the UNIVERSITY of GREENWICH

COMP1434 Data Warehousing

Trevor Evans & Mohamed Alkaisi

University of Greenwich

4/15/2020

Table of Contents

Trevor Evans: Member A

Mohamed Alkaisi: Member B

DEMONSTRATION VIDEOS:

Trevor - https://gre.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=dcb85239-acd3-42ba-b862-ab9e00bf253a

Mohamed - https://gre.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=0c9fd0f3-bfdb-447e-881e-ab9e00fa85fb

Table of Figures

Snowflake Schema

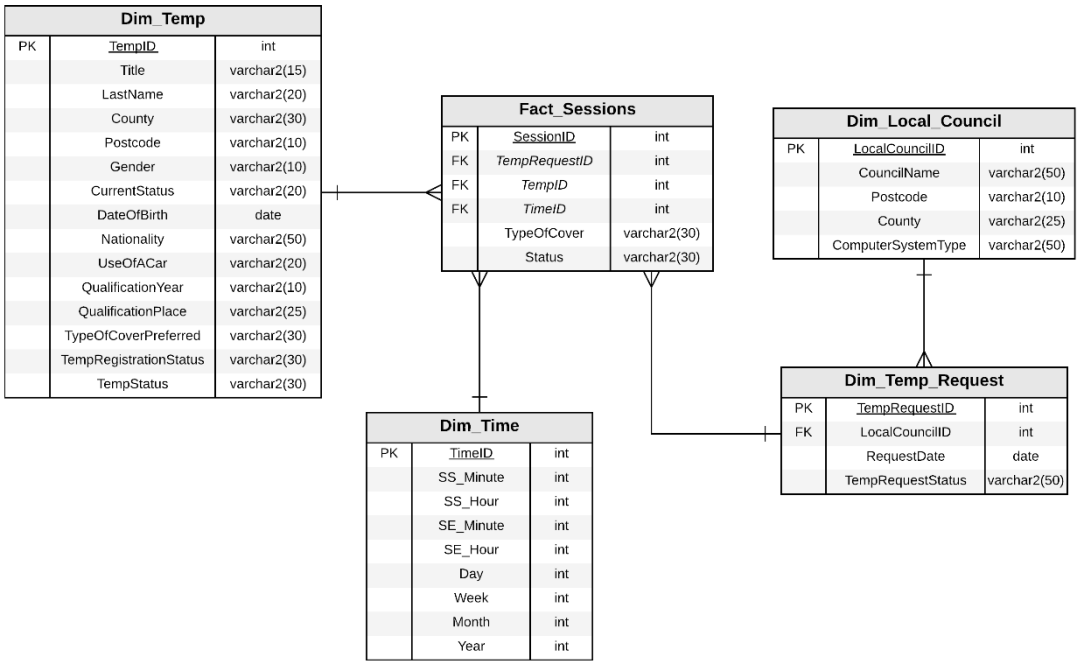

Figure 1 Star Schema Entity-Relationship Diagram (ERD)

Link to ERD: https://www.lucidchart.com/invitations/accept/fff76b97-90a9-4a2e-b351-523fbaddb2b6

Upon initial inspection of the ERD provided in the coursework specification it can be seen that the *sessions* table links everything together and therefore will be the source for the FACT table in the data warehouse. The tables and columns selected for the dimension tables in the data warehouse are deliberately chosen based on the queries from the coursework specification. Each of the queries specifies a time period that the results need to be ordered by, this data will be extracted from the original tables and imported in the *time* dimension table in the data warehouse.

The first query requests the number of sessions filled by type of temp cover. The table required for this query is *sessions*, which contains the type of cover column.

The second query requests the number of temp care worker requests made by council. The tables required for this are *temp request*, which contains the actual temp request entities to count, and *local council*, which contains the name of the councils.

The third query requests the number of temp care worker requests filled by county. So, this query will require the same tables as the second query. The difference is that this query requests the names of the counties instead of the names of the councils.

The fourth query requests the number of temp care worker requests filled by council. This query requires the same tables and columns as the second query.

The fifth query requests the number of temp care worker requests which were cancelled. This query only requires *sessions* and *temp request*.

Therefore, the essential tables needed for the data warehouse are *sessions, temp request,* and *local council*. The decision to include *temp* has been made in the case that, in the future, if information about the temps is needed, it will already be there and ready.

Extract Transform Load (ETL)

```
CREATE TABLE TEMP_REQUEST ( TempRequestID int, LocalCouncil_ID int, Request_date date, Start_date varchar2(50), End_date varchar2(50),
Monday_AM varchar2(50), Monday_PM varchar2(50), Tuesday_AM varchar2(50), Tuesday_PM varchar2(50), Wednesday_AM varchar2(50),
Wednesday_PM varchar2(50), Thursday_AM varchar2(50), Thursday_PM varchar2(50), Friday_AM varchar2(50), Friday_PM varchar2(50),
Saturday_AM varchar2(50), Request_Status int, Number_of_weeks int, Comments varchar2(100), PRIMARY KEY (TempRequestID ) );
CREATE INDEX "FK1" ON TEMP_REQUEST (LocalCouncil_ID);
CREATE TABLE REFERENCES ( ReferenceID int, TempID int, RefereeID int, Date_Reference_Sent varchar2(50),
Date_Reference_Received varchar2(50), Telephone_made varchar2(50), PRIMARY KEY (ReferenceID) );
CREATE TABLE REFEREE_DETAILS ( RefereeID int, First_Name varchar2(50), Last_Name varchar2(50),
Address_Line_1 varchar2(50), Adress_Line_2 varchar2(50), Town varchar2(50), County varchar2(50),
Postcode varchar2(50), Tel varchar2(50), PRIMARY KEY (RefereeID) );
CREATE TABLE CAREPROVIDER_COMPUTER_SYSTEM ( ComputerSystemID int, ComputerSystemDescription varchar2(50),PRIMARY KEY (ComputerSystemID) );
CREATE TABLE EMPLOYEE_DETAILS ( Employee_ID int, LocalCouncil_ID int, Title varchar2(50), First_Name varchar2(50),
Last_Name varchar2(50), Status varchar2(50), YearQualified varchar2(4), Initials varchar2(50), show varchar2(50),
deleted varchar2(50), PRIMARY KEY (Employee_ID) );
CREATE INDEX "FK2" ON EMPLOYEE_DETAILS (LocalCouncil_ID);
CREATE TABLE TEMP ( TempID int, Title varchar2(50), First_Name varchar2(50), Last_Name varchar2(50),
Address_Line_4 varchar2(50),Address_Line_2 varchar2(50), Town varchar2(50), County varchar2(50),
 Postcode varchar2(50), Tel_Home varchar2(50), Fax_Home varchar2(50), Tel_Work varchar2(50), Fax_Work varchar2(50),
 Mobile_Phone varchar2(50), Date_application_was_sent varchar2(50), Gender varchar2(50),
Current_Status varchar2(50), Date_of_birth date, Nationality varchar2(50), Use_of_a_car varchar2(50),Qualification_Year varchar2(50),
Qualification_Place varchar2(50), Type_of_Cover_Preferred varchar2(50),
 Monday_AM varchar2(50), Monday_PM varchar2(50), Tuesday_AM varchar2(50), Tuesday_PM varchar2(50), Wednesday_AM varchar2(50),
 Wednesday_PM varchar2(50), Thursday_AM varchar2(50), Thursday_PM varchar2(50),
Friday_AM varchar2(50), Friday_PM varchar2(50), Saturday_AM varchar2(50),Temp_Registration_Status varchar2(50),
Date_application_received varchar2(50), TempStatus varchar2(50), PRIMARY KEY(TempID) );
CREATE TABLE EMPLOYEE_TITLE ( TitleID int, TitleDescription varchar2(50), PRIMARY KEY (TitleID) );
CREATE TABLE SESSIONS ( SessionID int, RequestID int, TempID int, SessionDate date, SessionStart timestamp,
SessionEnd timestamp, Status varchar2(50), Type varchar2(50), EmployeeCovered varchar2(50), DoPrint varchar2(50) ,
Year int,Month int,Week int,Day int,SS_Hour int,SS_Minutes int,SE_Hour int,SE_Minutes int, PRIMARY KEY (SessionID) );
CREATE TABLE LOCAL_COUNCIL ( LocalCouncil_ID int, LocalCouncilName varchar2(50), Address_Line_1 varchar2(50),
Address_Line_2 varchar2(50), Town varchar2(50), County varchar2(50), Postcode varchar2(50), Locality varchar2(50),
Telephone varchar2(50), Fax varchar2(50), Bypass varchar2(50), Approximate_List_Size varchar2(50),
Type_of_Computer_System varchar2(50), Computer_Used_in_Consultations varchar2(50), Appointment_System varchar2(50),
Leaflets varchar2(50), Date_of_Last_Update varchar2(50), TypeofLocalCouncil varchar2(50), PRIMARY KEY (LocalCouncil_ID) );
CREATE TABLE TEMP_NATIONALITY ( NationalityID int, NationalityDescription varchar2(50), PRIMARY KEY (NationalityID) );
CREATE TABLE "TYPE_OF_TEMP_COVER" ( TypeofCoverID int, CoverDescription varchar2(50), PRIMARY KEY (TypeofCoverID ) );
 CREATE TABLE TEMP_AGE_BAND ( Age_band_name varchar2(50),Age_band_low varchar2(50), Age_band_high varchar2(50));
CREATE TABLE TEMP_COMPATABILITY_DETAILS( TempID int,BranchID int, Compatibility varchar2(50));
CREATE TABLE TEMP_CURRENT_STATUS ( CurrentStatusID int,CurrentStatusDescription varchar2(50));
CREATE TABLE TEMP_GENDER( GenderID int,GenderDescription varchar2(50));
CREATE TABLE TEMP_MEETING_DATE  ( Next_Locemdoc_meeting_date varchar2(50));
CREATE TABLE TEMP_PERMANENT_JOB_TYPE  ( PermanentJobID int,Type_of_Permanent_Job varchar2(50));
CREATE TABLE TEMP_REASONS_FOR_ARCHIVING  ( ReasonforArchivingID int, ReasonforArchivingDescription varchar(50));
CREATE TABLE TEMP_REGISTRATION_STATUS( LocumRegistrationStatusID int,LocumRegistrationStatusDescription varchar(50) );
CREATE TABLE TEMP_REPORT_DATES ( First_date varchar2(50),  Last_date varchar2(50));
CREATE TABLE TEMP_REQUEST_STATUS ( RequestStatusID int,RequestStatusDescription varchar2(50));
CREATE TABLE TEMP_STATUS  ( TempStatusID int, TempStatusDescription varchar2(50));
```

Figure 2 Original Database Schema Code

Before the data can be loaded into SQL Developer, the schema needs to be created. This is done so by running a SQL file that contains all the create table commands as well as creates the primary key and foreign key constraints as shown in Figure 2.

There are many tools available to perform ETL of data such as Amazon's AWS Glue or Microsoft's SQL Server Integrated Services (SSIS). Since the work is being hosted in an Oracle SQL Developer database, it makes the most sense to use an Oracle ETL tool to import the data. Therefore, the ETL of data in this work is carried out by Oracle's SQL Loader tool. However, the original file provided is a Microsoft Access Database (MDB) file and MDB files are not compatible with SQL Loader so, the tables from the original file are exported into individual comma separated value (CSV) files. SQL Loader requires a user's login information and a control file to determine where the data is coming from, how to transform it, and where to import it. Control files were generated for each of the tables.

```
load data
infile 'G:\SQL_Loader_Files\csv\TEMP.csv' "str '\r\n'"
truncate
into table TEMP
fields terminated by ','
OPTIONALLY ENCLOSED BY '"' AND '"'
trailing nullcols
          ( TempID,
            Title,
            First_Name CHAR(4000),
            Last_Name CHAR(4000),
            Address_Line_4 CHAR(4000),
            Address_Line_2 CHAR(4000),
            Town CHAR(4000),
            County CHAR(4000),
            Postcode CHAR(4000),
            Tel_Home CHAR(4000),
            Fax_Home CHAR(4000),
            Tel_Work CHAR(4000),
            Fax_Work CHAR(4000),
            Mobile_Phone CHAR(4000),
            Date_application_was_sent  DATE "DD/MM/YYYY",
            Gender,
            Current_Status,
            Date_of_birth DATE "DD/MM/YYYY",
            Nationality,
            Use_of_a_car CHAR(4000),
            Qualification_Year CHAR(4000),
            Qualification_Place CHAR(4000),
            Type_of_Cover_Preferred,
            Monday_AM CHAR(4000),
            Monday_PM CHAR(4000),
            Tuesday_AM CHAR(4000),
            Tuesday_PM CHAR(4000),
            Wednesday_AM CHAR(4000),
            Wednesday_PM CHAR(4000),
            Thursday_AM CHAR(4000),
            Thursday_PM CHAR(4000),
            Friday_AM CHAR(4000),
            Friday_PM CHAR(4000),
            Saturday_AM CHAR(4000),
            Temp_Registration_Status,
            Date_application_received  DATE "DD/MM/YYYY",
            TempStatus
          )
```

Figure 3 Control File (Temp. ctl)

Figure 2 shows the instructions inside the TEMP control file. The instructions at the top of the file indicate that the data from the CSV file, TEMP, is to be truncated into the table TEMP. The long list of columns and datatypes in the file match the columns and datatypes in the table TEMP to ensure all the data gets properly imported.

```
sqlldr te4695k/te4695k@obiwan
CONTROL=G:\SQL_Loader_Files\ctls\TEMP.ctl
LOG=G:\SQL_Loader_Files\logs\TEMP.log
BAD=G:\SQL_Loader_Files\bads\TEMP.bad skip=1
```

Figure 4 SQLLDR Command

The SQLLDR command shown in Figure 3 has the user's login information on the first line, the control file specified on the second line, and has two more parameters on the next two lines. These additional parameters specify where to store the LOG and BAD files. The LOG file will provide a detailed receipt of the SQLLDR transaction, specifying if the data was imported successfully or if it was not imported successfully and why it was not. The BAD file will contain the data that was not successfully imported.

There are many tables from the original schema, therefore a batch file is created containing separate SQLLDR commands for the ETL of each table. Simply running the batch file transforms and imports all the data from the original CSV files into the tables in SQL Developer.

Data Cleansing

Temp

The *temp* table has many foreign key relations with other tables. In order to minimize the number of tables in the data warehouse, the foreign key columns in *temp* have been denormalized to contain the actual values from the parent tables. The denormalized columns are title, current status, nationality, type of cover preferred, temp registration status, and temp status. These columns have some null values so those are replaced with 'NOT KNOWN' to prevent returning null values in queries. Rows in *temp* that have a null town and county, or a null postcode are deleted.

When inspecting the data, some values in the postcode column are not formatted correctly or are not valid postcodes at all. First, using regular expressions, the rows containing invalid postcodes are deleted because there are only a few of them. Regular expressions are also used to update and validate the format of the remaining postcodes. The regular expression first removes all spaces from the postcode leaving only the letters and numbers. Then the length of the postcode is counted, if the length is seven characters then a space is reinserted after the first four characters and if the length is six characters then a space is inserted after the first three characters.

Local Council

The decision is made to include a location column for the data warehouse; the column chosen is county because it contained the most accurate data. The address and town columns contain many rows with invalid values. When inspecting the data, a pattern is discovered. The pattern is that whenever a row has a null county, the correct county value has been improperly stored in the town column. Therefore, a cursor to loop through those rows is written to move the correct county value from the town column into the county column.

The postcode column is also selected to be in the data warehouse. So, using regular expressions, the postcodes are validated using the same methods as in the *temp* table's postcodes.

The type of computer system is also included in the data warehouse therefore the column has its foreign key values denormalized to the actual values from the parent table.

A pattern of improperly stored data is discovered for *local council* too. Where the council name column is null, the correct council name value is stored in the address line 2 column. Therefore, a cursor is written to iterate through those rows the move the correct council name value into the council name column.

Temp Request

The temp request id numbers in this table are 7 digits long, which is 4 digits longer than the primary key values in every other table. Additionally, the first 4 digits of each primary key are the same 4 digits. The only unique differences are in the hundredths, tenths, and one's position of the primary keys, so the decision is made to use the SUBSTR SQL function to remove the first 4 digits of each primary key in *temp request*.

Sessions

The status column contains four different values: booked, unbooked, branch cancelled, and temp cancelled. The reason or party responsible for a cancellation is not relevant at this point in time, therefore the decision is made to update all unbooked, branch cancelled, and temp cancelled to 'cancelled' leaving only two values in the column: booked and cancelled.

Sessions has a foreign key relation with *temp request*. Since the primary keys in *temp request* have their first 4 digits trimmed, the same is done for the *temp request* foreign key values in *sessions*.

The type of cover foreign key values have been denormalized to their actual values from the parent table, *type of temp cover*.

Data Warehouse

```
CREATE TABLE Dim_Temp ( TempID int, Title varchar2(15) NOT NULL, LastName varchar2(20) NOT NULL, County varchar2(30) NOT NULL,
Postcode varchar2(10) NOT NULL, Gender varchar2(10)  NOT NULL, CurrentStatus varchar2(20) NOT NULL, DateOfBirth date ,
Nationality varchar2(50) NOT NULL, UseOfACar VARCHAR2(20) NOT NULL, QualificationYear varchar2(10) NOT NULL,
QualificationPlace varchar2(25) NOT NULL, TypeOfCoverPreferred varchar2(30) NOT NULL,
TempRegistrationStatus varchar2(30) NOT NULL,TempStatus varchar2(30) NOT NULL, PRIMARY KEY (TempID));
CREATE TABLE Dim_Local_Council ( LocalCouncilID int, CouncilName varchar2(50)  NOT NULL, Postcode varchar2(10) NOT NULL,
County varchar2(25)  NOT NULL, ComputerSystemType varchar2(50) NOT NULL, PRIMARY KEY (LocalCouncilID));
CREATE TABLE Dim_Time ( TimeID int , SS_Minute int NOT NULL, SS_Hour int NOT NULL, SE_Minute int NOT NULL,
SE_Hour int NOT NULL, Day int NOT NULL, Week int NOT NULL, Month int NOT NULL, Year int NOT NULL, PRIMARY KEY (TimeID));
CREATE TABLE Dim_Temp_Request ( TempRequestID int, LocalCouncilID int NOT NULL REFERENCES Dim_Local_Council(LocalCouncilID),
RequestDate date, TempRequestStatus varchar2(50)  NOT NULL, PRIMARY KEY (TempRequestID));
CREATE TABLE Fact_Sessions ( SessionID int, TempRequestID int  NOT NULL REFERENCES Dim_Temp_Request(TempRequestID),
TempID int  NOT NULL REFERENCES Dim_Temp(TempID), TimeID int  REFERENCES Dim_Time(TimeID), TypeOfCover varchar2(30) NOT NULL,
Status varchar2(30) NOT NULL, PRIMARY KEY (SessionID));

CREATE SEQUENCE S_Time_PK
START WITH 1
INCREMENT BY 1
CACHE 10;

CREATE OR REPLACE TRIGGER T_Time_PK
BEFORE INSERT
ON Dim_Time
REFERENCING NEW AS NEW
FOR EACH ROW
BEGIN
  IF(:NEW.TimeID IS NULL) THEN
  SELECT S_Time_PK.NEXTVAL
  INTO :NEW.TimeID
  FROM dual;
  END IF;
END;
```

Figure 5 Data Warehousing Schema Code

Since it is established that *sessions* is the fact table, it will have foreign key relations to other tables in the data warehouse. *Fact_Sessions* contains foreign keys to *dim_temp*, *dim_temp_request*, and *dim_time*.

Since the primary keys for the *time* dimension are being created in the data warehouse as opposed to being created in the database and copied into the data warehouse they are imported into the fact table separately. Therefore when populating the fact table, two cursors are used, one retrieves the data from the original session table and the other retrieves the primary keys from the *time* dimension table for the foreign key relations.

The *time* dimension table in the data warehouse needs to get separate values for the year, month, week, day, session start time (hours), session start time (minutes), session end time (hours), and session end time(minutes) for each session. This data can be found in the *sessions* table under the columns session date, session start, and session end. A procedure is written using the EXTRACT SQL function to extract the needed values from their original columns and inserted separately into new columns for each specific value.

The time values being stored in the *time* dimension column need primary keys. The other tables in the data warehouse get their primary keys from the original relational database tables but *time* is not in the original schema therefor the primary key needs to increment. SQL Developer does not currently have an autoincrement function for primary keys, therefore a sequence and trigger are needed. The sequence is set to start with one and increment by one and the trigger is configured so that anytime a new record is inserted into the *time* dimension table the next value in the sequence will be generated and used as the new record's primary key.

PL/SQL Code Listings

All PL/SQL code listings in this work are stored in a package called *COMP1434_CLEANING*. The procedure at the top of the package is called *RUN_ME_ONLY* and the code inside is only calls to the other procedures. This allows the entire package to be run by simply running one procedure.

```
PROCEDURE RUN_ME_ONLY AS

BEGIN
    prep_Sessions();
    Cleaning_temp();
    Delete_Update_Null();
    clean_local_council();
    clean_Temp_request();
    Clean_Session();
    validate_postcode_format();
    TIME_DATE_EXTRACTION();
    DW_INSERT();
END RUN_ME_ONLY;
```

Figure 6 Procedure RUN_ME_ONLY

The next procedure is named *PREP_SESSIONS*. The purpose of this procedure is to delete rows from the *sessions* table where the temp id is null or zero. The decision to create this small procedure is explained in the 'Discussion of Problems' section of this report.

```
PROCEDURE prep_Sessions AS

BEGIN
    EXECUTE IMMEDIATE 'delete from sessions WHERE tempid = 0 OR tempid is null';
    EXECUTE IMMEDIATE 'alter TABLE sessions ADD FOREIGN KEY (tempid) REFERENCES temp(tempid) ON DELETE CASCADE';
END;
```

Figure 7 Procedure PREP_SESSIONS

The purpose of the remaining procedures is to clean data and insert it into the data warehouse tables. The first table to be cleaned is *temp*. *Temp* has many more columns being used for the data warehouse than any of the other tables and thus is the largest procedure.

```
PROCEDURE Cleaning_temp AS
      dr        employee_title.titledescription%TYPE;
      mr        employee_title.titledescription%TYPE;
      miss      employee_title.titledescription%TYPE;
      ms        employee_title.titledescription%TYPE;
      prof      employee_title.titledescription%TYPE;
      mrs       employee_title.titledescription%TYPE;
      nums      temp.title%TYPE;
      tempidd   temp.tempid%TYPE;
      tempiddd  temp.tempid%TYPE;
      genders   temp.gender%TYPE;
      addre     temp.address_line_2%TYPE;
      tempidddd temp.tempid%TYPE;
      cstatuss  temp.current_status%TYPE;
      v_county  temp.county%TYPE ;
      v_town    temp.town%TYPE;
      v_type    temp.type_of_cover_preferred%TYPE;
      v_rgs     temp.temp_registration_status%TYPE;
      v_date    temp.date_of_birth%TYPE;
      v_status  temp_status.tempstatusdescription%TYPE;
      v_check   BOOLEAN;
      v_count   NUMBER;
      v_code    NUMBER;
      v_error   VARCHAR(64);
      v_postcode temp.postcode%TYPE;
      v_id       temp.tempid%TYPE;
      v_length  Number;
      v_check   BOOLEAN;
```

Figure 8 Procedure CLEANING_TEMP (variables)

```
CURSOR temp_postcode IS
  SELECT regexp_replace(postcode, '[[:space:]]*',''), length(regexp_replace(postcode,
      '[[:space:]]*','')),tempid from temp;
CURSOR sess_requestid IS
  SELECT title, tempid FROM temp;
CURSOR v_gender IS
  SELECT gender, tempid FROM temp;
CURSOR v_tempst IS
  SELECT current_status, tempid FROM temp;
CURSOR v_nation IS
  SELECT nationality, tempid, address_line_2 FROM temp;
CURSOR v_countyid IS
  SELECT tempid FROM temp WHERE county IS NULL;
CURSOR v_phonenum IS
  SELECT tel_home, tempid FROM temp WHERE tel_home IS NULL;
CURSOR v_cover IS
  SELECT type_of_cover_preferred, tempid FROM temp;
CURSOR v_regid IS
  SELECT temp_registration_status, tempid FROM temp;
CURSOR v_tempstatus IS
  SELECT tempstatus, tempid FROM temp;
```

Figure 9 Procedure CLEANING_TEMP (cursors)

```
OPEN sess_requestid;
LOOP
    FETCH sess_requestid INTO nums, tempidd;
    IF sess_requestid%FOUND THEN
      IF nums IS NULL THEN
        UPDATE temp SET title = 'N/A' WHERE tempid = tempidd;
      ELSIF nums = '1' THEN
        UPDATE temp SET title = 'DR' WHERE tempid = tempidd;
      ELSIF nums = '2' THEN
        UPDATE temp SET title = 'MR' WHERE tempid = tempidd;
      ELSIF nums = '3' THEN
        UPDATE temp SET title = 'Miss' WHERE tempid = tempidd;
      ELSIF nums = '3' THEN
        UPDATE temp SET title = 'Mrs' WHERE  tempid = tempidd;
      ELSIF nums = '4' THEN
        UPDATE temp SET title = 'MS' WHERE  tempid = tempidd;
      ELSIF nums = '5' THEN
        UPDATE temp SET title = 'Prof' WHERE  tempid = tempidd;
      ELSIF nums = '6' THEN
        UPDATE temp SET title = 'MRS' WHERE  tempid = tempidd;
      END IF;
    END IF;
    EXIT WHEN sess_requestid%NOTFOUND;
END LOOP;
CLOSE sess_requestid;
```

Figure 10 Procedure CLEANING_TEMP (cursor SESS_REQUESTID)

```
OPEN v_gender;
LOOP
    FETCH v_gender INTO genders, tempiddd;
    IF v_gender%FOUND THEN
      IF genders = '1' THEN
        UPDATE temp SET gender = 'FEMALE' WHERE  tempid = tempiddd;
      ELSIF genders = '2' THEN
        UPDATE temp SET gender = 'MALE' WHERE  tempid = tempiddd;
      END IF;
    END IF;
    EXIT WHEN v_gender%NOTFOUND;
END LOOP;
CLOSE v_gender;
```

Figure 11 Procedure CLEANING_TEMP (cursor V_GENDER)

```
OPEN v_tempst;
LOOP
    FETCH v_tempst INTO cstatuss, tempidddd;
    IF v_tempst%FOUND THEN
      IF cstatuss = '1' THEN
        UPDATE temp SET current_status = 'Principal Full-Time' WHERE tempid = tempidddd;
      ELSIF cstatuss = '2' THEN
        UPDATE temp SET current_status = 'Principal Part-Time' WHERE  tempid = tempidddd;
      ELSIF cstatuss = '3' THEN
        UPDATE temp SET current_status = 'Assistant' WHERE  tempid = tempidddd;
      ELSIF cstatuss = '4' THEN
        UPDATE temp SET current_status = 'Temp' WHERE tempid = tempidddd;
      ELSIF cstatuss = '5' THEN
        UPDATE temp SET current_status = 'Retainer' WHERE  tempid = tempidddd;
      ELSIF cstatuss = '6' THEN
        UPDATE temp SET current_status = 'Retired' WHERE  tempid = tempidddd;
      ELSIF cstatuss = '7' THEN
        UPDATE temp SET current_status = 'Other' WHERE  tempid = tempidddd;
      ELSIF cstatuss = '8' THEN
        UPDATE temp SET current_status = 'Not Known' WHERE  tempid = tempidddd;
      ELSIF cstatuss = '0' THEN
        UPDATE temp SET current_status = 'Not Known' WHERE  tempid = tempidddd;
      END IF;
    END IF;
    EXIT WHEN v_tempst%NOTFOUND;
END LOOP;
CLOSE v_tempst;
```

Figure 12 Procedure CLEANING_TEMP (cursor V_TEMPST)

```
    OPEN v_nation;
    LOOP
        FETCH v_nation INTO cstatuss, tempidddd, addre;
        IF v_nation%FOUND THEN
          IF cstatuss = '1' THEN
            UPDATE temp SET nationality = 'British' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '2' THEN
            UPDATE temp SET nationality = 'Irish' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '3' THEN
            UPDATE temp SET nationality = 'Hindi' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '4' THEN
            UPDATE temp SET nationality = 'Arabic' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '5' THEN
            UPDATE temp SET    nationality = 'African' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '6' THEN
            UPDATE temp SET nationality = 'Greek' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '7' THEN
            UPDATE temp SET nationality = 'Not Known' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '8' THEN
            UPDATE temp SET nationality = 'New Zealand' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '9' THEN
            UPDATE temp SET nationality = 'South Africa' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '10' THEN
            UPDATE temp SET nationality = 'European' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '11' THEN
            UPDATE temp SET nationality = 'Chinese' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '12' THEN
            UPDATE temp SET nationality = 'Australian' WHERE  tempid = tempidddd;
          ELSIF cstatuss = '13' THEN
            UPDATE temp SET nationality = 'Indian' WHERE  tempid = tempidddd;
          ELSIF cstatuss IS NULL THEN
            UPDATE temp SET nationality = 'Not Known' WHERE  tempid = tempidddd;
          END IF;
        END IF;
        EXIT WHEN v_nation%NOTFOUND;
    END LOOP;
    CLOSE v_nation;
```

Figure 13Procedure CLEANING_TEMP (cursor V_NATION)

```
OPEN v_countyid;
LOOP
    FETCH v_countyid INTO tempidddd;
    IF v_countyid%FOUND THEN
      SELECT town INTO v_county FROM temp WHERE  tempid = tempidddd;
      UPDATE temp SET county = v_county WHERE  tempid = tempidddd;
    END IF;
    EXIT WHEN v_countyid%NOTFOUND;
END LOOP;
CLOSE v_countyid;
```

Figure 14 Procedure CLEANING_TEMP(cursor V_COUNTYID)

```
OPEN v_cover;
LOOP
    FETCH v_cover INTO v_type, tempidddd;
    IF v_cover%FOUND THEN
      IF v_type = '1' THEN
        UPDATE temp SET type_of_cover_preferred = 'Shop Assistant' WHERE  tempid = tempidddd;
      ELSIF v_type = '2' THEN
        UPDATE temp SET    type_of_cover_preferred = 'Floor Manager' WHERE  tempid = tempidddd;
      ELSIF v_type = '3' THEN
        UPDATE temp SET type_of_cover_preferred = 'Window Dresser ' WHERE  tempid = tempidddd;
      ELSIF v_type = '4' THEN
        UPDATE temp SET type_of_cover_preferred = 'Stores Assistant' WHERE  tempid = tempidddd;
      ELSIF v_type = '5' THEN
        UPDATE temp SET type_of_cover_preferred = 'Cleaner' WHERE  tempid = tempidddd;
      ELSIF v_type = '6' THEN
        UPDATE temp SET type_of_cover_preferred = 'Cashier' WHERE  tempid = tempidddd;
      ELSIF v_type = '0' THEN
        UPDATE temp SET type_of_cover_preferred = 'N/A' WHERE  tempid = tempidddd;
      END IF;
    END IF;
    EXIT WHEN v_cover%NOTFOUND;
END LOOP;
CLOSE v_cover;
```

Figure 15 Procedure CLEANING_TEMP (cursor V_COVER)

```
OPEN v_regid;
LOOP
    FETCH v_regid INTO v_rgs, tempidddd;
    IF v_regid%FOUND THEN
      IF v_rgs = '1' THEN
        UPDATE temp SET temp_registration_status = 'Waiting for Application Form' WHERE  tempid = tempidddd;
      ELSIF v_rgs = '2' THEN
        UPDATE temp SET temp_registration_status = 'None' WHERE  tempid = tempidddd;
      ELSIF v_rgs = '3' THEN
        UPDATE temp SET temp_registration_status = 'Waiting for Both References' WHERE  tempid = tempidddd;
      ELSIF v_rgs = '4' THEN
        UPDATE temp SET temp_registration_status = 'Waiting for 1 Reference' WHERE  tempid = tempidddd;
      ELSIF v_rgs = '5' THEN
        UPDATE temp SET temp_registration_status = 'Waiting for Both References' WHERE  tempid = tempidddd;
          ELSIF v_rgs = '6' THEN
        UPDATE temp SET temp_registration_status = 'Waiting for a decision' WHERE  tempid = tempidddd;
        ELSIF v_rgs = '0'  THEN
        UPDATE temp SET temp_registration_status = 'None' WHERE  tempid = tempidddd;
         ELSIF v_rgs = '14'  THEN
        UPDATE temp SET temp_registration_status = 'None' WHERE  tempid = tempidddd;
      ELSIF v_rgs IS NULL THEN
        UPDATE temp SET temp_registration_status = 'None' WHERE  tempid = tempidddd;
      END IF;
    END IF;
    EXIT WHEN v_regid%NOTFOUND;
END LOOP;
CLOSE v_regid;
```

Figure 16 Procedure CLEANING_TEMP (cursor V_REGID)

```
OPEN v_tempstatus;
LOOP
    FETCH v_tempstatus INTO v_status, tempidddd;
    IF v_tempstatus%FOUND THEN
      IF v_status = '1' THEN
        UPDATE temp SET tempstatus = 'Live' WHERE  tempid = tempidddd;
      ELSIF v_status = '2' THEN
        UPDATE temp SET tempstatus = 'Archive' WHERE  tempid = tempidddd;
      END IF;
    END IF;
    EXIT WHEN v_tempstatus%NOTFOUND;
    END LOOP;
CLOSE v_tempstatus;
```

Figure 17 Procedure CLEANING_TEMP (cursor V_TEMPSTATUS)

```
OPEN temp_postcode;
LOOP
    FETCH temp_postcode INTO v_postcode,v_length,v_id ;
    IF temp_postcode%FOUND THEN
      IF v_length = 6 THEN
        UPDATE temp SET  postcode = substr(v_postcode, 1, 3) || ' ' || substr(regexp_replace(v_postcode, '[[:space:]]*',''), 4)
        WHERE tempid = v_id;
      ELSIF v_length = 7 THEN
        UPDATE temp SET  postcode = substr(v_postcode, 1, 4) || ' ' || substr(regexp_replace(v_postcode, '[[:space:]]*',''), 5)
        WHERE tempid = v_id;
      ELSE
        DELETE FROM temp WHERE tempid = v_id;
      END IF;
    END IF;
    EXIT WHEN temp_postcode%NOTFOUND;
  END LOOP;
  CLOSE temp_postcode;

EXCEPTION
WHEN OTHERS THEN
         v_code := SQLCODE;

         v_error := Substr(SQLERRM, 1, 64);

         dbms_output.Put_line('ERROR CODE ='
                              || v_code
                              || ' : '
                              ||v_error);
END cleaning_temp;
```

Figure 18 Procedure CLEANIING_TEMP (cursor TEMP_POSTCODE)

```
PROCEDURE Delete_Update_Null AS
BEGIN
    UPDATE temp SET title = 'NOT_KNOWN' WHERE title IS NULL;
    UPDATE temp SET gender = 'NOT_KNOWN' WHERE gender IS NULL OR gender = '0';
    UPDATE temp SET GENDER = 'NOT_KNOWN' WHERE  gender ='0' OR gender IS NUll;
    UPDATE temp SET qualification_place = 'NOT_KNOWN' WHERE qualification_place IS NUll;
    UPDATE temp SET qualification_year = 'NOT_KNOWN'  WHERE qualification_year IS NULL;
    UPDATE sessions SET status ='cancelled' WHERE status ='Unbooked' or status ='Temp Cancelled' or status = 'Branch Cancelled';
    UPDATE sessions SET status = 'booked' WHERE status ='Booked';
    DELETE FROM temp WHERE town IS NULL AND county IS NULL OR postcode IS NULL;
    UPDATE temp SET tel_home = 0 WHERE tel_home is null;
    COMMIT;
END Delete_Update_Null;
```

Figure 19 Procedure DELETE_UPDATE_NULL

```
PROCEDURE clean_local_council
AS
  v_county          local_council.county%TYPE;
  v_localcouncil_id local_council.localcouncil_id%TYPE;
  v_county          local_council.county%TYPE;
  v_town            local_council.town%TYPE;
  county_town       local_council.town%TYPE;
  v_id              local_council.localcouncil_id%TYPE;
  v_length          NUMBER;
  v_postcode        local_council.postcode%TYPE;
  v_cs              local_council.type_of_computer_system%TYPE;
  v_councilname     local_council.address_line_2%TYPE;
  CURSOR v_countyid IS
    SELECT localcouncil_id FROM local_council WHERE county IS NULL;
  CURSOR local_postcode IS
    SELECT Regexp_replace(postcode, '[[:space:]]*', ''), Length(Regexp_replace(postcode, '[[:space:]]*', '')), localcouncil_id
    FROM   local_council;
  CURSOR v_local IS
    SELECT type_of_computer_system, localcouncil_id FROM local_council;
  CURSOR c_councilname IS
    SELECT  localcouncil_id,address_line_2 FROM local_council WHERE localcouncilname IS NULL;
```

Figure 20 Procedure CLEAN_LOCAL_COUNCIL (variables and cursors)

```
OPEN v_countyid;
LOOP
    FETCH v_countyid INTO v_localcouncil_id;
    IF v_countyid%FOUND THEN
      SELECT town INTO county_town FROM local_council WHERE localcouncil_id = v_localcouncil_id AND county IS NULL;
      UPDATE local_council SET county = county_town WHERE localcouncil_id = v_localcouncil_id;
      DELETE FROM local_council WHERE town IS NULL AND county IS NULL OR postcode IS NULL;
      UPDATE local_council SET telephone = 0 WHERE telephone IS NULL;
    END IF;
    EXIT WHEN v_countyid%NOTFOUND;
END LOOP;
CLOSE v_countyid;
```

Figure 21 Procedure CLEAN_LOCAL_COUNCIL (cursor V_COUNTYID)

```
OPEN local_postcode;
LOOP
    FETCH local_postcode INTO v_postcode, v_length, v_id;
    IF local_postcode%FOUND THEN
      IF v_length = 6 THEN
        UPDATE local_council SET postcode = Substr(v_postcode, 1, 3) || ' ' || Substr(Regexp_replace(v_postcode,
            '[[:space:]]*', ''), 4) WHERE localcouncil_id = v_id;
      ELSIF v_length = 7 THEN
        UPDATE local_council SET  postcode = Substr(v_postcode, 1, 4) || ' ' || Substr(Regexp_replace(v_postcode,
            '[[:space:]]*', ''), 5) WHERE  localcouncil_id = v_id;
      ELSE
        DELETE FROM local_council WHERE localcouncil_id = v_id;
      END IF;
    END IF;
    EXIT WHEN local_postcode%NOTFOUND;
END LOOP;
CLOSE local_postcode;
```

Figure 22 Procedure CLEAN_LOCAL_COUNCIL (cursor LOCAL_POSTCODE)

```
OPEN v_local;
LOOP
    FETCH v_local INTO v_cs, v_id;
    IF v_local%FOUND THEN
      IF v_cs IS NULL THEN
        UPDATE local_council SET type_of_computer_system = 'Not Known' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '0' THEN
        UPDATE local_council SET  type_of_computer_system = 'Not Known' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '1' THEN
        UPDATE local_council SET type_of_computer_system = 'Windows 95' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '2' THEN
        UPDATE local_council SET type_of_computer_system = 'Windows 3.xx' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '3' THEN
        UPDATE local_council SET type_of_computer_system = 'Dos' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '4' THEN
        UPDATE local_council SET type_of_computer_system = 'Not Known' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '5' THEN
        UPDATE local_council SET type_of_computer_system = 'VAMP' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '6' THEN
        UPDATE local_council SET type_of_computer_system = 'Meditel' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '7' THEN
        UPDATE local_council SET type_of_computer_system = 'Micro Solutions GP Plus' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '8' THEN
        UPDATE local_council SET type_of_computer_system = 'HMC' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '9' THEN
        UPDATE local_council SET type_of_computer_system = 'Emis' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '10' THEN
        UPDATE local_council SET type_of_computer_system = 'MCS' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '11' THEN
        UPDATE local_council SET type_of_computer_system = 'Genisyst' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '12' THEN
        UPDATE local_council SET type_of_computer_system = 'Seetec' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '13' THEN
        UPDATE local_council SET type_of_computer_system = 'Amsys' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '14' THEN
        UPDATE local_council SET type_of_computer_system = 'GP Exeter' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '15' THEN
        UPDATE local_council SET type_of_computer_system = 'Seven' WHERE  localcouncil_id = v_id;
      ELSIF v_cs = '16' THEN
        UPDATE local_council SET type_of_computer_system = 'Medusa' WHERE  localcouncil_id = v_id;
      END IF;
```

Figure 23 Procedure CLEAN_LOCAL_COUNCIL (cursor V_LOCAL)

```
OPEN c_councilname;
LOOP
    FETCH c_councilname INTO  v_id,v_councilname;
    IF c_councilname%FOUND THEN
        UPDATE local_council SET localcouncilname = v_councilname WHERE  localcouncil_id = v_id;
    END IF;
    EXIT WHEN c_councilname%NOTFOUND;
END LOOP;
CLOSE c_councilname;
```

Figure 24 Procedure CLEAN_LOCAL_COUNCIL (cursor V_LOCAL)

```
PROCEDURE clean_Temp_request AS
  v_reg_num       temp_request.temprequestid%TYPE;
  v_id            temp_request.temprequestid%TYPE;
  v_requeststatus temp_request.request_status%TYPE;
  CURSOR v_temprequest IS
    SELECT Substr(temprequestid, -3), temprequestid FROM temp_request;
BEGIN
    OPEN v_temprequest;
    LOOP
        FETCH v_temprequest INTO v_reg_num, v_id;
        IF v_temprequest%FOUND THEN
          UPDATE temp_request SET temprequestid = v_reg_num WHERE  temprequestid = v_id;
        END IF;
        EXIT WHEN v_temprequest%NOTFOUND;
    END LOOP;
    CLOSE v_temprequest;
END clean_Temp_request;
```

Figure 25 Procedure CLEAN_TEMP_REQUEST

```
PROCEDURE Clean_Session AS
    v_reg_num    sessions.requestid%TYPE;
    ids          sessions.sessionid%TYPE;
    idss          sessions.sessionid%TYPE;
    v_requestid   temp_request.temprequestid%TYPE;
    ids_temprequestid         temp_request.temprequestid%TYPE;
    v_type    temp.type_of_cover_preferred%TYPE;
    v_check   BOOLEAN;
    v_count   NUMBER;
    v_alter  varchar2(100);
    v_cursor INTEGER;
    v_code    NUMBER;
    v_error   VARCHAR(64);
    CURSOR sess_requestid IS
     SELECT SubSTR(requestid, -3),sessionid FROM sessions;
    CURSOR temp_requestid IS
     SELECT SubSTR(temprequestid, -3),temprequestid FROM temp_Request;
    CURSOR v_cover IS
     SELECT type, sessionid FROM sessions;
```

Figure 26 Procedure CLEAN_SESSION (variables and cursors)

```
OPEN sess_requestid;
LOOP
    FETCH sess_requestid INTO v_reg_num,ids;
    IF sess_requestid%FOUND THEN
       UPDATE sessions SET requestid = v_reg_num WHERE sessionid = ids;
    END IF;
    EXIT WHEN sess_requestid%NOTFOUND;
  END LOOP;
  CLOSE sess_requestid;
```

Figure 27 Procedure CLEAN_SESSION (cursor SESS_REQUESTID)

```
OPEN temp_requestid;
LOOP
    FETCH temp_requestid INTO  v_requestid, ids_temprequestid ;
      IF temp_requestid%FOUND THEN
        UPDATE temp_Request SET temprequestid = v_requestid WHERE temprequestid = ids_temprequestid;
        END IF;
        EXIT WHEN temp_requestid%NOTFOUND;
END LOOP;
CLOSE temp_requestid;
```

Figure 28 Procedure CLEAN_SOLUTION (cursor TEMP_REQUESTID)

```
    OPEN  v_cover;
    LOOP
        FETCH  v_cover  INTO v_type, idss;
        IF  v_cover%FOUND THEN
          IF   v_type = 1 THEN
            UPDATE sessions SET   type = 'Shop Assistant' WHERE sessionid = idss;
          ELSIF   v_type = 2 THEN
            UPDATE sessions SET type = 'Floor Manager' WHERE sessionid = idss;
          ELSIF   v_type = 3 THEN
            UPDATE sessions SET type = 'Window Dresser' WHERE sessionid = idss;
          ELSIF   v_type = 4 THEN
            UPDATE sessions SET type = 'Stores Assistant' WHERE sessionid = idss;
           ELSIF   v_type  = 0 THEN
            UPDATE sessions SET type = 'N/A' WHERE sessionid = idss;
          END IF;
        END IF;
        EXIT WHEN  v_cover%NOTFOUND;
    END LOOP;
    CLOSE  v_cover;

  EXCEPTION
    WHEN OTHERS THEN
              v_code := SQLCODE;

              v_error := Substr(SQLERRM, 1, 64);

              dbms_output.Put_line('ERROR CODE ='
                                   || v_code
                                   || ' : '
                                   ||v_error);
  END Clean_Session;
```

Figure 29 Procedure CLEAN_SOLUTION (cursor V_COVER)

```
PROCEDURE validate_postcode_format AS
    v_postCode temp.postcode%TYPE;
    v_tempid temp.tempid%TYPE;

    CURSOR v_pc IS
        SELECT  postcode,tempid  FROM TEMP;
    BEGIN
    OPEN v_pc;
    LOOP
        FETCH  v_pc INTO v_postCode,v_tempid;
        IF v_pc%FOUND THEN
            IF v_postCode = 'GIR 0AA' THEN
                DBMS_OUTPUT.PUT_LINE('');
            END IF;
            IF LENGTH( v_postCode ) > 8 THEN
                DBMS_OUTPUT.PUT_LINE(v_postCode || 'invalid PostCode ' || '2'||' '||'ID ='|| v_tempid);
            END IF;
            IF REGEXP_LIKE(v_postCode, '^[[:alpha:]]{1,2}[[:digit:]]{1,2}[[:alpha:]]{0,1}[[:space:]]{1}[[:digit:]]{1}[[:alpha:]]{2}')
            THEN
                DBMS_OUTPUT.PUT_LINE('');
            ELSE
                DBMS_OUTPUT.PUT_LINE('');
            END IF;
        END IF;
        EXIT WHEN v_pc%NOTFOUND;
      END LOOP;
CLOSE v_pc;

END;
```

Figure 30 Procedure VALIDATE_POSTCODE_FORMAT

```
PROCEDURE TIME_DATE_EXTRACTION AS

    v_year      sessions.year%TYPE;
    v_month     sessions.month%TYPE;
    v_week      sessions.week%TYPE;
    v_day       sessions.day%TYPE;
    v_se_hour       sessions.se_hour%TYPE;
    v_se_minutes    sessions.se_minutes%TYPE;
    v_ss_hour       sessions.ss_hour%TYPE;
    v_ss_minutes    sessions.ss_minutes%TYPE;
    v_sessionID sessions.sessionid%TYPE;

    CURSOR c_sessionDate IS
        SELECT EXTRACT(year FROM sessiondate),EXTRACT(month FROM sessiondate),EXTRACT(day FROM sessiondate),EXTRACT(HOUR FROM sessionstart),
        EXTRACT(minute FROM sessionstart),EXTRACT(HOUR FROM sessionend),EXTRACT(minute FROM sessionend), sessionid FROM sessions;
    CURSOR c_sessionWeek IS
        SELECT to_number(to_char(to_date(sessiondate, 'DD/MM/YYYY'), 'WW')), sessionid FROM sessions;
```

Figure 31 Procedure TIME_DATE_EXTRACTION (variables and cursors)

```
OPEN c_sessionDate;
    LOOP
        FETCH c_sessionDate INTO v_year,v_month,v_day,v_ss_hour,v_ss_minutes,v_se_hour,v_se_minutes,v_sessionID;
        IF c_sessionDate%FOUND THEN
        UPDATE sessions SET year = v_year, month=v_month,day=v_day, ss_hour = v_ss_hour,ss_minutes = v_ss_minutes,
            se_hour = v_se_hour,se_minutes=v_se_minutes WHERE  sessionid = v_sessionID;
        END IF;
        EXIT WHEN c_sessionDate%NOTFOUND;
    END LOOP;
    CLOSE c_sessionDate;
```

Figure 32 Procedure TIME_DATE_EXTRACTION (cursor C_SESSIONDATE)

```
OPEN c_sessionWeek;
   LOOP
       FETCH c_sessionWeek INTO  v_week,v_sessionID;
       IF c_sessionWeek%FOUND THEN
           UPDATE sessions SET week = v_week WHERE  sessionid = v_sessionID;
       END IF;
       EXIT WHEN c_sessionWeek%NOTFOUND;
   END LOOP;
   CLOSE c_sessionWeek;
END TIME_DATE_EXTRACTION;
```

Figure 33 Procedure TIME_DATE_EXTRACTION (cursor C_SESSIONWEEK)

```
PROCEDURE DW_Insert AS

v_tempid   temp.tempid%TYPE;
v_title    temp.title%TYPE;
v_ln       temp.last_name%TYPE;
v_county   temp.county%TYPE;
v_pc       temp.postcode%TYPE;
v_gender   temp.gender%TYPE;
v_cs       temp.current_status%TYPE;
v_dob      temp.date_of_birth%TYPE;
v_nationality   temp.nationality%TYPE;
v_car       temp.use_of_a_car%TYPE;
v_qy        temp.qualification_year%TYPE;
v_qp        temp.qualification_place%TYPE;
v_type      temp.type_of_cover_preferred%TYPE;
v_trs       temp.temp_registration_status%TYPE;
v_ts        temp.tempstatus%TYPE;

CURSOR C_DIM_TEMP IS
select tempid,title,last_name,county,postcode,gender,current_status,date_of_birth,nationality,use_of_a_car,qualification_year,
qualification_place,type_of_cover_preferred,temp_registration_status,tempstatus from temp;
```

Figure 34 Procedure DW_INSERT (variables for 1st cursor)

```
v_councilid   local_council.localcouncil_id%TYPE;
v_councilname local_council.localcouncilname%TYPE;
v_local_pc          local_council.postcode%TYPE;
v_local_county      local_council.county%TYPE;
v_local_type         local_council.type_of_computer_system%TYPE;
CURSOR C_DIM_Local_Council IS
select localcouncil_id,localcouncilname,postcode,county,type_of_computer_system from local_council;
```

Figure 35 Procedure DW_INSERT (variables for 2nd cursor)

```
v_requestid temp_request.temprequestid%TYPE;
v_lcid      temp_request.localcouncil_id%TYPE;
v_requestdate temp_request.request_date%TYPE;
v_requeststatus temp_request.request_status%TYPE;

CURSOR C_DIM_TempRequest IS
select temprequestid,localcouncil_id,request_date,request_status from temp_request;
```

Figure 36 Procedure DW_INSERT (variables for 3rd cursor)

```
v_sessionid sessions.sessionid%TYPE;
v_session_requestid sessions.requestid%TYPE;
v_session_tempid   sessions.tempid%TYPE;
v_session_type    sessions.type%TYPE;
v_session_status   sessions.status%TYPE;


CURSOR c_Fact_Session IS
select sessionid,requestid,tempid,type,status  from sessions;
```

Figure 37 Procedure DW_INSERT (variables for 4th cursor)

```
v_sID   sessions.sessionid%TYPE;
v_year  sessions.year%TYPE;
v_month sessions.month%TYPE;
v_week  sessions.week%TYPE;
v_day   sessions.day%TYPE;
v_ss_hour sessions.ss_hour%TYPE;
v_ss_minutes sessions.ss_minutes%TYPE;
v_se_hour   sessions.se_hour%TYPE;
v_se_minutes sessions.se_minutes%TYPE;
v_timeid    dim_time.timeid%TYPE;
CURSOR c_DIM_TIME IS
SELECT sessionid,year,month,week,day,ss_hour,ss_minutes,se_hour,se_minutes from  sessions;

CURSOR c_timeid IS
   SELECT timeid  FROM dim_time;
```

Figure 38 Procedure DW_INSERT (variables for 5th cursor)

```
OPEN C_DIM_TEMP;
   LOOP
      FETCH C_DIM_TEMP INTO v_tempid,v_title,v_ln,v_local_county,v_local_pc,v_gender,v_cs,v_dob,
          v_nationality,v_car,v_qy,v_qp,v_local_type,v_trs,v_ts;
      IF C_DIM_TEMP%FOUND THEN
      INSERT INTO dim_temp (tempid,title,lastname,county,postcode,gender,currentstatus,dateofbirth,
          nationality,useofacar,qualificationyear,
      qualificationplace,typeofcoverpreferred,TempRegistrationStatus,tempstatus)VALUES(v_tempid,v_title,
          v_ln,v_local_county,v_local_pc,v_gender,v_cs,v_dob,v_nationality,v_car,v_qy,v_qp,v_local_type,v_trs,v_ts);
     END IF;
      EXIT WHEN C_DIM_TEMP%NOTFOUND;
   END LOOP;
CLOSE C_DIM_TEMP;
```

Figure 39 Procedure DW_INSERT (cursor C_DIM_TEMP)

```
OPEN C_DIM_Local_Council;
   LOOP
       FETCH C_DIM_Local_Council INTO  v_councilid,v_councilname,v_pc,v_county,v_type;
       IF C_DIM_Local_Council%FOUND THEN
       INSERT INTO dim_local_council (localcouncilid,councilname,postcode,county,computersystemtype)
           VALUES(v_councilid,v_councilname,v_pc,v_county,v_type);
       END IF;
       EXIT WHEN C_DIM_Local_Council%NOTFOUND;
   END LOOP;
   CLOSE C_DIM_Local_Council;
```

Figure 40 Procedure DW_INSERT (cursor C_DIM_LOCAL_COUNCIL)

```
OPEN C_DIM_TempRequest;
   LOOP
       FETCH C_DIM_TempRequest INTO  v_requestid,v_lcid ,v_requestdate,v_requeststatus;
       IF C_DIM_TempRequest%FOUND THEN
       INSERT INTO dim_temp_request (temprequestid,localcouncilid,requestdate,temprequeststatus)
           VALUES(v_requestid,v_lcid ,v_requestdate,v_requeststatus);
       END IF;
       EXIT WHEN C_DIM_TempRequest%NOTFOUND;
   END LOOP;
   CLOSE C_DIM_TempRequest;
```

Figure 41 Procedure DW_INSERT (cursor C_DIM_TEMPREQUEST)

```
OPEN c_Fact_Session;
   LOOP
       FETCH c_Fact_Session  INTO  v_sessionid,v_session_requestid,v_session_tempid,
           v_session_type,v_session_status;
       IF c_Fact_Session %FOUND THEN
       INSERT INTO fact_sessions (sessionid,temprequestid,tempid,typeofcover,status)
           VALUES(v_sessionid,v_session_requestid,v_session_tempid,v_session_type,v_session_status);
       END IF;
       EXIT WHEN c_Fact_Session%NOTFOUND;
   END LOOP;
   CLOSE c_Fact_Session;
```

Figure 42 Procedure DW_INSERT (cursor C_FACT_SESSION)

```
OPEN c_DIM_TIME;
   LOOP
       FETCH c_DIM_TIME  INTO v_sID,v_year,v_month,v_week,v_day,v_ss_hour,v_ss_minutes,
           v_se_hour,v_se_minutes;
       IF c_DIM_TIME%FOUND THEN
       INSERT INTO dim_time (year,month,week,day,ss_hour,ss_minute,se_hour,se_minute)
           VALUES(v_year,v_month,v_week,v_day,v_ss_hour,v_ss_minutes,v_se_hour,v_se_minutes);
       END IF;
CLOSE c_DIM_TIME;
```

Figure 43 Procedure DW_INSERT (cursor C_DIM_TIME)

```
       OPEN c_timeid;
              LOOP
              FETCH c_timeid INTO v_timeid;
              IF C_timeid%FOUND THEN

              UPDATE fact_sessions SET timeid =v_timeid WHERE sessionid = v_sID;
              END IF;
                 EXIT WHEN c_timeid%NOTFOUND;
              END LOOP;
      CLOSE c_timeid;
              EXIT WHEN c_DIM_TIME%NOTFOUND;
          END LOOP;

      End;
      END COMP1434_CLEANING;
```

Figure 44 Procedure DW_INSERT (cursor C_TIMEID)

SQL Scripts for Queries

```
/* 1. The number of sessions filled by type of temp cover by month. */
SELECT COUNT(fact_sessions.sessionid), fact_sessions.typeofcover, dim_time.Month
FROM fact_sessions
INNER JOIN dim_time ON fact_sessions.timeid = dim_time.timeid
GROUP BY dim_time.month, fact_sessions.typeofcover
ORDER BY dim_time.month;
```

Figure 45 Query 1

```
/* 2. The number of temp care worker requests made by council for each week. */
SELECT COUNT(fact_sessions.temprequestid), dim_local_council.councilname, dim_time.week
FROM fact_sessions
INNER JOIN dim_temp_request ON fact_sessions.temprequestid = dim_temp_request.temprequestid
INNER JOIN dim_local_council ON dim_temp_request.localcouncilid = dim_local_council.localcouncilid
INNER JOIN DIM_TIME ON fact_sessions.timeid = dim_time.timeid
GROUP BY dim_local_council.councilname, dim_time.week
ORDER BY dim_time.week;
```

Figure 46 Query 2

```
/* 3. The number of temp care worker requests filled by county for each month. */
SELECT COUNT(fact_sessions.temprequestid)AS "REQUESTS",dim_local_council.county,dim_time.month
FROM fact_sessions
INNER JOIN dim_temp_request ON fact_sessions.temprequestid = dim_temp_request.temprequestid
INNER JOIN dim_local_council ON dim_temp_request.localcouncilid = dim_local_council.localcouncilid
INNER JOIN DIM_TIME ON fact_sessions.timeid = dim_time.timeid
WHERE NOT fact_sessions.status = 'cancelled'
GROUP BY dim_local_council.county,dim_time.month, fact_sessions.status
ORDER BY dim_time.month;
```

Figure 47 Query 3

```
/* 4. The number of temp care worker requests filled by council each week. */
SELECT COUNT(fact_sessions.temprequestid) AS "REQUESTS",dim_local_council.councilname,dim_time.week
FROM fact_sessions
INNER JOIN dim_temp_request ON fact_sessions.temprequestid = dim_temp_request.temprequestid
INNER JOIN dim_local_council ON dim_temp_request.localcouncilid = dim_local_council.localcouncilid
INNER JOIN DIM_TIME ON fact_sessions.timeid = dim_time.timeid
WHERE NOT fact_sessions.status = 'cancelled'
GROUP BY dim_local_council.councilname, dim_time.week, fact_sessions.status
ORDER BY dim_time.week;
```

Figure 48 Query 4

```
/* 5. The number of temp care worker requests which were cancelled each month. */
SELECT COUNT(fact_sessions.temprequestid)AS "CANCELLED REQUESTS",dim_time.month,fact_sessions.status
FROM fact_sessions
INNER JOIN dim_temp_request ON fact_sessions.temprequestid = dim_temp_request.temprequestid
INNER JOIN DIM_TIME ON fact_sessions.timeid = dim_time.timeid
WHERE fact_sessions.status = 'cancelled'
GROUP BY dim_time.month, fact_sessions.status
ORDER BY dim_time.month;
```

Figure 49 Query 5

Bonus

Time Dimension Table in Data Warehouse

Dim_Time		
PK	TimeID	int
	SS_Minute	int
	SS_Hour	int
	SE_Minute	int
	SE_Hour	int
	Day	int
	Week	int
	Month	int
	Year	int

Figure 50 Time Dimension Table (ERD)

TIMEID	SS_MINUTE	SS_HOUR	SE_MINUTE	SE_HOUR	DAY	WEEK	MONTH	YEAR
1	30	8	30	10	2	14	4	2011
2	0	16	0	18	2	14	4	2011
3	30	8	30	10	5	14	4	2011
4	30	8	30	10	8	14	4	2011
5	0	16	0	18	8	14	4	2011
6	30	8	30	10	8	14	4	2011
7	30	8	30	10	9	15	4	2011
8	0	16	0	18	9	15	4	2011
9	30	8	30	10	9	15	4	2011

Figure 51 Time Dimension Table (Data Warehouse Data)

Create Materialised View

```
/* BONUS. The average length of a session cover by month. */
CREATE MATERIALIZED VIEW Avg_sessionlength
BUILD IMMEDIATE
REFRESH ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT fact_sessions.typeofcover, round(AVG(((se_hour*60+se_minute)-(ss_hour*60+ss_minute)))) AS "Length(Minutes)", dim_time.month
FROM dim_time
INNER JOIN fact_sessions ON dim_time.timeid = fact_sessions.timeid
GROUP BY fact_sessions.typeofcover, dim_time.month
ORDER BY dim_time.month;
```

Figure 52 Bonus Query with Materialised View

SQL *Loader Data Population

SQL Loader is not used to populate the temp data warehouse table, but SQL Loader is used to populate all the original tables with the original data.

Tool to Automate Cleansing

A package is created containing many procedures to clean the tables containing the data that is to be used in the data warehouse. A final procedure “RUN_ME_ONLY” is written containing calls to all the other procedure. So, running “RUN_ME_ONLY” automates the entire cleaning process as well as imports the cleaned data into the data warehouse.

Discussion of Problems

The first problem faced during this coursework was the export of the data from the MDB file. There was not a way to export the entire database or schema autonomously, so each table had to be manually exported to the CSV files.

The generated control files for the SQLLDR commands generated all the column data types as CHAR(4000). This was not an issue for most columns, but it was a problem for any data meant to be stored as a date. Therefore, the control files had to be manually modified to specify the date datatype for the date columns.

As mentioned in the 'PL / SQL Code Listings' section of this report, a procedure called *PREP_SESSIONS* containing only two lines of code was written because of a discovered problem. The problem was with the temp id foreign key relation between *sessions* and *temp*. *Sessions* being the child table of the relationship contained many rows that had 0 in the temp id column. The problem is that there is no record in *temp* (the parent table) that has a temp id of 0. The issue was discovered when trying to create the foreign key relationship; an error message would display saying the parent record cannot be found. This meant that the tables needed to be initially created without establishing the relationship, then importing the data, then deleting the rows from *sessions* that had a temp id of 0 and then altering *sessions* to add a foreign key constraint.

The addresses in the database had so many problems. Many addresses were in the wrong columns and many of them were not in a correct format. Many of the postcodes were not in a valid postcode format. The towns and counties were also a mess. After doing some research, a downloadable file from the postal service containing all UK addresses and postcodes exists and can be used to validate addresses in the database. However, the downloadable file costs £400 and therefore was not purchased nor used during the implementation of this work.