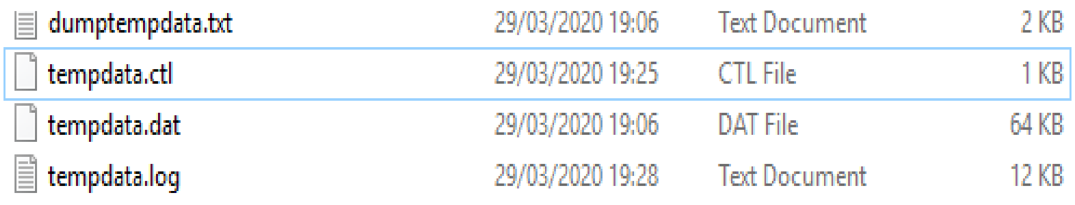
Task completed in 0.374 seconds

```
SQL> @ G:\dumptempdata.txt
SQL> set feedback off
SQL> set pagesize 0
SQL> set termout off
```

file(temp.data).

Once that is complete, we then ran the command @G:\dumptempdata.txt file on our sql oracle to enable for the data file to be saved as the flat file required. Figure 38 below shows the execution of the command and Figure 40 shows the successful download of the files into the G drive.

Figure 38 shows the execution of command

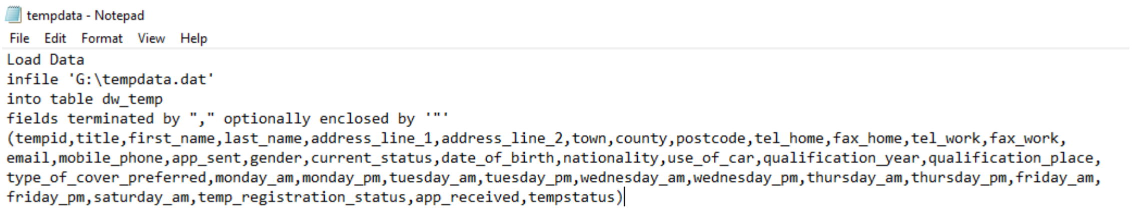


dumptempdata.txt	29/03/2020 19:06	Text Document	2 KB
tempdata.ctl	29/03/2020 19:25	CTL File	1 KB
tempdata.dat	29/03/2020 19:06	DAT File	64 KB
tempdata.log	29/03/2020 19:28	Text Document	12 KB

Figure 40 shows the execution of command that is created

Now that we have the files that allowed us to dump the data into a flat file. It is time to make good use of the data by populating it into the new table that we have created. The final step involves us having to create a control file that will regulate the process of populating the data into the new table. Figure …… below shows the control file (.ctl), that we used to make that happen. The file specifies that the temp.dat data file will be loaded and put into and populated into our dw_temp table.

The final stage will then be to execute the code in order to see that our rows of data have successfully been transferred into our new dw_temp table. The command prompt will notify us with the successful uploads of data and all the errors and unsuccessful upload will then be stated in the BAD file(.bad) and in the log. Figure 41 shows the successfully upload of 100 rows into our new file hence a successful upload of the data from the original table using the flat files to transfer our data.

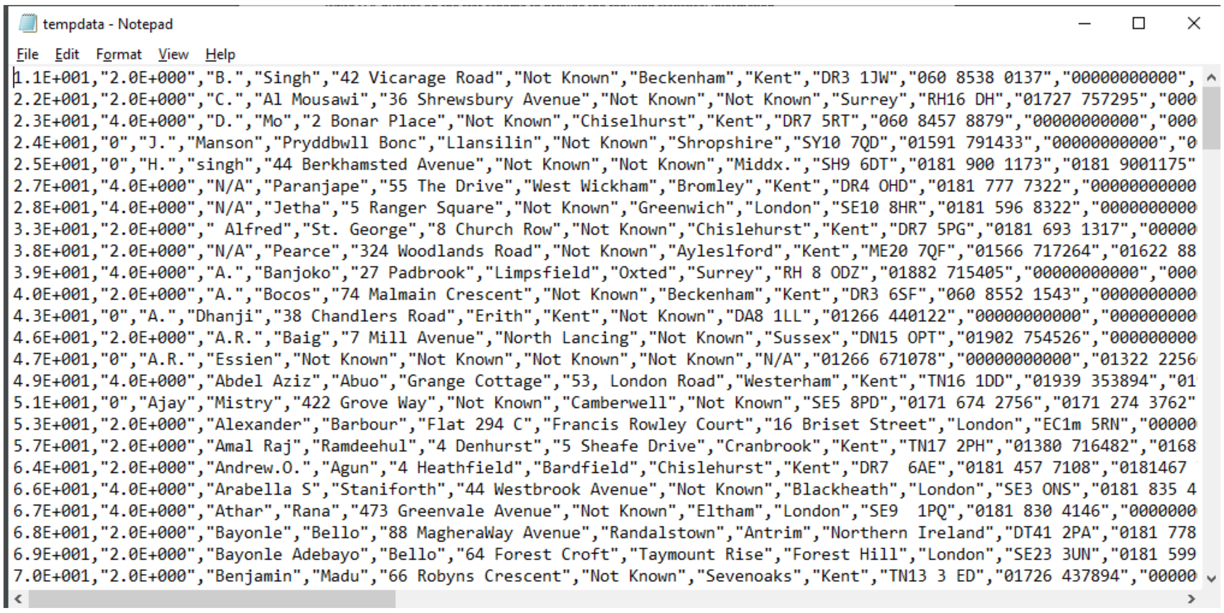


```
Load Data
infile 'G:\tempdata.dat'
into table dw_temp
fields terminated by "," optionally enclosed by '"'
(tempid,title,first_name,last_name,address_line_1,address_line_2,town,county,postcode,tel_home,fax_home,tel_work,fax_work,
email,mobile_phone,app_sent,gender,current_status,date_of_birth,nationality,use_of_car,qualification_year,qualification_place,
type_of_cover_preferred,monday_am,monday_pm,tuesday_am,tuesday_pm,wednesday_am,wednesday_pm,thursday_am,thursday_pm,friday_am,
friday_pm,saturday_am,temp_registration_status,app_received,tempstatus)
```

Figure 41 shows the code to transfer the data into the table

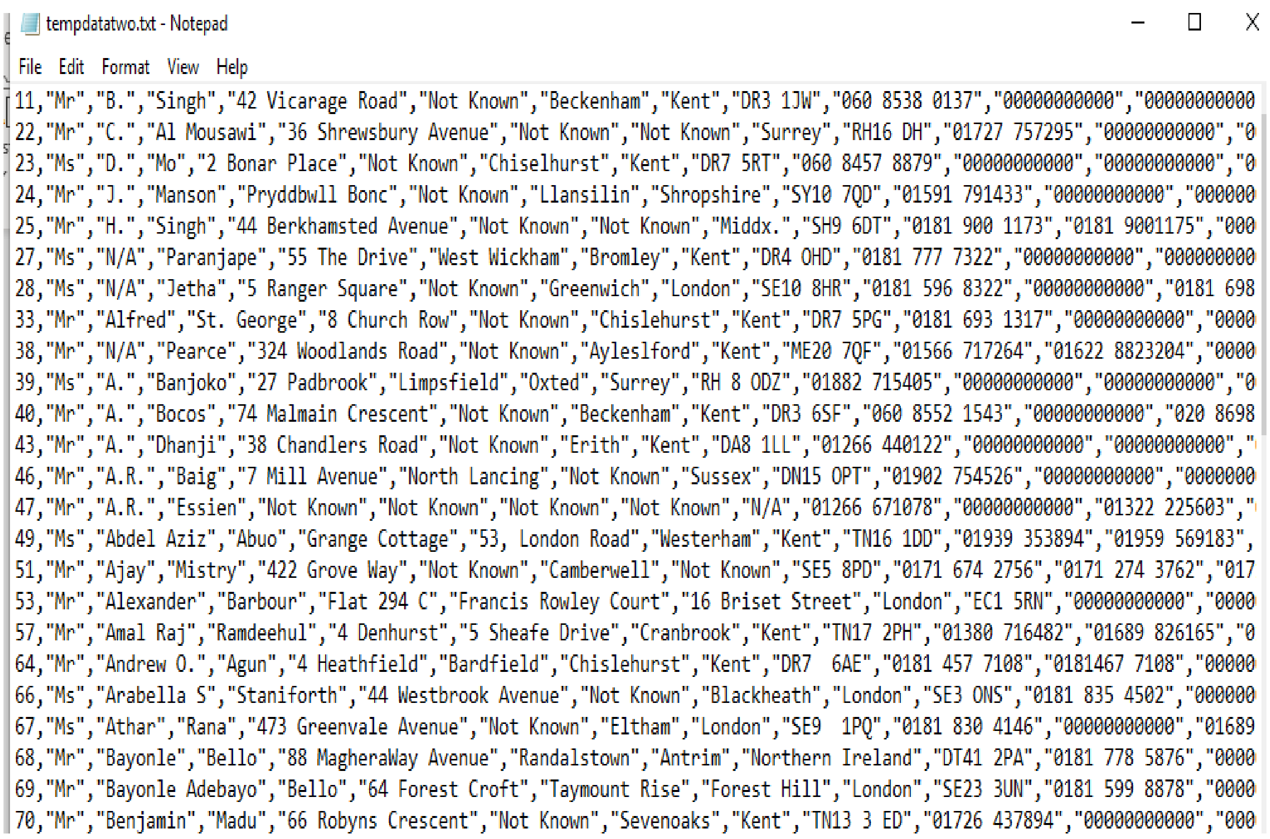
Data File

The data files are the most important as they contain the data for which is transferred from one table to another. Figure 42 below shows the data file for the temp table before it was cleansed and populated where as Figure 43 shows the new data file after the table has been cleansed and ready to be populated to the new dw_temp table. Figure 44 shows that the transferred using SQL Script Loader has been successful.



```
1.1E+001,"2.0E+000","B.","Singh","42 Vicarage Road","Not Known","Beckenham","Kent","DR3 1JW","060 8538 0137","00000000000",
2.2E+001,"2.0E+000","C.","Al Mousawi","36 Shrewsbury Avenue","Not Known","Not Known","Surrey","RH16 DH","01727 757295","000
2.3E+001,"4.0E+000","D.","Mo","2 Bonar Place","Not Known","Chiselhurst","Kent","DR7 5RT","060 8457 8879","00000000000","000
2.4E+001,"0","J.","Manson","Pryddbwll Bonc","Llansilin","Not Known","Shropshire","SY10 7QD","01591 791433","00000000000","0
2.5E+001,"0","H.","singh","44 Berkhamsted Avenue","Not Known","Not Known","Middx.","SH9 6DT","0181 900 1173","0181 9001175"
2.7E+001,"4.0E+000","N/A","Paranjape","55 The Drive","West Wickham","Bromley","Kent","DR4 OHD","0181 777 7322","0000000000
2.8E+001,"4.0E+000","N/A","Jetha","5 Ranger Square","Not Known","Greenwich","London","SE10 8HR","0181 596 8322","000000000
3.3E+001,"2.0E+000"," Alfred","St. George","8 Church Row","Not Known","Chislehurst","Kent","DR7 5PG","0181 693 1317","00000
3.8E+001,"2.0E+000","N/A","Pearce","324 Woodlands Road","Not Known","Ayleslford","Kent","ME20 7QF","01566 717264","01622 88
3.9E+001,"4.0E+000","A.","Banjoko","27 Padbrook","Limpsfield","Oxted","Surrey","RH 8 ODZ","01882 715405","00000000000","000
4.0E+001,"2.0E+000","A.","Bocos","74 Malmain Crescent","Not Known","Beckenham","Kent","DR3 6SF","060 8552 1543","0000000000
4.3E+001,"0","A.","Dhanji","38 Chandlers Road","Erith","Kent","Not Known","DA8 1LL","01266 440122","00000000000","000000000
4.6E+001,"2.0E+000","A.R.","Baig","7 Mill Avenue","North Lancing","Not Known","Sussex","DN15 OPT","01902 754526","000000000
4.7E+001,"0","A.R.","Essien","Not Known","Not Known","Not Known","Not Known","N/A","01266 671078","00000000000","01322 2256
4.9E+001,"4.0E+000","Abdel Aziz","Abuo","Grange Cottage","53, London Road","Westerham","Kent","TN16 1DD","01939 353894","01
5.1E+001,"0","Ajay","Mistry","422 Grove Way","Not Known","Camberwell","Not Known","SE5 8PD","0171 674 2756","0171 274 3762"
5.3E+001,"2.0E+000","Alexander","Barbour","Flat 294 C","Francis Rowley Court","16 Briset Street","London","EC1m 5RN","00000
5.7E+001,"2.0E+000","Amal Raj","Ramdeehul","4 Denhurst","5 Sheafe Drive","Cranbrook","Kent","TN17 2PH","01380 716482","0168
6.4E+001,"2.0E+000","Andrew.O.","Agun","4 Heathfield","Bardfield","Chislehurst","Kent","DR7  6AE","0181 457 7108","0181467
6.6E+001,"4.0E+000","Arabella S","Staniforth","44 Westbrook Avenue","Not Known","Blackheath","London","SE3 ONS","0181 835 4
6.7E+001,"4.0E+000","Athar","Rana","473 Greenvale Avenue","Not Known","Eltham","London","SE9  1PQ","0181 830 4146","0000000
6.8E+001,"2.0E+000","Bayonle","Bello","88 MagheraWay Avenue","Randalstown","Antrim","Northern Ireland","DT41 2PA","0181 778
6.9E+001,"2.0E+000","Bayonle Adebayo","Bello","64 Forest Croft","Taymount Rise","Forest Hill","London","SE23 3UN","0181 599
7.0E+001,"2.0E+000","Benjamin","Madu","66 Robyns Crescent","Not Known","Sevenoaks","Kent","TN13 3 ED","01726 437894","00000
```

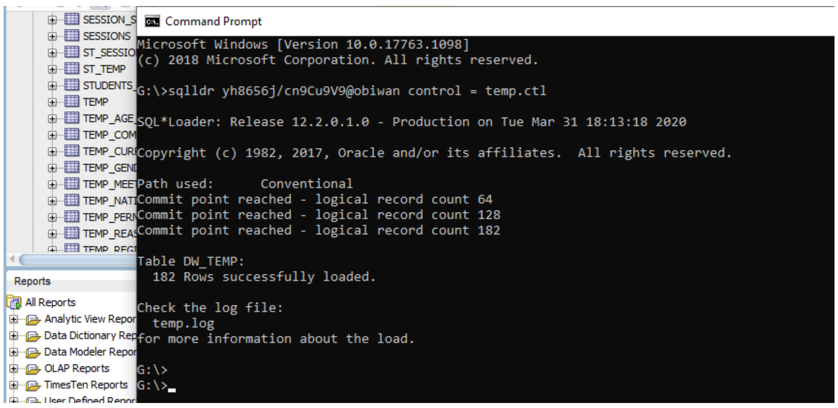
Figure 42 shows the data before cleansed



```
11,"Mr","B.","Singh","42 Vicarage Road","Not Known","Beckenham","Kent","DR3 1JW","060 8538 0137","00000000000","00000000000
22,"Mr","C.","Al Mousawi","36 Shrewsbury Avenue","Not Known","Not Known","Surrey","RH16 DH","01727 757295","00000000000","0
23,"Ms","D.","Mo","2 Bonar Place","Not Known","Chiselhurst","Kent","DR7 5RT","060 8457 8879","00000000000","00000000000","0
24,"Mr","J.","Manson","Pryddbwll Bonc","Not Known","Llansilin","Shropshire","SY10 7QD","01591 791433","00000000000","000000
25,"Mr","H.","Singh","44 Berkhamsted Avenue","Not Known","Not Known","Middx.","SH9 6DT","0181 900 1173","0181 9001175","000
27,"Ms","N/A","Paranjape","55 The Drive","West Wickham","Bromley","Kent","DR4 OHD","0181 777 7322","00000000000","000000000
28,"Ms","N/A","Jetha","5 Ranger Square","Not Known","Greenwich","London","SE10 8HR","0181 596 8322","00000000000","0181 698
33,"Mr","Alfred","St. George","8 Church Row","Not Known","Chislehurst","Kent","DR7 5PG","0181 693 1317","00000000000","0000
38,"Mr","N/A","Pearce","324 Woodlands Road","Not Known","Aylesford","Kent","ME20 7QF","01566 717264","01622 8823204","0000
39,"Ms","A.","Banjoko","27 Padbrook","Limpsfield","Oxted","Surrey","RH 8 ODZ","01882 715405","00000000000","00000000000","0
40,"Mr","A.","Bocos","74 Malmain Crescent","Not Known","Beckenham","Kent","DR3 6SF","060 8552 1543","00000000000","020 8698
43,"Mr","A.","Dhanji","38 Chandlers Road","Not Known","Erith","Kent","DA8 1LL","01266 440122","00000000000","00000000000","
46,"Mr","A.R.","Baig","7 Mill Avenue","North Lancing","Not Known","Sussex","DN15 OPT","01902 754526","00000000000","0000000
47,"Mr","A.R.","Essien","Not Known","Not Known","Not Known","Not Known","N/A","01266 671078","00000000000","01322 225603","
49,"Ms","Abdel Aziz","Abuo","Grange Cottage","53, London Road","Westerham","Kent","TN16 1DD","01939 353894","01959 569183",
51,"Mr","Ajay","Mistry","422 Grove Way","Not Known","Camberwell","Not Known","SE5 8PD","0171 674 2756","0171 274 3762","017
53,"Mr","Alexander","Barbour","Flat 294 C","Francis Rowley Court","16 Briset Street","London","EC1 5RN","00000000000","0000
57,"Mr","Amal Raj","Ramdeehul","4 Denhurst","5 Sheafe Drive","Cranbrook","Kent","TN17 2PH","01380 716482","01689 826165","0
64,"Mr","Andrew O.","Agun","4 Heathfield","Bardfield","Chislehurst","Kent","DR7  6AE","0181 457 7108","0181467 7108","00000
66,"Ms","Arabella S","Staniforth","44 Westbrook Avenue","Not Known","Blackheath","London","SE3 ONS","0181 835 4502","000000
67,"Ms","Athar","Rana","473 Greenvale Avenue","Not Known","Eltham","London","SE9  1PQ","0181 830 4146","00000000000","01689
68,"Mr","Bayonle","Bello","88 MagheraWay Avenue","Randalstown","Antrim","Northern Ireland","DT41 2PA","0181 778 5876","0000
69,"Mr","Bayonle Adebayo","Bello","64 Forest Croft","Taymount Rise","Forest Hill","London","SE23 3UN","0181 599 8878","0000
70,"Mr","Benjamin","Madu","66 Robyns Crescent","Not Known","Sevenoaks","Kent","TN13 3 ED","01726 437894","00000000000","000
```

Figure 43 shows the data that has been cleansed after

Running SQL on Command Prompt



```
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

G:\>sqlldr yh8656j/cn9Cu9V9@obiwan control = temp.ctl

SQL*Loader: Release 12.2.0.1.0 - Production on Tue Mar 31 18:13:18 2020

Copyright (c) 1982, 2017, Oracle and/or its affiliates.  All rights reserved.

Path used:      Conventional
Commit point reached - logical record count 64
Commit point reached - logical record count 128
Commit point reached - logical record count 182

Table DW_TEMP:
  182 Rows successfully loaded.

Check the log file:
  temp.log
for more information about the load.

G:\>
G:\>_
```

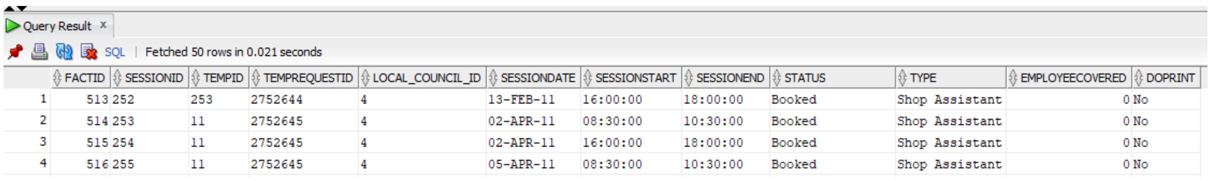
Figure 44 shows that this has been successfully transferred.

Write the create table statement that will Partition the Fact table by year.

You can see that this statement below has been added to the fact table. Please refer yourself to Figure 5 and 6 to show the result of what has been obtained. This was executed by using a simple SELECT statement to show what has been stored within these partitions. As you can see, this was completed by range for no less data than each year using the correct format.

```
partition  by RANGE(SESSIONDATE)
            (
partition SESSIONS_2011 values less than (To_Date('31-12-2011', 'DD-MM-YYYY')),
partition SESSIONS_2012 values  less than(To_Date('31-12-2012', 'DD-MM-YYYY')));
```

```
SELECT * FROM DW_FACT_SESSION PARTITION(SESSIONS_2011);
```



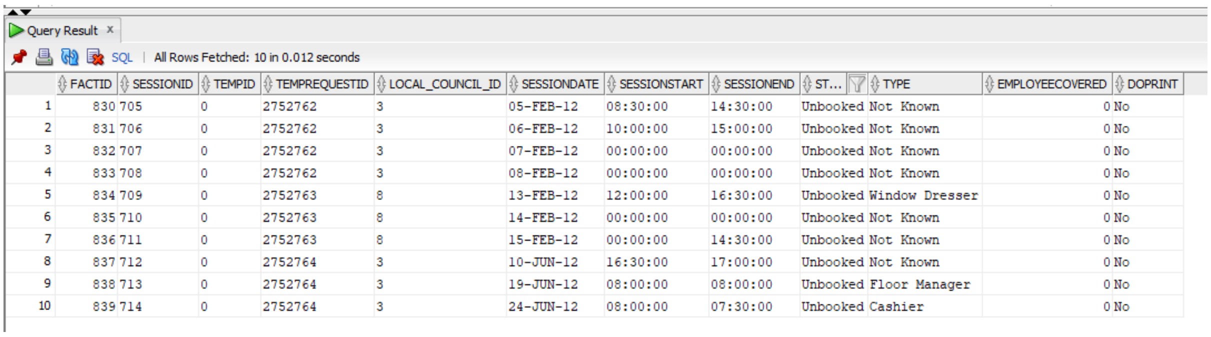
Query Result

Fetched 50 rows in 0.021 seconds

	FACTID	SESSIONID	TEMPID	TEMPREQUESTID	LOCAL_COUNCIL_ID	SESSIONDATE	SESSIONSTART	SESSIONEND	STATUS	TYPE	EMPLOYEECOVERED	DOPRINT
1	513	252	253	2752644	4	13-FEB-11	16:00:00	18:00:00	Booked	Shop Assistant	0	No
2	514	253	11	2752645	4	02-APR-11	08:30:00	10:30:00	Booked	Shop Assistant	0	No
3	515	254	11	2752645	4	02-APR-11	16:00:00	18:00:00	Booked	Shop Assistant	0	No
4	516	255	11	2752645	4	05-APR-11	08:30:00	10:30:00	Booked	Shop Assistant	0	No

Figure 45 shows sample by year 11 only in the fact table

```
SELECT * FROM DW_FACT_SESSION PARTITION(SESSIONS_2012);
```



Query Result

All Rows Fetched: 10 in 0.012 seconds

	FACTID	SESSIONID	TEMPID	TEMPREQUESTID	LOCAL_COUNCIL_ID	SESSIONDATE	SESSIONSTART	SESSIONEND	ST...	TYPE	EMPLOYEECOVERED	DOPRINT
1	830	705	0	2752762	3	05-FEB-12	08:30:00	14:30:00	Unbooked	Not Known	0	No
2	831	706	0	2752762	3	06-FEB-12	10:00:00	15:00:00	Unbooked	Not Known	0	No
3	832	707	0	2752762	3	07-FEB-12	00:00:00	00:00:00	Unbooked	Not Known	0	No
4	833	708	0	2752762	3	08-FEB-12	00:00:00	00:00:00	Unbooked	Not Known	0	No
5	834	709	0	2752763	8	13-FEB-12	12:00:00	16:30:00	Unbooked	Window Dresser	0	No
6	835	710	0	2752763	8	14-FEB-12	00:00:00	00:00:00	Unbooked	Not Known	0	No
7	836	711	0	2752763	8	15-FEB-12	00:00:00	14:30:00	Unbooked	Not Known	0	No
8	837	712	0	2752764	3	10-JUN-12	16:30:00	17:00:00	Unbooked	Not Known	0	No
9	838	713	0	2752764	3	19-JUN-12	08:00:00	08:00:00	Unbooked	Floor Manager	0	No
10	839	714	0	2752764	3	24-JUN-12	08:00:00	07:30:00	Unbooked	Cashier	0	No

Figure 46 shows sample by year 12 in the fact table

Problems Faced

A Issues

This section looks at some of the issues that have been encountered in the implementation of the data warehouse. An issue that had occurred in the early stages of the process during ETL, was the exporting data from Microsoft access. We were using the lab computers by following the simply guide given to us, it was still not possible as the service we require was not available. We required the 'oracle in 64' obiwan data source and the options were everything but that. At first, we tried using excel to export that data and then import it into oracle ODBC but that ended up giving us an incorrect reading of that data. A solution we then had was to try it through VMware virtual desktop which eventually ended up working. Shortly after the situation was rectified and we were able to export the original data source, should we need to export any data from that point onwards. This issue is quite like the session table as the session start, and end times were both giving default dates of '30-dec-99' rather than the actual times of the session. This would affect the data warehouse as the bonus section requires that we find an average time of the sessions to then be put into a materialized view, hence making it viewable by the government who want to see information on sessions. The result of this aided our search to find a solution. We change the data type from date to time and then exported the table so it would show the time in the '24HH:MI:SS' format such as '16:00:00'. Once this worked, we were able to continue to view and use this time segment and further expand its relevance in other areas. With the issues in ETL rectified, we were then able to resume cleansing the data and carrying out the other tasks.

B Issues

Within this section, I will have to look at the issues that I faced during the cleansing and implementation of dimensions and fact table. At the start of staging this, both of us had a look at the data types that were incorrect. This was noted to make sure the accuracy of the data is put within the correct data type. For example, some columns had doubles within them when they should have had integers. However, some issues that came about was trying to switch from the use of Integers and Varchars. This was a consistently switching to make sure that this was correct. This was due to when the data was transferred to the dimension. This was causing consistent errors. Therefore, I had to recreate the table, and transfer the data due to the inconsistent of data types that were misleading at first. Another issue that may have been misleading was the inconsistent data that got transferred from Access to Oracle. Many of the data that got transferred was inconsistent due to reasons of the transfer. Therefore, when of the data that had been cleansed of each table, this needed to be referenced back each time for every table. An example that has been mentioned above was within the session table. One of them columns were converted to a date, when the data was a time within the original database within Access. Therefore, this was converted back to do this. Another issue that had been consistent throughout when trying to create the table was the fact table. This was completed differently because I tried to use the cursor for this, it kept giving me different errors. Therefore, after doing research, I realised that I can merge tables whilst inserting this. Therefore, instead of inserting this and failing, I made sure that the select statement was correct. Consequently, I made sure that the select statement that had been issued was correct and made sure that the rows match. On the other hand, I realised later on within this that I had too many links within the fact table. This issue was that I had used foreign key with employee details when the council table has already used the employee details with this. I made reference back to the schema and got rid of the employee detail that was already duplicate to make sure that each of the dimensions match. Once this was complete, I had all rows connected.

Bonus Issues

An issue that occurred in the partitioning element of the bonus section is that when the temp table was used to create a partition there were many errors that did not allow for that table to be partitioned. We figured that it could be the date format that is an issue, but this was the early stages of implementing the partition. The solution was then to first identify the fact table, which in our case was the session table and then try to partition that table by year. Having a quick scroll through the fact session table, we then realize that it would be a lot simpler to partition that by year as it only had two years in there anyways. The spec required us to partition the fact table by year and that is exactly what we had done. An issue that occurred was also in the querying of our data. We initially done the queries during the cleansing stage to further speed up the process but was giving us some inaccurate reading as we checked the table to verify the information. The queries were correct so we decided to then run the queries again that we had saved to see the difference once we have completed cleansing and that was the solution which later allowed us to provide the queries for the government to see information such as the number of sessions filled by council per month.

Another issue that came across within bonuses as that to create a materialized view for the duration and average. We were not able to join both results together. This was a consistent error that we kept getting whilst trying to do this bonus. Therefore, as you may have been mentioned above, we have split both materialized view in order to do this. This was an issue with grouping both select statements together. For some unknown reason, it did not let me do this. Therefore, this could pout put in as an improvement for future references.

Evaluation (A)

In this documentation, it is evident that we have completed all of the requirements to the best of our abilities. This was no easy task but required much time especially for tasks such as cleansing and querying the correct information for the government to see the relevant information. The ETL process was also a big part as the other stages would not be possible without the correct exporting and importation of the data source into the data warehouse. It was fundamental that we understood the requirements and proceed with the creation of the data warehouse and all other additional aspects such as using a flat file for the temp table to try and gain an understand on other ways of transferring data from one table to another using a data and control file through SQL loader and command prompt. The tutorials were very useful as this allowed for us to put the understanding, we had in these tutorials to then apply that to our coursework to further boost the quality of our data warehouse. An existing aspect was creating the extended timetable, as it gathered information from the session date and split that single date into many columns which described things such as the day, the month, the day of week etc. By using that, we were then able to query by a given time period such as sessions per week or requests per month etc. The tasks were separate into A and B between me and my colleague and the rest we shared equally but in the end we both worked hard to gain the best results out of this implemented data warehouse and hope that it pays off.

Overall, I very much enjoyed the process of this whole coursework as it was interesting and is a new challenge as I only have experience in SQL when it comes to programming. I have used databases many times by the level of sophistication in data warehouse made me see a new light and gain a new experience into how advanced databases work and how organized a data warehouse is to make it much easier for organizations. I hope to use this new skill set in the foreseeable future.

Evaluation (B)

I would also like to mention that the only task that had not been attempted were the last bonus where it was to create an automated cleansing tool. This was unable to do. However, as always, it would be noted as a future reference to make improvements on this task. I would also like to mention that the number of tables that have been condensed to 9 is incredible. This was also necessary as some of these tables were unnecessary and not needed. As mentioned, tables and columns were dropped for a reason as many columns were empty. This was inconsistent and not needed within the table. However, some columns were updated with default data of which tables that were needed.

This was a fundamental challenge to make sure how tasks were split with my colleague. At the start, I saw this as a challenge. However, I made sure that the specification and tasks that were given, both of us were happy to do this. Therefore, splitting tasks were easy and both students, including myself, were happy to do the tasks that have been completed. We also both went the extra mile to make sure that both data warehousing was consistent. We also made sure that the bonus were attempted too. Even though we have missed one, as mentioned, it was essential to gain the experience and make the improvement on this matter too. We both realise that we have chosen a complex schema in Snowflake; however, I felt that the tables and views that were included have made it easier to join and view.

Overall, I would like to say that it has been a pleasure to learn a different side of data processing. I felt that within this coursework, working together with my colleague, Yunus; has been enjoyable to work and solve each task. This process was a challenge to see and have different experience of SQL from a different side. I have used different levels of creating databases. However, I can see that this was an interesting experience to see a different side of SQL and how to manage data from Oracle-side. I hope to use data warehousing within the near future.

Tasks Allocation

Tasks	Student Completed	Student ID
Member A – ETL Export the data from a Microsoft Access database into Oracle	Yunus Hassan	000880204
Member A – from staging area to Dimensions and Fact tables using cursor	Yunus Hassan	000880204
Member B – Data Cleansing	Usman Basharat	000874782
Member B – Implementation of Dimensions & Fact	Usman Basharat	000874782
Please notice that everything else have been attempted together. Issues and Evaluation have been noted of which student has completed which section.		

References

Golfarelli, M. and Rizzi, S., 2009. *Data Warehouse Design*. New York: McGraw-Hill, pp.pg.1-pg.10.

Smallcombe, M., 2020. *Snowflake Schemas Vs. Star Schemas: What Are They And How Are They Different?*. [online] Xplenty. Available at: <https://www.xplenty.com/blog/snowflake-schemas-vs-star-schemas-what-are-they-and-how-are-they-different/#whatssnowflake> [Accessed 29 March 2020].