FIT5195

Major group project

Dishi Jain-30759307
Sarthak Sareen - 30761182

MONASH University
Information Technology

GROUP ASSIGNMENT COVER SHEET

Student ID Number	Surname	Given Names
30761182	Sareen	**Sarthak**
30759307	Jain	**Dishi**

***** Please include the names of all other group members.

Unit name and code	FIT 5195 Bussiness intelligence and data warehousing S1 2020	
Title of assignment	FIT5195 Major Assignment - Sem 1/2020	
Lecturer/tutor	Lecturer - Agnes Haryanto tutor – Farah Kabir	
Tutorial day and time	Tuesday 6-8 pm	**Campus Caulfield**
Is this an authorised group assignment?	☒ **Yes** ☐ **No**	
Has any part of this assignment been previously submitted as part of another unit/course?	☐ **Yes** ☒ **No**	
Due Date 16/06/2020		**Date submitted 16/06/2020**

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date) .... Signature of lecturer/tutor ....

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  i. provide to another member of faculty and any external marker; and/or
  ii. submit it to a text matching software; and/or
  iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature .... *Date*……………………………………

* delete (iii) if not applicable

Signature _____Sarthak Sareen_______________Date:______16/06/2020_______________

Signature _______________Dishi jain_________________Date:____16/06/2020____

Signature ____________________Date:___________________ Signature _________________________________Date:_______

Updated: 17 Jun 2014

MONASH University
Information Technology

Signature ____________________Date:______________________ Signature _________________________________Date:_______

Privacy Statement
The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au

MONASH University
Information Technology

Contribution Declaration Form

(to be completed by all team members)

Please fill in the form with the contribution from each student towards the assignment.

1 NAME AND CONTRIBUTION DETAILS

Student ID	Student Name	Contribution Percentage
30761182	SARTHAK SAREEN	50 %
30759307	DISHI JAIN	50 %

2 DECLARATION

We declare that:

- The information we have supplied in or with this form is complete and correct.
- We understand that the information we have provided in this form will be used for individual assessment of the assignment.

3 SIGNATURE

Signatures

Sarthak sareen

Dishijain

Day Month

Year

Date

16/06/20

Updated: 17 Jun 2014

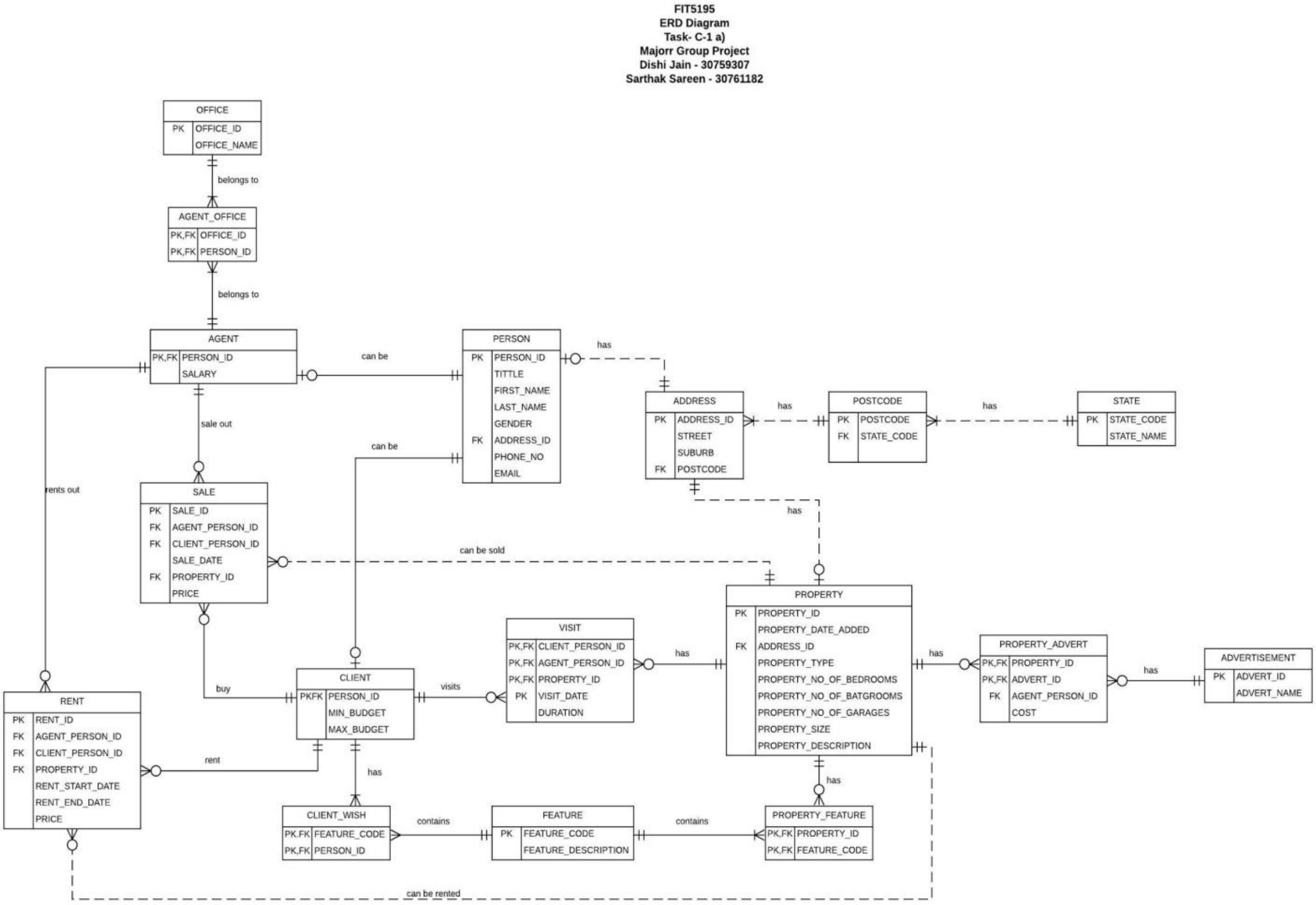
DETAILS OF ORACLE ACCOUNT -
Username - S30759307
Password- student

Contribution-
ERD Design 50% (Dishi) and 50% (Sarthak)
Data Cleaning 50% (Dishi) and 50% (Sarthak)
Star schema implementation 50% (Dishi) and 50% (Sarthak)
Report 50% (Dishi) and 50% (Sarthak)
OLAP query 50% (Dishi) and 50% (Sarthak)

C. Tasks

Task 1 out of 4

a) The final ERD diagram looks like –



Task 1 out of 4
b) DATA CLEANING

Data cleaning has been carried out by exploring the tables in the original operational database.
For exploring each table the following queries have been used -

1. SELECT column_name, count(*)
   FROM monre.table
   GROUP BY column_name
   HAVING COUNT(*) > 1;

2. select * from monre.table where end_date < start_date;

3. select * from monre.table where max_budget < min_budget;

4. Select * from monre.table where gender not in (‘Male’,’Female’);

5. Select * from monre.table where title not in (‘Ms’,’Mrs’,’Mr’,’Dr’);

6. select * from monre.table where column_name like 'null';

7. select * from monre.table where column_name is null;

8. select column_name_foreign_attr from monre.foreign_table where column_name_foreign_attr not in (select column_name_foreign_attr from monre.original_table);

9. select * from monre.table order by column_name desc;

10. select * from monre.table order by column_name;

The queries have been used to explore each and every table in the operational database. Respective exploration,correction and screenshots of data before cleaning and screenshots of data after cleaning is provided below.

Table - MonRE.address

1. a) Exploring

To check if an address_id is repeated twice SQL query used is -

```
SELECT address_id , count(*)
        FROM monre.address
        GROUP BY address_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each address_id appears only once in the table.

2. a) Exploring

To check if a street is repeated twice SQL query used is -

```
SELECT street, suburb,postcode, count(*)
        FROM monre.address
        GROUP BY street,suburb,postcode
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found for 39 rows which hence have the same street name,suburb and postcode but from the above exploration we can confirm that they have different address_id. As they have different address_id we can say that they are different addresses and hence no correction is done.

3. a) Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.address where address_id like 'null';
select * from monre.address where address_id is null;

select * from monre.address where street like 'null';
select * from monre.address where street is null;
select * from monre.address where suburb like 'null';
select * from monre.address where suburb is null;


select * from monre.address where postcode like 'null';
select * from monre.address where postcode is null;
```

3. b) Result

No null values found.

4. a) Exploration

To check if the postcode written in the monre.address table rows actually exist in the monre.postcode table, SQL query used is -

```
select postcode from monre.address where postcode not in (select postcode from monre.postcode);
```

4. b)Result

No output seen. Hence we can say that all the postcodes present in the monre.address table are valid and are present in the monre.postcode table.

5. a)Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.address order by address_id desc;
select * from monre.address order by address_id ;

select * from monre.address order by street desc;
select * from monre.address order by street;

select * from monre.address order by suburb desc;
select * from monre.address order by suburb;

select * from monre.address order by postcode desc;
select * from monre.address order by postcode ;
```

5. b) Result

No error seen

Table - MonRE.agent

1. a) Exploring

To check if an person_id is repeated twice SQL query used is -

```
SELECT person_id , count(*)
        FROM monre.agent
        GROUP BY person_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each person_id appears only once in the table.

2. a) Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.agent  where person_id like 'null';
select * from monre.agent where person_id is null;

select * from monre.agent where salary like 'null';
select * from monre.agent where salary is null;
```

2. b)Result

No null values found.

3. a) Exploration

To check if the person_id written in the monre.agent table rows actually exist in the monre.person table, SQL query used is -

select person_id from monre.agent where person_id not in (select person_id from monre.person);

3. b)Result

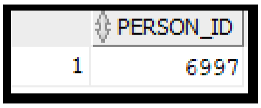
Output seen is a person_id = 6997. This tells us that monre.agent has a person_id entry as 6997 which actually doesn't exist in the monre.person table's person_id column. Hence it is an illegal entry in the monre.agent table.

3. c)Removing Error

A copy of the table monre.agent is created and then the error is removed from it.
For copying the SQL query used is - create table clean_agent as select * from monre.agent
The error is removed using the SQL - delete from clean_agent where person_id not in (select person_id from monre.person);

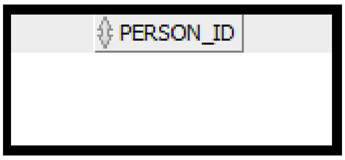
3. d)Screenshots of data before and after cleaning

Before -



	PERSON_ID
1	6997

After -
Using query select person_id from clean_agent where person_id not in (select person_id from monre.person);



	PERSON_ID

Hence no such person id exists in new clean_agent which is not present in monre.person. Hence all legal entries only.

4. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

select * from monre.agent order by person_id desc;
select * from monre.agent order by person_id ;

select * from monre.agent order by salary desc;
select * from monre.agent order by salary;

4. b)Result

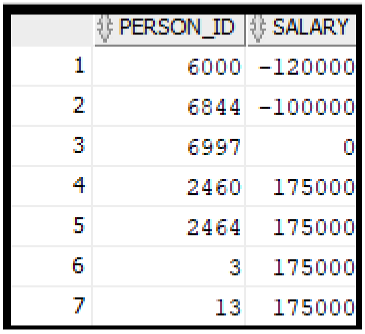
Using the query - select * from monre.agent order by salary; it was seen that there are some salary values that are 0 or negative in number. To get such salary values, SQL query used is - select * from MonRE.agent where salary <= 0. This seems to be an error as the salary of an agent cannot be 0 or a negative number.

4. c)Removing Error

The error is removed using the SQL - delete from clean_agent where salary <= 0;

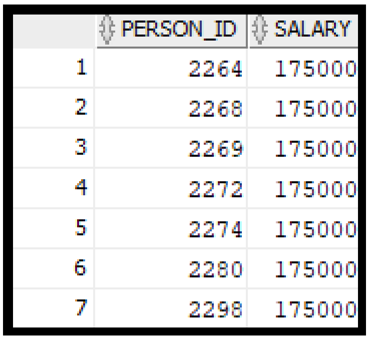
4. d)Screenshots of data before and after cleaning

Before -



	PERSON_ID	SALARY
1	6000	-120000
2	6844	-100000
3	6997	0
4	2460	175000
5	2464	175000
6	3	175000
7	13	175000

After -



	PERSON_ID	SALARY
1	2264	175000
2	2268	175000
3	2269	175000
4	2272	175000
5	2274	175000
6	2280	175000
7	2298	175000

Hence no 0 or negative salaries present.

Table - MonRE.agent_office

1. a) Exploring

To check if a person_id is repeated twice SQL query used is -

```
SELECT person_id , count(*)
        FROM monre.agent_office
        GROUP BY person_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each person_id appears only once in the table.

2. a) Exploration

To check if an office_id is repeated twice SQL query used is -

```
SELECT office_id , count(*)
        FROM monre.agent_office
        GROUP BY office_id
        HAVING COUNT(*) > 1;
```

2. b) Result

No repetition found. Hence each office_id appears only once in the table

3. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.agent_office  where person_id like 'null';
select * from monre.agent_office where person_id is null;

select * from monre.agent_office where office_id like 'null';
select * from monre.agent_office where office_id is null;
```

3. b)Result

No null values found.

4. a) Exploration

To check if the person_id written in the monre.agent_office table rows actually exist in the monre.person table, SQL query used is -

```
select person_id from monre.agent_office where person_id not in (select person_id from monre.person);
```

4. b)Result

Output seen is a person_id = 6997. This tells us that monre.agent_office has a person_id entry as 6997 which actually doesn’t exist in the monre.person table’s person_id column. Hence it is an illegal entry in the monre.agent_office table.

4. c)Removing Error

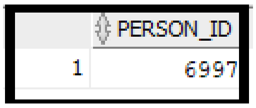
A copy of the table monre.agent_office is created and then the error is removed from it.

For copying the SQL query used is - create table clean_agent_office as select * from monre.agent_office

The error is removed using the SQL - delete from clean_agent_office where person_id not in (select person_id from monre.person);

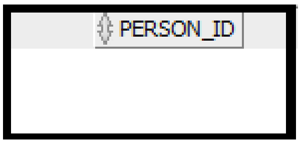
4. d)Screenshots of data before and after cleaning

Before -



```
   PERSON_ID
1       6997
```

After -



```
   PERSON_ID
```

No output hence now all valid person id present in clean_agent_office.

5. a) Exploration

To check if the office_id written in the monre.agent_office table rows actually exist in the monre.office table, SQL query used is -

select office_id from monre.agent_office where office_id not in (select office_id from monre.office);

5. b)Result

No output seen. Hence we can say that all the office_id present in the monre.agent_office table are valid and are present in the monre.office table.

6. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

select * from monre.agent_office order by person_id desc;
select * from monre.agent_office order by person_id ;

select * from monre.agent_office order by office_id desc;
select * from monre.agent_office order by office_id ;

6. b)Result

No errors seen

Table - MonRE.client

1. a) Exploring

To check if a person_id is repeated twice SQL query used is -

```
SELECT person_id , count(*)
        FROM monre.client
        GROUP BY person_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each person_id appears only once in the table.

2. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

select * from monre.client where person_id like 'null';
select * from monre.client where person_id is null;

select * from monre.client where max_budget like 'null';
select * from monre.client where max_budget is null;

select * from monre.client where min_budget like 'null';
select * from monre.client where min_budget is null;

2. b)Result

No null values found.

3. a) Exploration

To check if the person_id written in the monre.client table rows actually exist in the monre.person table, SQL query used is -

select person_id from monre.client where person_id not in (select person_id from monre.person);

3. b)Result

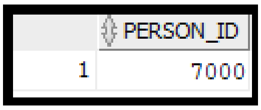
Output seen is a person_id = 7000. This tells us that monre.client has a person_id entry as 7000 which actually doesn't exist in the monre.person table's person_id column. Hence it is an illegal entry in the monre.client table.

3. c)Removing Error

A copy of the table monre.client is created and then the error is removed from it.
For copying the SQL query used is - create table clean_client as select * from monre.client
The error is removed using the SQL - delete from clean_client where person_id not in (select person_id from monre.person);

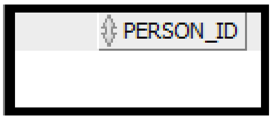
3. d)Screenshots of data before and after cleaning

Before -



	PERSON_ID
1	7000

After -



	PERSON_ID

4. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

select * from monre.client order by person_id desc;
select * from monre.client order by person_id ;

select * from monre.client order by max_budget desc;
select * from monre.client order by max_budget ;

select * from monre.client order by min_budget desc;
select * from monre.client order by min_budget ;

4. b)Result

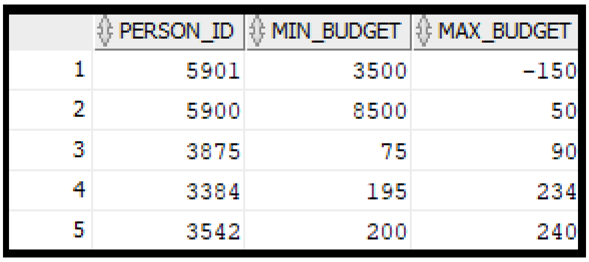
From the SQL query - select * from monre.client order by max_budget ; it was seen that a value in max_budget has a negative value of -150. This cannot be true as the budget cannot be a negative number.

4. c)Removing Error

The error is removed using the SQL - delete from clean_client where max_budget < 0 or min_budget < 0;

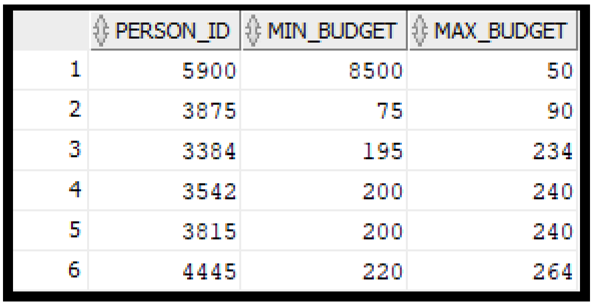
4. d)Screenshots of data before and after cleaning

Before -



	PERSON_ID	MIN_BUDGET	MAX_BUDGET
1	5901	3500	-150
2	5900	8500	50
3	3875	75	90
4	3384	195	234
5	3542	200	240

After -



	PERSON_ID	MIN_BUDGET	MAX_BUDGET
1	5900	8500	50
2	3875	75	90
3	3384	195	234
4	3542	200	240
5	3815	200	240
6	4445	220	264

5. a) Exploration

For comparison of max_budget and min_budget, the SQL query used is -

select * from monre.client where max_budget < min_budget;

5. b)Result

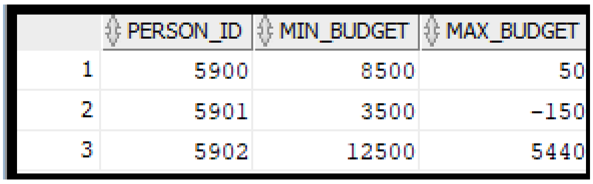
From the SQL query it was found that there are 3 entries in the monre.client table that have max_budget less than the min_budget.

5. c)Removing Error

The error is removed using the SQL - delete from clean_client where max_budget < min_budget ;

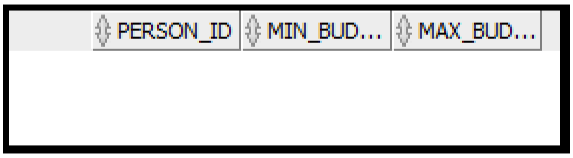
5. d)Screenshots of data before and after cleaning

Before -



	PERSON_ID	MIN_BUDGET	MAX_BUDGET
1	5900	8500	50
2	5901	3500	-150
3	5902	12500	5440

After -



	PERSON_ID	MIN_BUD...	MAX_BUD...

Hence using query - select * from clean_client where max_budget < min_budget; we see no output

Table - MonRE.client_wish

1. a) Exploring

To check if a feature_code is repeated twice SQL query used is -

```
SELECT feature_code , count(*)
        FROM monre.client_wish
        GROUP BY feature_code
        HAVING COUNT(*) > 1;
```

1. b) Result

Repetition found. Which is valid as a particular feature can be a wish for multiple clients.

2. a) Exploring

To check if a person_id is repeated twice SQL query used is -

```
SELECT person_id , count(*)
        FROM monre.client_wish
        GROUP BY person_id
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. Which is valid as a particular client(person) can have multiple features as his/her wish.

3. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.client_wish  where person_id like 'null';
select * from monre.client_wish where person_id is null;

select * from monre.client_wish where feature_code like 'null';
select * from monre.client_wish where feature_code is null;
```

3. b)Result

No null values found.

4. a) Exploration

To check if the person_id written in the monre.client_wish table rows actually exist in the monre.person table, SQL query used is -

select person_id from monre.client_wish where person_id not in (select person_id from monre.person);

4. b)Result

No output seen. Hence we can say that all the person_id present in the monre.client_wish table are valid and are present in the monre.office table.

5. a) Exploration

To check if the feature_code written in the monre.client_wish table rows actually exist in the monre.feature table, SQL query used is -

select feature_code from monre.client_wish where feature_code not in (select feature_code from monre.feature);

5. b)Result

No output seen. Hence we can say that all the feature_code present in the monre.client_wish table are valid and are present in the monre.feature table.

6. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.client_wish order by person_id desc;
select * from monre.client_wish order by person_id ;

select * from monre.client_wish order by feature_code desc;
select * from monre.client_wish order by feature_code
```

6. b)Result

No errors seen.

Table - MonRE.feature

1. a) Exploring

To check if a feature_code is repeated twice SQL query used is -

```
SELECT feature_code , count(*)
        FROM monre.feature
        GROUP BY feature_code
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each feature_code appears only once in the table.

2. a) Exploring

To check if a feature_description is repeated twice SQL query used is -

```
SELECT feature_description , count(*)
        FROM monre.feature
        GROUP BY feature_description
        HAVING COUNT(*) > 1;
```

2. b) Result

No repetition found. Hence each feature_description appears only once in the table. .

3. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.feature  where feature_description like 'null';
select * from monre.feature where feature_description is null;

select * from monre.feature where feature_code like 'null';
```

select * from monre.feature where feature_code is null;

3. b)Result

No null values found.

4. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.feature order by feature_description desc;
select * from monre.feature order by feature_description ;

select * from monre.feature order by feature_code desc;
select * from monre.feature order by feature_code
```

4. b)Result

From the query - select * from monre.feature order by feature_code desc; it is seen that a feature_description has value of ‘Fake Feature’. This is not a valid feature and hence is taken as an error. The feature_code for the corresponding feature_description is 726.

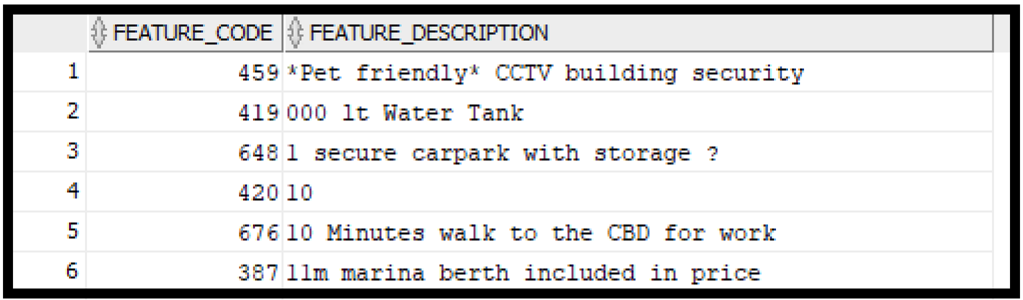
From the query - select * from monre.feature order by feature_description ;
Another error that was seen is that feature_description has a value of 10. This is not a valid feature and hence is taken as an error. The feature_code for the corresponding feature_description is 420.

4. c) Removing Error

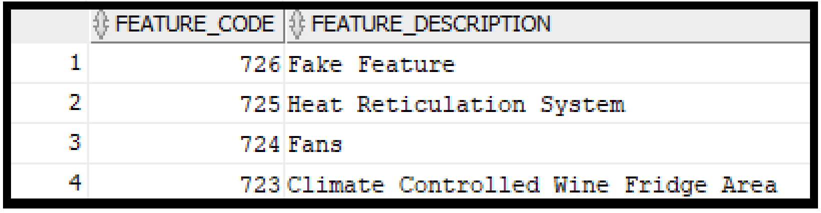
A copy of the table monre.feature is created and then the error is removed from it.
For copying the SQL query used is - create table clean_feature as select * from monre.feature;
The error is removed using the SQL - delete from clean_feature where feature_code = 726 or feature_code = 420;

4. d)Screenshots of data before and after cleaning

Before -

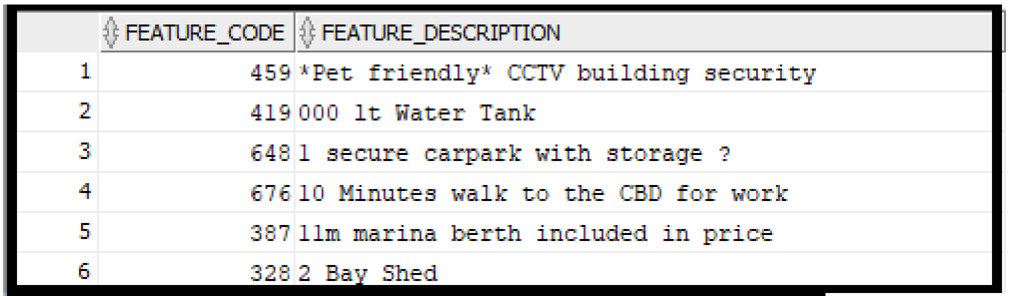


	FEATURE_CODE	FEATURE_DESCRIPTION
1	459	*Pet friendly* CCTV building security
2	419	000 lt Water Tank
3	648	1 secure carpark with storage ?
4	420	10
5	676	10 Minutes walk to the CBD for work
6	387	11m marina berth included in price

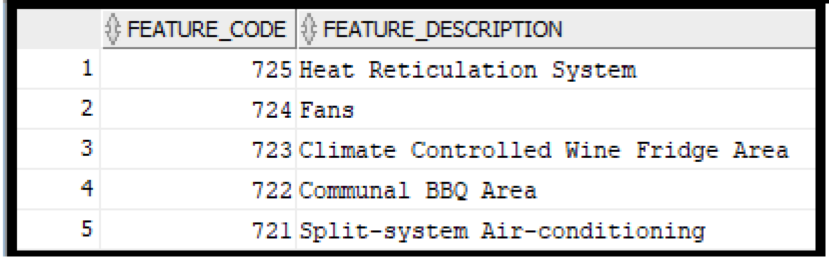


	FEATURE_CODE	FEATURE_DESCRIPTION
1	726	Fake Feature
2	725	Heat Reticulation System
3	724	Fans
4	723	Climate Controlled Wine Fridge Area

After -



	FEATURE_CODE	FEATURE_DESCRIPTION
1	459	*Pet friendly* CCTV building security
2	419	000 lt Water Tank
3	648	1 secure carpark with storage ?
4	676	10 Minutes walk to the CBD for work
5	387	11m marina berth included in price
6	328	2 Bay Shed



	FEATURE_CODE	FEATURE_DESCRIPTION
1	725	Heat Reticulation System
2	724	Fans
3	723	Climate Controlled Wine Fridge Area
4	722	Communal BBQ Area
5	721	Split-system Air-conditioning

Table - MonRE.office

1. a) Exploring

To check if an office_id is repeated twice SQL query used is -

```
SELECT office_id , count(*)
        FROM monre.office
        GROUP BY office_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each office_id appears only once in the table.

2. a) Exploring

To check if an office_name is repeated twice SQL query used is -

```
SELECT office_name , count(*)
        FROM monre.office
        GROUP BY office_name
        HAVING COUNT(*) > 1;
```

2. b) Result

No repetition found. Hence each office_id appears only once in the table.

3. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.office  where office_id like 'null';
select * from monre.office where office_id is null;

select * from monre.office where office_name like 'null';
select * from monre.office where office_name is null;
```

3. b)Result

No null values found.

4. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.office order by office_id desc;
select * from monre.office order by office_id ;

select * from monre.office order by office_name desc;
select * from monre.office order by office_name
```

4. b)Result

No errors seen.

Table - MonRE.person

1. a) Exploring

To check if a person_id is repeated twice SQL query used is -

```
SELECT person_id , count(*)
        FROM monre.person
        GROUP BY person_id
        HAVING COUNT(*) > 1;
```

1. b) Result

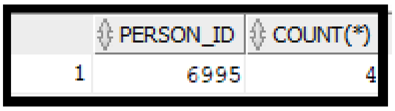
Repetition found. The person_id with a value of 6995 is repeated 4 times in the table with duplicate values. This is an error and needs to be removed.

1. c) Removing Error

A copy of the table monre.person is created using distinct which will remove the error.
For copying the SQL query used is - create table clean_person as select distinct * from monre.person;

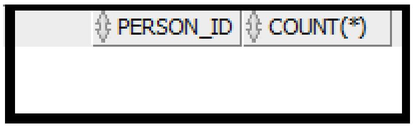
1. d)Screenshots of data before and after cleaning

Before -



	PERSON_ID	COUNT(*)
1	6995	4

After -



	PERSON_ID	COUNT(*)

Hence now no repetition seen

2. a) Exploring

To check if an address_id is repeated twice SQL query used is -

```
SELECT address_id , count(*)
        FROM monre.person
        GROUP BY address_id
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. The address_id with a value of 12650 is repeated 4 times in the table with duplicate values. This is an error and needs to be removed. This error corresponds to the error above where person_id = 6995 was repeated 4 times. Hence by removing the above error this error will also be removed.

3. a) Exploring

To check if the gender lies within Male and Female values only,SQL query used is -

select * from monre.person where gender not in ('Male','Female');

3. b) Result

No errors found

4. a) Exploring

To check if the titles of the names lies within Mr,Mrs,Ms and Dr,,SQL query used is -

select * from monre.person where title not in ('Ms','Mrs','Mr','Dr');

4. b) Result

One row entry found that has a title as null. Which can be valid as a person might not have a title to his/her name.

5. a)Exploring

To check if the address_id written in the monre.person table rows actually exist in the monre.address table, SQL query used is -

select address_id from monre.person where address_id not in (select address_id from monre.address);

5. b)Result

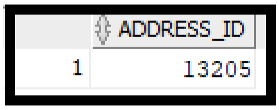
Output seen is an address_id = 13205. This tells us that monre.person has an address_id entry as 13205 which actually doesn't exist in the monre.address table's address_id column. Hence it is an illegal entry in the monre.person table. The corresponding value of person_id is 7001

5. c)Removing Error

The error is removed using the SQL - delete from clean_person where address_id not in (select address_id from monre.address);

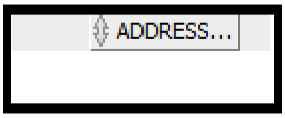
5. d)Screenshots of data before and after cleaning

Before -



```
     ADDRESS_ID
1         13205
```

After -



```
     ADDRESS...
```

Hence by using select address_id from clean_person where address_id not in (select address_id from monre.address); we now see no errors

6. a)Exploration

To check for null values in each column and values that are filled as 'null' , SQL query used is -

select * from monre.person where person_id like 'null';
select * from monre.person where person_id is null;

select * from monre.person where title like 'null';
select * from monre.person where title is null;

select * from monre.person where first_name like 'null';
select * from monre.person where first_name is null;

select * from monre.person where last_name like 'null';
select * from monre.person where last_name is null;

select * from monre.person where gender like 'null';
select * from monre.person where gender is null;

select * from monre.person where address_id like 'null';
select * from monre.person where address_id is null;

select * from monre.person where phone_no like 'null';
select * from monre.person where phone_no is null;

select * from monre.person where email like 'null';
select * from monre.person where email is null;

6. b)Result

No null values found.

7. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

select * from monre.person order by person_id desc;
select * from monre.person order by person_id ;

```
select * from monre.person  order by title desc;
select * from monre.person order by title;

select * from monre.person order by first_name desc;
select * from monre.person order by first_name ;

select * from monre.person  order by last_name desc;
select * from monre.person order by last_name;

select * from monre.person order by gender desc;
select * from monre.person order by gender ;

select * from monre.person  order by address_id desc;
select * from monre.person order by address_id;

select * from monre.person order by phone_no desc;
select * from monre.person order by phone_no ;

select * from monre.person  order by email desc;
select * from monre.person order by email;
```

7. b)Result

No errors seen.

Table - MonRE.postcode

1. a) Exploring

To check if a postcode is repeated twice SQL query used is -

```
SELECT postcode , count(*)
        FROM monre.postcode
        GROUP BY postcode
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each postcode is present in only one state.

2. a) Exploring

To check if an state_code is repeated twice SQL query used is -

```
SELECT state_code , count(*)
        FROM monre.postcode
        GROUP BY state_code
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. This is valid, as a state can have multiple postcodes of locations inside it.

3. a)Exploring

To check if the state_code written in the monre.postcode table rows actually exist in the monre.state table, SQL query used is -

```
select state_code from monre.postcode where state_code not in (select state_code from monre.state);
```

3. b)Result

No output seen. Hence valid. So only valid state_codes are present in each row of the monre.postcode table.

4. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.postcode  where postcode like 'null';
select * from monre.postcode where postcode  is null;

select * from monre.postcode  where state_code like 'null';
select * from monre.postcode where state_code is null;
```

4. b)Result

No null values found.

5. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.postcode order by postcode desc;
select * from monre.postcode order by postcode ;

select * from monre.postcode  order by state_code desc;
select * from monre.postcode order by state_code;
```

5. b)Result

No errors seen.

Table - MonRE.property

1. a) Exploring

To check if a property_id is repeated twice SQL query used is -

```
SELECT property_id , count(*)
        FROM monre.property
        GROUP BY property_id
        HAVING COUNT(*) > 1;
```

1. b) Result

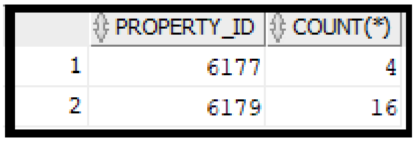
Repetition found. The property_id with a value of 6177 and 6179 is repeated 4 times and 16 times respectively in the table with duplicate values. This is an error and needs to be removed.

1. c) Removing Error

A copy of the table monre.property is created using distinct which will remove the error.
For copying the SQL query used is - create table clean_property as select distinct * from monre.property;

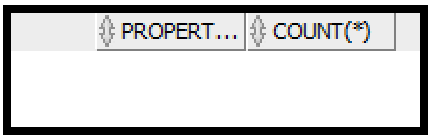
1. d)Screenshots of data before and after cleaning

Before -



	PROPERTY_ID	COUNT(*)
1	6177	4
2	6179	16

After -



PROPERT...	COUNT(*)

Hence now no repetitions seen

2. a) Exploring

To check if an address_id is repeated twice SQL query used is -

```
SELECT address_id , count(*)
        FROM monre.property
        GROUP BY address_id
        HAVING COUNT(*) > 1;
```

2. b) Result

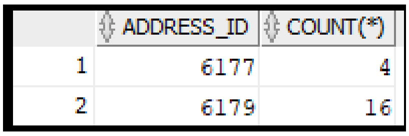
Repetition found. The address_id with a value of 6177 and 6179 is repeated 4 times and 16 times respectively in the table with duplicate values. This is an error and needs to be removed. While removing the above error where property_id was repeated, this error also gets removed.

2. c) Removing Error

A copy of the table monre.property is created using distinct which will remove the error.
For copying the SQL query used is - create table clean_property as select distinct * from monre.property;
This has already been done, hence there is no need to repeat this again.

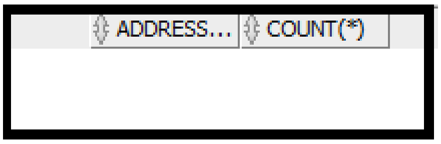
2. d)Screenshots of data before and after cleaning

Before -



	ADDRESS_ID	COUNT(*)
1	6177	4
2	6179	16

After -



ADDRESS...	COUNT(*)

3. a)Exploring

To check if the address_id written in the monre.property table rows actually exist in the monre.address table, SQL query used is -

```
select address_id from monre.property where address_id not in (select address_id from monre.address);
```

3. b)Result

No Output seen. Hence all address_ids that appear in the monre.property table's address_id column are valid and are present in the monre.address table.

4. a)Exploration

To check for null values in each column and values that are filled as 'null' , SQL query used is -

```
select * from monre.property  where property_id like 'null';
select * from monre.property where person_id  is null;

select * from monre.property  where property_date_added like 'null';
select * from monre.property where property_date_added is null;

select * from monre.property  where ADDRESS_ID like 'null';
select * from monre.property where ADDRESS_ID is null;

select * from monre.property  where PROPERTY_TYPE like 'null';
select * from monre.property where PROPERTY_TYPE  is null;

select * from monre.property  where PROPERTY_NO_OF_BEDROOMS like 'null';
select * from monre.property where PROPERTY_NO_OF_BEDROOMS is null;

select * from monre.property  where PROPERTY_NO_OF_BATHROOMS like 'null';
select * from monre.property where PROPERTY_NO_OF_BATHROOMS is null;

select * from monre.property  where PROPERTY_NO_OF_GARAGES like 'null';
select * from monre.property where PROPERTY_NO_OF_GARAGES is null;

select * from monre.property  where PROPERTY_SIZE like 'null';
select * from monre.property where PROPERTY_SIZE is null;

select * from monre.property  where PROPERTY_DESCRIPTION like 'null';
select * from monre.property where PROPERTY_DESCRIPTION is null;
```

4. b)Result

No unusual null values found.

5. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.property order by property_id desc;
select * from monre.property order by property_id ;
select * from monre.property  order by property_date_added desc;
select * from monre.property order by property_date_added ;

select * from monre.property order by ADDRESS_ID desc;
select * from monre.property order by ADDRESS_ID ;

select * from monre.property  order by PROPERTY_TYPE desc;
select * from monre.property order by PROPERTY_TYPE ;

select * from monre.property order by PROPERTY_NO_OF_BEDROOMS desc;
select * from monre.property order by PROPERTY_NO_OF_BEDROOMS ;

select * from monre.property  order by PROPERTY_NO_OF_BATHROOMS desc;
select * from monre.property order by PROPERTY_NO_OF_BATHROOMS ;

select * from monre.property order by PROPERTY_NO_OF_GARAGES desc;
select * from monre.property order by PROPERTY_NO_OF_GARAGES ;

select * from monre.property  order by PROPERTY_SIZE desc;
select * from monre.property order by PROPERTY_SIZE ;

select * from monre.property  order by PROPERTY_DESCRIPTION desc;
select * from monre.property order by PROPERTY_DESCRIPTION ;
```

5. b)Result

No unusual errors seen.

Table - MonRE.property_advert

1. a) Exploring

To check if a property_id is repeated twice SQL query used is -

```
SELECT property_id , count(*)
        FROM monre.property_advert
        GROUP BY property_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence each property_id is present only once.

2. a) Exploring

To check if an advert_id is repeated twice SQL query used is -

```
SELECT advert_id, count(*)
        FROM monre.property_advert
        GROUP BY advert_id
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. This is valid, as a particular advert_id can be for multiple properties.

3. a) Exploring

To check if an agent_person_id is repeated twice SQL query used is -

```
SELECT agent_person_id , count(*)
        FROM monre.property_advert
        GROUP BY agent_person_id
        HAVING COUNT(*) > 1;
```

3. b) Result

Repetition found. This is valid, as a particular agent_person_id can advertise for multiple properties.

4. a)Exploring

To check if the property_id written in the monre.property_advert table rows actually exist in the monre.property table, SQL query used is -

```
select property_id from monre.property_advert where property_id not in (select property_id from monre.property);
```

4. b)Result

No output seen. Hence valid.

5. a)Exploring

To check if the advert_id written in the monre.property_advert table rows actually exist in the monre.advertisement table, SQL query used is -

```
select advert_id from monre.property_advert where advert_id not in (select advert_id from monre.advertisement);
```

5. b)Result

No output seen. Hence valid.

6. a)Exploring

To check if the agent_person_id written in the monre.property_advert table rows actually exist in the monre.agent table, SQL query used is -

```
select agent_person_id from monre.property_advert where agent_person_id not in (select person_id from monre.agent);
```

6. b)Result

No output seen. Hence valid.

7. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.property_advert  where property_id like 'null';
select * from monre.property_advert where property_id  is null;

select * from monre.property_advert  where advert_id like 'null';
select * from monre.property_advert where advert_id is null;

select * from monre.property_advert  where agent_person_id like 'null';
select * from monre.property_advert where agent_person_id is null;

select * from monre.property_advert  where cost like 'null';
select * from monre.property_advert where cost is null;
```

7. b)Result

No null values found.

8. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.property_advert order by property_id desc;
select * from monre.postproperty_advert code order by property_id ;

select * from monre.property_advert order by advert_id desc;
select * from monre.postproperty_advert code order by advert_id ;

select * from monre.property_advert  order by agent_person_id desc;
select * from monre.property_advert order by agent_person_id ;


select * from monre.property_advert order by cost desc;
select * from monre.postproperty_advert code order by cost;
```

8. b)Result

No errors seen.

Table - MonRE.property_feature

1. a) Exploring

To check if a property_id is repeated twice SQL query used is -

```
SELECT property_id , count(*)
        FROM monre.property_feature
        GROUP BY property_id
        HAVING COUNT(*) > 1;
```

1. b) Result

Repetition found. This is valid as a property can have multiple features.

2. a) Exploring

To check if a feature_code is repeated twice SQL query used is -

```
SELECT feature_code , count(*)
        FROM monre.property_feature
        GROUP BY feature_code
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. This is valid, as a particular feature_code can be for multiple properties.

3. a)Exploring

To check if the property_id written in the monre.property_feature table rows actually exist in the monre.property table, SQL query used is -

```
select property_id from monre.property_feature where property_id not in (select property_id from monre.property);
```

3. b)Result

No output seen. Hence valid.

4. a)Exploring

To check if the feature_code written in the monre.property_feature table rows actually exist in the monre.feature table, SQL query used is -

```
select feature_code from monre.property_feature where feature_code not in (select feature_code from monre.feature);
```

4. b)Result

No output seen. Hence valid.

5. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.property_feature  where property_id like 'null';
select * from monre.property_feature where property_id  is null;

select * from monre.property_feature  where feature_code like 'null';
select * from monre.property_feature where feature_code is null;
```

5. b)Result

No null values found.

6. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.property_feature order by property_id desc;
select * from monre.property_feature code order by property_id ;

select * from monre.property_feature order by feature_code desc;
select * from monre.property_feature code order by feature_code ;
```

6. b)Result

No errors seen.

Table - MonRE.rent

1. a) Exploring

To check if a rent_id is repeated twice SQL query used is -

```
SELECT rent_id , count(*)
        FROM monre.rent
        GROUP BY rent_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence all are valid.

2. a) Exploring

To check if a agent_person_id is repeated twice SQL query used is -

```
SELECT agent_person_id , count(*)
        FROM monre.rent
        GROUP BY agent_person_id
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. This is valid as an agent can be responsible for renting out multiple properties. Hence all are valid.

3. a) Exploring

To check if a client_person_id is repeated twice SQL query used is -

```
SELECT client_person_id , count(*)
        FROM monre.rent
        GROUP BY client_person_id
        HAVING COUNT(*) > 1;
```

3. b) Result

Repetition found 1637 times of ony null client_person_id. This is valid as when a property is not rented out the client_person_id will be null. Hence not an error.

4. a) Exploring

To check if a property_id is repeated twice SQL query used is -

```
SELECT property_id, count(*)
        FROM monre.rent
        GROUP BY property_id
        HAVING COUNT(*) > 1;
```

4. b) Result

Repetition found 2 times of property_id = 5741. This is valid as the property has two records of not being rented and then being rented out. Hence not an error.

5. a)Exploring

To check if the agent_person_id written in the monre.rent table rows actually exist in the monre.agent table, SQL query used is -

select agent_person_id from monre.rent where agent_person_id not in (select person_id from monre.agent);

5. b)Result

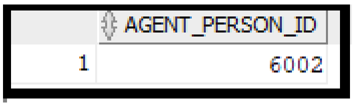
Output seen is an agent_person_id = 6002. This tells us that monre.rent has a agent_person_id entry as 6002 which actually doesn't exist in the monre.agent table's person_id column. Hence it is an illegal entry in the monre.rent table.

5. c)Removing Error

A copy of the table monre.agerent nt is created and then the error is removed from it.
For copying the SQL query used is - create table clean_rent as select * from monre.rent;
The error is removed using the SQL - delete from clean_rent where agent_person_id not in (select person_id from monre.agent);

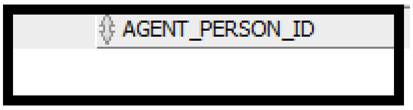
5. d)Screenshots of data before and after cleaning

Before -



	AGENT_PERSON_ID
1	6002

After -



	AGENT_PERSON_ID

6. a)Exploring

To check if the client_person_id written in the monre.rent table rows actually exist in the monre.client table, SQL query used is -

select client_person_id from monre.rent where client_person_id not in (select person_id from monre.client);

6. b)Result

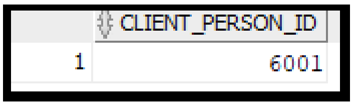
Output seen is an client_person_id = 6001. This tells us that monre.rent has a client_person_id entry as 6001 which actually doesn't exist in the monre.client table's person_id column. Hence it is an illegal entry in the monre.rent table.

6. c)Removing Error

The error is removed using the SQL - delete from clean_rent where client_person_id not in (select person_id from monre.client);

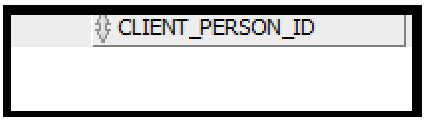
6. d)Screenshots of data before and after cleaning

Before -



	CLIENT_PERSON_ID
1	6001

After -



	CLIENT_PERSON_ID

7. a)Exploring

To check if the property_id written in the monre.rent table rows actually exist in the monre.property table, SQL query used is -

select property_id from monre.rent where property_id not in (select property_id from monre.property);

7. b)Result

No output seen. Hence all property entries are valid.

8. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.rent  where rent_id like 'null';
select * from monre.rent where rent_id  is null;

select * from monre.rent  where AGENT_PERSON_ID like 'null';
select * from monre.rent where AGENT_PERSON_ID is null;

select * from monre.rent  where CLIENT_PERSON_ID like 'null';
select * from monre.rent where CLIENT_PERSON_ID is null;

select * from monre.rent  where PROPERTY_ID like 'null';
select * from monre.rent where PROPERTY_ID  is null;

select * from monre.rent  where RENT_START_DATE like 'null';
select * from monre.rent where RENT_START_DATE is null;

select * from monre.rent  where RENT_END_DATE like 'null';
select * from monre.rent where RENT_END_DATE          is null;

select * from monre.rent  where price like 'null';
select * from monre.rent where price is null;
```

8. b)Result

No unusual null values found.

9. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.rent order by rent_id desc;
select * from monre.rent order by rent_id;

select * from monre.rent  order by AGENT_PERSON_ID desc;
select * from monre.rent order by AGENT_PERSON_ID;

select * from monre.rent order by CLIENT_PERSON_ID desc;
select * from monre.rent order by CLIENT_PERSON_ID;

select * from monre.rent  order by PROPERTY_ID desc;
select * from monre.rent order by PROPERTY_ID;

select * from monre.rent order by RENT_START_DATE desc;
select * from monre.rent order by RENT_START_DATE ;

select * from monre.rent  order by RENT_END_DATE desc;
select * from monre.rent order by RENT_END_DATE;

select * from monre.rent order by price desc;
select * from monre.rent order by price;
```

9. b)Result

No unusual errors seen.

10. a) Exploration

To check out of range or invalid values in the date column, the SQL query used is -

```
select * from monre.rent where rent_end_date < rent_start_date;
```

10. b)Result

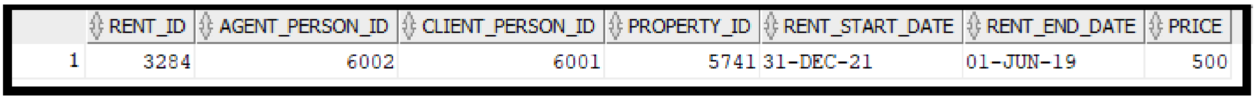
A row seen which has rent_start_date = 31-DEC-21 and rent_end_date = 01-JUN-19. This is invalid and needs to be removed.

10. c)Removing Error

The error is removed using the SQL - delete from clean_rent where rent_end_date < rent_start_date;

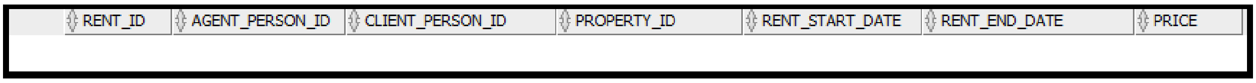
10. d)Screenshots of data before and after cleaning

Before -



	RENT_ID	AGENT_PERSON_ID	CLIENT_PERSON_ID	PROPERTY_ID	RENT_START_DATE	RENT_END_DATE	PRICE
1	3284	6002	6001	5741	31-DEC-21	01-JUN-19	500

After -



RENT_ID	AGENT_PERSON_ID	CLIENT_PERSON_ID	PROPERTY_ID	RENT_START_DATE	RENT_END_DATE	PRICE

Table - MonRE.sale

1. a) Exploring

To check if a sale_id is repeated twice SQL query used is -

```
SELECT sale_id, count(*)
        FROM monre.sale
        GROUP BY sale_id
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence all are valid.

2. a) Exploring

To check if a agent_person_id is repeated twice SQL query used is -

```
SELECT agent_person_id , count(*)
        FROM monre.sale
        GROUP BY agent_person_id
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. This is valid as an agent can be responsible for selling out multiple properties. Hence all are valid.

3. a) Exploring

To check if a client_person_id is repeated twice SQL query used is -

```
SELECT client_person_id , count(*)
        FROM monre.sale
        GROUP BY client_person_id
        HAVING COUNT(*) > 1;
```

3. b) Result

Repetition found 2009 times of ony null client_person_id. This is valid as when a property is not sold out the client_person_id will be null. Hence not an error.

4. a) Exploring

To check if a property_id is repeated twice SQL query used is -

```
SELECT property_id, count(*)
        FROM monre.sale
        GROUP BY property_id
        HAVING COUNT(*) > 1;
```

4. b) Result

No errors seen.

5. a)Exploring

To check if the agent_person_id written in the monre.sale table rows actually exist in the monre.agent table, SQL query used is -

```
select agent_person_id from monre.sale  where agent_person_id not in (select person_id from monre.agent);
```

5. b)Result

No errors seen. Hence all agent_person_ids are valid

6. a)Exploring

To check if the client_person_id written in the monre.sale table rows actually exist in the monre.client table, SQL query used is -

```
select client_person_id from monre.sale where client_person_id not in (select person_id from monre.client);
```

6. b)Result

No errors seen. Hence all client_person_ids are valid

7. a)Exploring

To check if the property_id written in the monre.sale table rows actually exist in the monre.property table, SQL query used is -

```
select property_id from monre.sale  where property_id not in (select property_id from monre.property);
```

7. b)Result

No output seen. Hence all property entries are valid.

8. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.sale  where sale_id like 'null';
select * from monre.sale where sale_id  is null;

select * from monre.sale  where AGENT_PERSON_ID like 'null';
select * from monre.sale where AGENT_PERSON_ID is null;

select * from monre.sale  where CLIENT_PERSON_ID like 'null';
select * from monre.sale where CLIENT_PERSON_ID is null;

select * from monre.sale  where PROPERTY_ID like 'null';
select * from monre.sale where PROPERTY_ID  is null;

select * from monre.sale  where SALE_DATE like 'null';
select * from monre.sale where SALE_DATE  is null;

select * from monre.sale  where price like 'null';
select * from monre.sale where price is null;
```

8. b)Result

No unusual null values found.

9. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.sale order by sale_id desc;
select * from monre.sale order by sale_id;

select * from monre.sale  order by AGENT_PERSON_ID desc;
select * from monre.sale order by AGENT_PERSON_ID;

select * from monre.sale order by CLIENT_PERSON_ID desc;
select * from monre.sale order by CLIENT_PERSON_ID;

select * from monre.sale  order by PROPERTY_ID desc;
select * from monre.sale order by PROPERTY_ID;

select * from monre.sale order by SALE_DATE desc;
select * from monre.sale order by SALE_DATE ;

select * from monre.sale order by price desc;
select * from monre.sale order by price;
```

9. b)Result

No unusual errors seen.

Table - MonRE.state

1. a) Exploring

To check if a state_code is repeated twice SQL query used is -

```
SELECT state_code , count(*)
        FROM monre.state
        GROUP BY state_code
        HAVING COUNT(*) > 1;
```

1. b) Result

No repetition found. Hence all are valid.

2. a) Exploring

To check if a state_name is repeated twice SQL query used is -

```
SELECT state_name, count(*)
        FROM monre.state
        GROUP BY state_name
        HAVING COUNT(*) > 1;
```

2. b) Result

No repetition found. Hence all are valid.

3. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.state  where state_code like 'null';
select * from monre.state where state_code  is null;

select * from monre.state  where state_name like 'null';
select * from monre.state where state_name  is null;
```

3. b) Result

No errors seen.

4. **a) Exploration**

To check out of range or invalid values in each column, the SQL query used is -

```
select * from monre.state order by state_code desc;
select * from monre.state order by state_code ;

select * from monre.state  order by state_name desc;
select * from monre.state order by state_name ;
```

4. b) Result

No errors seen.

Table - MonRE.visit

1. a) Exploring

To check if a agent_person_id is repeated twice SQL query used is -

```
SELECT agent_person_id , count(*)
        FROM monre.visit
        GROUP BY agent_person_id
        HAVING COUNT(*) > 1;
```

1. b) Result

Repetition found. This is valid as an agent can be responsible for visiting of multiple properties. Hence all are valid.

2. a) Exploring

To check if a client_person_id is repeated twice SQL query used is -

```
SELECT client_person_id , count(*)
        FROM monre.visit
        GROUP BY client_person_id
        HAVING COUNT(*) > 1;
```

2. b) Result

Repetition found. This is valid as a client can visit multiple properties. Hence all are valid.

3. a) Exploring

To check if a property_id is repeated twice SQL query used is -

```
SELECT property_id, count(*)
        FROM monre.visit
        GROUP BY property_id
        HAVING COUNT(*) > 1;
```

3. b) Result

Repetition found. This is valid as a property can be visited by multiple agents/clients. Hence all are valid.

4. a)Exploring

To check if the agent_person_id written in the monre.visit table rows actually exist in the monre.agent table, SQL query used is -

select agent_person_id from monre.visit where agent_person_id not in (select person_id from monre.agent);

4. b)Result

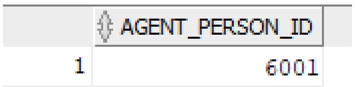
Output seen is an agent_person_id = 6001. This tells us that monre.visit has an agent_person_id entry as 6001 which actually doesn't exist in the monre.agent table's person_id column. Hence it is an illegal entry in the monre.visit table.

4. c)Removing Error

A copy of the table monre.visit is created and then the error is removed from it.
For copying the SQL query used is - create table clean_visit as select * from monre.visit
The error is removed using the SQL - delete from clean_visit where agent_person_id not in (select person_id from monre.agent);

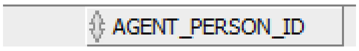
4. d)Screenshots of data before and after cleaning

Before -



	AGENT_PERSON_ID
1	6001

After -



	AGENT_PERSON_ID

Hence invalid entry removed

5. a)Exploring

To check if the client_person_id written in the monre.visit table rows actually exist in the monre.client table, SQL query used is -

select client_person_id from monre.visit where client_person_id not in (select person_id from monre.client);

5. b)Result

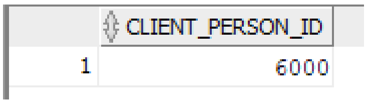
Output seen is an client_person_id = 6000. This tells us that monre.visit has a client_person_id entry as 6000 which actually doesn't exist in the monre.client table's person_id column. Hence it is an illegal entry in the monre.visit table.

5. c)Removing Error

The error is removed using the SQL - delete from clean_visit where client_person_id not in (select person_id from monre.client);

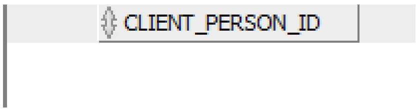
5. d)Screenshots of data before and after cleaning

Before -



	CLIENT_PERSON_ID
1	6000

After -



	CLIENT_PERSON_ID

6. a)Exploring

To check if the property_id written in the monre.visit table rows actually exist in the monre.property table, SQL query used is -

select property_id from monre.visit where property_id not in (select property_id from monre.property);

6. b)Result

No output seen. Hence all property entries are valid.

7. a)Exploration

To check for null values in each column and values that are filled as ‘null’ , SQL query used is -

```
select * from monre.visit  where AGENT_PERSON_ID like 'null';
select * from monre.visit where AGENT_PERSON_ID is null;

select * from monre.visit  where CLIENT_PERSON_ID like 'null';
select * from monre.visit where CLIENT_PERSON_ID is null;

select * from monre.visit  where PROPERTY_ID like 'null';
select * from monre.visit where PROPERTY_ID  is null;

select * from monre.visit  where VISIT_DATE like 'null';
select * from monre.visit where VISIT_DATE  is null;

select * from monre.visit  where duration like 'null';
select * from monre.visit where duration is null;
```

7. b)Result

No unusual null values found.

8. a) Exploration

To check out of range or invalid values in each column, the SQL query used is -

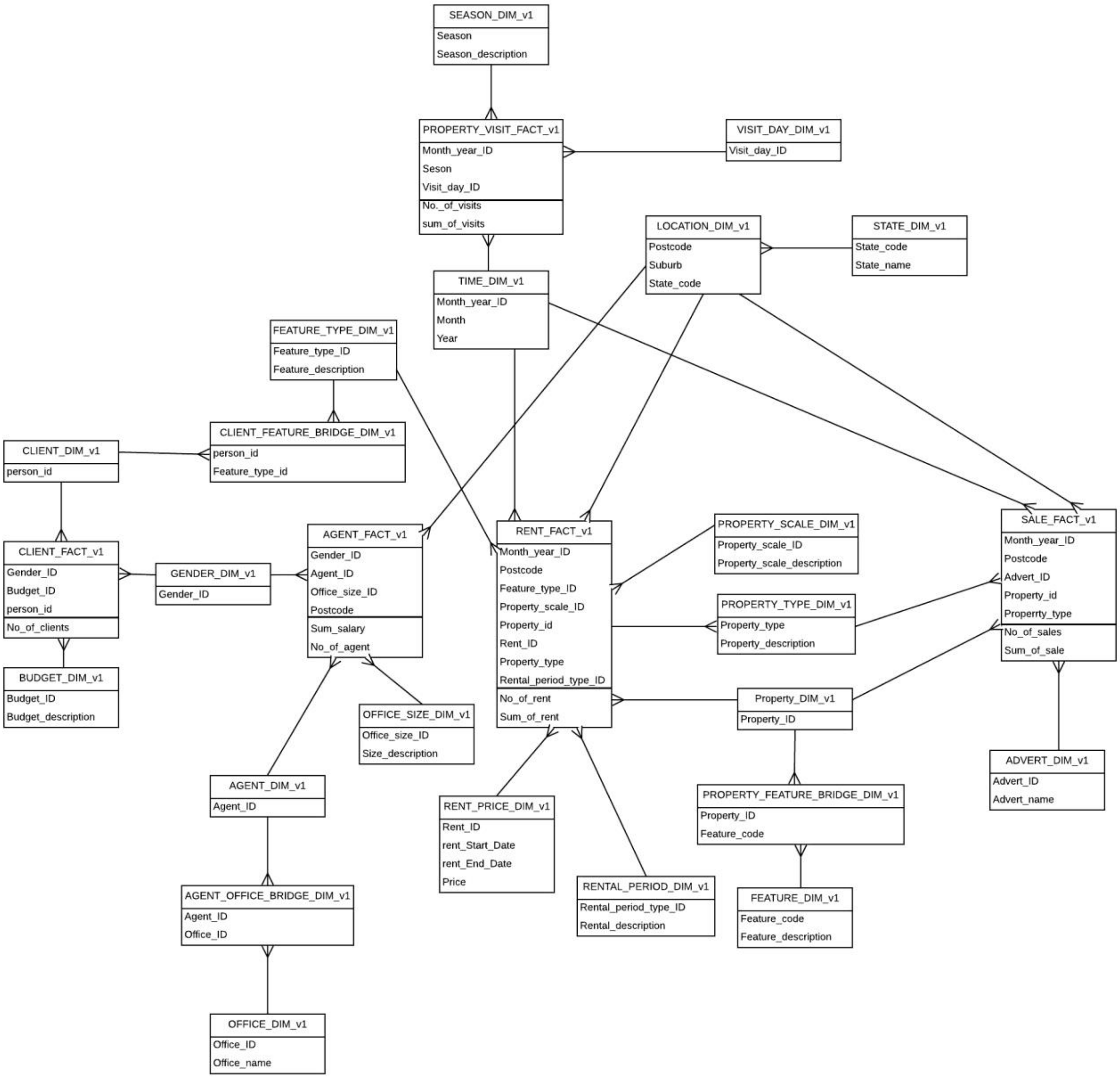
```
        select * from monre.visit  order by AGENT_PERSON_ID desc;
        select * from monre.visit order by AGENT_PERSON_ID;

        select * from monre.visit order by CLIENT_PERSON_ID desc;
        select * from monre.visit order by CLIENT_PERSON_ID;

        select * from monre.visit  order by PROPERTY_ID desc;
        select * from monre.visit order by PROPERTY_ID;

        select * from monre.visit order by VISIT_DATE desc;
        select * from monre.visit order by VISIT_DATE ;

        select * from monre.visit order by duration desc;
        select * from monre.visit order by duration;
```

8. b)Result

No unusual errors seen.

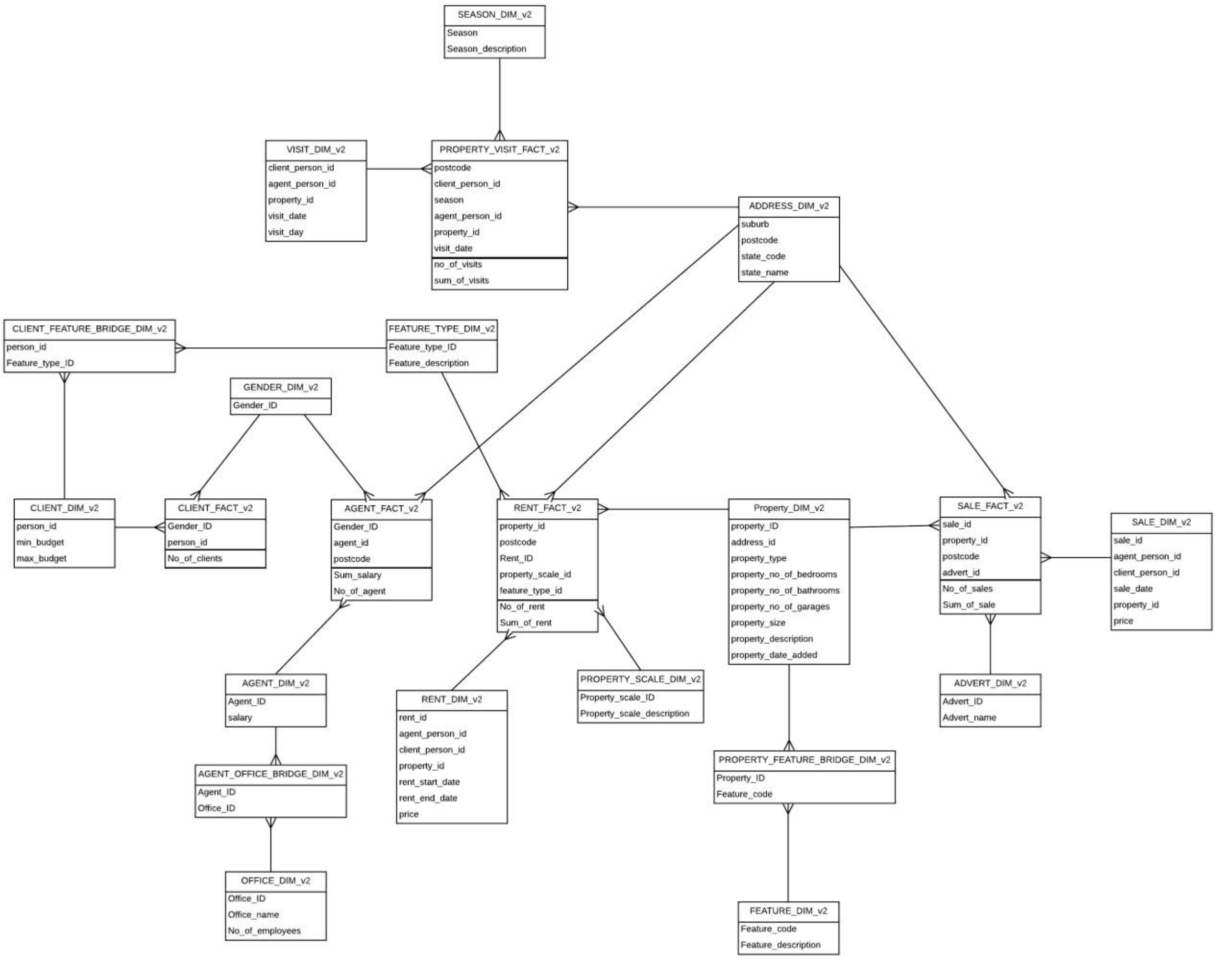
Task 1 out of 4
c) Two versions of star schema diagrams
VERSION 1 -

FIT5195
Star Schema Version-1
Task- C-1 c)
Majorr Group Project
Dishi Jain - 30759307
Sarthak Sareen - 30761182



VERSION -2

FIT5195
Star Schema Version-2
Task- C-1 c)
Majorr Group Project
Dishi Jain - 30759307
Sarthak Sareen - 30761182



Task 1 out of 4
d) Explanation for using hierarchy

Hierarchy has been chosen in the Dimensions - Location_DIM and State_DIM. This is because a particular location/postcode will have only one state_code. However a state_code can have multiple location addresses inside it. Also location_DIM is more detailed showing the exact location postcode and suburb whereas the state_DIM is more general showing just the state_codes. Hence we see a natural traversal path from location_dim to state_dim. The hierarchy is hence created between the two.

Task 1 out of 4
e) Reason of SCD Type chosen for temporal dimension

The temporal dimension used in the star schema is Rent_Price_DIM. This DIM will store the changing prices of the property being rented out. It contains the rent_id, rent_start_date, rent_end_date and the price. The type chosen for temporal dimension is Type2. This temporal dimension contains the information about the history of the prices of the properties being rented and hence being type2 it is directly connected to the Fact Table. The reason for choosing this Type2 was because it was making the most significance to choose this type and also because it includes the addition of just one dimension into the star schema. Fewer the extra dimensions added the better the star schema. Hence as the entire history is saved in the Type2 SCD, we choose this type. It is important to know the current price, previous price and all the other previouses prices of the rent for a particular property. Hence an entire record is required to maintain. Hence the Type2 SCD is chosen.

Task 1 out of 4

f) Difference between the two versions of Star Schema

The Version 1 star schema is the one which has Level 2 aggregation. It hence has a high level of groupings when we take a look at the dimension attributes. As it has a high level of aggregation hence the granularity is low. This Version is hence less detailed. It contains overall 18 dimensions and 5 fact tables. The connections between them are made according to the questions given to us for creating the data warehouse.

The Version 2 star schema is the one which has Level 0 aggregation. It hence has a low level of groupings when we take a look at the dimension attributes. As it has a low level of aggregation hence the granularity is high. This Version is hence more detailed. It contains overall 14 dimensions and 5 fact tables. The connections between them are made according to the questions given to us for creating the data warehouse.

Task 2 out of 4

a) SQL for creating Version 1 i.e. with Level 2 Aggregation star schemaTask 2 out of 4

```
--CREATING STAR SCHEMA VERSION 1 Level 2 Aggregation
drop table gender_dim_v1;
drop table budget_dim_v1;
drop table season_dim_v1;
drop table location_dim_v1;
drop table visit_day_dim_v1;
drop table state_dim_v1;
drop table office_size_dim_v1;
drop table feature_type_dim_v1;
drop table property_scale_dim_v1;
drop table property_type_dim_v1;
drop table property_dim_v1;
drop table property_feature_bridge_dim_v1;
drop table feature_dim_v1;
drop table property_advert_bridge_dim_v1;
drop table advert_dim_v1;
drop table client_dim_v1;
drop table client_feature_bridge_dim_v1;
drop table client_tempfact;
drop table client_fact_v1;
drop table time_dim_v1;
drop table property_visit_tempfact;
drop table property_visit_fact_v1;
drop table rent_tempfact;
drop table rent_fact_v1;
drop table sale_tempfact;
drop table sale_fact_v1;
drop table property_dim_v1;
drop table property_feature_bridge_dim_v1;
drop table feaure_dim_v1;
drop table agent_tempfact;
drop table agent_fact_v1;
drop table rental_period_dim_v1;
drop table rent_price_dim_v1;
drop table office_dim_v1;
drop table agent_dim_v1;
drop table agent_office_bridge_dim_v1;


--creating gender dim
create table gender_dim_v1
( gender_id varchar2(30));

--inserting values in gender dim
insert into gender_dim_v1 values('male');
insert into gender_dim_v1 values('female');

--creating budget dim
create table budget_dim_v1
( budget_id varchar(30),
  budget_description varchar2(30));

--inserting values in budget dim
insert into budget_dim_v1 values ('low','0 to 1000');
insert into budget_dim_v1 values ('medium','1001 to 100000');
insert into budget_dim_v1 values ('high','100001 to 10000000');


--creating season dim
create table season_dim_v1
( season varchar2(30),
  season_description varchar2(30));

--inserting values in season dim
insert into season_dim_v1 values('winter','june-july-august');
insert into season_dim_v1 values('summer','dec-jan-feb');
insert into season_dim_v1 values('autum','march-april-may');
insert into season_dim_v1 values('spring','sept-oct-nov');

--creating location dim
create table location_dim_v1 as select a.postcode,a.suburb,p.state_code
from clean_address a, clean_postcode p
where p.postcode=a.postcode;

--creating visit day dim
create table visit_day_dim_v1
( visit_day_id varchar(30));

--inserting values in visit day dim
insert into visit_day_dim_v1 values('monday');
insert into visit_day_dim_v1 values('tuesday');
insert into visit_day_dim_v1 values('wednesday');
insert into visit_day_dim_v1 values('thursday');
insert into visit_day_dim_v1 values('friday');
insert into visit_day_dim_v1 values('saturday');
insert into visit_day_dim_v1 values('sunday');

--creating state dim
create table state_dim_v1 as select state_code,state_name
from clean_state;

--creating office size dim
create table office_size_dim_v1
( office_size_id varchar2(30),
  size_description varchar2(30));

--creating office size dim
insert into office_size_dim_v1 values ('small','less than 4 employess');
insert into office_size_dim_v1 values ('medium','4 - 12 employess');
insert into office_size_dim_v1 values ('large','more than 12 employess');

--creating feature type dim
create table feature_type_dim_v1
( feature_type_id varchar2(30),
 feature_description varchar(30));

--inserting values in feature type dim
insert into feature_type_dim_v1 values ('basic',' less than 10 features');
insert into feature_type_dim_v1 values ('standard ','10-20 features');
insert into feature_type_dim_v1 values ('luxurious',' more than 20 features');

--creating property scale dim
create table property_scale_dim_v1
( property_scale_id varchar2(30),
 property_scale_description varchar(30));

--inseting values in property scale dim
insert into property_scale_dim_v1 values ('extra small',' <= 1 bedroom');
insert into property_scale_dim_v1 values ('small ','2-3 bedrooms');
insert into property_scale_dim_v1 values ('medium',' 3-6 bedrooms');
insert into property_scale_dim_v1 values ('large',' 6-10 bedrooms');
insert into property_scale_dim_v1 values ('extra large','> 10 bedrooms');

--creating table property type dim
create table property_type_dim_v1 as select distinct property_type,property_description
from clean_property;

--creating property_dim
create table property_dim_v1 as select distinct property_id from clean_property;
```

```
--creating property feature bridge dim
create table property_feature_bridge_dim_v1 as select * from clean_property_feature;


--creating feature dim
create table feature_dim_v1 as select distinct * from clean_feature;

--creating advert dim
create table advert_dim_v1 as select distinct * from clean_advertisement;

--creating client dim
create table client_dim_v1 as select person_id from clean_client;

--creating client feature bridge dim
create table client_feature_bridge_dim_v1 as select FEATURE_CODE as feature_type_id,
person_id from clean_client_wish;

--creating client tempfact
create table client_tempfact as select c.person_id, p.gender as
gender_id,c.max_budget,c.min_budget from clean_person p , clean_client c
where p.person_id=c.person_id;

--adding columns to client temp fact
alter table client_tempfact add(budget_id varchar(30));

update client_tempfact set budget_id ='low' where max_budget <= 1000;
update client_tempfact set budget_id ='medium' where max_budget >1000 and max_budget<=
100000;
update client_tempfact set budget_id ='high' where max_budget >100000 and
max_budget<=10000000;


--creating client fact
create table client_fact_v1 as select person_id,gender_id,budget_id, count(person_id) as
no_of_clients from client_tempfact
group by gender_id,budget_id,person_id;

--creating time dim
create table time_dim_v1 as select distinct to_char(visit_date,'mm') || to_char(visit_date,'yy') as
month_year_id,to_char(visit_date,'mm') as month,to_char(visit_date,'yy') as year from
clean_visit
union select distinct to_char(rent_start_date,'mm') || to_char(rent_start_date,'yy') as
month_year_id,to_char(rent_start_date,'mm') as month,to_char(rent_start_date,'yy') as year
from clean_rent union
select distinct to_char(rent_end_date,'mm') || to_char(rent_end_date,'yy') as
month_year_id,to_char(rent_end_date,'mm') as month,to_char(rent_end_date,'yy') as year
from clean_rent union
```

```
select distinct  to_char(sale_date,'mm') || to_char(sale_date,'yy') as
month_year_id,to_char(sale_date,'mm') as month,to_char(sale_date,'yy') as year  from
clean_sale;


--creating property visit temp fact
create table property_visit_tempfact as select v.client_person_id,v.agent_person_id,
count(v.property_id) as no_of_visits,TO_date(v.visit_date,'dd-mon-yy') AS
visit_date,a.postcode from clean_visit v,clean_property p,clean_address a
where p.property_id=v.property_id and a.address_id=p.address_id
group by v.client_person_id,v.agent_person_id,TO_date(v.visit_date,'dd-mon-yy'),a.postcode;

--altering the visit temp fat
alter table property_visit_tempfact add(season varchar(30), month_year_id
varchar(30),visit_day_id varchar(30));

--updating the visit temp fact
update property_visit_tempfact
set month_year_id =  to_char(visit_date,'mm') ||  to_char(visit_date,'yy');

UPDATE property_visit_tempfact
SET
season = 'summer'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'dec',
'jan',
'feb'
);

UPDATE property_visit_tempfact
SET
season = 'winter'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'jun',
'jul',
'aug'
);

UPDATE property_visit_tempfact
SET
season = 'spring'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'sep',
'oct',
'nov'
);

UPDATE property_visit_tempfact
SET
season = 'autum'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'mar',
'apr',
'may'
);

UPDATE property_visit_tempfact
SET
visit_day_id = 'monday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%MONDAY%';

UPDATE property_visit_tempfact
SET
visit_day_id = 'tuesday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%TUESDAY%';


UPDATE property_visit_tempfact
SET
visit_day_id = 'wednesday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%WEDNESDAY%';

UPDATE property_visit_tempfact
SET
visit_day_id = 'thursday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%THURSDAY%';

UPDATE property_visit_tempfact
SET
visit_day_id = 'friday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%FRIDAY%';

UPDATE property_visit_tempfact
SET
visit_day_id = 'saturday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%SATURDAY%';
```

```
UPDATE property_visit_tempfact
SET
visit_day_id = 'sunday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%SUNDAY%';

--ceating property visit fact
create table property_visit_fact_v1 as select
month_year_id,season,visit_day_id,no_of_visits,sum(no_of_visits) as sum_of_visits
from property_visit_tempfact
group by month_year_id,season,visit_day_id,no_of_visits;

--creating rental period dim
create table rental_period_dim_v1
( rental_period_type_id varchar2(30),
  rental_description varchar2(30));

--inserting values in rental period dim
insert into rental_period_dim_v1 values ('short','< 6 months');
insert into rental_period_dim_v1 values ('medium','6-12 months');
insert into rental_period_dim_v1 values ('large','> 12 months');

--creating rent temp fact
create table rent_tempfact as select t.rent_id, p.property_id,
p.property_no_of_bedrooms,p.property_type,f.feature_code,t.price,a.postcode,to_date(t.rent_start_date,'dd-mm-yy') as start_date,to_date(t.rent_end_date,'dd-mm-yy') as end_date
from clean_rent t,
clean_property p,clean_property_feature f, clean_address a where
p.property_id=t.property_id and
p.property_id=f.property_id and
p.address_id=a.address_id;

--altering rent temp fact
alter table rent_tempfact add(feature_type_id varchar(30), property_scale_id
varchar(30),calc_price number(20,2),month_year_id varchar(30),rental_period_type_id
varchar(30));


--updating rent temp fact
update  rent_tempfact set feature_type_id ='very basic' where property_id in (select property_id
from monre.property_feature group by property_id having count(*) < 10);
update  rent_tempfact set feature_type_id ='standard' where property_id in (select property_id
from monre.property_feature group by property_id having count(*) between 11 and 20);
update  rent_tempfact set feature_type_id ='luxurious' where property_id in (select property_id
from monre.property_feature group by property_id having count(*) > 20);
```

```
update rent_tempfact set property_scale_id ='extra small' where property_no_of_bedrooms <=
1;
update rent_tempfact set property_scale_id ='small' where property_no_of_bedrooms > 1 and
property_no_of_bedrooms <=2;
update rent_tempfact set property_scale_id ='medium' where property_no_of_bedrooms > 2
and property_no_of_bedrooms <=6;
update rent_tempfact  set property_scale_id ='large' where property_no_of_bedrooms > 6 and
property_no_of_bedrooms <=10;
update rent_tempfact  set property_scale_id ='extra large' where property_no_of_bedrooms >
10;


update rent_tempfact  set calc_price = (to_date(end_date,'dd-mon-yy') - to_date(start_date,'dd-
mon-yy')) /7 * price;

update rent_tempfact set rental_period_type_id = 'short' where (to_date(end_date,'dd-mon-yy')
- to_date(start_date,'dd-mon-yy'))/30 <= 6 ;
update rent_tempfact set rental_period_type_id = 'medium' where (to_date(end_date,'dd-mon-
yy') - to_date(start_date,'dd-mon-yy'))/30 between 7 and 12 ;
update rent_tempfact set rental_period_type_id = 'large' where (to_date(end_date,'dd-mon-yy')
- to_date(start_date,'dd-mon-yy'))/30 > 12 ;

update rent_tempfact
set month_year_id =  to_char(start_date,'mm') ||  to_char(start_date,'yy');
--creating temporal dim (rent_price_dim) to store history price of properties being rented
create table rent_price_dim_v1 as select rent_id,to_date(rent_start_date,'dd-mm-yy') as
rent_start_date,to_date(rent_end_date,'dd-mm-yy') as rent_end_date, price from clean_rent;

--creating rent fact
create table rent_fact_v1 as select month_year_id,postcode,rent_id, property_id, property_type
, feature_type_id, property_scale_id,rental_period_type_id, count(rent_id) as no_of_rent,
sum(calc_price) as sum_of_rent from rent_tempfact
group by  month_year_id,postcode,rent_id, property_id, property_type , feature_type_id,
property_scale_id,rental_period_type_id;

--creating table sale temp fact
create table sale_tempfact as select to_date(p.property_date_added,'dd-mm-yy') as
advertised_date ,p.property_id,s.sale_id,
a.postcode,d.advert_id,p.property_type ,s.price,s.sale_date from
clean_property p, clean_address a, clean_sale s , clean_property_advert d where
p.property_id=s.property_id and
s.property_id=d.property_id and
p.address_id=a.address_id;

alter table sale_tempfact add(month_year_id varchar(30));

update sale_tempfact set month_year_id = (to_char(advertised_date,'mm') ||
to_char(advertised_date,'yy'));
```

```
--creating sale fact
create table sale_fact_v1 as select
month_year_id,postcode,property_id,property_type,advert_id,count(sale_id) as
no_of_sales,sum(price) as sum_of_sale
from sale_tempfact group by month_year_id,postcode,property_id,property_type,advert_id;

--creating feature dim
create table feaure_dim_v1 as select distinct * from clean_feature;

--creating office dim  dim
create table office_dim_v1 as select distinct * from clean_office;

--creating agent dim
create table agent_dim_v1 as select person_id as agent_id from clean_agent;

--creating agent office bridge dim
create table agent_office_bridge_dim_v1 as select person_id as agent_id , office_id from
clean_agent_office;

--creating agent temp fact
create table agent_tempfact as select g.person_id as agent_id
,g.salary,a.postcode,o.office_id,p.gender as gender_id
from clean_agent g, clean_agent_office o,clean_person p, clean_address a
where p.person_id=o.person_id and
p.person_id=g.person_id and
p.address_id=a.address_id;


--altering the table agent temp fact
alter table agent_tempfact add(office_size_id varchar(30));

--updating the agent temp fact
update agent_tempfact set office_size_id ='small' where office_id in (select office_id from
monre.agent_office group by office_id having count(*) <= 4 );
update agent_tempfact set office_size_id ='medium' where office_id in (select office_id from
monre.agent_office group by office_id having count(*) between 5 and 12 );
update agent_tempfact set office_size_id ='large' where office_id in (select office_id from
monre.agent_office group by office_id having  count(*) > 12);

--creating table agent fact
create table agent_fact_v1
as select postcode , agent_id , gender_id ,office_size_id , sum(salary) as sum_salary,
count(agent_id) as no_of_agent
from agent_tempfact
group by postcode ,  agent_id ,gender_id , office_size_id ;
```

b) SQL for creating Version 2 i.e. with Level 0 Aggregation star schema
2 out of 4

```
--CREATING STAR SCHEMA VERSION 2 LEVEL 0 AGGREGATION

drop table gender_dim_v2;
drop table season_dim_v2;
drop table property_visit_fact_v2;
drop table visit_dim_v2;
drop table address_dim_v2;
drop table client_feature_bridge_v2;
drop table feature_type_dim_v2;
drop table client_dim_v2;
drop table client_fact_v2;
drop table agent_fact_v2;
drop table rent_fact_v2;
drop table property_dim_v2;
drop table sale_fact_v2;
drop table sale_dim_v2;
drop table agent_dim_v2;
drop table rent_dim_v2;
drop table property_scale_dim_v2;
drop table property_feature_bridge_dim_v2;
drop table advert_dim_v2;
drop table agent_office_bridge_dim_v2;
drop table office_dim_v2;
drop table feature_dim_v2;
drop table visit_tempdim_v2;
drop table client_tempfact_v2;
drop table property_visit_tempfact_v2;
drop table rent_tempfact_v2;
drop table sale_tempfact_v2;
drop table agent_tempfact_v2;
drop table client_feature_bridge_dim_v2;
drop table agent_fact_dim_v2;




--creating gender dim
create table gender_dim_v2
( gender_id varchar2(30));

insert into gender_dim_v2 values('Male');
insert into gender_dim_v2 values('Female');

--creating client dim version 2
```

```
create table client_dim_v2 as select person_id,min_budget,max_budget from clean_client;


--creating visit temp dim
create table visit_tempdim_v2 as select
client_person_id,agent_person_id,property_id,to_date(visit_date,'dd-mm-yy') as visit_date ,
duration from clean_visit;

--altering visit temp dim
alter table visit_tempdim_v2 add (visit_day varchar2(30));

--updating the visit temp dim
UPDATE visit_tempdim_v2
SET
visit_day = 'monday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%MONDAY%';

UPDATE visit_tempdim_v2
SET
visit_day = 'tuesday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%TUESDAY%';


UPDATE visit_tempdim_v2
SET
visit_day = 'wednesday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%WEDNESDAY%';

UPDATE visit_tempdim_v2
SET
visit_day = 'thursday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%THURSDAY%';

UPDATE visit_tempdim_v2
SET
visit_day = 'friday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%FRIDAY%';

UPDATE visit_tempdim_v2
SET
visit_day = 'saturday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%SATURDAY%';
```

```
UPDATE visit_tempdim_v2
SET
visit_day = 'sunday'
WHERE
TO_CHAR(visit_date, 'DAY') like '%SUNDAY%';

--creating visit dim version 2
create table visit_dim_v2 as select
client_person_id,agent_person_id,property_id,visit_day,to_date(visit_date,'dd-mm-yy') as
visit_date
from visit_tempdim_v2;

--creating address dim version 2
create table address_dim_v2 as select a.postcode, a.suburb,s.state_code,s.state_name
from clean_address a,clean_state s,clean_postcode p
where a.postcode=p.postcode and
s.state_code=p.state_code;

--creating agent dim version 2
create table agent_dim_v2 as select person_id as agent_id,salary from clean_agent;

--creating agent office bridge   dim version 2
create table agent_office_bridge_dim_v2 as select person_id as agent_id , office_id from
clean_agent_office;

--creating office dim version 2
create table office_dim_v2 as select distinct o.office_id,office_name,count(person_id) as
no_of_employees from clean_agent_office a,clean_office o
where o.office_id=a.office_id
group by o.office_id,office_name;

--creating rent dim version 2
create table rent_dim_v2 as select
rent_id,agent_person_id,client_person_id,property_id,rent_start_date,rent_end_date,price
from clean_rent;

--creating property_dim version 2
create table property_dim_v2 as select
property_id,address_id,property_type,property_no_of_bedrooms,property_no_of_bathrooms,pr
operty_no_of_garages,
property_size,property_description,property_date_added from clean_property;

--creating property feature bridge dim version 2
create table property_feature_bridge_dim_v2 as select * from clean_property_feature;
```

```
--creating feature dim version 2
create table feature_dim_v2 as select distinct * from clean_feature;

--creating advert dim
create table advert_dim_v2 as select distinct * from clean_advertisement;

--creating property scale dim version 2
create table property_scale_dim_v2
( property_scale_id varchar2(30),
 property_scale_description varchar(30));

--inseting values in property scale dim
insert into property_scale_dim_v2 values ('extra small',' <= 1 bedroom');
insert into property_scale_dim_v2 values ('small ','2-3 bedrooms');
insert into property_scale_dim_v2 values ('medium',' 3-6 bedrooms');
insert into property_scale_dim_v2 values ('large',' 6-10 bedrooms');
insert into property_scale_dim_v2 values ('extra large','> 10 bedrooms');


create table season_dim_v2
( season varchar2(30),
  season_description varchar2(30));

--inserting values in season dim
insert into season_dim_v2 values('winter','june-july-august');
insert into season_dim_v2 values('summer','dec-jan-feb');
insert into season_dim_v2 values('autum','march-april-may');
insert into season_dim_v2 values('spring','sept-oct-nov');

--creating client tempfact version 2
create table client_tempfact_v2 as select c.person_id, p.gender as
gender_id,c.max_budget,c.min_budget from clean_person p , clean_client c
where p.person_id=c.person_id;

--creating client fact version 2
create table client_fact_v2 as select person_id,gender_id,count(person_id) as no_of_clients
from client_tempfact_v2
group by gender_id,person_id;

--creating property visit temp fact
create table property_visit_tempfact_v2 as select
v.client_person_id,v.property_id,v.agent_person_id,a.postcode,
count(v.property_id) as no_of_visits,TO_date(v.visit_date,'dd-mon-yy') AS visit_date from
clean_visit v,clean_property p,clean_address a
where p.property_id=v.property_id and a.address_id=p.address_id
group by v.client_person_id,v.property_id,v.agent_person_id,TO_date(v.visit_date,'dd-mon-yy'),a.postcode;
```

```
--altering the visit temp fact
alter table property_visit_tempfact_v2 add(season varchar(30));


UPDATE property_visit_tempfact_v2
SET
season = 'summer'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'dec',
'jan',
'feb'
);

UPDATE property_visit_tempfact_v2
SET
season = 'winter'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'jun',
'jul',
'aug'
);

UPDATE property_visit_tempfact_v2
SET
season = 'spring'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'sep',
'oct',
'nov'
);

UPDATE property_visit_tempfact_v2
SET
season = 'autum'
WHERE
TO_CHAR(visit_date, 'mon') IN (
'mar',
'apr',
'may'
);


--ceating property visit fact
create table property_visit_fact_v2 as select
postcode,property_id,client_person_id,agent_person_id,season,
to_date(visit_date,'dd-mm-yy') as visit_date,no_of_visits,sum(no_of_visits) as sum_of_visits
from property_visit_tempfact_v2
group by  postcode,property_id,client_person_id,agent_person_id,season,
to_date(visit_date,'dd-mm-yy'),no_of_visits;

--creating rent temp fact
create table rent_tempfact_v2 as select t.rent_id, p.property_id,
p.property_no_of_bedrooms,f.feature_code,a.postcode,t.price,to_date(t.rent_start_date,'dd-
mm-yy') as start_date,to_date(t.rent_end_date,'dd-mm-yy') as end_date
from clean_rent t,
clean_property p,clean_property_feature f , clean_address a where
p.property_id=t.property_id and
p.property_id=f.property_id and
p.address_id = a.address_id;

--altering rent temp fact
alter table rent_tempfact_v2 add(feature_type_id varchar(30), property_scale_id
varchar(30),calc_price number(20,2));



--updating rent temp fact
update  rent_tempfact_v2 set feature_type_id ='very basic' where property_id in (select
property_id from monre.property_feature group by property_id having count(*) < 10);
update  rent_tempfact_v2 set feature_type_id ='standard' where property_id in (select
property_id from monre.property_feature group by property_id having count(*) between 11 and
20);
update  rent_tempfact_v2 set feature_type_id ='luxurious' where property_id in (select
property_id from monre.property_feature group by property_id having count(*) > 20);

update rent_tempfact_v2 set property_scale_id ='extra small' where property_no_of_bedrooms
<= 1;
update rent_tempfact_v2 set property_scale_id ='small' where property_no_of_bedrooms > 1
and property_no_of_bedrooms <=2;
update rent_tempfact_v2 set property_scale_id ='medium' where property_no_of_bedrooms >
2 and property_no_of_bedrooms <=6;
update rent_tempfact_v2 set property_scale_id ='large' where property_no_of_bedrooms > 6
and property_no_of_bedrooms <=10;
update rent_tempfact_v2 set property_scale_id ='extra large' where property_no_of_bedrooms
> 10;



update rent_tempfact_v2  set calc_price = (to_date(end_date,'dd-mon-yy') -
to_date(start_date,'dd-mon-yy')) /7 * price;

--creating rent fact
create table rent_fact_v2 as select postcode,rent_id, property_id,
property_scale_id,feature_type_id, count(rent_id) as no_of_rent,
sum(calc_price) as sum_of_rent from rent_tempfact_v2
```

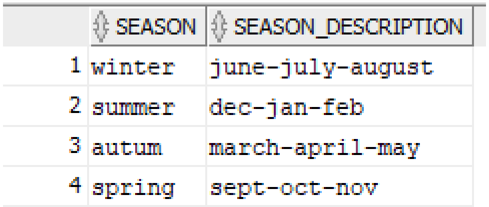
```
group by postcode,rent_id, property_id, property_scale_id,feature_type_id;


--creating table sale temp fact
create table sale_tempfact_v2 as select p.property_id,s.sale_id,
a.postcode,d.advert_id,s.price from
clean_property p, clean_address a, clean_sale s , clean_property_advert d where
p.property_id=s.property_id and
s.property_id=d.property_id and
p.address_id=a.address_id;


--creating sale fact
create table sale_fact_v2 as select postcode,property_id,sale_id,advert_id,count(sale_id) as
no_of_sales,sum(price) as sum_of_sale
from sale_tempfact_v2 group by postcode,property_id,sale_id,advert_id;

--creating sale dim version 2
create table sale_dim_v2 as select * from clean_sale;

--creating agent tempfact version 2
create table agent_tempfact_v2 as select a.postcode,a.salary ,a.person_id as agent_id,gender
as gender_id
from clean_agent a,clean_address a,clean_person p
where p.person_id=a.person_id and
a.address_id=p.address_id;

--creating agent fact dim
create table agent_fact_v2 as select postcode,agent_id,gender_id,count(agent_id) as
no_of_agent,sum(salary) as sum_salary
from agent_tempfact_v2 group by postcode,agent_id,gender_id;

--creating client feature bridge dim
create table client_feature_bridge_dim_v2 as select FEATURE_CODE as feature_type_id,
person_id from clean_client_wish;


--creating feature type dim
create table feature_type_dim_v2
( feature_type_id varchar2(30),
 feature_description varchar(30));

--inserting values in feature type dim
insert into feature_type_dim_v2 values ('basic',' less than 4 features');
insert into feature_type_dim_v2 values ('standard ','10-20 features');
insert into feature_type_dim_v2 values ('luxurious',' more than 20 features');
```

COMMIT;

c) Screenshots of tables

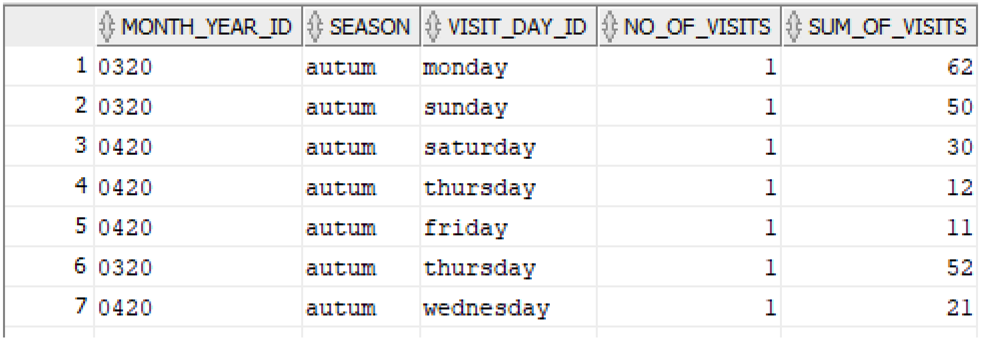
LEVEL 1 (VERSION 1)

SEASON_DIM_V1



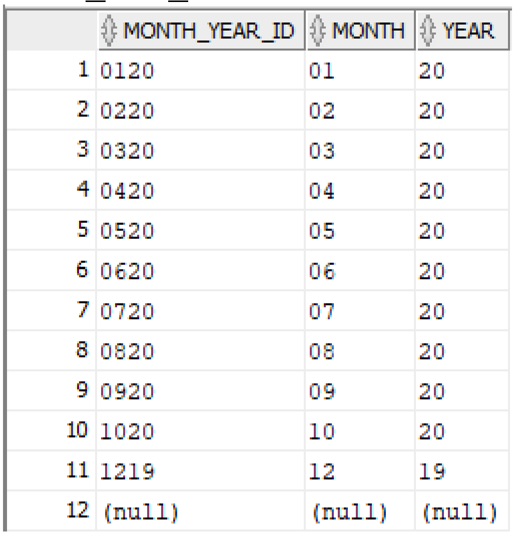
	SEASON	SEASON_DESCRIPTION
1	winter	june-july-august
2	summer	dec-jan-feb
3	autum	march-april-may
4	spring	sept-oct-nov

PROPERTY_VISIT_FACT_V1



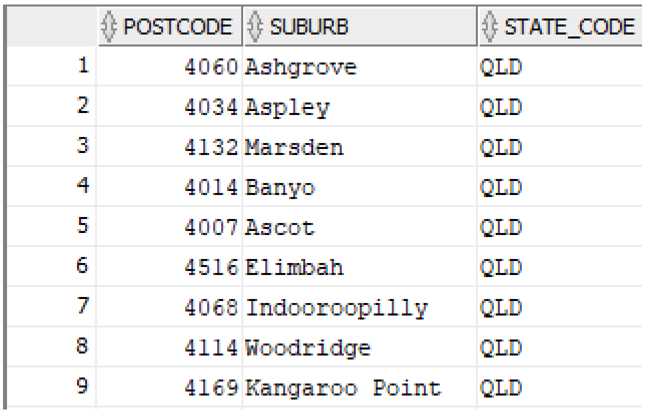
	MONTH_YEAR_ID	SEASON	VISIT_DAY_ID	NO_OF_VISITS	SUM_OF_VISITS
1	0320	autum	monday	1	62
2	0320	autum	sunday	1	50
3	0420	autum	saturday	1	30
4	0420	autum	thursday	1	12
5	0420	autum	friday	1	11
6	0320	autum	thursday	1	52
7	0420	autum	wednesday	1	21

TIME_DIM_V1



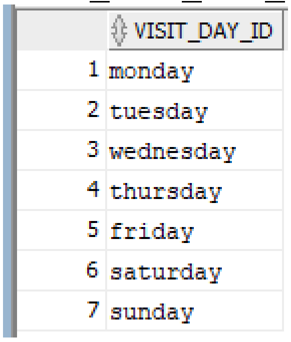
	MONTH_YEAR_ID	MONTH	YEAR
1	0120	01	20
2	0220	02	20
3	0320	03	20
4	0420	04	20
5	0520	05	20
6	0620	06	20
7	0720	07	20
8	0820	08	20
9	0920	09	20
10	1020	10	20
11	1219	12	19
12	(null)	(null)	(null)

LOCATION_DIM_V1



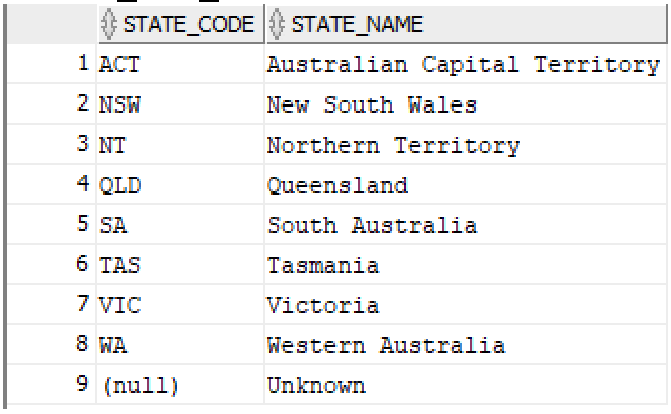
	POSTCODE	SUBURB	STATE_CODE
1	4060	Ashgrove	QLD
2	4034	Aspley	QLD
3	4132	Marsden	QLD
4	4014	Banyo	QLD
5	4007	Ascot	QLD
6	4516	Elimbah	QLD
7	4068	Indooroopilly	QLD
8	4114	Woodridge	QLD
9	4169	Kangaroo Point	QLD

VISIT_DAY_DIM_V1



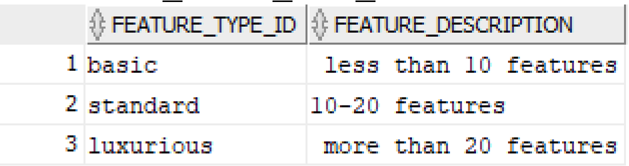
	VISIT_DAY_ID
1	monday
2	tuesday
3	wednesday
4	thursday
5	friday
6	saturday
7	sunday

STATE_DIM_V1



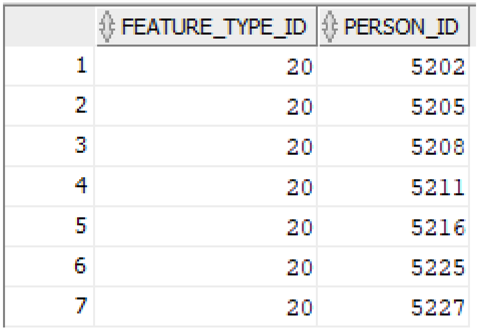
	STATE_CODE	STATE_NAME
1	ACT	Australian Capital Territory
2	NSW	New South Wales
3	NT	Northern Territory
4	QLD	Queensland
5	SA	South Australia
6	TAS	Tasmania
7	VIC	Victoria
8	WA	Western Australia
9	(null)	Unknown

FEATURE_TYPE_DIM_V1



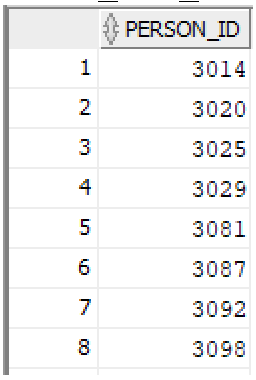
	FEATURE_TYPE_ID	FEATURE_DESCRIPTION
1	basic	less than 10 features
2	standard	10-20 features
3	luxurious	more than 20 features

client_feature_bridge_DIM_V1



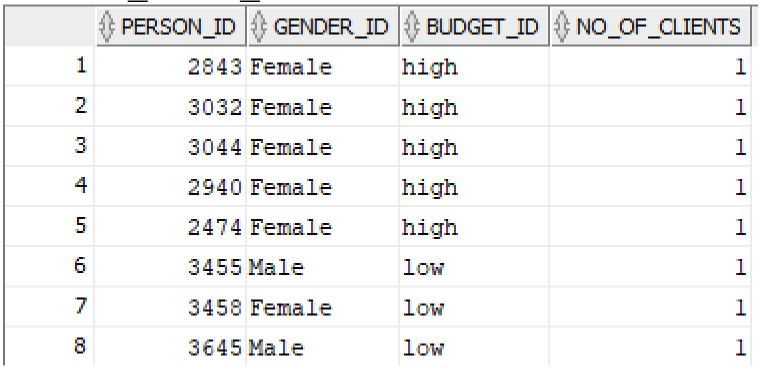
	FEATURE_TYPE_ID	PERSON_ID
1	20	5202
2	20	5205
3	20	5208
4	20	5211
5	20	5216
6	20	5225
7	20	5227

Client_dim_v1



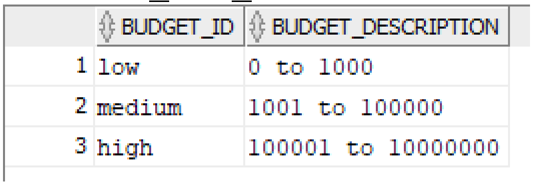
	PERSON_ID
1	3014
2	3020
3	3025
4	3029
5	3081
6	3087
7	3092
8	3098

CLIENT_FACT_V1



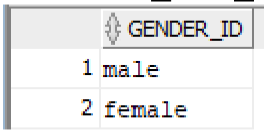
	PERSON_ID	GENDER_ID	BUDGET_ID	NO_OF_CLIENTS
1	2843	Female	high	1
2	3032	Female	high	1
3	3044	Female	high	1
4	2940	Female	high	1
5	2474	Female	high	1
6	3455	Male	low	1
7	3458	Female	low	1
8	3645	Male	low	1

BUDGET_DIM_V1



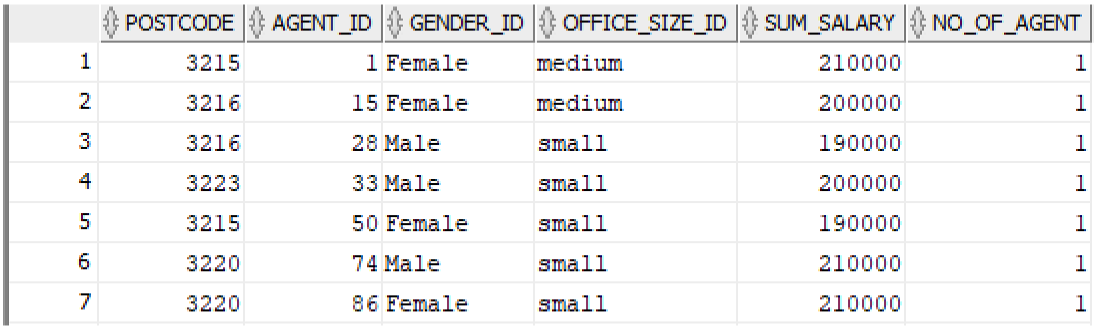
	BUDGET_ID	BUDGET_DESCRIPTION
1	low	0 to 1000
2	medium	1001 to 100000
3	high	100001 to 10000000

GENDER_DIM_V1



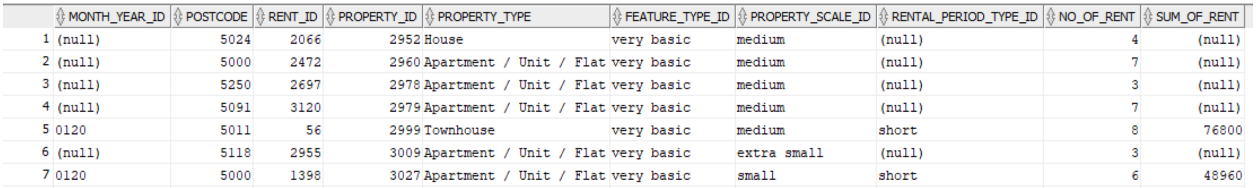
	GENDER_ID
1	male
2	female

AGENT_FACT_V1



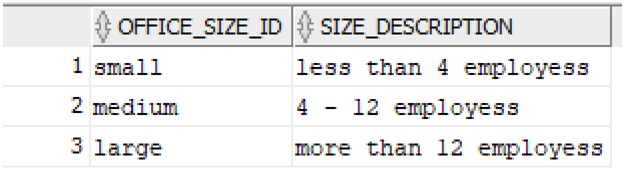
	POSTCODE	AGENT_ID	GENDER_ID	OFFICE_SIZE_ID	SUM_SALARY	NO_OF_AGENT
1	3215	1	Female	medium	210000	1
2	3216	15	Female	medium	200000	1
3	3216	28	Male	small	190000	1
4	3223	33	Male	small	200000	1
5	3215	50	Female	small	190000	1
6	3220	74	Male	small	210000	1
7	3220	86	Female	small	210000	1

RENT_FACT_V1



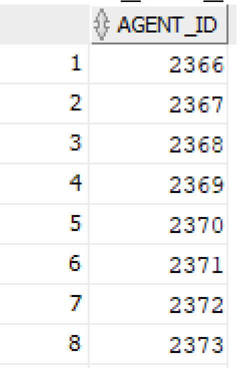
	MONTH_YEAR_ID	POSTCODE	RENT_ID	PROPERTY_ID	PROPERTY_TYPE	FEATURE_TYPE_ID	PROPERTY_SCALE_ID	RENTAL_PERIOD_TYPE_ID	NO_OF_RENT	SUM_OF_RENT
1	(null)	5024	2066	2952	House	very basic	medium	(null)	4	(null)
2	(null)	5000	2472	2960	Apartment / Unit / Flat	very basic	medium	(null)	7	(null)
3	(null)	5250	2697	2978	Apartment / Unit / Flat	very basic	medium	(null)	3	(null)
4	(null)	5091	3120	2979	Apartment / Unit / Flat	very basic	medium	(null)	7	(null)
5	0120	5011	56	2999	Townhouse	very basic	medium	short	8	76800
6	(null)	5118	2955	3009	Apartment / Unit / Flat	very basic	extra small	(null)	3	(null)
7	0120	5000	1398	3027	Apartment / Unit / Flat	very basic	small	short	6	48960

OFFICE_SIZE_DIM_V1



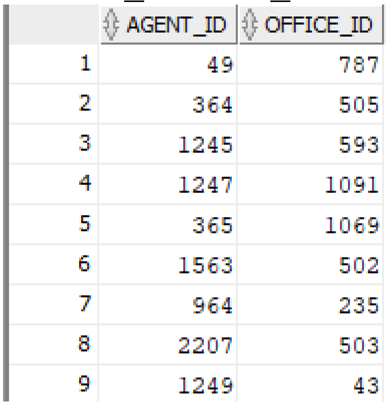
	OFFICE_SIZE_ID	SIZE_DESCRIPTION
1	small	less than 4 employess
2	medium	4 - 12 employess
3	large	more than 12 employess

AGENT_DIM_V1



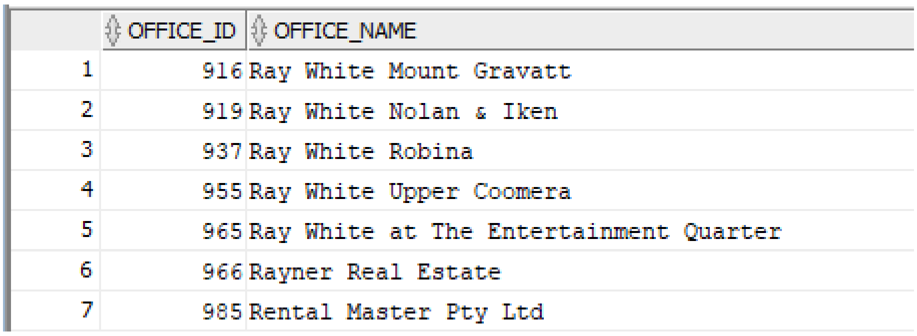
	AGENT_ID
1	2366
2	2367
3	2368
4	2369
5	2370
6	2371
7	2372
8	2373

AGENT_OFFICE_BRIDGE_DIM_V1



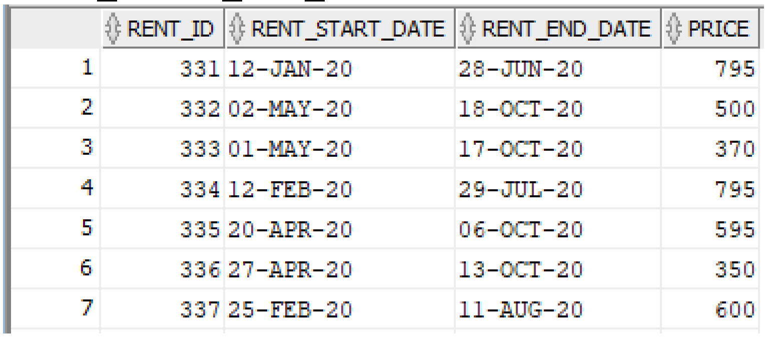
	AGENT_ID	OFFICE_ID
1	49	787
2	364	505
3	1245	593
4	1247	1091
5	365	1069
6	1563	502
7	964	235
8	2207	503
9	1249	43

OFFICE_DIM_V1



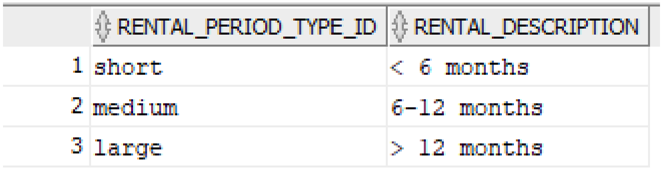
	OFFICE_ID	OFFICE_NAME
1	916	Ray White Mount Gravatt
2	919	Ray White Nolan & Iken
3	937	Ray White Robina
4	955	Ray White Upper Coomera
5	965	Ray White at The Entertainment Quarter
6	966	Rayner Real Estate
7	985	Rental Master Pty Ltd

RENT_PRICE_DIM_V1



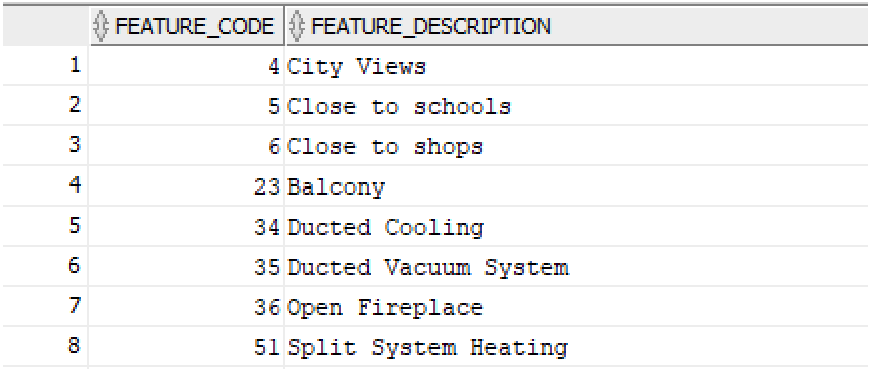
	RENT_ID	RENT_START_DATE	RENT_END_DATE	PRICE
1	331	12-JAN-20	28-JUN-20	795
2	332	02-MAY-20	18-OCT-20	500
3	333	01-MAY-20	17-OCT-20	370
4	334	12-FEB-20	29-JUL-20	795
5	335	20-APR-20	06-OCT-20	595
6	336	27-APR-20	13-OCT-20	350
7	337	25-FEB-20	11-AUG-20	600

RENTAL_PERIOD_DIM_V1



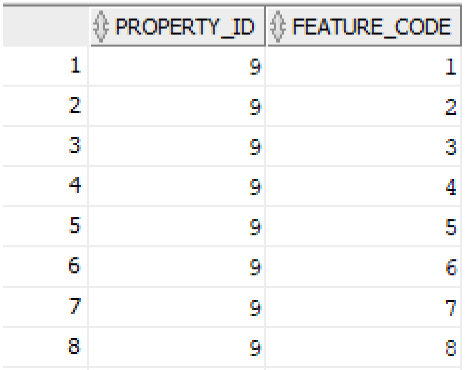
	RENTAL_PERIOD_TYPE_ID	RENTAL_DESCRIPTION
1	short	< 6 months
2	medium	6-12 months
3	large	> 12 months

FEATURE_DIM_V1



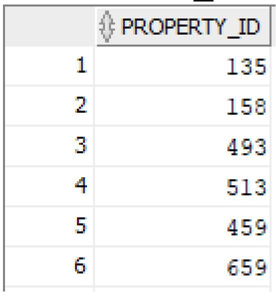
	FEATURE_CODE	FEATURE_DESCRIPTION
1	4	City Views
2	5	Close to schools
3	6	Close to shops
4	23	Balcony
5	34	Ducted Cooling
6	35	Ducted Vacuum System
7	36	Open Fireplace
8	51	Split System Heating

PROPERTY_FEATURE_BRIDGE_DIM_V1



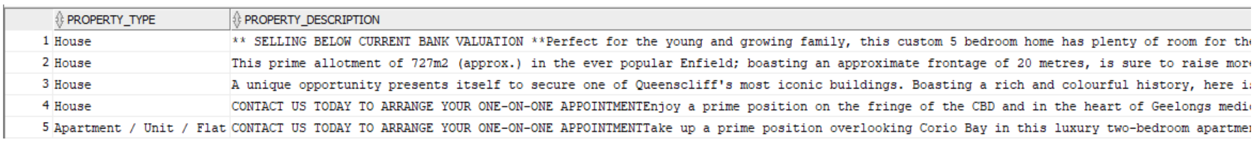
	PROPERTY_ID	FEATURE_CODE
1	9	1
2	9	2
3	9	3
4	9	4
5	9	5
6	9	6
7	9	7
8	9	8

PROPERTY_DIM_V1



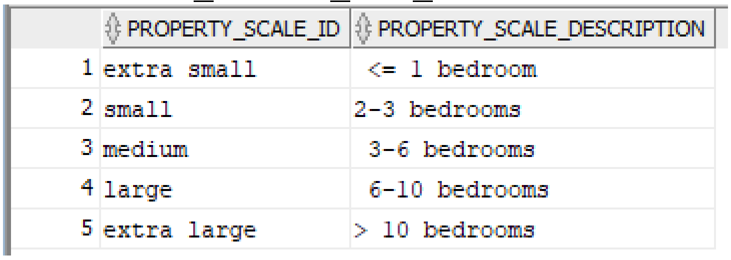
	PROPERTY_ID
1	135
2	158
3	493
4	513
5	459
6	659

PROPERTY_TYPE_DIM_V1



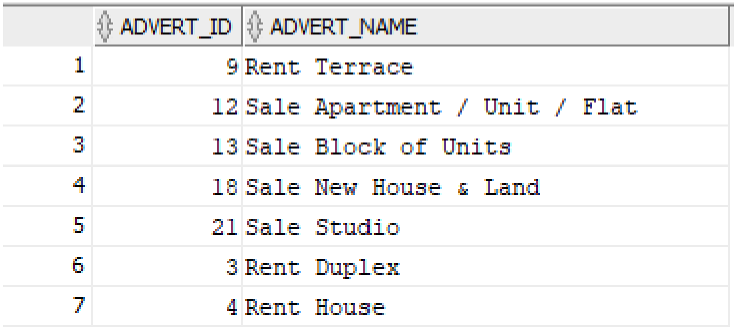
	PROPERTY_TYPE	PROPERTY_DESCRIPTION
1	House	** SELLING BELOW CURRENT BANK VALUATION **Perfect for the young and growing family, this custom 5 bedroom home has plenty of room for th
2	House	This prime allotment of 727m2 (approx.) in the ever popular Enfield; boasting an approximate frontage of 20 metres, is sure to raise mor
3	House	A unique opportunity presents itself to secure one of Queenscliff's most iconic buildings. Boasting a rich and colourful history, here i
4	House	CONTACT US TODAY TO ARRANGE YOUR ONE-ON-ONE APPOINTMENTEnjoy a prime position on the fringe of the CBD and in the heart of Geelongs medi
5	Apartment / Unit / Flat	CONTACT US TODAY TO ARRANGE YOUR ONE-ON-ONE APPOINTMENTTake up a prime position overlooking Corio Bay in this luxury two-bedroom apartme

PROPERTY_SCALE_DIM_V1



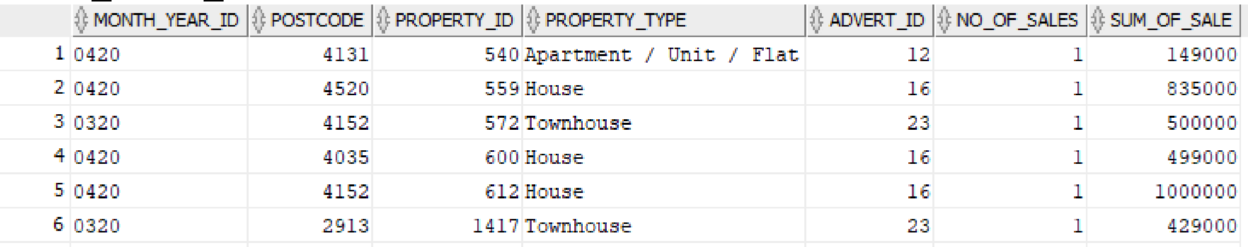
	PROPERTY_SCALE_ID	PROPERTY_SCALE_DESCRIPTION
1	extra small	<= 1 bedroom
2	small	2-3 bedrooms
3	medium	3-6 bedrooms
4	large	6-10 bedrooms
5	extra large	> 10 bedrooms

ADVERT_DIM_V1



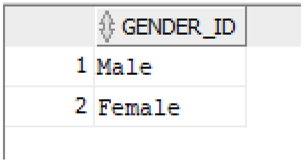
	ADVERT_ID	ADVERT_NAME
1	9	Rent Terrace
2	12	Sale Apartment / Unit / Flat
3	13	Sale Block of Units
4	18	Sale New House & Land
5	21	Sale Studio
6	3	Rent Duplex
7	4	Rent House

SALE_FACT_V1



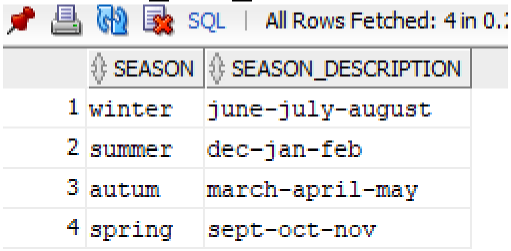
	MONTH_YEAR_ID	POSTCODE	PROPERTY_ID	PROPERTY_TYPE	ADVERT_ID	NO_OF_SALES	SUM_OF_SALE
1	0420	4131	540	Apartment / Unit / Flat	12	1	149000
2	0420	4520	559	House	16	1	835000
3	0320	4152	572	Townhouse	23	1	500000
4	0420	4035	600	House	16	1	499000
5	0420	4152	612	House	16	1	1000000
6	0320	2913	1417	Townhouse	23	1	429000

LEVEL 0 TABLES (VERSION 2)
GENDER_DIM_V2



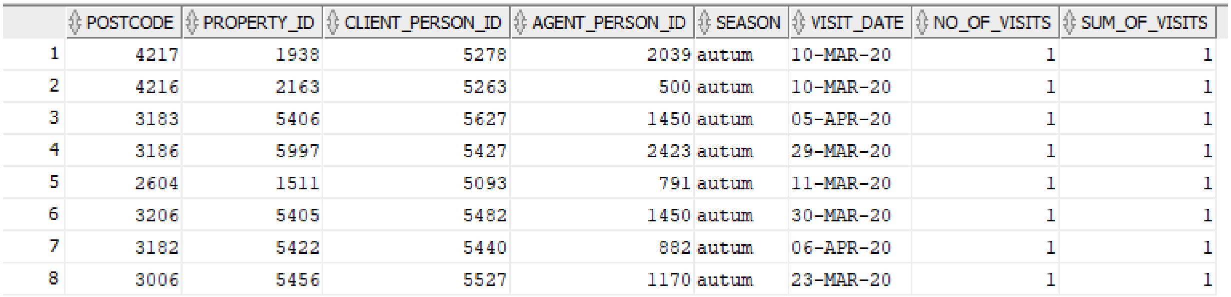
	GENDER_ID
1	Male
2	Female

SEASON_DIM_V2



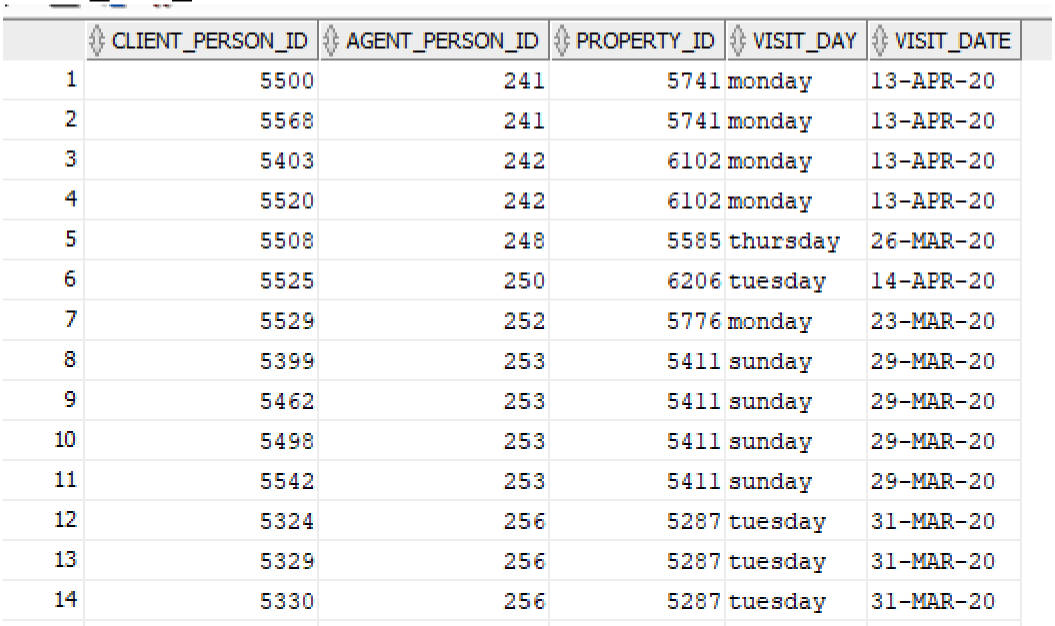
	SEASON	SEASON_DESCRIPTION
1	winter	june-july-august
2	summer	dec-jan-feb
3	autum	march-april-may
4	spring	sept-oct-nov

PROPERTY_VISIT_FACT_V2



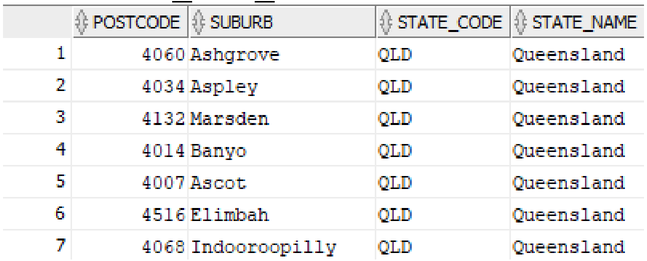
	POSTCODE	PROPERTY_ID	CLIENT_PERSON_ID	AGENT_PERSON_ID	SEASON	VISIT_DATE	NO_OF_VISITS	SUM_OF_VISITS
1	4217	1938	5278	2039	autum	10-MAR-20	1	1
2	4216	2163	5263	500	autum	10-MAR-20	1	1
3	3183	5406	5627	1450	autum	05-APR-20	1	1
4	3186	5997	5427	2423	autum	29-MAR-20	1	1
5	2604	1511	5093	791	autum	11-MAR-20	1	1
6	3206	5405	5482	1450	autum	30-MAR-20	1	1
7	3182	5422	5440	882	autum	06-APR-20	1	1
8	3006	5456	5527	1170	autum	23-MAR-20	1	1

VISIT_DIM_V2



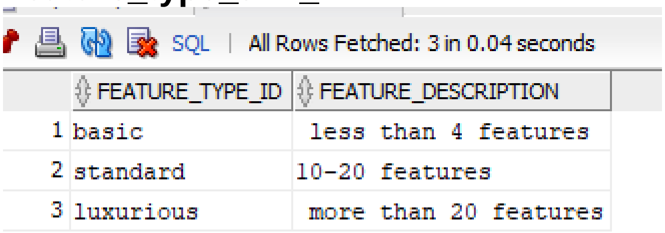
	CLIENT_PERSON_ID	AGENT_PERSON_ID	PROPERTY_ID	VISIT_DAY	VISIT_DATE
1	5500	241	5741	monday	13-APR-20
2	5568	241	5741	monday	13-APR-20
3	5403	242	6102	monday	13-APR-20
4	5520	242	6102	monday	13-APR-20
5	5508	248	5585	thursday	26-MAR-20
6	5525	250	6206	tuesday	14-APR-20
7	5529	252	5776	monday	23-MAR-20
8	5399	253	5411	sunday	29-MAR-20
9	5462	253	5411	sunday	29-MAR-20
10	5498	253	5411	sunday	29-MAR-20
11	5542	253	5411	sunday	29-MAR-20
12	5324	256	5287	tuesday	31-MAR-20
13	5329	256	5287	tuesday	31-MAR-20
14	5330	256	5287	tuesday	31-MAR-20

ADDRESS_DIM_V2



	POSTCODE	SUBURB	STATE_CODE	STATE_NAME
1	4060	Ashgrove	QLD	Queensland
2	4034	Aspley	QLD	Queensland
3	4132	Marsden	QLD	Queensland
4	4014	Banyo	QLD	Queensland
5	4007	Ascot	QLD	Queensland
6	4516	Elimbah	QLD	Queensland
7	4068	Indooroopilly	QLD	Queensland

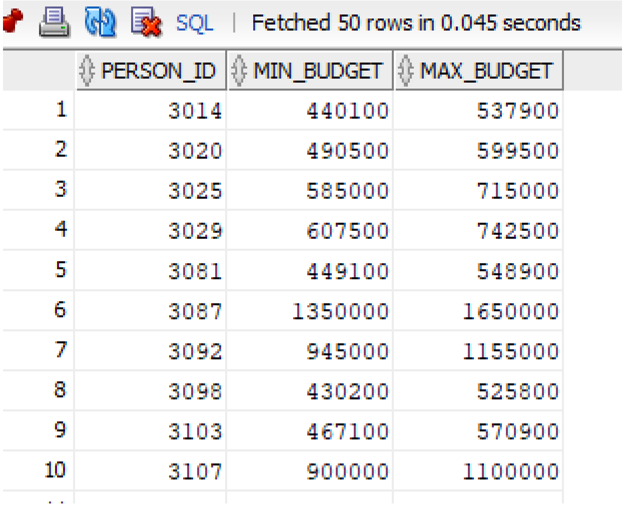
Feature_type_dim_v2



SQL | All Rows Fetched: 3 in 0.04 seconds

	FEATURE_TYPE_ID	FEATURE_DESCRIPTION
1	basic	less than 4 features
2	standard	10-20 features
3	luxurious	more than 20 features

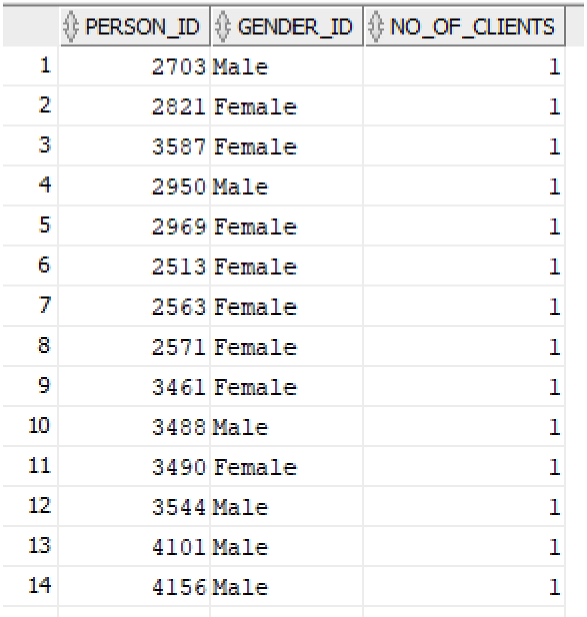
CLIENT_DIM_V2



SQL | Fetched 50 rows in 0.045 seconds

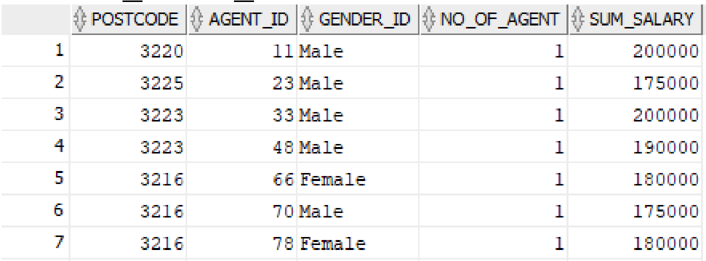
	PERSON_ID	MIN_BUDGET	MAX_BUDGET
1	3014	440100	537900
2	3020	490500	599500
3	3025	585000	715000
4	3029	607500	742500
5	3081	449100	548900
6	3087	1350000	1650000
7	3092	945000	1155000
8	3098	430200	525800
9	3103	467100	570900
10	3107	900000	1100000

CLIENT_FACT_V2



	PERSON_ID	GENDER_ID	NO_OF_CLIENTS
1	2703	Male	1
2	2821	Female	1
3	3587	Female	1
4	2950	Male	1
5	2969	Female	1
6	2513	Female	1
7	2563	Female	1
8	2571	Female	1
9	3461	Female	1
10	3488	Male	1
11	3490	Female	1
12	3544	Male	1
13	4101	Male	1
14	4156	Male	1

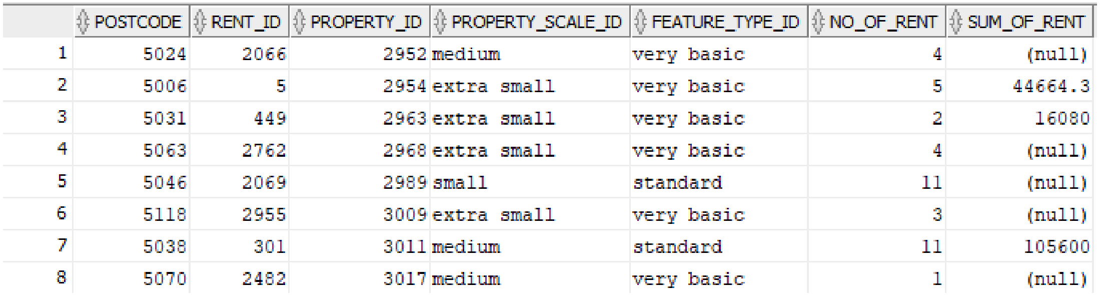
AGENT_FACT_V2



	POSTCODE	AGENT_ID	GENDER_ID	NO_OF_AGENT	SUM_SALARY
1	3220	11	Male	1	200000
2	3225	23	Male	1	175000
3	3223	33	Male	1	200000
4	3223	48	Male	1	190000
5	3216	66	Female	1	180000
6	3216	70	Male	1	175000
7	3216	78	Female	1	180000

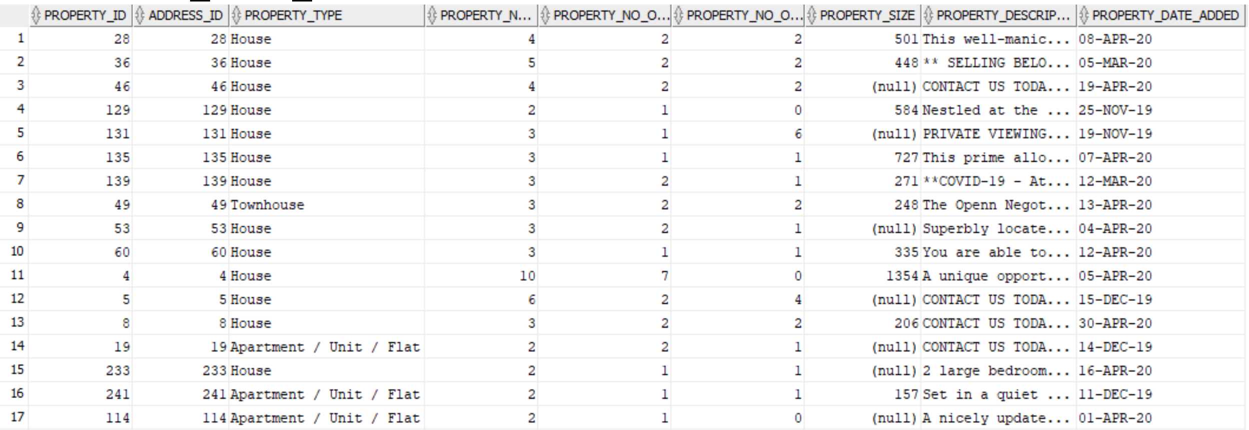
MONASH University
Information Technology

RENT_FACT_V2



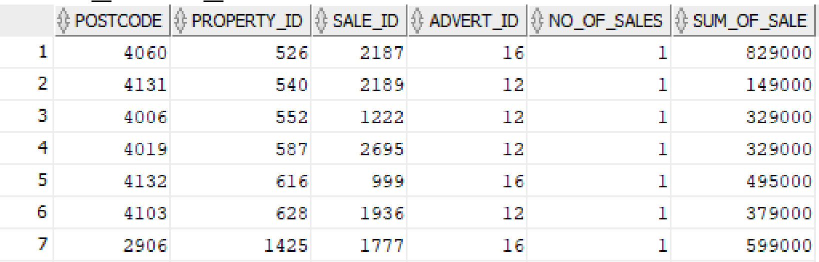
	POSTCODE	RENT_ID	PROPERTY_ID	PROPERTY_SCALE_ID	FEATURE_TYPE_ID	NO_OF_RENT	SUM_OF_RENT
1	5024	2066	2952	medium	very basic	4	(null)
2	5006	5	2954	extra small	very basic	5	44664.3
3	5031	449	2963	extra small	very basic	2	16080
4	5063	2762	2968	extra small	very basic	4	(null)
5	5046	2069	2989	small	standard	11	(null)
6	5118	2955	3009	extra small	very basic	3	(null)
7	5038	301	3011	medium	standard	11	105600
8	5070	2482	3017	medium	very basic	1	(null)

PROPERTY_TIME_V2



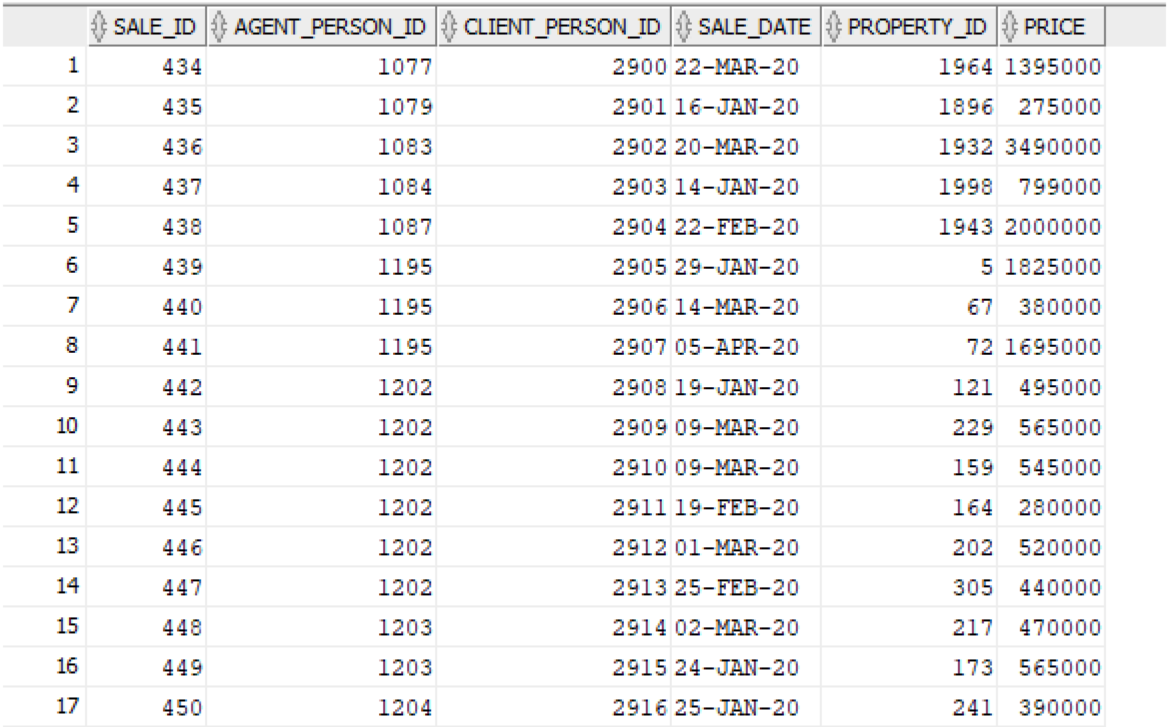
	PROPERTY_ID	ADDRESS_ID	PROPERTY_TYPE	PROPERTY_N...	PROPERTY_NO_O...	PROPERTY_NO_O...	PROPERTY_SIZE	PROPERTY_DESCRIP...	PROPERTY_DATE_ADDED
1	28	28	House	4	2	2	501	This well-manic...	08-APR-20
2	36	36	House	5	2	2	448	** SELLING BELO...	05-MAR-20
3	46	46	House	4	2	2	(null)	CONTACT US TODA...	19-APR-20
4	129	129	House	2	1	0	584	Nestled at the ...	25-NOV-19
5	131	131	House	3	1	6	(null)	PRIVATE VIEWING...	19-NOV-19
6	135	135	House	3	1	1	727	This prime allo...	07-APR-20
7	139	139	House	3	2	1	271	**COVID-19 - At...	12-MAR-20
8	49	49	Townhouse	3	2	2	248	The Openn Negot...	13-APR-20
9	53	53	House	3	2	1	(null)	Superbly locate...	04-APR-20
10	60	60	House	3	1	1	335	You are able to...	12-APR-20
11	4	4	House	10	7	0	1354	A unique opport...	05-APR-20
12	5	5	House	6	2	4	(null)	CONTACT US TODA...	15-DEC-19
13	8	8	House	3	2	2	206	CONTACT US TODA...	30-APR-20
14	19	19	Apartment / Unit / Flat	2	2	1	(null)	CONTACT US TODA...	14-DEC-19
15	233	233	House	2	1	1	(null)	2 large bedroom...	16-APR-20
16	241	241	Apartment / Unit / Flat	2	1	1	157	Set in a quiet ...	11-DEC-19
17	114	114	Apartment / Unit / Flat	2	1	0	(null)	A nicely update...	01-APR-20

SALE_FACT_V2



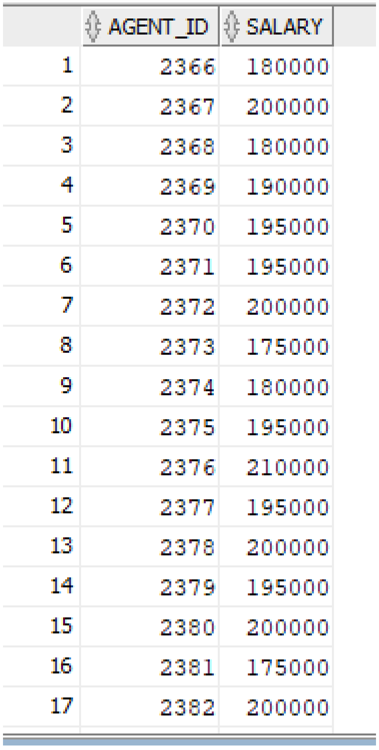
	POSTCODE	PROPERTY_ID	SALE_ID	ADVERT_ID	NO_OF_SALES	SUM_OF_SALE
1	4060	526	2187	16	1	829000
2	4131	540	2189	12	1	149000
3	4006	552	1222	12	1	329000
4	4019	587	2695	12	1	329000
5	4132	616	999	16	1	495000
6	4103	628	1936	12	1	379000
7	2906	1425	1777	16	1	599000

SALE_DIM_V2



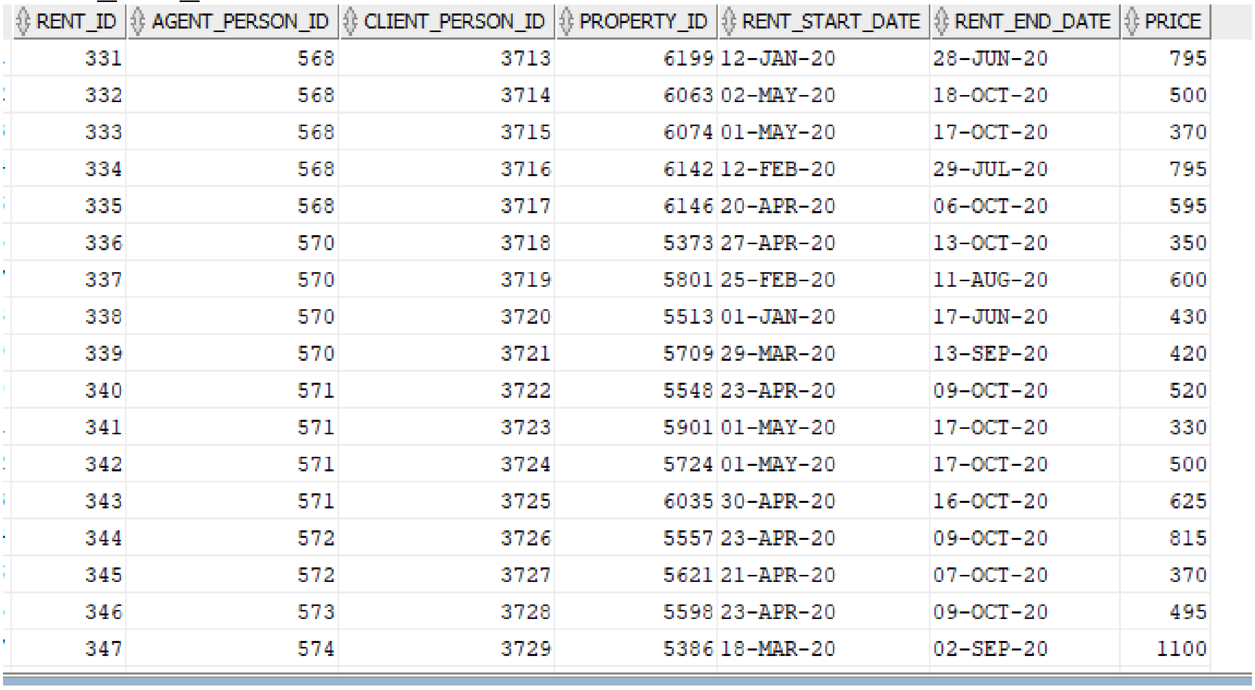
	SALE_ID	AGENT_PERSON_ID	CLIENT_PERSON_ID	SALE_DATE	PROPERTY_ID	PRICE
1	434	1077	2900	22-MAR-20	1964	1395000
2	435	1079	2901	16-JAN-20	1896	275000
3	436	1083	2902	20-MAR-20	1932	3490000
4	437	1084	2903	14-JAN-20	1998	799000
5	438	1087	2904	22-FEB-20	1943	2000000
6	439	1195	2905	29-JAN-20	5	1825000
7	440	1195	2906	14-MAR-20	67	380000
8	441	1195	2907	05-APR-20	72	1695000
9	442	1202	2908	19-JAN-20	121	495000
10	443	1202	2909	09-MAR-20	229	565000
11	444	1202	2910	09-MAR-20	159	545000
12	445	1202	2911	19-FEB-20	164	280000
13	446	1202	2912	01-MAR-20	202	520000
14	447	1202	2913	25-FEB-20	305	440000
15	448	1203	2914	02-MAR-20	217	470000
16	449	1203	2915	24-JAN-20	173	565000
17	450	1204	2916	25-JAN-20	241	390000

AGENT_DIM_V2



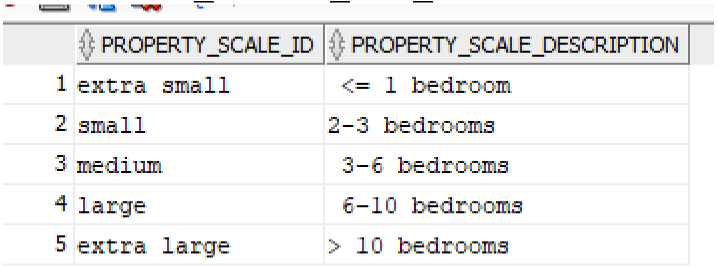
	AGENT_ID	SALARY
1	2366	180000
2	2367	200000
3	2368	180000
4	2369	190000
5	2370	195000
6	2371	195000
7	2372	200000
8	2373	175000
9	2374	180000
10	2375	195000
11	2376	210000
12	2377	195000
13	2378	200000
14	2379	195000
15	2380	200000
16	2381	175000
17	2382	200000

RENT_DIM_V2



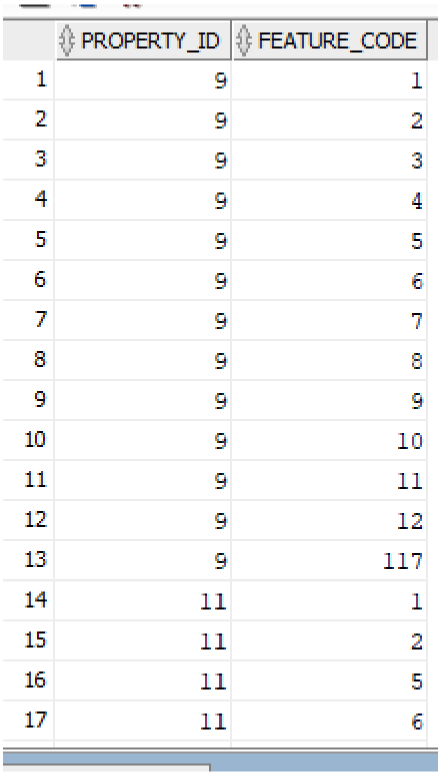
RENT_ID	AGENT_PERSON_ID	CLIENT_PERSON_ID	PROPERTY_ID	RENT_START_DATE	RENT_END_DATE	PRICE
331	568	3713	6199	12-JAN-20	28-JUN-20	795
332	568	3714	6063	02-MAY-20	18-OCT-20	500
333	568	3715	6074	01-MAY-20	17-OCT-20	370
334	568	3716	6142	12-FEB-20	29-JUL-20	795
335	568	3717	6146	20-APR-20	06-OCT-20	595
336	570	3718	5373	27-APR-20	13-OCT-20	350
337	570	3719	5801	25-FEB-20	11-AUG-20	600
338	570	3720	5513	01-JAN-20	17-JUN-20	430
339	570	3721	5709	29-MAR-20	13-SEP-20	420
340	571	3722	5548	23-APR-20	09-OCT-20	520
341	571	3723	5901	01-MAY-20	17-OCT-20	330
342	571	3724	5724	01-MAY-20	17-OCT-20	500
343	571	3725	6035	30-APR-20	16-OCT-20	625
344	572	3726	5557	23-APR-20	09-OCT-20	815
345	572	3727	5621	21-APR-20	07-OCT-20	370
346	573	3728	5598	23-APR-20	09-OCT-20	495
347	574	3729	5386	18-MAR-20	02-SEP-20	1100

PROPERTY_SCALE_DIM_V2



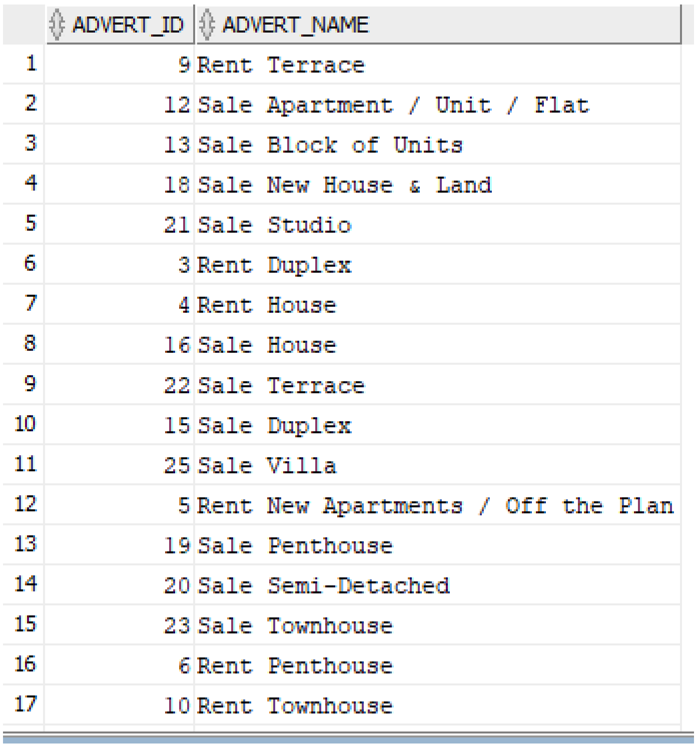
	PROPERTY_SCALE_ID	PROPERTY_SCALE_DESCRIPTION
1	extra small	<= 1 bedroom
2	small	2-3 bedrooms
3	medium	3-6 bedrooms
4	large	6-10 bedrooms
5	extra large	> 10 bedrooms

PROPERTY_FEATURE_BRIDGE_DIM_V2



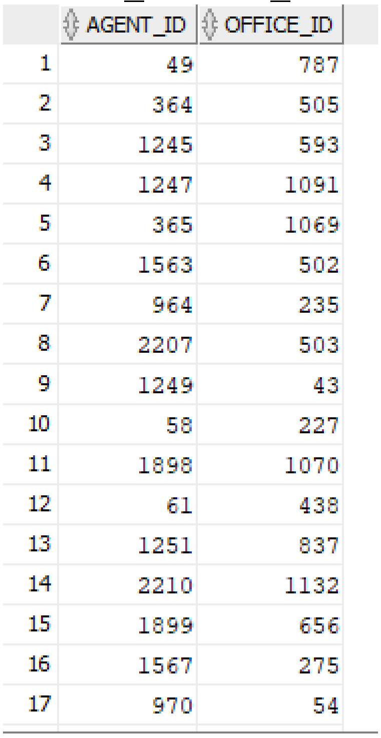
	PROPERTY_ID	FEATURE_CODE
1	9	1
2	9	2
3	9	3
4	9	4
5	9	5
6	9	6
7	9	7
8	9	8
9	9	9
10	9	10
11	9	11
12	9	12
13	9	117
14	11	1
15	11	2
16	11	5
17	11	6

ADVERT_DIM_V2



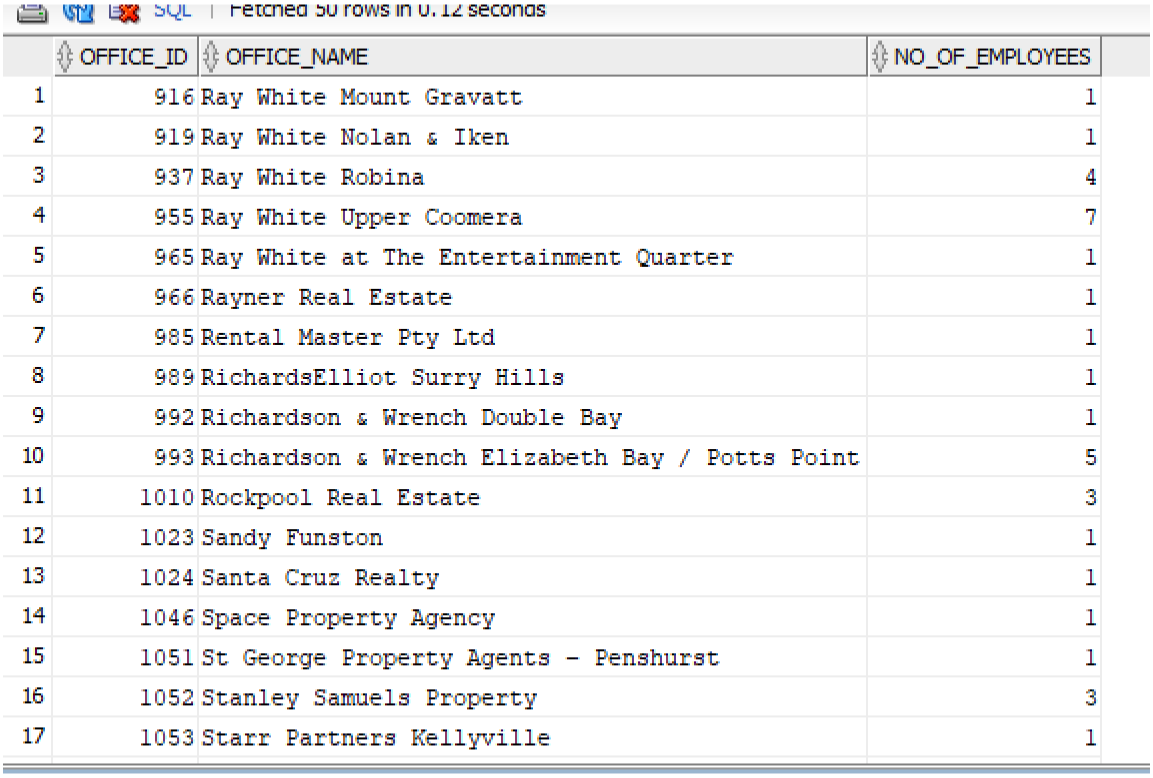
	ADVERT_ID	ADVERT_NAME
1	9	Rent Terrace
2	12	Sale Apartment / Unit / Flat
3	13	Sale Block of Units
4	18	Sale New House & Land
5	21	Sale Studio
6	3	Rent Duplex
7	4	Rent House
8	16	Sale House
9	22	Sale Terrace
10	15	Sale Duplex
11	25	Sale Villa
12	5	Rent New Apartments / Off the Plan
13	19	Sale Penthouse
14	20	Sale Semi-Detached
15	23	Sale Townhouse
16	6	Rent Penthouse
17	10	Rent Townhouse

AGENT_OFFICE_BRIDGE_DIM_V2



	AGENT_ID	OFFICE_ID
1	49	787
2	364	505
3	1245	593
4	1247	1091
5	365	1069
6	1563	502
7	964	235
8	2207	503
9	1249	43
10	58	227
11	1898	1070
12	61	438
13	1251	837
14	2210	1132
15	1899	656
16	1567	275
17	970	54

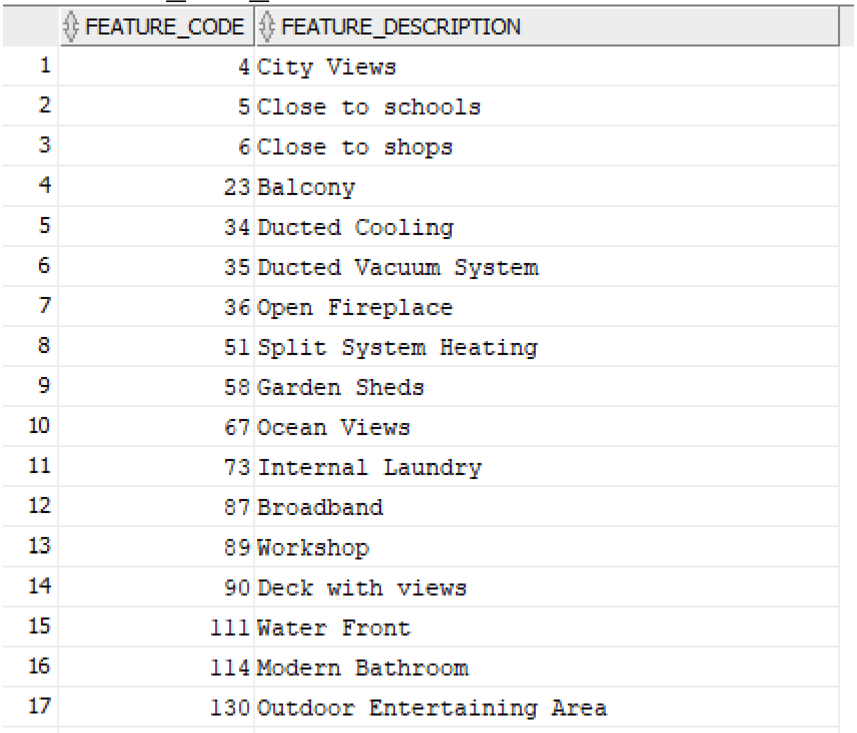
OFFICE_DIM_V2



SQL | Fetched 50 rows in 0.12 seconds

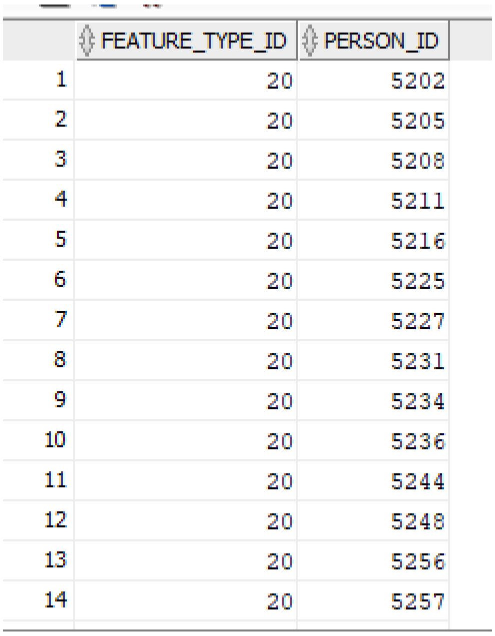
	OFFICE_ID	OFFICE_NAME	NO_OF_EMPLOYEES
1	916	Ray White Mount Gravatt	1
2	919	Ray White Nolan & Iken	1
3	937	Ray White Robina	4
4	955	Ray White Upper Coomera	7
5	965	Ray White at The Entertainment Quarter	1
6	966	Rayner Real Estate	1
7	985	Rental Master Pty Ltd	1
8	989	RichardsElliot Surry Hills	1
9	992	Richardson & Wrench Double Bay	1
10	993	Richardson & Wrench Elizabeth Bay / Potts Point	5
11	1010	Rockpool Real Estate	3
12	1023	Sandy Funston	1
13	1024	Santa Cruz Realty	1
14	1046	Space Property Agency	1
15	1051	St George Property Agents - Penshurst	1
16	1052	Stanley Samuels Property	3
17	1053	Starr Partners Kellyville	1

FEATURE_DIM_V2



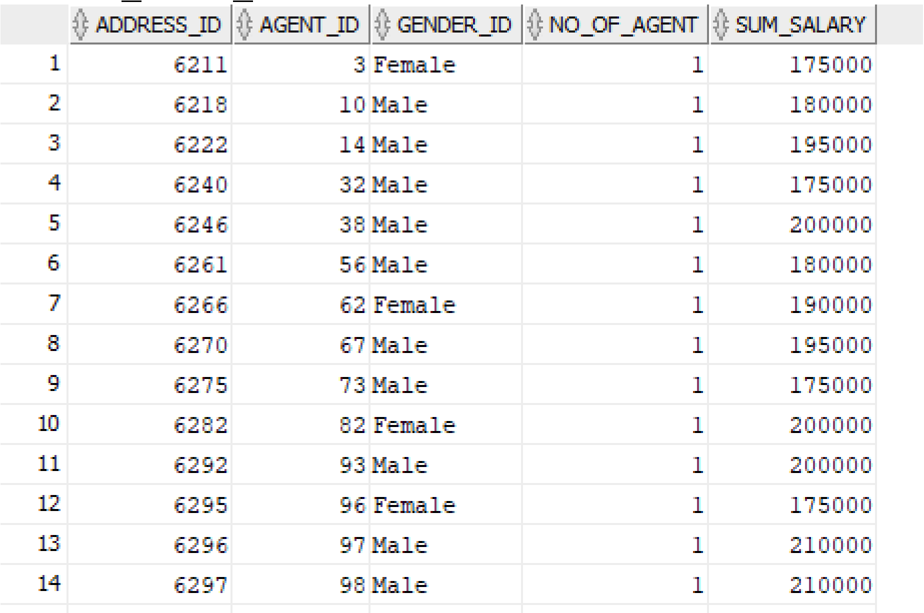
	FEATURE_CODE	FEATURE_DESCRIPTION
1	4	City Views
2	5	Close to schools
3	6	Close to shops
4	23	Balcony
5	34	Ducted Cooling
6	35	Ducted Vacuum System
7	36	Open Fireplace
8	51	Split System Heating
9	58	Garden Sheds
10	67	Ocean Views
11	73	Internal Laundry
12	87	Broadband
13	89	Workshop
14	90	Deck with views
15	111	Water Front
16	114	Modern Bathroom
17	130	Outdoor Entertaining Area

CLIENT_FEATURE_BRIDGE_DIM_V2



	FEATURE_TYPE_ID	PERSON_ID
1	20	5202
2	20	5205
3	20	5208
4	20	5211
5	20	5216
6	20	5225
7	20	5227
8	20	5231
9	20	5234
10	20	5236
11	20	5244
12	20	5248
13	20	5256
14	20	5257

AGENT_FACT_V2



	ADDRESS_ID	AGENT_ID	GENDER_ID	NO_OF_AGENT	SUM_SALARY
1	6211	3	Female	1	175000
2	6218	10	Male	1	180000
3	6222	14	Male	1	195000
4	6240	32	Male	1	175000
5	6246	38	Male	1	200000
6	6261	56	Male	1	180000
7	6266	62	Female	1	190000
8	6270	67	Male	1	195000
9	6275	73	Male	1	175000
10	6282	82	Female	1	200000
11	6292	93	Male	1	200000
12	6295	96	Female	1	175000
13	6296	97	Male	1	210000
14	6297	98	Male	1	210000

Task 3 out of 4

VERSION 1 -

a) Simple Reports

a) Query questions in English -

Report 1 - top 5 Sum of rent for each feature type and property scale

Report 2 - top 5% sales in each location and time period

Report 3 - number of visits in each season in each day

b) Explanation why it is important to management

Query/Report 1 can be used by the management to get the top sum of rents for the combinations of feature type and scale of the property. Using this information the management can know which top 5 combinations give the most number of rents and can use that combination to achieve maximum rents. Hence feature count and scale of property can be useful in determining the case of which combination produces the most rent.

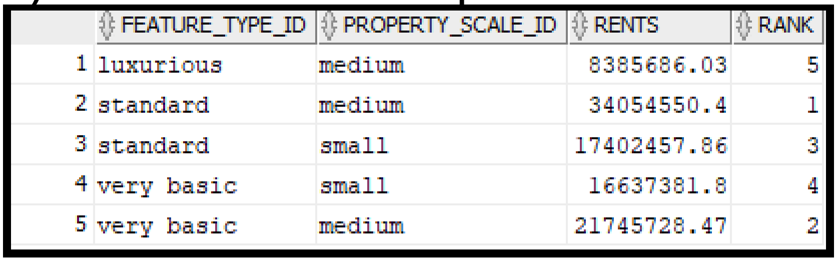
Query/Report 2 can be used by the management to get the best locations and time period where sales were maximum. It can help the management to determine any relationships between the two factors location and time period. Hence the management can do further analysis about which region/location has the most sales based on the different months/time period. It will hence help the management to make decisions as to when to advertise more in which area so that maximum sales are seen.

Query/Report 3 can be used by the management to determine which days and which seasons see the most number of visits for a property. This can help the management to focus on certain seasons and certain days and find which best combination sees the most number of visits. It would help the management to determine which combination has most number of visits and hence they can use that information to arrange more inspection times during the most popular combinations.

c) SQL Commands - Version 1 Report 1

```
--VERSION 1 REPORT 1 top 5 Sum of rent for each feature type and property scale
SELECT *
FROM
(SELECT FEATURE_TYPE_ID,PROPERTY_SCALE_ID,sum(SUM_OF_RENT) as RENTS,
RANK() OVER (ORDER BY  sum(SUM_OF_RENT)  DESC ) AS RANK
FROM rent_fact_v1
GROUP BY FEATURE_TYPE_ID,PROPERTY_SCALE_ID
having sum(SUM_OF_RENT) > 0
order by FEATURE_TYPE_ID)
WHERE RANK <= 5;
```

d.) Screenshots- Version 1 Report 1

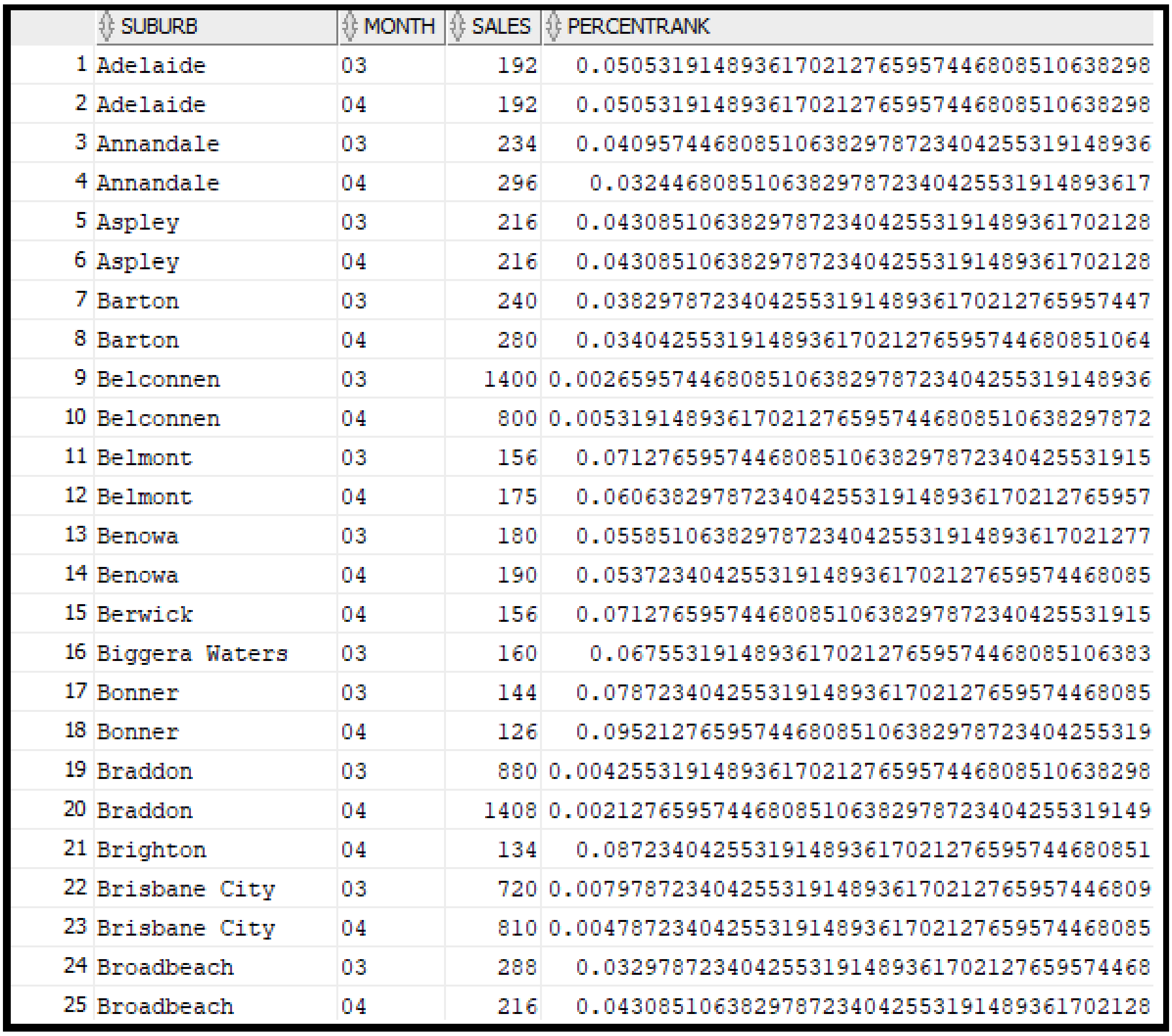


	FEATURE_TYPE_ID	PROPERTY_SCALE_ID	RENTS	RANK
1	luxurious	medium	8385686.03	5
2	standard	medium	34054550.4	1
3	standard	small	17402457.86	3
4	very basic	small	16637381.8	4
5	very basic	medium	21745728.47	2

c) SQL Commands - Version 1 Report 2

```
--VERSION 1 REPORT 2 top 5% sales in each location and time period


SELECT *
FROM (
SELECT
l.suburb as SUBURB,
t.Month as MONTH, sum(f.no_of_sales) AS SALES,
percent_rank() over
(order by sum(f.no_of_sales) desc) as PercentRank
FROM sale_fact_v1 f ,time_dim_v1 t, location_dim_v1 l
WHERE f.month_year_id = t.month_year_id and
f.postcode = l.postcode
GROUP BY l.suburb ,t.Month
having sum(f.no_of_sales) > 0
order by l.suburb
) WHERE PercentRank < 0.1;
```

d.) Screenshots- Version 1 Report 2

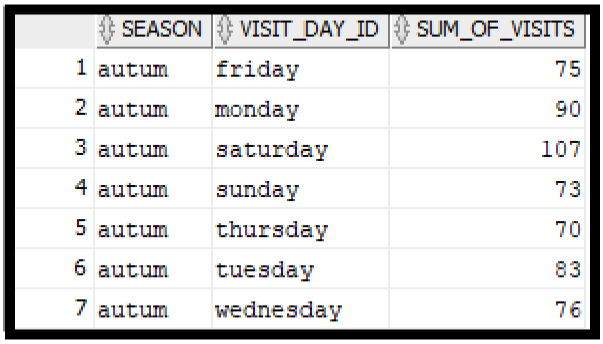


	SUBURB	MONTH	SALES	PERCENTRANK
1	Adelaide	03	192	0.0505319148936170212765957446808510638298
2	Adelaide	04	192	0.0505319148936170212765957446808510638298
3	Annandale	03	234	0.0409574468085106382978723404255319148936
4	Annandale	04	296	0.032446808510638297872340425531914893617
5	Aspley	03	216	0.0430851063829787234042553191489361702128
6	Aspley	04	216	0.0430851063829787234042553191489361702128
7	Barton	03	240	0.0382978723404255319148936170212765957447
8	Barton	04	280	0.0340425531914893617021276595744680851064
9	Belconnen	03	1400	0.002659574468085106382978723404255319148936
10	Belconnen	04	800	0.005319148936170212765957446808510638297872
11	Belmont	03	156	0.0712765957446808510638297872340425531915
12	Belmont	04	175	0.0606382978723404255319148936170212765957
13	Benowa	03	180	0.0558510638297872340425531914893617021277
14	Benowa	04	190	0.0537234042553191489361702127659574468085
15	Berwick	04	156	0.0712765957446808510638297872340425531915
16	Biggera Waters	03	160	0.067553191489361702127659574468085106383
17	Bonner	03	144	0.0787234042553191489361702127659574468085
18	Bonner	04	126	0.0952127659574468085106382978723404255319
19	Braddon	03	880	0.004255319148936170212765957446808510638298
20	Braddon	04	1408	0.002127659574468085106382978723404255319149
21	Brighton	04	134	0.0872340425531914893617021276595744680851
22	Brisbane City	03	720	0.007978723404255319148936170212765957446809
23	Brisbane City	04	810	0.004787234042553191489361702127659574468085
24	Broadbeach	03	288	0.0329787234042553191489361702127659574468
25	Broadbeach	04	216	0.0430851063829787234042553191489361702128

c) SQL Commands - Version 1 Report 3

```
--VERSION 1 REPORT 3 number of visits in each season in each day
select SEASON,VISIT_DAY_ID, sum(no_of_visits) as sum_of_visits
from property_visit_fact_v1
group by SEASON,VISIT_DAY_ID
order by season,VISIT_DAY_ID;
```

d.) Screenshots- Version 1 Report 3



	SEASON	VISIT_DAY_ID	SUM_OF_VISITS
1	autum	friday	75
2	autum	monday	90
3	autum	saturday	107
4	autum	sunday	73
5	autum	thursday	70
6	autum	tuesday	83
7	autum	wednesday	76

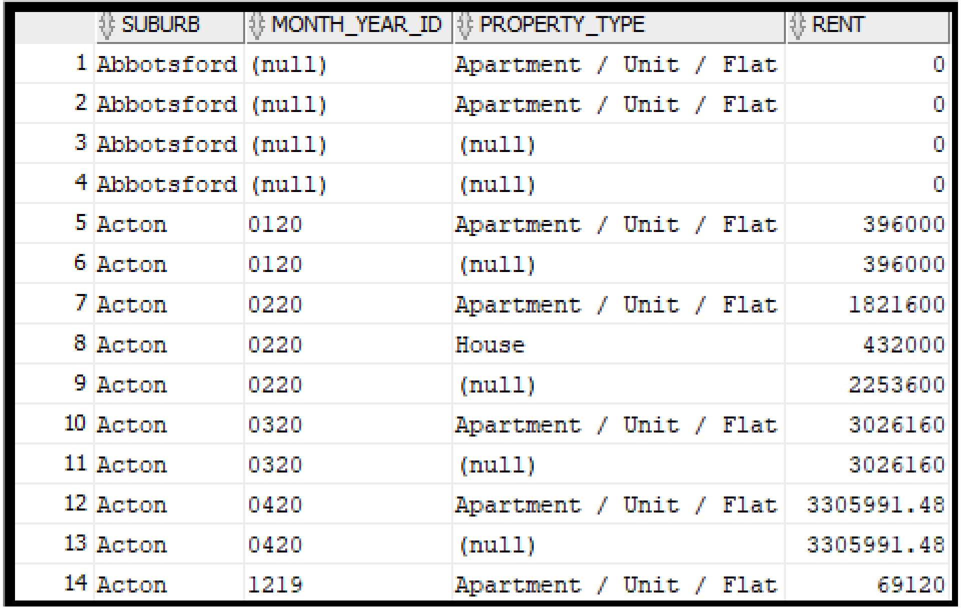
b) Reports with proper sub-totals:

REPORT 4

SQL-

```
--VERSION 1 REPORT 4
--the sub-total and total rental fees from each suburb, time period, and property type using cube
SELECT l.suburb, f.month_year_id ,f.property_type , NVL(SUM(sum_of_rent),0)
as RENT
FROM rent_fact_v1 f,location_dim_v1 l
WHERE f.postcode = l.postcode
GROUP BY CUBE(l.suburb, f.month_year_id ,f.property_type)
order by l.suburb;
```

Screenshot Version 1 Report 4



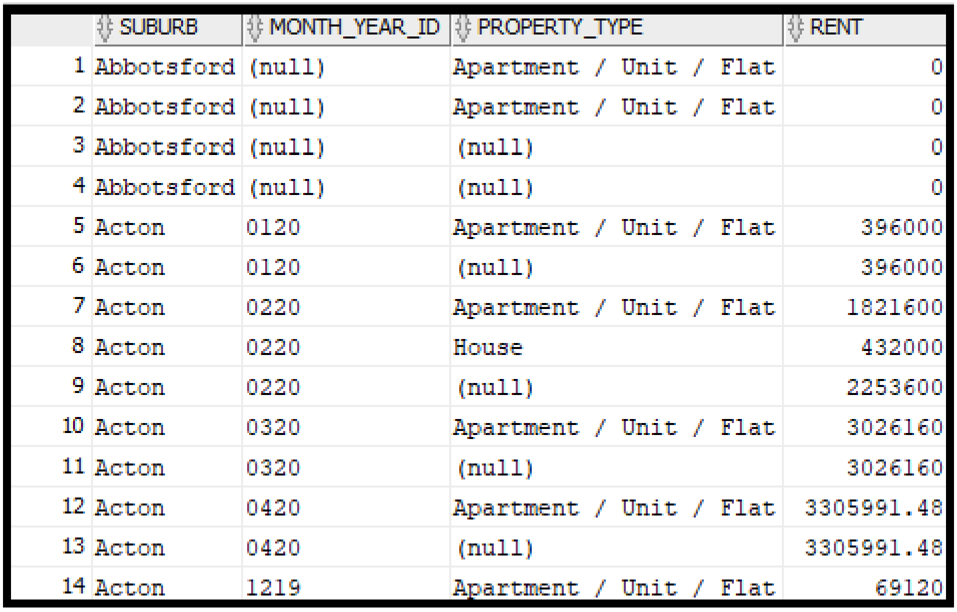
	SUBURB	MONTH_YEAR_ID	PROPERTY_TYPE	RENT
1	Abbotsford	(null)	Apartment / Unit / Flat	0
2	Abbotsford	(null)	Apartment / Unit / Flat	0
3	Abbotsford	(null)	(null)	0
4	Abbotsford	(null)	(null)	0
5	Acton	0120	Apartment / Unit / Flat	396000
6	Acton	0120	(null)	396000
7	Acton	0220	Apartment / Unit / Flat	1821600
8	Acton	0220	House	432000
9	Acton	0220	(null)	2253600
10	Acton	0320	Apartment / Unit / Flat	3026160
11	Acton	0320	(null)	3026160
12	Acton	0420	Apartment / Unit / Flat	3305991.48
13	Acton	0420	(null)	3305991.48
14	Acton	1219	Apartment / Unit / Flat	69120

REPORT 5

SQL-

```
--VERSION 1 REPORT 5
----the sub-total and total rental fees from each suburb, time period, and property type using partial cube

SELECT l.suburb, f.month_year_id ,f.property_type , NVL(SUM(sum_of_rent),0)
as RENT
FROM rent_fact_v1 f,location_dim_v1 l
WHERE f.postcode = l.postcode
GROUP BY l.suburb , CUBE(f.month_year_id ,f.property_type)
order by l.suburb;
```

Screenshot Version 1 Report 5



	SUBURB	MONTH_YEAR_ID	PROPERTY_TYPE	RENT
1	Abbotsford	(null)	Apartment / Unit / Flat	0
2	Abbotsford	(null)	Apartment / Unit / Flat	0
3	Abbotsford	(null)	(null)	0
4	Abbotsford	(null)	(null)	0
5	Acton	0120	Apartment / Unit / Flat	396000
6	Acton	0120	(null)	396000
7	Acton	0220	Apartment / Unit / Flat	1821600
8	Acton	0220	House	432000
9	Acton	0220	(null)	2253600
10	Acton	0320	Apartment / Unit / Flat	3026160
11	Acton	0320	(null)	3026160
12	Acton	0420	Apartment / Unit / Flat	3305991.48
13	Acton	0420	(null)	3305991.48
14	Acton	1219	Apartment / Unit / Flat	69120

REPORT 6 -

REPORT 7 -

a) Explanation in English

VERSION 1 REPORT 6- total number of sales from each suburb, time period and advertisement type using rollup
VERSION 1 REPORT 7- total number of sales from each suburb, time period and advertisement type using partial rollup

b) Explanation why it is important to management

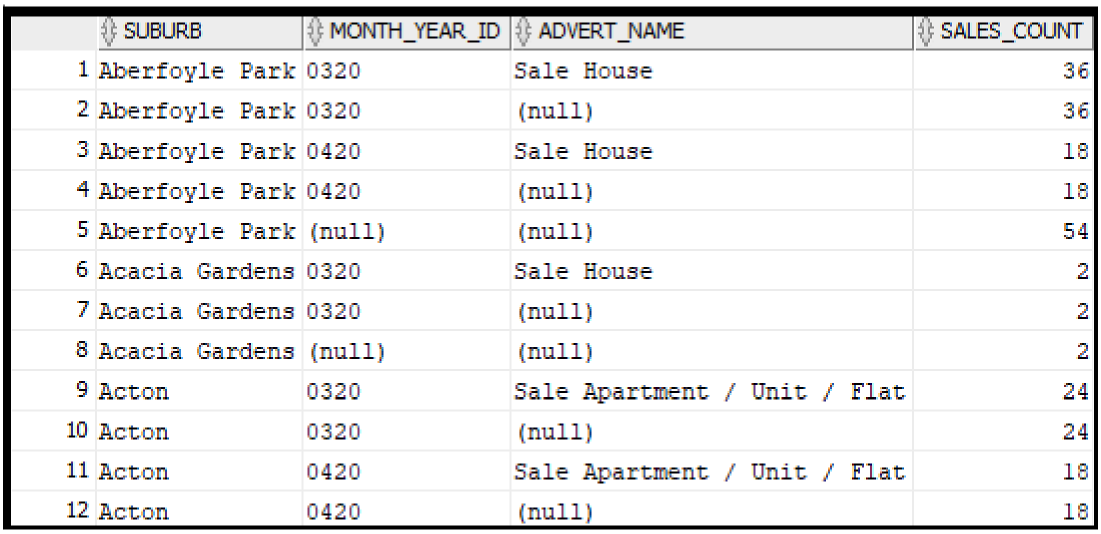
Report 6 can be used by the management to determine how each suburb has a dependency on time period and advertisement type. Using rollup we can find the sum of the number of sales for combinations being formed by rollup using the sequence suburb,time period(month and year) and advertisement name. The management can hence determine which kind of advertisement has what kind of an effect in which time period for the suburbs. Hence a sales count can be determined by the management for the given combinations which it can use to make decisions.

Report 7 can be used by the management to again find the sales count of the combinations in partial rollup such that suburb ROLLUP (time period , advertisement name) is the pattern followed. Hence suburb is fixed and combinations of time period and advertisement is used as taken in rollup. The management can hence use count of these sales (sum of sales) to determine how each suburb has an effect based on the combinations of time period and advertisement. The management can hence advertise particular kinds of ads in some particular suburbs that show the most sales count.

c) SQL Commands Version 1 Report 6

```
--VERSION 1 REPORT 6
--total number of sales from each suburb, time period and advertisement type using rollup
SELECT l.suburb, f.month_year_id ,a.advert_name , SUM(no_of_sales)
as SALES_COUNT
FROM sale_fact_v1 f,location_dim_v1 l, advert_dim_v1 a
WHERE f.postcode = l.postcode and
f.advert_id = a.advert_id
GROUP BY ROLLUP(l.suburb, f.month_year_id ,a.advert_name)
order by l.suburb;
```

d) Screenshots Version 1 Report 6

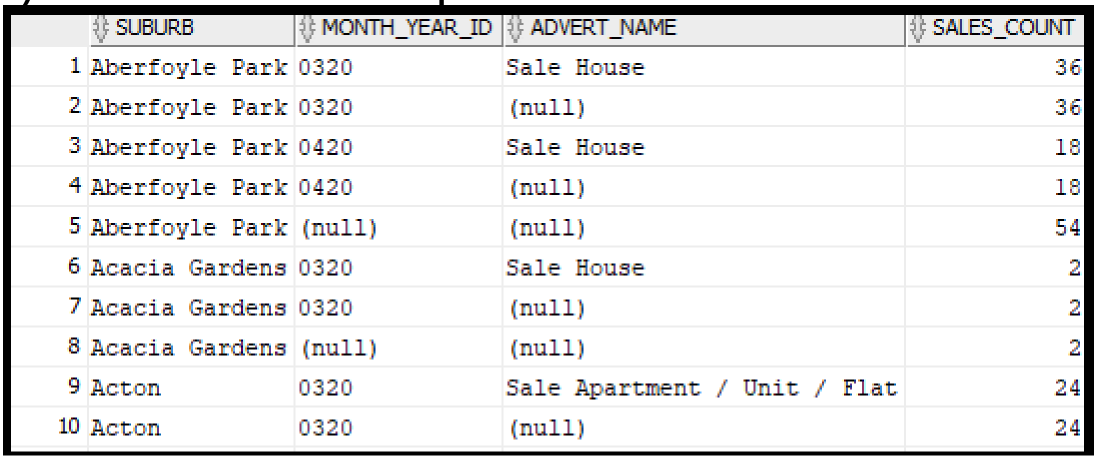


	SUBURB	MONTH_YEAR_ID	ADVERT_NAME	SALES_COUNT
1	Aberfoyle Park	0320	Sale House	36
2	Aberfoyle Park	0320	(null)	36
3	Aberfoyle Park	0420	Sale House	18
4	Aberfoyle Park	0420	(null)	18
5	Aberfoyle Park	(null)	(null)	54
6	Acacia Gardens	0320	Sale House	2
7	Acacia Gardens	0320	(null)	2
8	Acacia Gardens	(null)	(null)	2
9	Acton	0320	Sale Apartment / Unit / Flat	24
10	Acton	0320	(null)	24
11	Acton	0420	Sale Apartment / Unit / Flat	18
12	Acton	0420	(null)	18

c) SQL Commands Version 1 Report 7

```
--VERSION 1 REPORT 7
--total number of sales from each suburb, time period and advertisement type using partial rollup
SELECT l.suburb, f.month_year_id ,a.advert_name , SUM(no_of_sales)
as SALES_COUNT
FROM sale_fact_v1 f,location_dim_v1 l, advert_dim_v1 a
WHERE f.postcode = l.postcode and
f.advert_id = a.advert_id
GROUP BY l.suburb, ROLLUP(f.month_year_id ,a.advert_name)
order by l.suburb;
```

d) Screenshots Version 1 Report 7



	SUBURB	MONTH_YEAR_ID	ADVERT_NAME	SALES_COUNT
1	Aberfoyle Park	0320	Sale House	36
2	Aberfoyle Park	0320	(null)	36
3	Aberfoyle Park	0420	Sale House	18
4	Aberfoyle Park	0420	(null)	18
5	Aberfoyle Park	(null)	(null)	54
6	Acacia Gardens	0320	Sale House	2
7	Acacia Gardens	0320	(null)	2
8	Acacia Gardens	(null)	(null)	2
9	Acton	0320	Sale Apartment / Unit / Flat	24
10	Acton	0320	(null)	24

c). Reports with moving and cumulative aggregates:

Report 8 -

SQL -

Screenshots-

Report 9 -

Report 10 -

a) Explanation in English

VERSION 1 REPORT 9
total number of rents and cumulative aggregate number of rents in melbourne for each month

VERSION 1 REPORT 10
total number of rents and moving aggregate number of rents for houses for each month

b) Explanation why it is important to management

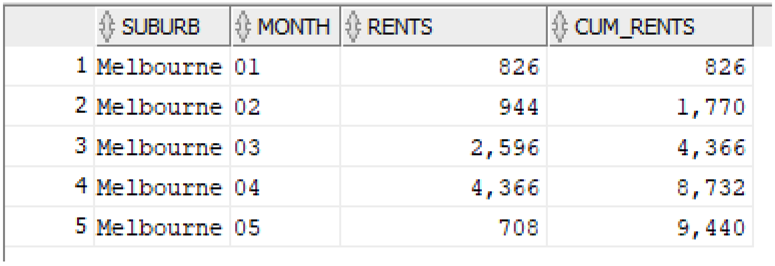
The Report 9 can be used by the management to find the individual rents and cumulative rents in Melbourne suburb for each month that we have data on. This can help the management to find out how the rent changes in each month for melbourne i.e. whether the trend is seen as increasing rent or decreasing rent or similar rent. Hence the management can determine the rents in melbourne to find the month which gets the most rent. Overall the management can also find the final rent sum that is obtained from Melbourne across the months.

Report 10 can be used by the management to find how the rent varies in the property type of Houses across the months. This can be used by the management to determine how the rent varies in the House types of properties across the months and hence the management can focus accordingly on House type properties. The management can also determine how the rent is moving aggregative for the current month and previous 2 months hence average is taken.

c) SQL Commands Version 1 Report 9

```
--VERSION 1 REPORT 9
--total number of rents and cumulative aggregate number of rents in melbourne for each month

SELECT l.suburb, t.month,
TO_CHAR (SUM(f.no_of_rent), '9,999,999,999') AS RENTS,
TO_CHAR (SUM(SUM(f.no_of_rent)) OVER
(ORDER BY l.suburb, t.month
ROWS UNBOUNDED PRECEDING),
'9,999,999,999') AS CUM_RENTS
FROM rent_fact_v1 f, location_dim_v1 l, time_dim_v1 t
WHERE f.postcode = l.postcode
AND f.month_year_id = t.month_year_id
AND l.suburb = 'Melbourne'
GROUP BY l.suburb, t.month
order by l.suburb;
```

d) Screenshot Version 1 Report 9

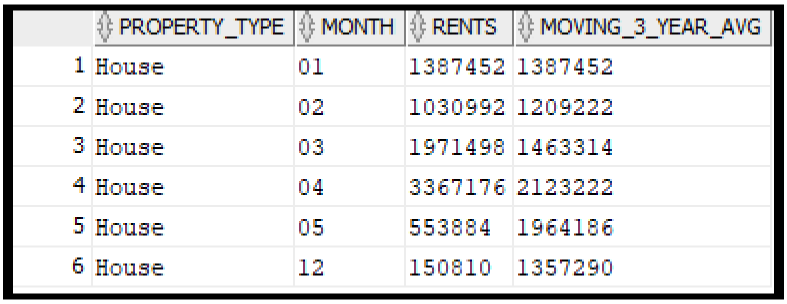


	SUBURB	MONTH	RENTS	CUM_RENTS
1	Melbourne	01	826	826
2	Melbourne	02	944	1,770
3	Melbourne	03	2,596	4,366
4	Melbourne	04	4,366	8,732
5	Melbourne	05	708	9,440

c) SQL Command Version 1 Report 10

```
--VERSION 1 REPORT 10
--total number of rents and moving aggregate number of rents for houses for each month

SELECT p.property_type, t.month,
TO_CHAR (SUM(f.no_of_rent)) AS RENTS,
TO_CHAR (AVG(SUM(f.no_of_rent)) OVER
(ORDER BY p.property_type, t.month
ROWS 2 PRECEDING)) AS MOVING_3_YEAR_AVG
FROM rent_fact_v1 f, property_type_dim_v1 p, time_dim_v1 t
WHERE f.property_type = p.property_type
AND f.month_year_id = t.month_year_id
AND p.property_type = 'House'
GROUP BY p.property_type, t.month
order by p.property_type;
```

d) Screenshot Version 1 Report 10



	PROPERTY_TYPE	MONTH	RENTS	MOVING_3_YEAR_AVG
1	House	01	1387452	1387452
2	House	02	1030992	1209222
3	House	03	1971498	1463314
4	House	04	3367176	2123222
5	House	05	553884	1964186
6	House	12	150810	1357290

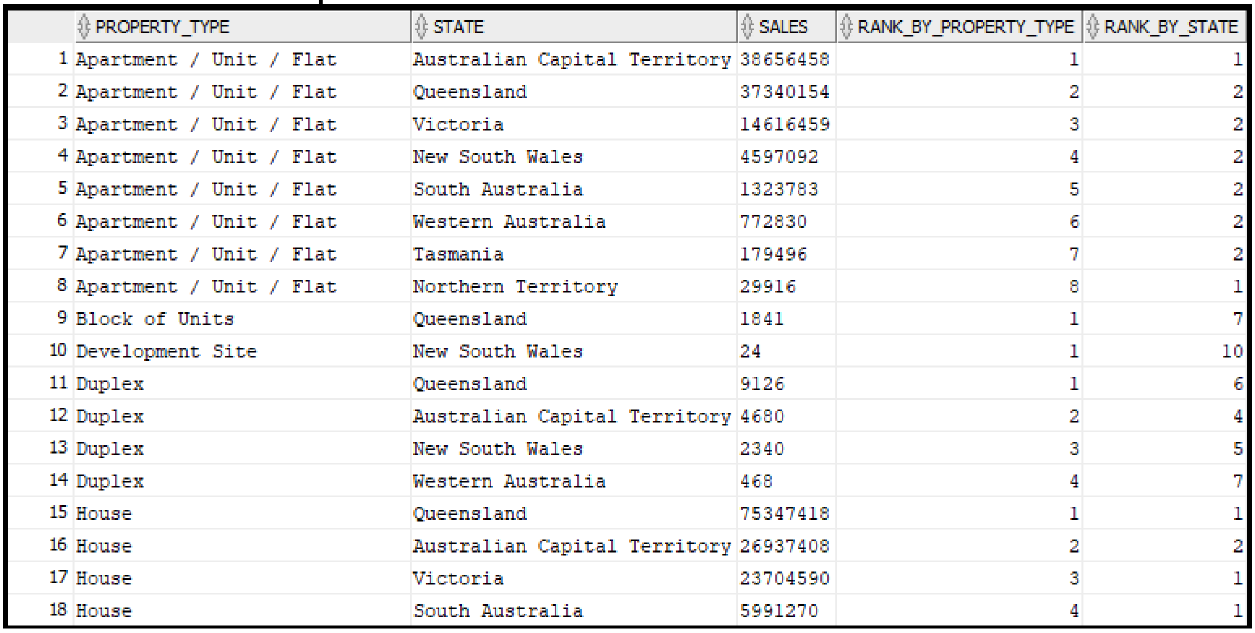
d) Reports with Partitions:

Report 11-
SQL commands

Version 1 Report 11 -

```
SELECT p.property_type, s.state_name  AS STATE,
TO_CHAR(SUM(f.no_of_sales)) AS SALES,
RANK() OVER (PARTITION BY p.property_type
ORDER BY SUM(f.no_of_sales) DESC) AS RANK_BY_PROPERTY_TYPE,
RANK() OVER (PARTITION BY s.state_name
ORDER BY SUM(f.no_of_sales) DESC) AS RANK_BY_STATE
FROM sale_fact_v1 f, property_type_dim_v1 p,location_dim_v1 l,state_dim_v1 s
WHERE f.property_type = p.property_type
and f.postcode = l.postcode
and l.state_code = s.state_code
GROUP BY p.property_type, s.state_name
order by p.property_type;
```

Screenshot Version 1 Report 11



	PROPERTY_TYPE	STATE	SALES	RANK_BY_PROPERTY_TYPE	RANK_BY_STATE
1	Apartment / Unit / Flat	Australian Capital Territory	38656458	1	1
2	Apartment / Unit / Flat	Queensland	37340154	2	2
3	Apartment / Unit / Flat	Victoria	14616459	3	2
4	Apartment / Unit / Flat	New South Wales	4597092	4	2
5	Apartment / Unit / Flat	South Australia	1323783	5	2
6	Apartment / Unit / Flat	Western Australia	772830	6	2
7	Apartment / Unit / Flat	Tasmania	179496	7	2
8	Apartment / Unit / Flat	Northern Territory	29916	8	1
9	Block of Units	Queensland	1841	1	7
10	Development Site	New South Wales	24	1	10
11	Duplex	Queensland	9126	1	6
12	Duplex	Australian Capital Territory	4680	2	4
13	Duplex	New South Wales	2340	3	5
14	Duplex	Western Australia	468	4	7
15	House	Queensland	75347418	1	1
16	House	Australian Capital Territory	26937408	2	2
17	House	Victoria	23704590	3	1
18	House	South Australia	5991270	4	1

Report 12

a) Explanation in English

ranking of each property scale based on the total number of rents and the ranking of each property type based on the total number of rents.

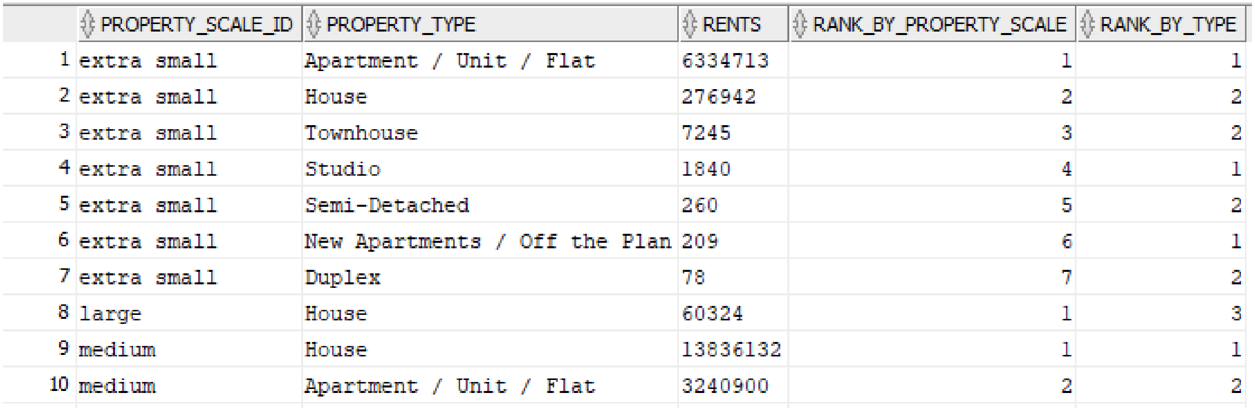
b) Explanation why it is important to management

The management would want this information to have the ranking of the different properties based on bedrooms(scale) on the sum of rents and to show rankings of the different property types based on the sum of rents. This information is hence helpful for the management to determine which property_scale receives the most rents and similarly which property type receives the most rent. The management can use this information to then have more properties with the property_scale that gives them the most profit. Hence ranking of the property_scale and the property types will help the management to make better decisions to attract the most rents.

c) SQL Command Version 1 Report 12

```
--VERSION 1 REPORT 12
        --ranking of each property scale based on the total number of rents and the ranking of --each property type
based on the
--total number of rents.
SELECT p.property_scale_id, pt.property_type  AS PROPERTY_TYPE,
TO_CHAR(SUM(f.no_of_rent)) AS RENTS,
RANK() OVER (PARTITION BY p.property_scale_id
ORDER BY SUM(f.no_of_rent) DESC) AS RANK_BY_PROPERTY_SCALE,
RANK() OVER (PARTITION BY pt.property_type
ORDER BY SUM(f.no_of_rent) DESC) AS RANK_BY_TYPE
FROM rent_fact_v1 f, property_scale_dim_v1 p,property_type_dim_v1 pt
WHERE f.property_scale_id = p.property_scale_id
and f.property_type = pt.property_type
GROUP BY p.property_scale_id, pt.property_type
order by p.property_scale_id;
```

d) Screenshot Version 1 Report 12



	PROPERTY_SCALE_ID	PROPERTY_TYPE	RENTS	RANK_BY_PROPERTY_SCALE	RANK_BY_TYPE
1	extra small	Apartment / Unit / Flat	6334713	1	1
2	extra small	House	276942	2	2
3	extra small	Townhouse	7245	3	2
4	extra small	Studio	1840	4	1
5	extra small	Semi-Detached	260	5	2
6	extra small	New Apartments / Off the Plan	209	6	1
7	extra small	Duplex	78	7	2
8	large	House	60324	1	3
9	medium	House	13836132	1	1
10	medium	Apartment / Unit / Flat	3240900	2	2

Task 3 out of 4

VERSION 2 -

a) Simple Reports

a) Query questions in English -

Report 1 - top 5 Sum of rent for each feature type and property scale

Report 2 - top 5% sales in each location and time period

Report 3 - number of visits in each season in each day

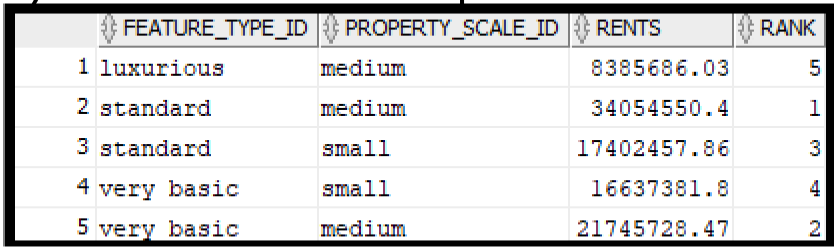
b) Explanation why it is important to management

Query/Report 1 can be used by the management to get the top sum of rents for the combinations of feature type and scale of the property. Using this information the management can know which top 5 combinations give the most number of rents and can use that combination to achieve maximum rents. Hence feature count and scale of property can be useful in determining the case of which combination produces the most rent.
Query/Report 2 can be used by the management to get the best locations and time period where sales were maximum. It can help the management to determine any relationships between the two factors location and time period. Hence the management can do further analysis about which region/location has the most sales based on the different months/time period. It will hence help the management to make decisions as to when to advertise more in which area so that maximum sales are seen.
Query/Report 3 can be used by the management to determine which days and which seasons see the most number of visits for a property. This can help the management to focus on certain seasons and certain days and find which best combination sees the most number of visits. It would help the management to determine which combination has most number of visits and hence they can use that information to arrange more inspection times during the most popular combinations.

c) SQL Commands - Version 2 Report 1

```
--VERSION 2 REPORT 1 top 5 Sum of rent for each feature type and property scale
SELECT *
FROM
(SELECT FEATURE_TYPE_ID,PROPERTY_SCALE_ID,sum(SUM_OF_RENT) as RENTS,
RANK() OVER (ORDER BY  sum(SUM_OF_RENT)  DESC ) AS RANK
FROM rent_fact_v2
GROUP BY FEATURE_TYPE_ID,PROPERTY_SCALE_ID
having sum(SUM_OF_RENT) > 0
order by FEATURE_TYPE_ID)
WHERE RANK <= 5;
```

d.) Screenshots- Version 2 Report 1

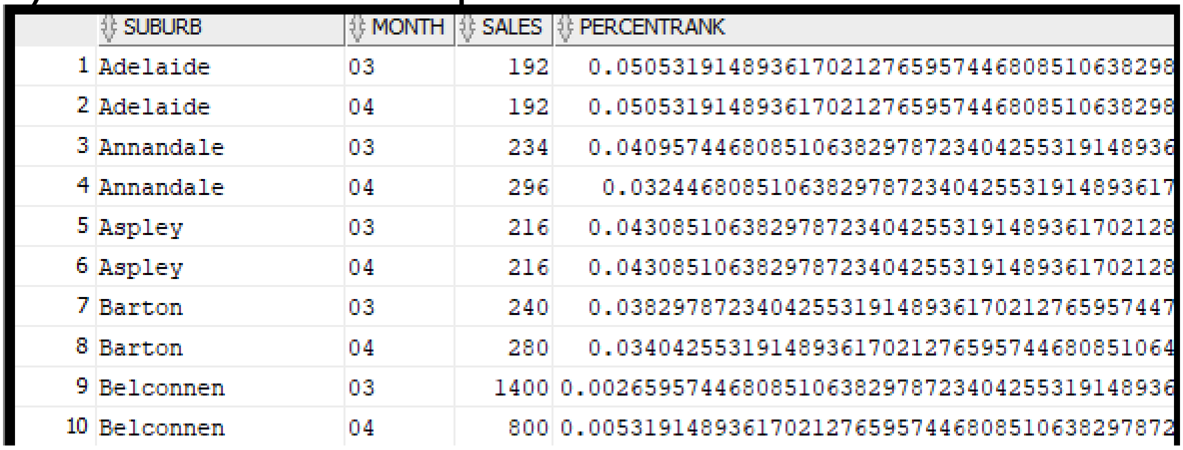


	FEATURE_TYPE_ID	PROPERTY_SCALE_ID	RENTS	RANK
1	luxurious	medium	8385686.03	5
2	standard	medium	34054550.4	1
3	standard	small	17402457.86	3
4	very basic	small	16637381.8	4
5	very basic	medium	21745728.47	2

c) SQL Commands - Version 2 Report 2

```
--VERSION 2 REPORT 2 top 5% sales in each location and time period


SELECT *
FROM (
SELECT
a.suburb as SUBURB,
to_char(p.property_date_added,'mm') as MONTH, sum(f.no_of_sales) AS SALES,
percent_rank() over
(order by sum(f.no_of_sales) desc) as PercentRank
FROM sale_fact_v2 f ,property_dim_v2 p, address_dim_v2 a
WHERE f.postcode = a.postcode and
f.property_id = p.property_id
GROUP BY a.suburb ,to_char(p.property_date_added,'mm')
having sum(f.no_of_sales) > 0
order by a.suburb
) WHERE PercentRank < 0.1;
```

d.) Screenshots- Version 2 Report 2

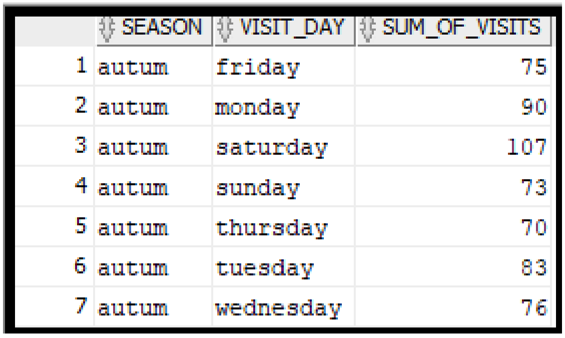


	SUBURB	MONTH	SALES	PERCENTRANK
1	Adelaide	03	192	0.0505319148936170212765957446808510638298
2	Adelaide	04	192	0.0505319148936170212765957446808510638298
3	Annandale	03	234	0.0409574468085106382978723404255319148936
4	Annandale	04	296	0.032446808510638297872340425531914893617
5	Aspley	03	216	0.0430851063829787234042553191489361702128
6	Aspley	04	216	0.0430851063829787234042553191489361702128
7	Barton	03	240	0.0382978723404255319148936170212765957447
8	Barton	04	280	0.0340425531914893617021276595744680851064
9	Belconnen	03	1400	0.002659574468085106382978723404255319148936
10	Belconnen	04	800	0.005319148936170212765957446808510638297872

c) SQL Commands - Version 2 Report 3

```
--VERSION 2 REPORT 3 number of visits in each season in each day
select f.SEASON,v.VISIT_DAY, sum(f.no_of_visits) as sum_of_visits
from property_visit_fact_v2 f, visit_dim_v2 v
where f.client_person_id = v.client_person_id and
f.agent_person_id = v.agent_person_id and
f.visit_date = v.visit_date and
f.property_id = v.property_id
group by f.SEASON,v.VISIT_DAY
order by f.season,v.VISIT_DAY;
```

d.) Screenshots- Version 2 Report 3



	SEASON	VISIT_DAY	SUM_OF_VISITS
1	autum	friday	75
2	autum	monday	90
3	autum	saturday	107
4	autum	sunday	73
5	autum	thursday	70
6	autum	tuesday	83
7	autum	wednesday	76

b) Reports with proper sub-totals:

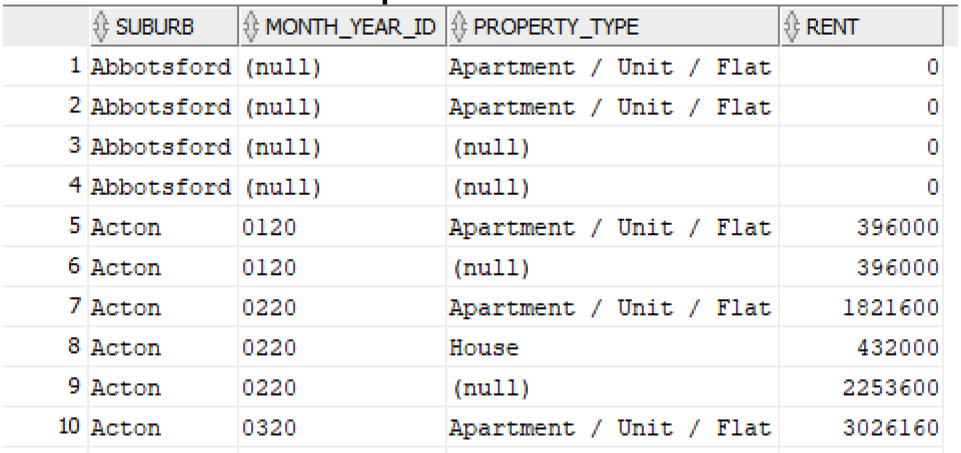
REPORT 4

SQL-

```
        --VERSION 2 REPORT 4
        --the sub-total and total rental fees from each suburb, time period, and property type --using cube
SELECT a.suburb, to_char(r.rent_start_date,'mm') || to_char(r.rent_start_date,'yy') as month_year_id ,p.property_type ,
NVL(SUM(f.sum_of_rent),0)
as RENT
FROM rent_fact_v2 f,address_dim_v2  a, rent_dim_v2 r, property_dim_v2 p
WHERE f.postcode = a.postcode and
```

```
f.property_id = p.property_id and
f.rent_id = r.rent_id
GROUP BY CUBE(a.suburb, to_char(r.rent_start_date,'mm') || to_char(r.rent_start_date,'yy') ,p.property_type)
order by a.suburb;
```

Screenshot Version 2 Report 4



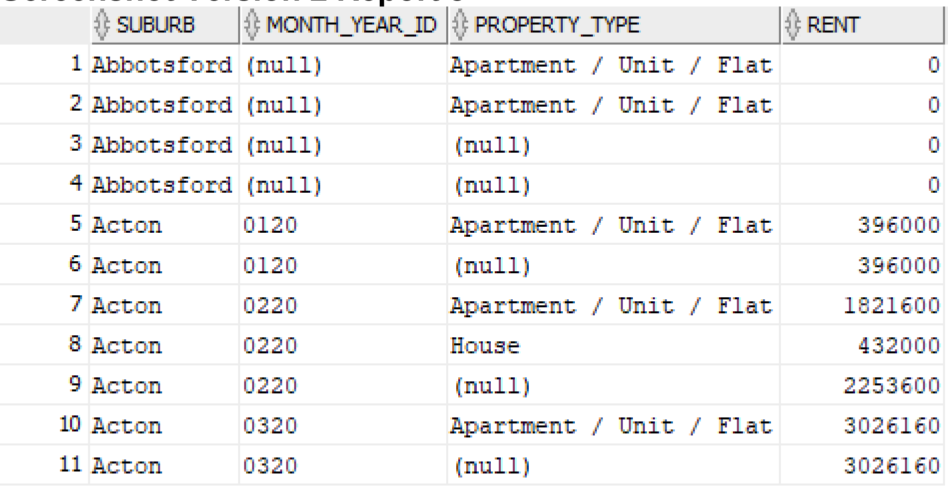
	SUBURB	MONTH_YEAR_ID	PROPERTY_TYPE	RENT
1	Abbotsford	(null)	Apartment / Unit / Flat	0
2	Abbotsford	(null)	Apartment / Unit / Flat	0
3	Abbotsford	(null)	(null)	0
4	Abbotsford	(null)	(null)	0
5	Acton	0120	Apartment / Unit / Flat	396000
6	Acton	0120	(null)	396000
7	Acton	0220	Apartment / Unit / Flat	1821600
8	Acton	0220	House	432000
9	Acton	0220	(null)	2253600
10	Acton	0320	Apartment / Unit / Flat	3026160

REPORT 5

SQL-

```
--VERSION 2 REPORT 5
----the sub-total and total rental fees from each suburb, time period, and property type using --partial cube

SELECT a.suburb, to_char(r.rent_start_date,'mm') || to_char(r.rent_start_date,'yy') as month_year_id ,p.property_type ,
NVL(SUM(sum_of_rent),0)
as RENT
FROM rent_fact_v2 f,rent_dim_v2 r , property_dim_v2 p, address_dim_v2 a
WHERE f.postcode = a.postcode and
f.rent_id = r.rent_id and
f.property_id = p.property_id
GROUP BY a.suburb , CUBE(to_char(r.rent_start_date,'mm') || to_char(r.rent_start_date,'yy') ,p.property_type)
order by a.suburb;
```

Screenshot Version 2 Report 5



	SUBURB	MONTH_YEAR_ID	PROPERTY_TYPE	RENT
1	Abbotsford	(null)	Apartment / Unit / Flat	0
2	Abbotsford	(null)	Apartment / Unit / Flat	0
3	Abbotsford	(null)	(null)	0
4	Abbotsford	(null)	(null)	0
5	Acton	0120	Apartment / Unit / Flat	396000
6	Acton	0120	(null)	396000
7	Acton	0220	Apartment / Unit / Flat	1821600
8	Acton	0220	House	432000
9	Acton	0220	(null)	2253600
10	Acton	0320	Apartment / Unit / Flat	3026160
11	Acton	0320	(null)	3026160

REPORT 6 -

REPORT 7 -

e) Explanation in English

VERSION 2 REPORT 6- total number of sales from each suburb, time period and advertisement type using rollup
VERSION 2 REPORT 7- total number of sales from each suburb, time period and advertisement type using partial rollup

f) Explanation why it is important to management

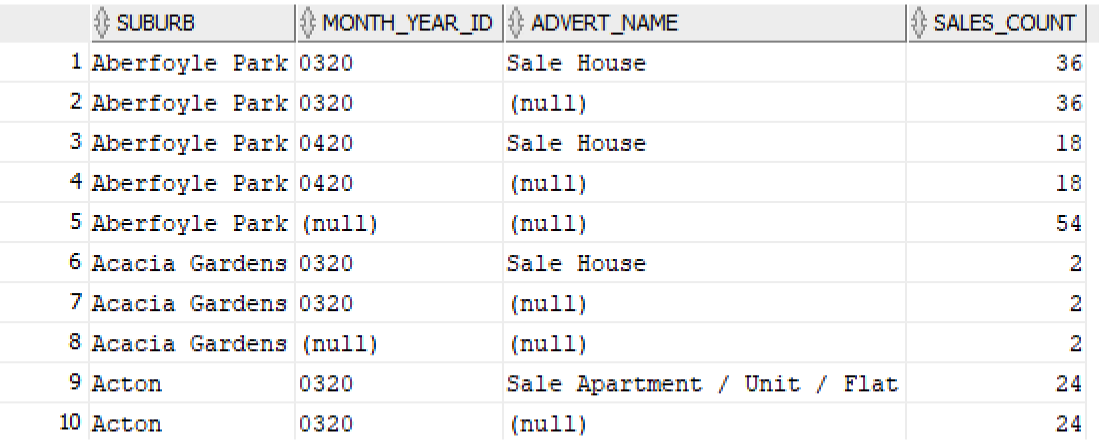
Report 6 can be used by the management to determine how each suburb has a dependency on time period and advertisement type. Using rollup we can find the sum of the number of sales for combinations being formed by rollup using the sequence suburb,time period(month and year) and advertisement name. The management can hence determine which kind of advertisement has what kind of an effect in which time period for the suburbs. Hence a sales count can be determined by the management for the given combinations which it can use to make decisions.

Report 7 can be used by the management to again find the sales count of the combinations in partial rollup such that suburb ROLLUP (time period , advertisement name) is the pattern followed. Hence suburb is fixed and combinations of time period and advertisement is used as taken in rollup. The management can hence use count of these sales (sum of sales) to determine how each suburb has an effect based on the combinations of time period and advertisement. The management can hence advertise particular kinds of ads in some particular suburbs that show the most sales count.

g) SQL Commands Version 2 Report 6

```
--VERSION 2 REPORT 6
--total number of sales from each suburb, time period and advertisement type using --rollup
SELECT a.suburb, to_char(p.property_date_added,'mm') || to_char(p.property_date_added,'yy') as month_year_id
,ad.advert_name , SUM(f.no_of_sales)
as SALES_COUNT
FROM sale_fact_v2 f,address_dim_v2 a, advert_dim_v2 ad, property_dim_v2 p
WHERE f.postcode = a.postcode and
f.advert_id = ad.advert_id and
f.property_id = p.property_id
GROUP BY ROLLUP(a.suburb, to_char(p.property_date_added,'mm') ||
to_char(p.property_date_added,'yy'),ad.advert_name)
order by a.suburb;
```

h) Screenshots Version 2 Report 6



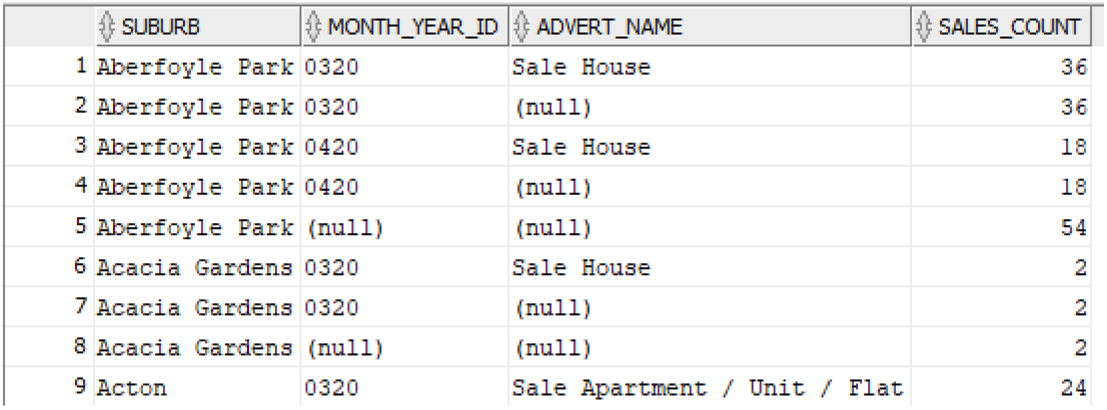
	SUBURB	MONTH_YEAR_ID	ADVERT_NAME	SALES_COUNT
1	Aberfoyle Park	0320	Sale House	36
2	Aberfoyle Park	0320	(null)	36
3	Aberfoyle Park	0420	Sale House	18
4	Aberfoyle Park	0420	(null)	18
5	Aberfoyle Park	(null)	(null)	54
6	Acacia Gardens	0320	Sale House	2
7	Acacia Gardens	0320	(null)	2
8	Acacia Gardens	(null)	(null)	2
9	Acton	0320	Sale Apartment / Unit / Flat	24
10	Acton	0320	(null)	24

c) SQL Commands Version 2 Report 7

```
--VERSION 2 REPORT 7
--total number of sales from each suburb, time period and advertisement type using partial rollup
SELECT a.suburb, to_char(p.property_date_added,'mm') || to_char(p.property_date_added,'yy') as month_year_id
,ad.advert_name ,  SUM(no_of_sales)
as SALES_COUNT
FROM sale_fact_v2 f,address_dim_v2 a, advert_dim_v2 ad, property_dim_v2 p
WHERE f.postcode = a.postcode and
f.advert_id = ad.advert_id and
f.property_id = p.property_id
```

```
GROUP BY a.suburb, ROLLUP(to_char(p.property_date_added,'mm') ||
to_char(p.property_date_added,'yy'),ad.advert_name)
order by a.suburb;
```

d) Screenshots Version 2 Report 7



	SUBURB	MONTH_YEAR_ID	ADVERT_NAME	SALES_COUNT
1	Aberfoyle Park	0320	Sale House	36
2	Aberfoyle Park	0320	(null)	36
3	Aberfoyle Park	0420	Sale House	18
4	Aberfoyle Park	0420	(null)	18
5	Aberfoyle Park	(null)	(null)	54
6	Acacia Gardens	0320	Sale House	2
7	Acacia Gardens	0320	(null)	2
8	Acacia Gardens	(null)	(null)	2
9	Acton	0320	Sale Apartment / Unit / Flat	24

c). Reports with moving and cumulative aggregates:

Report 8 -

SQL -

Screenshots-

Report 9 -

Report 10 -

e) Explanation in English

VERSION 2 REPORT 9
total number of rents and cumulative aggregate number of rents in melbourne for each month

VERSION 2 REPORT 10
total number of rents and moving aggregate number of rents for houses for each month

f) Explanation why it is important to management

The Report 9 can be used by the management to find the individual rents and cumulative rents in Melbourne suburb for each month that we have data on. This can help the management to find out how the rent changes in each month for melbourne i.e. whether the trend is seen as increasing rent or decreasing rent or similar rent. Hence the management can determine the rents in melbourne to find the month which gets the most rent. Overall the management can also find the final rent sum that is obtained from Melbourne across the months.

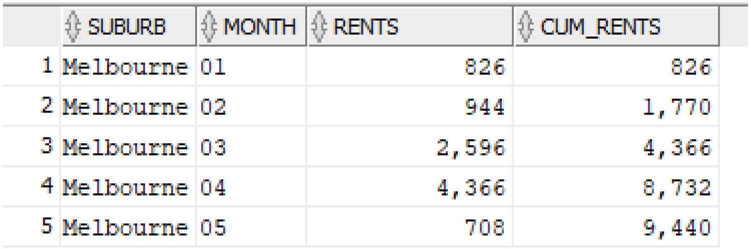
Report 10 can be used by the management to find how the rent varies in the property type of Houses across the months. This can be used by the management to determine how the rent varies in the House types of properties across the months and hence the management can focus accordingly on House type properties. The management can also determine how the rent is moving aggregative for the current month and previous 2 months hence average is taken.

g) SQL Commands Version 2 Report 9

```
--VERSION 2 REPORT 9
--total number of rents and cumulative aggregate number of rents in melbourne for each month
SELECT a.suburb, to_char(r.rent_start_date,'mm') as month,
TO_CHAR (SUM(f.no_of_rent), '9,999,999,999') AS RENTS,
TO_CHAR (SUM(SUM(f.no_of_rent)) OVER
(ORDER BY a.suburb, to_char(r.rent_start_date,'mm')
ROWS UNBOUNDED PRECEDING),
'9,999,999,999') AS CUM_RENTS
FROM rent_fact_v2 f, address_dim_v2 a, rent_dim_v2 r
```

```
WHERE f.postcode = a.postcode
AND f.rent_id = r.rent_id
AND a.suburb = 'Melbourne' and
to_char(r.rent_start_date,'mm') is not null
GROUP BY a.suburb, to_char(r.rent_start_date,'mm')
order by a.suburb;
```

h) Screenshot Version 2 Report 9

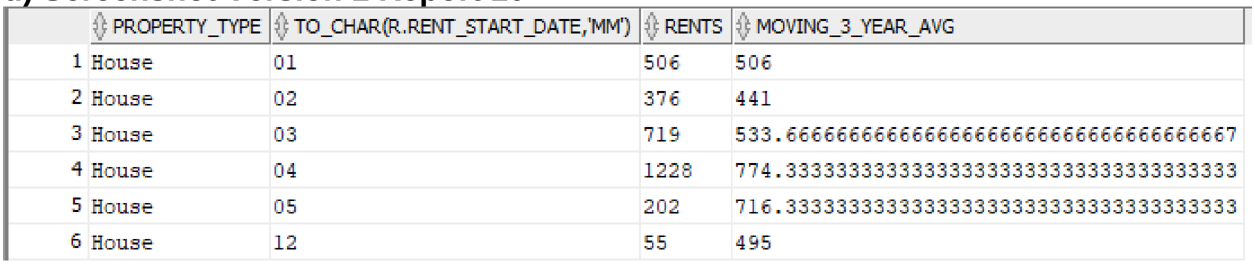


	SUBURB	MONTH	RENTS	CUM_RENTS
1	Melbourne	01	826	826
2	Melbourne	02	944	1,770
3	Melbourne	03	2,596	4,366
4	Melbourne	04	4,366	8,732
5	Melbourne	05	708	9,440

c) SQL Command Version 2 Report 10

```
--VERSION 2 REPORT 10
--total number of rents and moving aggregate number of rents for houses for --each month

SELECT p.property_type, to_char(r.rent_start_date,'mm'),
TO_CHAR (SUM(f.no_of_rent)) AS RENTS,
TO_CHAR (AVG(SUM(f.no_of_rent)) OVER
(ORDER BY p.property_type,to_char(r.rent_start_date,'mm')
ROWS 2 PRECEDING)) AS MOVING_3_YEAR_AVG
FROM rent_fact_v2 f, property_dim_v2 p, rent_dim_v2 r
WHERE f.property_id = p.property_id
AND f.rent_id = r.rent_id
AND p.property_type = 'House' and
to_char(r.rent_start_date,'mm') is not null
GROUP BY p.property_type, to_char(r.rent_start_date,'mm')
order by p.property_type;
```

d) Screenshot Version 1 Report 10



	PROPERTY_TYPE	TO_CHAR(R.RENT_START_DATE,'MM')	RENTS	MOVING_3_YEAR_AVG
1	House	01	506	506
2	House	02	376	441
3	House	03	719	533.666666666666666666666666666666666667
4	House	04	1228	774.333333333333333333333333333333333333
5	House	05	202	716.333333333333333333333333333333333333
6	House	12	55	495

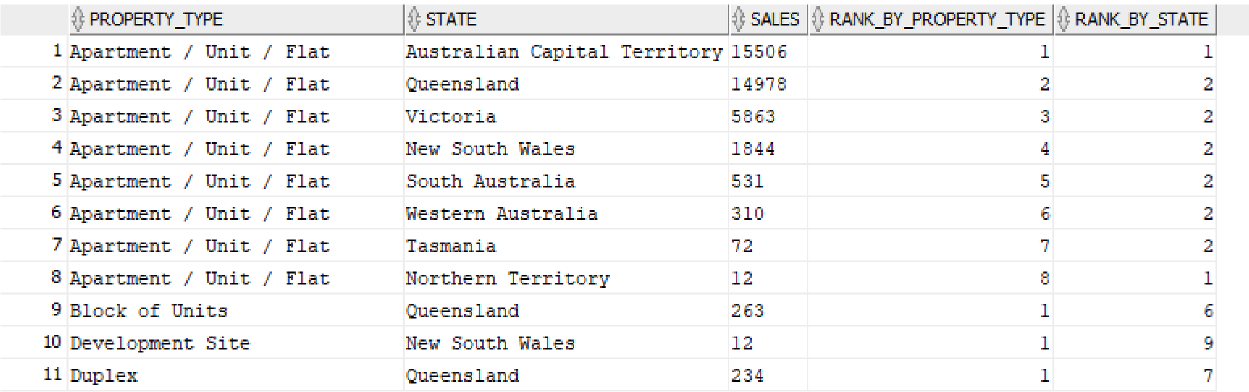
d) Reports with Partitions:

Report 11-
SQL commands

Version 2 Report 11 -

```
--VERSION 2 REPORT 11
--ranking of each property type based on the yearly total number of sales and the ranking of each state based on the yearly
--total number of sales.

SELECT p.property_type, s.state_name  AS STATE,
TO_CHAR(SUM(f.no_of_sales)) AS SALES,
RANK() OVER (PARTITION BY p.property_type
ORDER BY SUM(f.no_of_sales) DESC) AS RANK_BY_PROPERTY_TYPE,
RANK() OVER (PARTITION BY s.state_name
ORDER BY SUM(f.no_of_sales) DESC) AS RANK_BY_STATE
FROM sale_fact_v2 f, property_dim_v2 p, address_dim_v2 s
WHERE f.property_id = p.property_id
and f.postcode = s.postcode
GROUP BY p.property_type, s.state_name
order by p.property_type;
```

Screenshot Version 2 Report 11



	PROPERTY_TYPE	STATE	SALES	RANK_BY_PROPERTY_TYPE	RANK_BY_STATE
1	Apartment / Unit / Flat	Australian Capital Territory	15506	1	1
2	Apartment / Unit / Flat	Queensland	14978	2	2
3	Apartment / Unit / Flat	Victoria	5863	3	2
4	Apartment / Unit / Flat	New South Wales	1844	4	2
5	Apartment / Unit / Flat	South Australia	531	5	2
6	Apartment / Unit / Flat	Western Australia	310	6	2
7	Apartment / Unit / Flat	Tasmania	72	7	2
8	Apartment / Unit / Flat	Northern Territory	12	8	1
9	Block of Units	Queensland	263	1	6
10	Development Site	New South Wales	12	1	9
11	Duplex	Queensland	234	1	7

Report 12

c) Explanation in English

ranking of each property scale based on the total number of rents and the ranking of each property type based on the total number of rents.

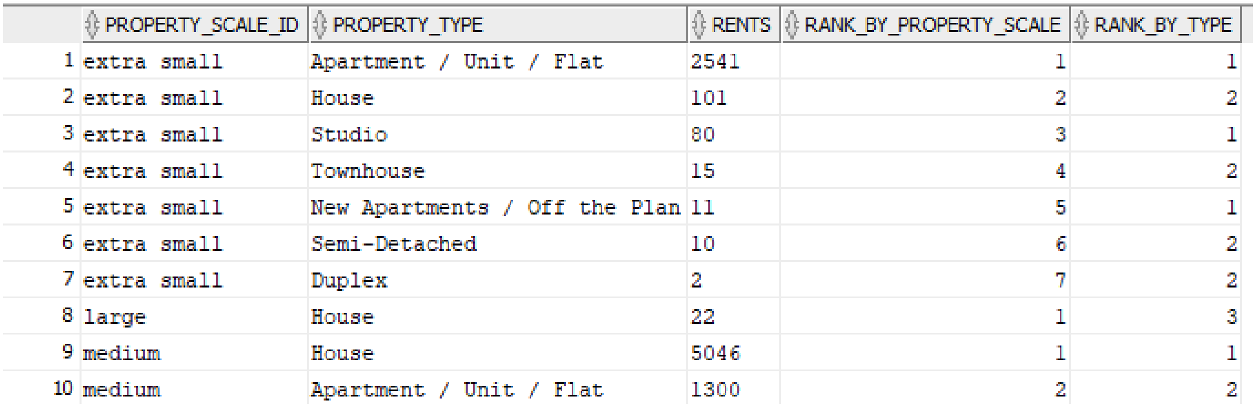
d) Explanation why it is important to management

The management would want this information to have the ranking of the different properties based on bedrooms(scale) on the sum of rents and to show rankings of the different property types based on the sum of rents. This information is hence helpful for the management to determine which property_scale receives the most rents and similarly which property type receives the most rent. The management can use this information to then have more properties with the property_scale that gives them the most profit. Hence ranking of the property_scale and the property types will help the management to make better decisions to attract the most rents.

c) SQL Command Version 2 Report 12

```
--VERSION 2 REPORT 12
--ranking of each property scale based on the total number of rents and the ranking of each property type based on the
--total number of rents.
SELECT p.property_scale_id, pt.property_type  AS PROPERTY_TYPE,
TO_CHAR(SUM(f.no_of_rent)) AS RENTS,
RANK() OVER (PARTITION BY p.property_scale_id
ORDER BY SUM(f.no_of_rent) DESC) AS RANK_BY_PROPERTY_SCALE,
RANK() OVER (PARTITION BY pt.property_type
ORDER BY SUM(f.no_of_rent) DESC) AS RANK_BY_TYPE
FROM rent_fact_v2 f, property_scale_dim_v2 p,property_dim_v2 pt
WHERE f.property_scale_id = p.property_scale_id
and f.property_id = pt.property_id
GROUP BY p.property_scale_id, pt.property_type
order by p.property_scale_id;
```

d) Screenshot Version 2 Report 12

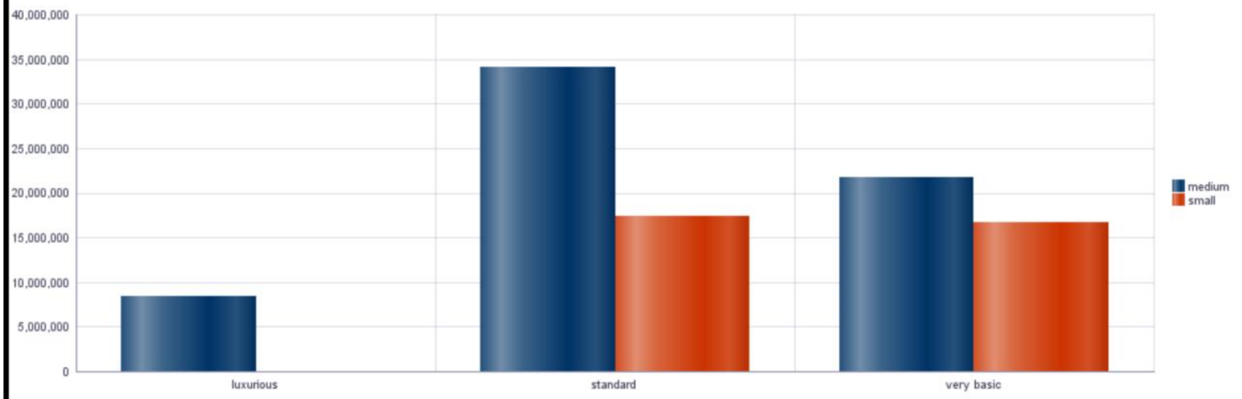


	PROPERTY_SCALE_ID	PROPERTY_TYPE	RENTS	RANK_BY_PROPERTY_SCALE	RANK_BY_TYPE
1	extra small	Apartment / Unit / Flat	2541	1	1
2	extra small	House	101	2	2
3	extra small	Studio	80	3	1
4	extra small	Townhouse	15	4	2
5	extra small	New Apartments / Off the Plan	11	5	1
6	extra small	Semi-Detached	10	6	2
7	extra small	Duplex	2	7	2
8	large	House	22	1	3
9	medium	House	5046	1	1
10	medium	Apartment / Unit / Flat	1300	2	2

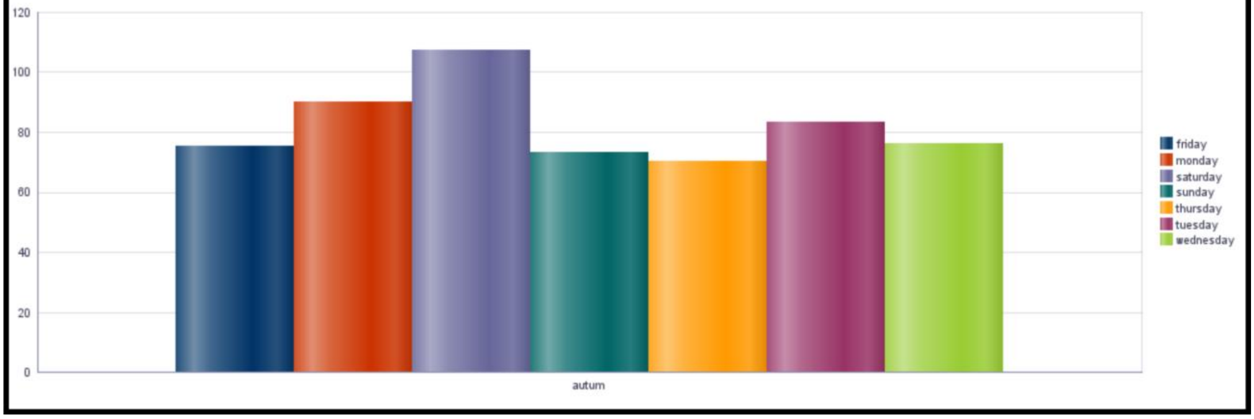
Task 4 out of 4

The following graphs attached are for the respective reports of Version 1.

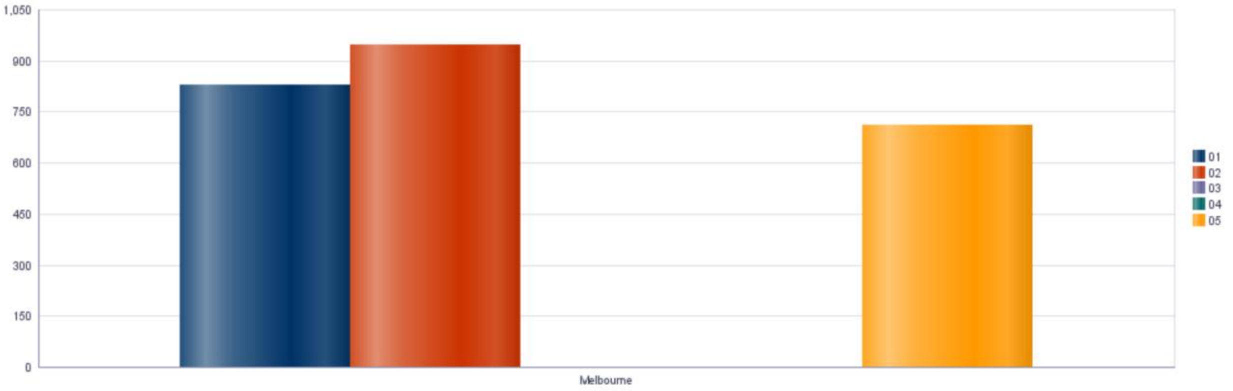
Report1



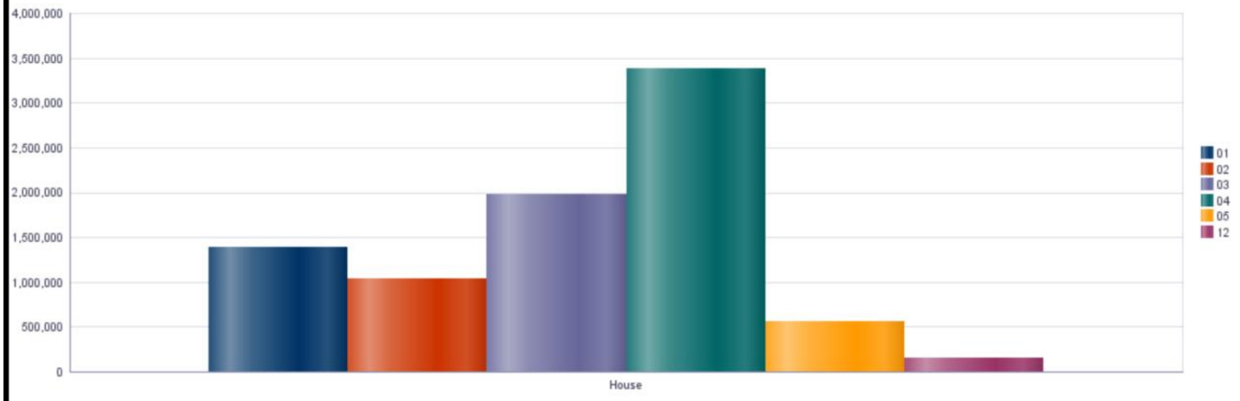
Report3



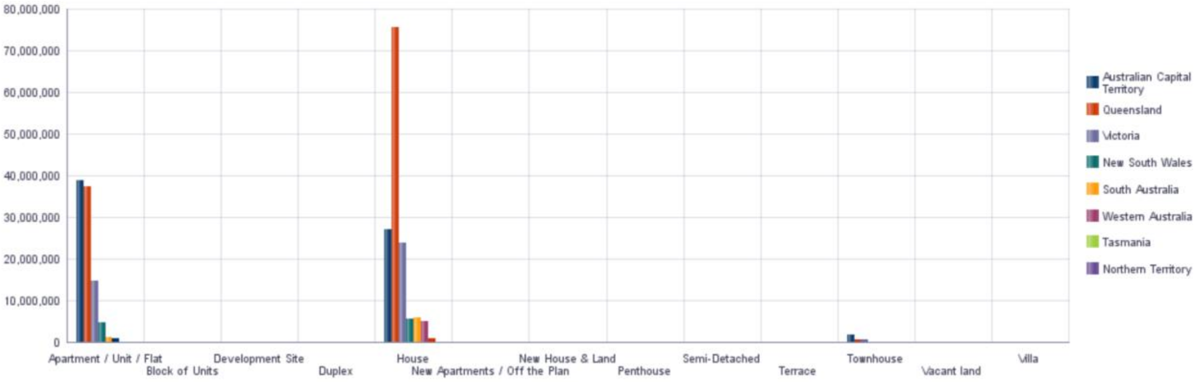
Report9



Report10



Report11


Apartment / Unit / Flat
Block of Units
Development Site
Duplex
House
New Apartments / Off the Plan
New House & Land
Penthouse
Semi-Detached
Terrace
Townhouse
Vacant land
Villa
Australian Capital Territory
Queensland
Victoria
New South Wales
South Australia
Western Australia
Tasmania
Northern Territory