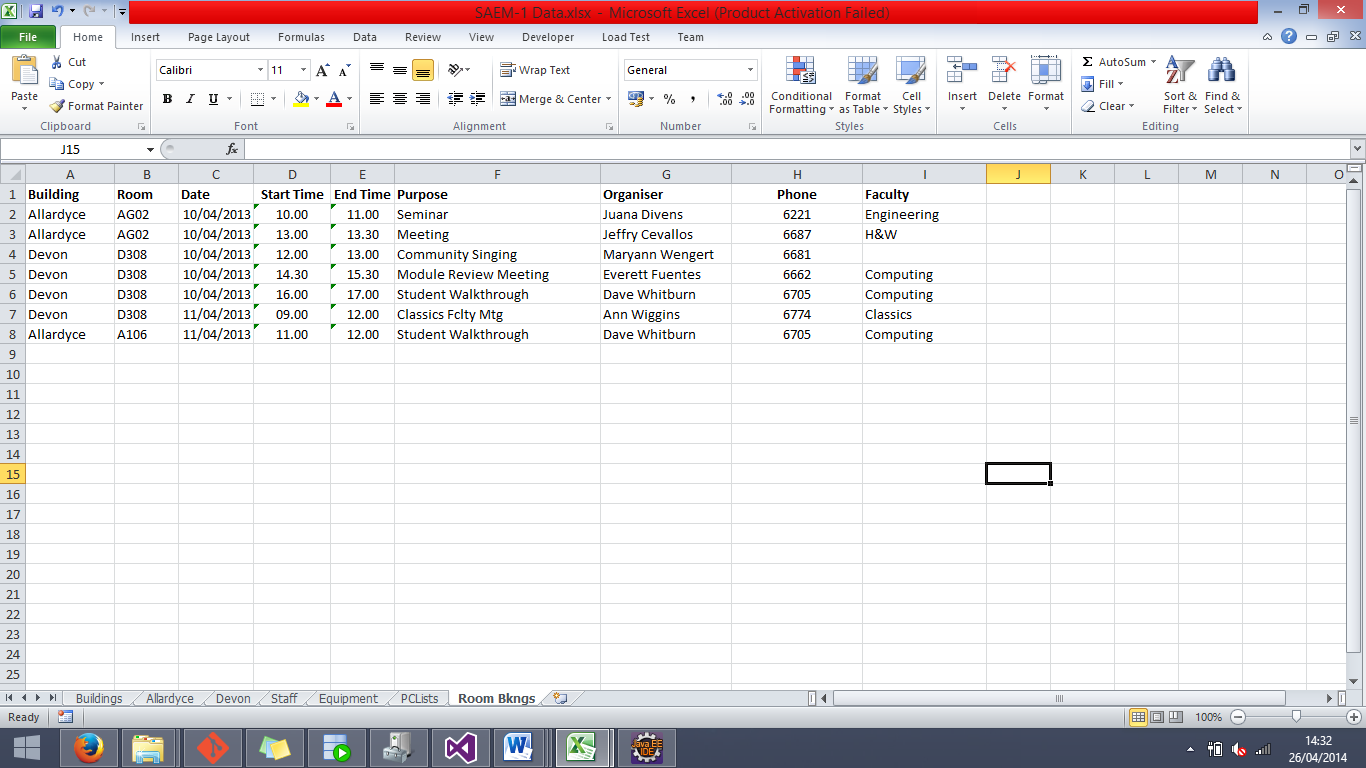
**Data Take-on Testing**

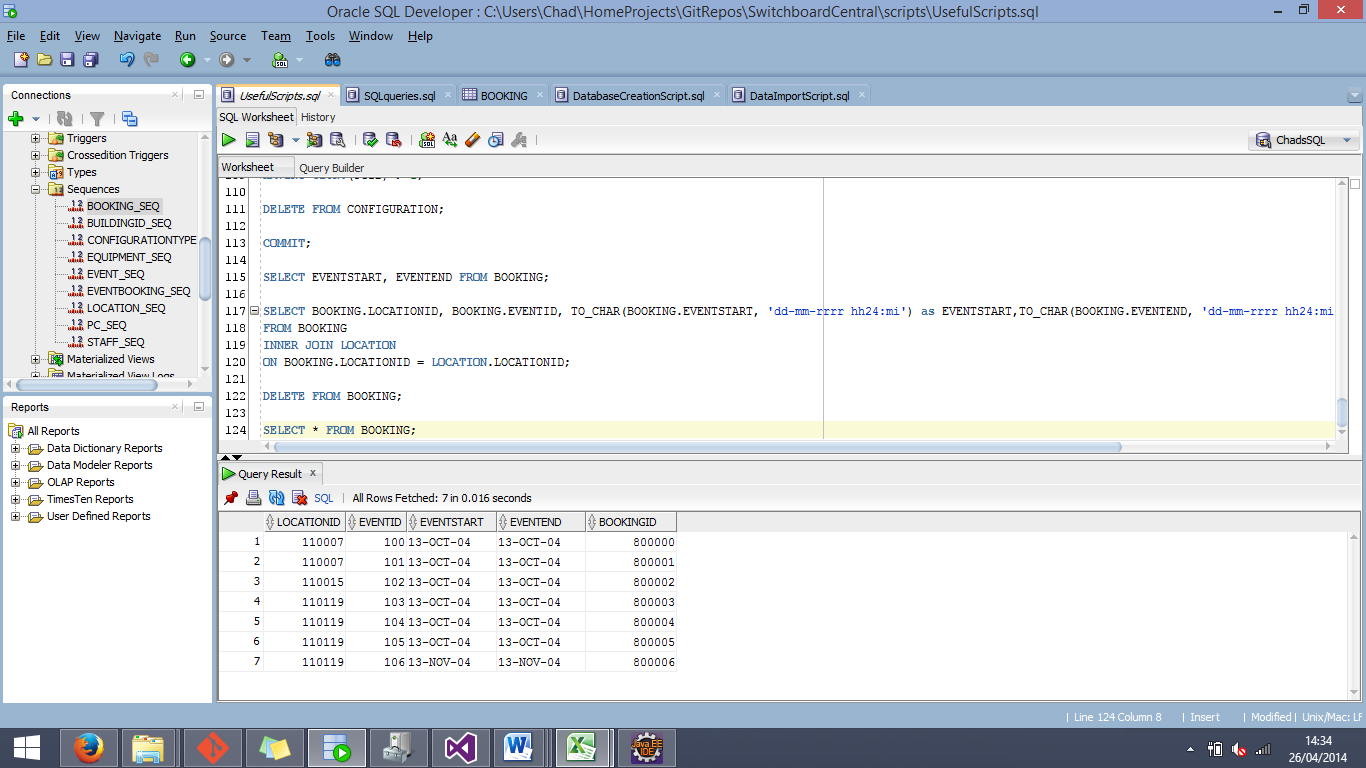
The first action we took when starting to test the data that we loaded was to look back on the source data that we were given and see if there were any aspects of it that would be able to be tested in the final database implementation to see if they were the same as they were in the source data.

As a result of that, we have compiled a series of tests that we performed on the loaded data inside of the database that could be compared to the raw data that we received in the form of an Excel document. From the results of these tests, we will be able to see if the data that we have loaded is valid and correct.

This testing will be split up into two sections, manual testing, and automated testing. The manual testing will largely be done by hand, performing actions such as simply checking the row counts or checking to see if a certain piece of equipment for a certain location has been loaded correctly. The automated testing will mostly be in the form of SQL scripts that can check things that would take a much longer time if they were to be done manually.

**Data Testing**

The first test that is performed on the loaded data is to check that the correct number of event bookings have been implemented into the database. This will be done by simply manually checking that the number of rows in the database is equal to the amount of rows in the source data.



As can be seen from the screenshots above, the amount of events in the final database is equal to the amount of data rows in the source document. There are 8 rows in the source document (8 minus the one row for column headers), and a total of 7 rows in the BOOKING database entity.

In an attempt to weave automated testing in with the manual testing, we made SQL scripts along the way that were able to perform the same action in the database, in order to more easily compare the data in the database with the source data.

SQL Script:

SELECT COUNT(\*) FROM EVENT;

------

The next series of manual tests revolved around making sure that the total amount of locations in the database is correct, and checking that each building had its correct number of locations

The first test was to compare the total amount of locations in the database to the cumulated amount of locations that were between the different building sheets in the Excel source file. We then compared the amount of locations that had been inserted into the database with our results, and made the table below to show the outcome of the tests.

|  |  |  |  |
| --- | --- | --- | --- |
| Buildings | Source Result | Database Result | Test Result |
| Devon | 96 | 96 | Matching Data |
| Allardyce | 36 | 36 | Matching Data |
| Total | 132 | 132 | Matching Data |

SQL Script:

SELECT COUNT(\*) FROM LOCATION WHERE BUILDINGID = (SELECT BUILDINGID FROM BUILDING WHERE BUILDINGNAME = :BName);

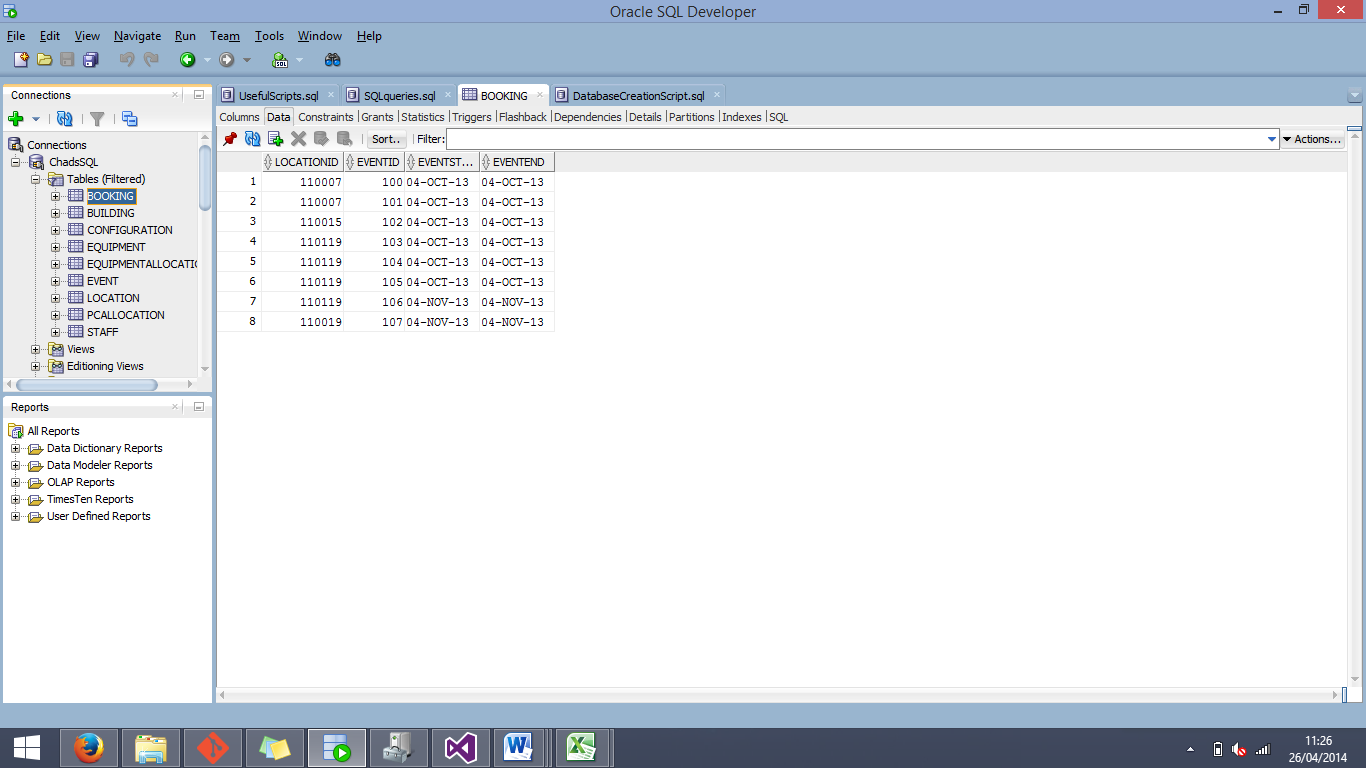
By using the bind variable to input the building name, we were able to check the amount of locations in each of the buildings, and by using this SQL script in conjunction of manual testing (counting the amount of locations in each of the sheets of the Excel file), we were able to prove that the loaded data is accurate in this scenario.

------

Another test that we performed was one exclusively on the database, and it was to check and see if it was possible to extract the date that was inserted into the database in a suitable format.

The data from the database that we used for this exercise was the EVENTSTART and EVENTEND fields from the BOOKING entity. To aid us in this test we created a script that would need to be used if we were going to be able to successfully extract the dates in a suitable format.

The default format of the date field data is shown below.

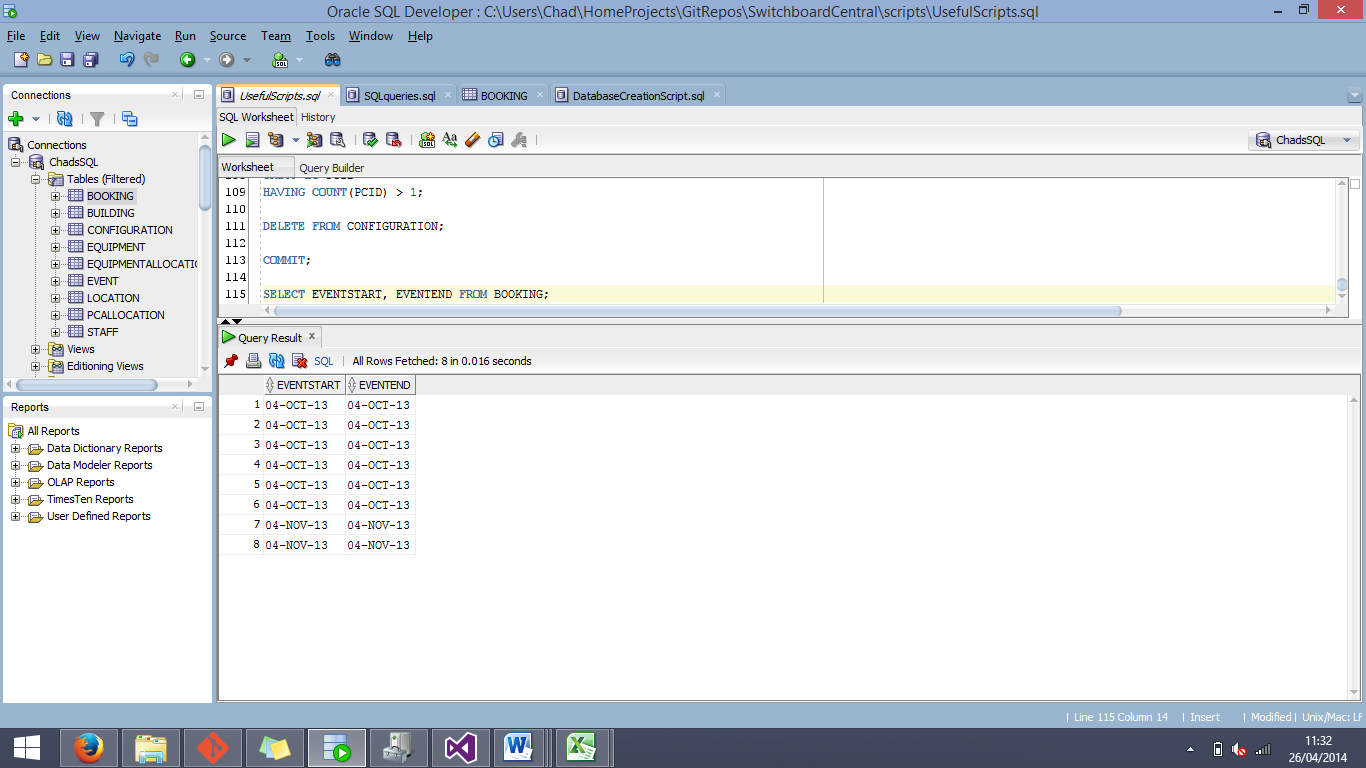


In order to see if this data would be able to show correctly by simply selecting the field, we first tested the following script:

SQL Script:

SELECT EVENTSTART, EVENTEND FROM BOOKING;

The result that was output is the following:



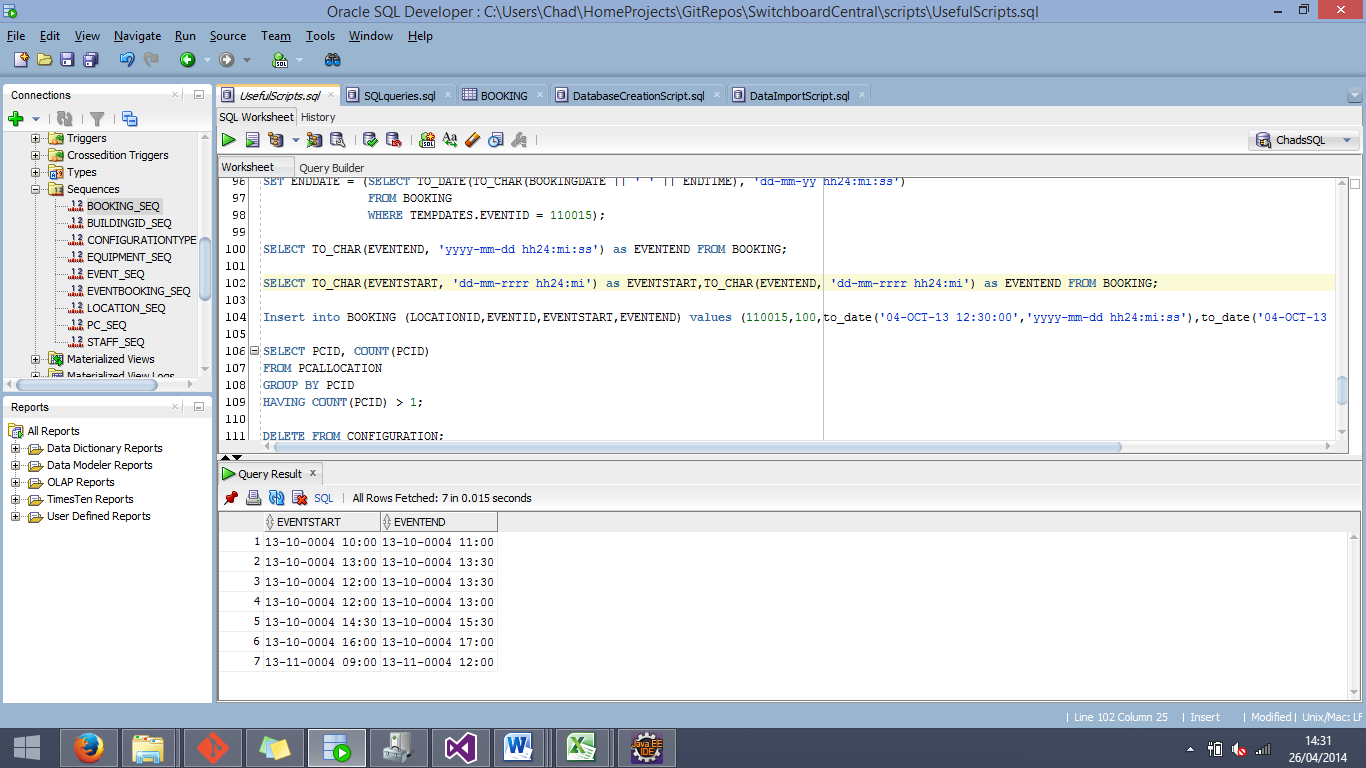
Now we know that we cannot extract the dates by normally selecting them from the BOOKING entity, we must test special methods of extracting, using functions to format the date on the way out.

To do this, we employed the TO\_CHAR() SQL function to format the outputted data to a format that would include the time as well as the date for the field.

SQL Script:

SELECT TO\_CHAR(EVENTSTART, 'dd-mm-rrrr hh24:mi') as EVENTSTART,TO\_CHAR(EVENTEND, 'dd-mm-rrrr hh24:mi') as EVENTEND FROM BOOKING;

As a result from this query, it can be seen that the data can be extrapolated in a valid format from the given date fields.



Now that it is confirmed that we are able to successfully extract the date and time from the BOOKING entity without errors, the next step will be to check and see if the data in the database correctly matches what was loaded from the source data.

To do this, we are going to combine manual checking with an SQL script to select all of the valid data from the database that would be needed to prove the authenticity of the test.

SQL Script:

SELECT BOOKINGID, EVENT.EVENTPURPOSE, TO\_CHAR(BOOKING.EVENTSTART, 'dd-mm-rrrr hh24:mi') as EVENTSTART,TO\_CHAR(BOOKING.EVENTEND, 'dd-mm-rrrr hh24:mi') as EVENTEND, LOCATION.ROOMNO

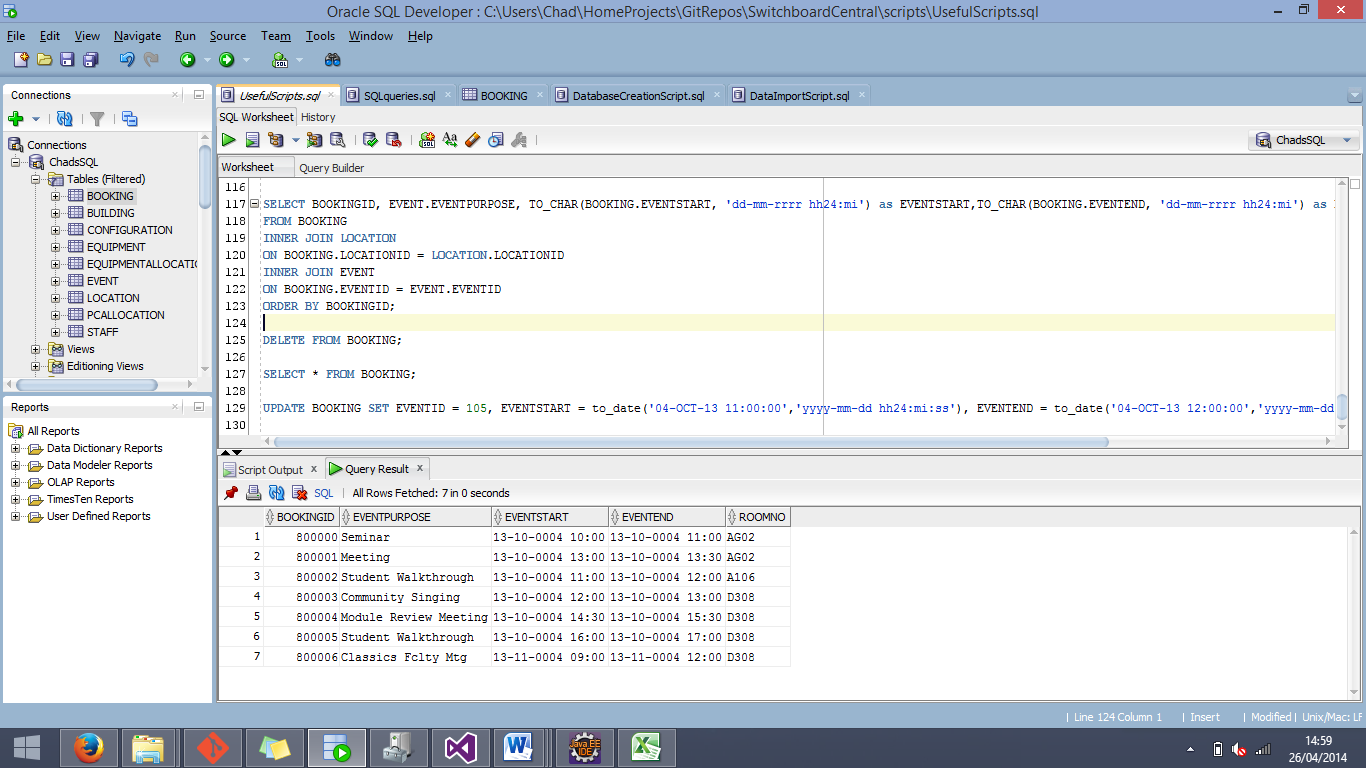
FROM BOOKING

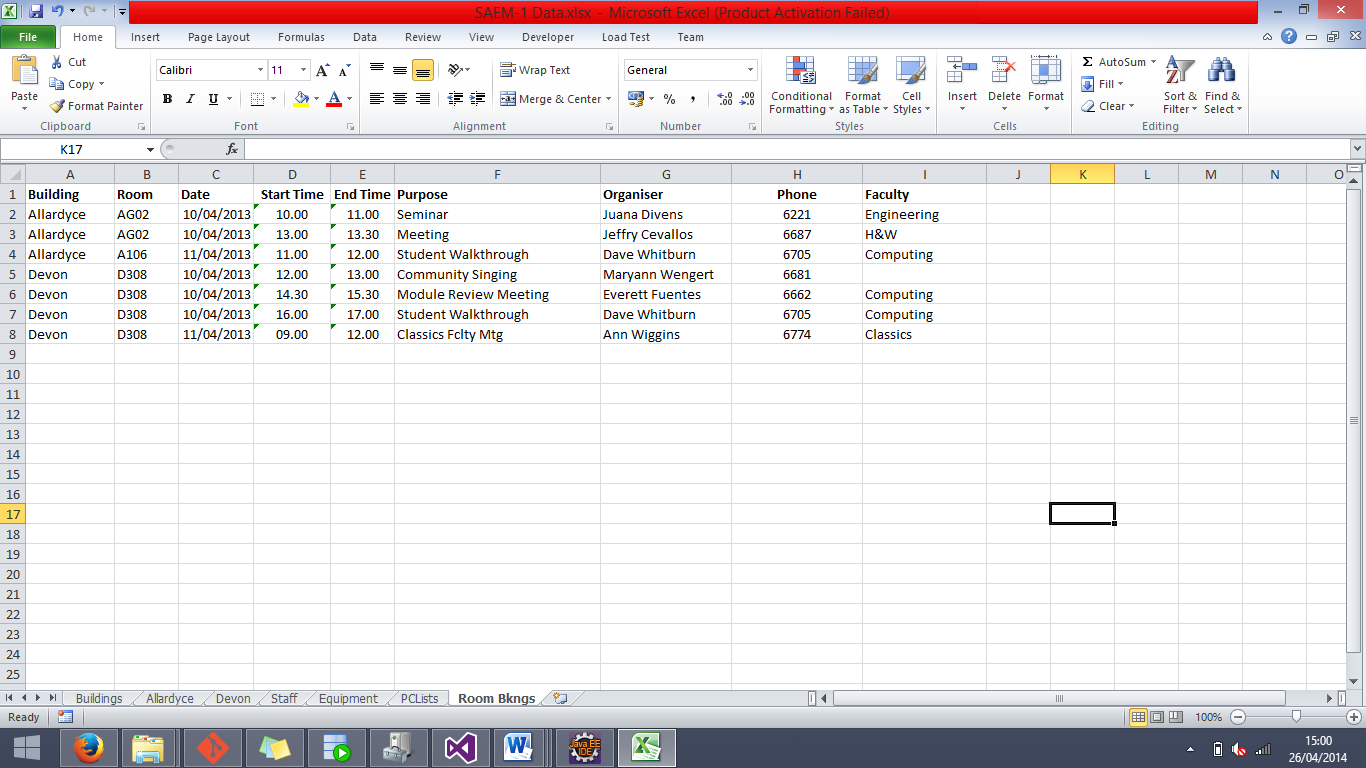
INNER JOIN LOCATION

ON BOOKING.LOCATIONID = LOCATION.LOCATIONID

INNER JOIN EVENT

ON BOOKING.EVENTID = EVENT.EVENTID;

From this script, we are able to see the ROOMNO, EVENTSTART, EVENTEND, and the IDs of the event and location that each of these bookings corresponds to.



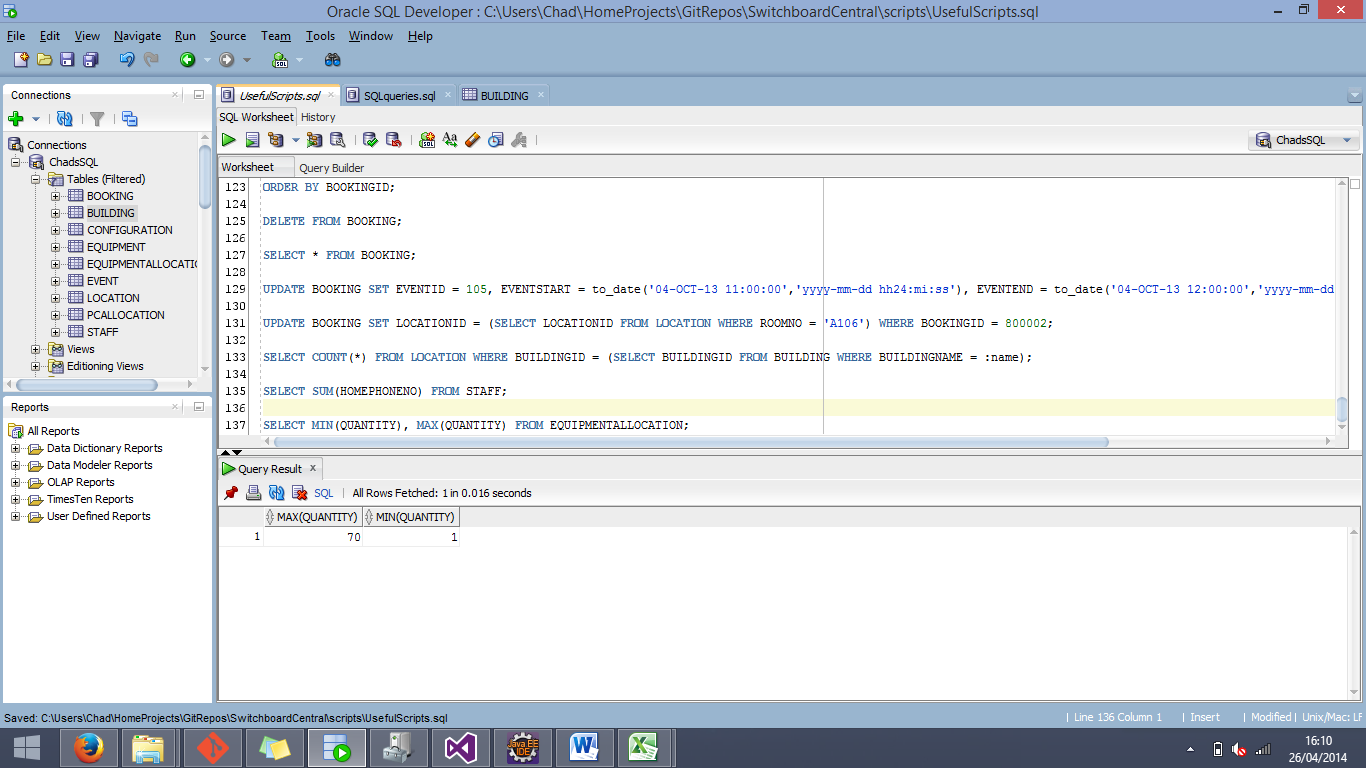
The data from the source Excel document and the data from the final database implementation is a match, proving that the loaded data is accurate in this section.

------

The next test that we scheduled to perform on the database was to further test the correctness of the data that we loaded. Our task was to check the range between the smallest amount quantity that is entered into the EQUIPMENTALLOCATION table, and the largest. To do this, we first incorporated the =MIN() and =MAX() functions inside of Excel to find the appropriate values, and then use a script inside of SQL Developer to find the corresponding values in the database.

SQL Script:

SELECT MIN(QUANTITY), MAX(QUANTITY) FROM EQUIPMENTALLOCATION;



|  |  |  |  |
| --- | --- | --- | --- |
| Source | Minimum Value | Maximum Value | Test Result |
| Source Data (Excel) | 1 | 70 | Matching Data |
| Database | 1 | 70 | Matching Data |

While admittedly this test is by no means completely conclusive as to whether or not the data inside the EQUIPMENTALLOCATION table is completely accurate, it does give at least some measure of assurance.

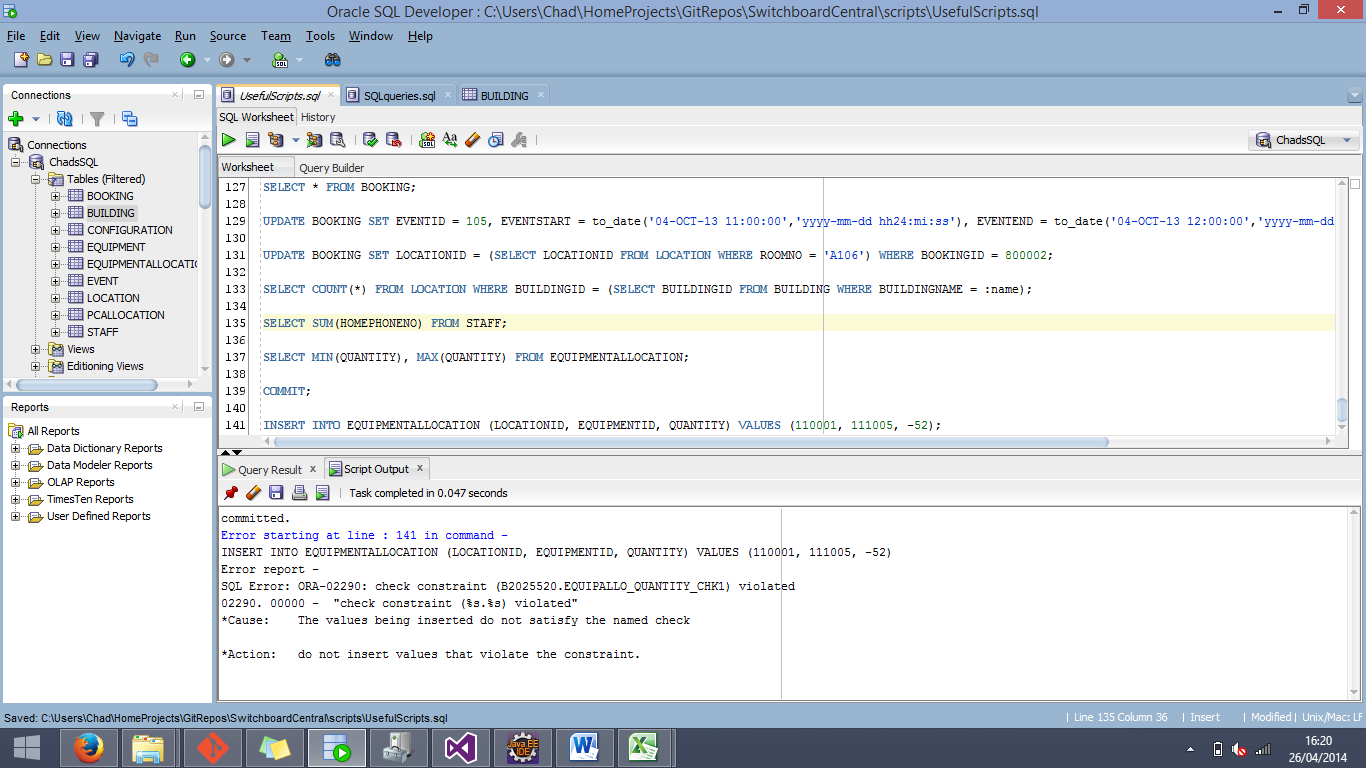
------

The next test that we performed on the database was one that was to make sure that any of the given CHECK constraints would perform their duty. For this test we chose the Quantity attribute of the EQUIPMENTALLOCATION entity, and attempting to insert an entry with a negative value in the given field. The expected result of this test is for the insert statement to fail due to a violated constraint.

SQL Script:

INSERT INTO EQUIPMENTALLOCATION (LOCATIONID, EQUIPMENTID, QUANTITY) VALUES (110001, 111005, -52);

The result of trying to apply the above script resulted in the following:



From this error, we can now confirm that the CHECK constraints associated with the Quantity field to prevent it from falling below 0 is effective.

The next series of tests involve performing SUM functions on both the Excel source document and the database. These tests will further enforce the assumption that we have loaded the data correctly. The test data will be shown in a series of tables below.

SUM Test 1: Staff Work Phone Numbers

|  |  |  |
| --- | --- | --- |
| Data source | WorkPhoneSUM | Test Result |
| Source File (Excel) | 118277 | Matching Data |
| Database | 118277 | Matching Data |

After comparing the SUMs of the respective columns, it can be confirmed that they are matching and that we are one step closer to proving the correctness of the loaded data.

SUM Test 2: Configuration Capacities

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Source | Horseshoe Capacity | Island Capacity | Row Capacity | Added Result | Test Result |
| Source File (Excel) | 80 | 718 | 1560 | 2358 | Matching Data |
| Database | 80 | 718 | 1560 | 2358 | Matching Data |

After comparing the SUMs of the respective columns, and the sum of those results, it can be confirmed that they are matching and that we are one step closer to proving the correctness of the loaded data.

SUM Test 3: Lengths and Widths of Locations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Source | Length | Width | Added Result | Test Result |
| Source File (Excel) | 1578.5 | 911.5 | 2490 | Matching Data |
| Database | 1578.5 | 911.5 | 2490 | Matching Data |

After comparing the SUMs of the respective columns, and the sum of those results, it can be confirmed that they are matching and that we are one step closer to proving the correctness of the loaded data.

------

The next test will once again call for us to manually check the source data that we were given with the data that is currently in the database. The purpose of this test is to check and see if the Primary Key of the STAFF entity has been correctly loaded into the EVENT entity in the form of an organiser reference.

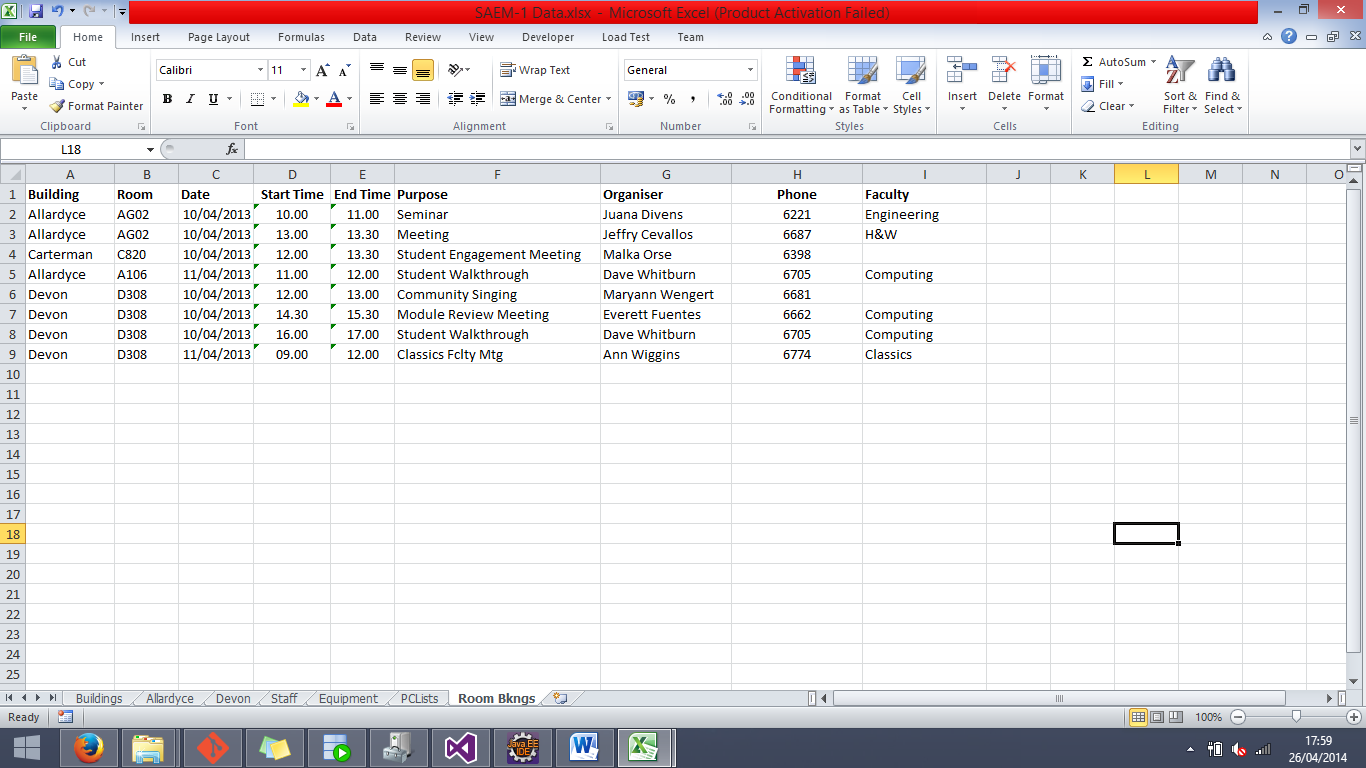
SQL Script:

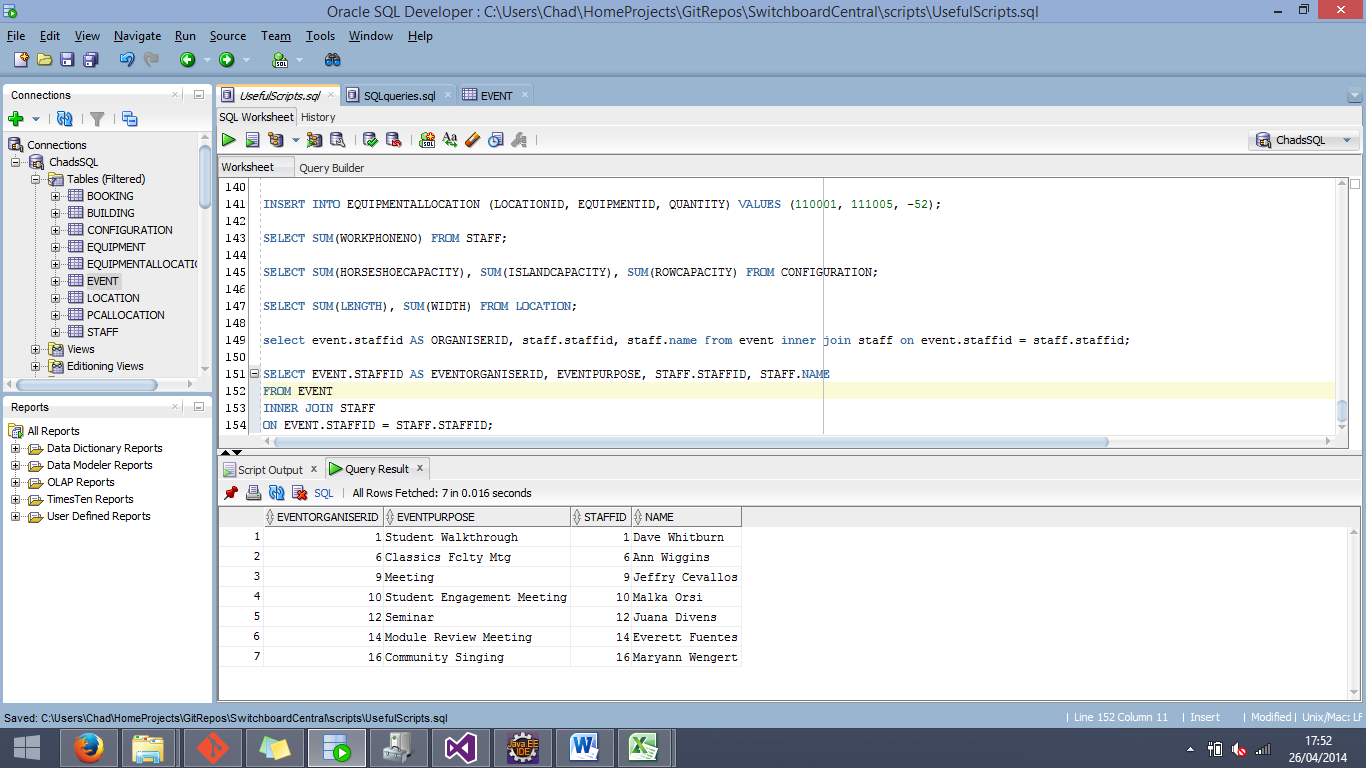
SELECT EVENT.STAFFID AS ORGANISERID, STAFF.STAFFID, STAFF.NAME

FROM EVENT

INNER JOIN STAFF

ON EVENT.STAFFID = STAFF.STAFFID;

From this SQL script we are able to see the version of StaffID that is listed in the EVENT entity, and compare it with the StaffID in the STAFF entity, along with the name of the associated staff. From this result we will be able to compare it with the source data in the Excel document to determine the accuracy of the loaded data.



As can be seen in the