

Monash University

Semester Examination Period
Faculty of Information Technology

EXAM CODES:

Sample-1

TITLE OF PAPER:

Sample-1

EXAM DURATION:

READING TIME:

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

- | | | | | |
|---|---|------------------------------------|---|--|
| <input type="checkbox"/> Berwick | <input checked="" type="checkbox"/> Clayton | <input type="checkbox"/> Malaysia | <input checked="" type="checkbox"/> Off Campus Learning | <input type="checkbox"/> Open Learning |
| <input checked="" type="checkbox"/> Caulfield | <input type="checkbox"/> Gippsland | <input type="checkbox"/> Peninsula | <input type="checkbox"/> Enhancement Studies | <input type="checkbox"/> Sth Africa |
| <input type="checkbox"/> Pharmacy | <input type="checkbox"/> Other (specify) | | | |

During an exam, you must not have in your possession, a book, notes, paper, calculator, pencil case, mobile phone or other material/item which has not been authorised for the exam or specifically permitted as noted below. Any material or item on your desk, chair or person will be deemed to be in your possession. You are reminded that possession of unauthorised materials in

No examination papers are to be removed from the room.

Attempt all questions. Write answers in the spaces provided. Rough work may be done on the back of pages in this paper. Any material written on the backs of pages will not normally be corrected. If an answer needs to overflow from its designated answer space to a blank page, clearly indicate that this is the case and that the material on the blank page is for correction. This paper must be handed up at the end of the examination, even if no questions are attempted. **There are 10 questions, each of which is worth 10 marks. The total is 100 marks.** This exam counts as 60% of the final assessment for the unit.

AUTHORISED MATERIALS

CALCULATORS

☐ YES☒ NO

OPEN BOOK

☐ YES☒ NO

SPECIFICALLY PERMITTED ITEMS

☐ YES☒ NO

Candidates must complete this section if required to write answers within this paper

STUDENT ID

DESK NUMBER

Question 1:

Monash International would like to analyse their policy in regard to English requirement for admission into a course. Monash International has the following data:

Table: Student_IELTS

Student ID	Student Name	Listening	Reading	Writing	Speaking	Overall
228493	Sooying Tan	6.5	6.5	6.0	7.0	6.5
229094	Xuebing Lu	5.5	5.5	5.5	5.5	5.5
231289	Amandh Kumar	6.0	7.0	6.0	7.0	6.5
234354	Agus Hidayat	5.5	6.0	6.0	6.5	6.0
234355	Budi Rahayu	7.0	7.0	7.0	7.0	7.0
...						
...						

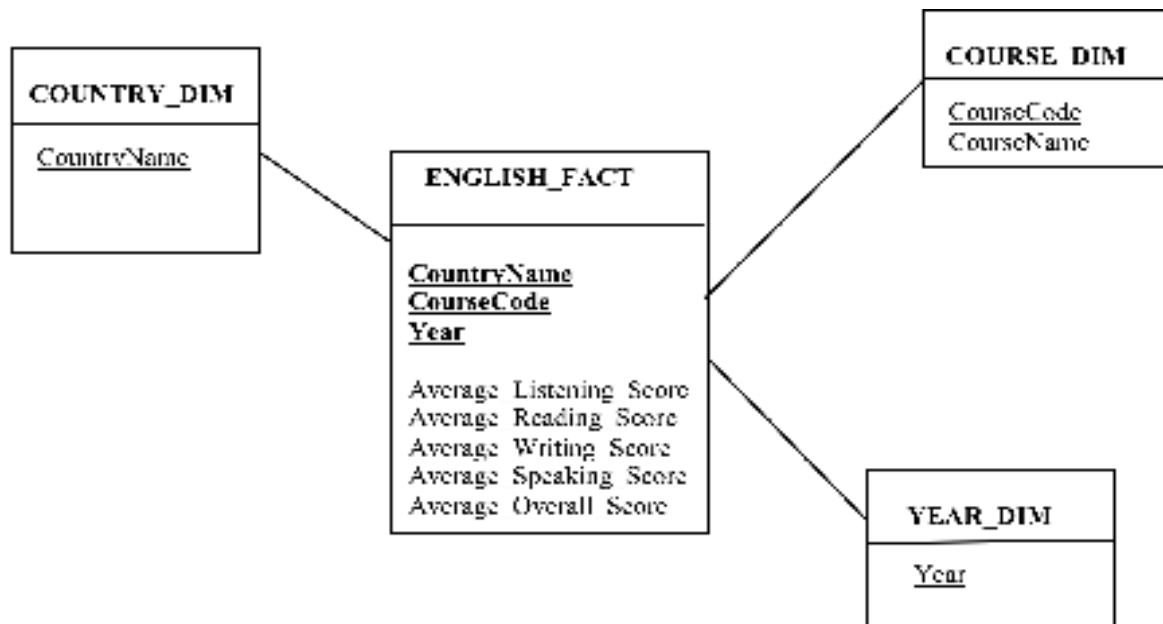
Table: Student_Course

Student ID	Student Name	Course	StarYear
228493	Sooying Tan	MBIS	2013
229094	Xuebing Lu	MBIS	2013
231289	Amandh Kumar	MIT	2013
234354	Agus Hidayat	MIT	2013
234355	Budi Rahayu	MIT	2013
...			
...			

Table: Student

Student ID	Student Name	Address	Suburb	Phone Number	Country
228493	Sooying Tan				Singapore
229094	Xuebing Lu				China
231289	Amandh Kumar				India
234354	Agus Hidayat				Indonesia
234355	Budi Rahayu				Indonesia
...					
...					

A data warehouse based on the above data has been created. A star schema is shown as follows:



Questions:

- The above star schema will not produce a correct analysis of the fact measures. Explain why. Explain your answer using more concrete examples or data.
- How do you correct this problem by changing the fact measures of the above star schema. Explain your solution using more concrete examples or data.

Write your answer here:

Question-a

If the operational database contains 10 students from Indonesia doing MIT in 2013, the fact table will contain only one entry, which aggregates these 10 students. The fact will aggregate the IELTS scores (i.e. listening, reading, writing, speaking, and overall score) and get the average (e.g. average listening, average reading, average writing, average speaking, and average overall score). This will be done for each of combination of country, course, and year.

However, if in the OLAP, we would like to get the average scores of students from a specific country, say Indonesia, then we will need to get all the records where country is Indonesia from the fact table. And then average it again.

The records in the fact table are already averaged. If we average on top of another average, the result will be incorrect.

Continue your answer here:

For example, the fact table of the above sample data is shown as follows:

Table: Fact

CountryName	CourseCode	Year	AvgList	AvgRead	AvgWrit	AvgSpeak	AvgOverall
China	MBIS	2013	5.5	5.5	5.5	5.5	5.5
India	MIT	2013	6.0	7.0	6.0	7.0	6.5
Indonesia	MIT	2013	6.25	6.5	6.5	6.75	6.5
Singapore	MBIS	2013	6.5	6.5	6.0	7.0	6.5

If we query the **Average Listening Score of MIT** students, then based on the above fact table, the average listening score is $(6.0+6.25)/2 = 6.125$.

Supposed in the operational database, there are three students from India and Indonesia (e.g. Amandh from India, and Agus and Budi from Indonesia):

Table: Student_IELTS

Student ID	Student Name	Listening	Reading	Writing	Speaking	Overall
231289	Amandh Kumar	6.0	7.0	6.0	7.0	6.5
234354	Agus Hidayat	5.5	6.0	6.0	6.5	6.0
234355	Budi Rahayu	7.0	7.0	7.0	7.0	7.0

If we look at these three students from India and Indonesia (Amand, Agus, and Budi), the average listening score is $(6.0+5.5+7.0)/3 = 6.17$.

This example shows that keeping the average score in the fact will produce incorrect OLAP query results.

Question-b.

Hence, average is not a good fact measures. We should not have average_reading, average_listening, etc in the fact.

On the other hand, we should have total_reading, total_listening, total_speaking, total_writing, and total_overall in the fact. We also need total_number_of_students in the fact.

So, in OLAP, if we want to get the **Average Listening Score of MIT** course students, we will get the **total_listening** and **total_number_of_students**, and then divide them to get the average. Using the example above, **total_listening** = $(6.0+5.5+7.0) = 18.5$, and **total_number_of_students** = 3. Then listening average is $18.5/3 = 6.17$

Question 2:

There is a toll way (or toll road) in a metropolitan city (such as CityLink or EastLink in Melbourne, or any similar toll roads in other major cities in the world). This toll way has a number of gates, where the motorist needs to pay. Every time a motorist passes through this toll gate, the registration number of the vehicle, vehicle type (e.g. car, bus, truck, etc), amount paid, and time, are recorded in the operational database.

A data warehouse needs to be built, for analysing the *revenue* from the toll payments. The management would like to drill down this revenue based on the *tollgate* (there is a number of toll gates along the toll way), *day of week* (e.g. weekdays, weekends), and *time period of a day* (e.g. peak hours, non-peak hours, late nights).

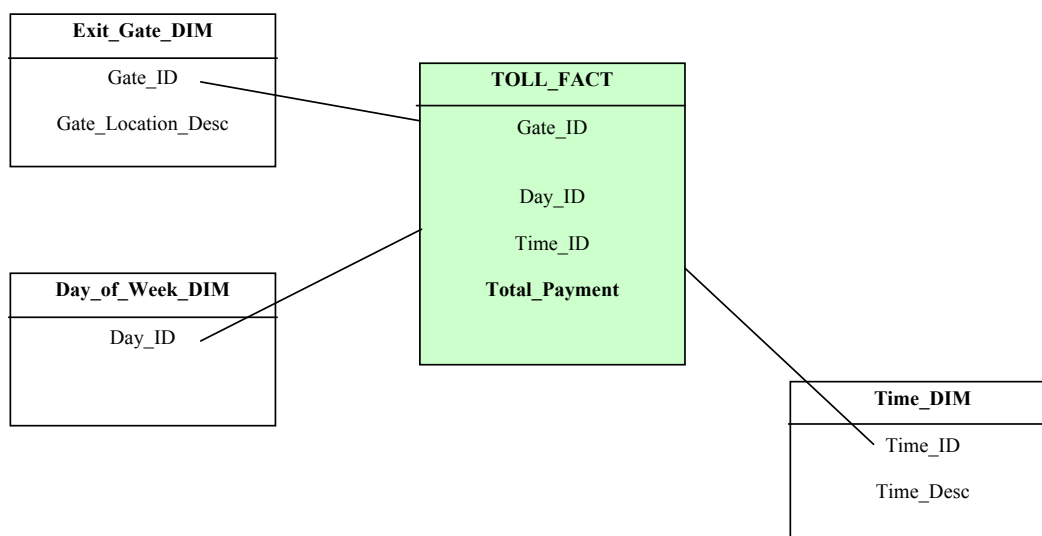
You are required to draw **three levels of star schemas** showing three different levels of aggregation for the above data warehouse. You also need to explain each of the three star schemas, by contrasting the level of aggregation. Level-0 star schema contains the most detailed data, whereas level-2 star schema is the highly aggregated (e.g. containing highly aggregated data).

Questions:

- Draw a level-2 star schema and explain why it is a level-2 schema
- Draw a level-1 star schema and explain why it is a level-1 schema. You may want to add a new dimension, called *vehicle* (e.g. cars, trucks, busses, etc). You need to also explain the difference between level-1 and level-2 schemas.
- Draw a level-0 star schema and explain why it is a level-0 schema. You also need to explain the difference between level-1 and level 0 schemas.

Write your answer here:

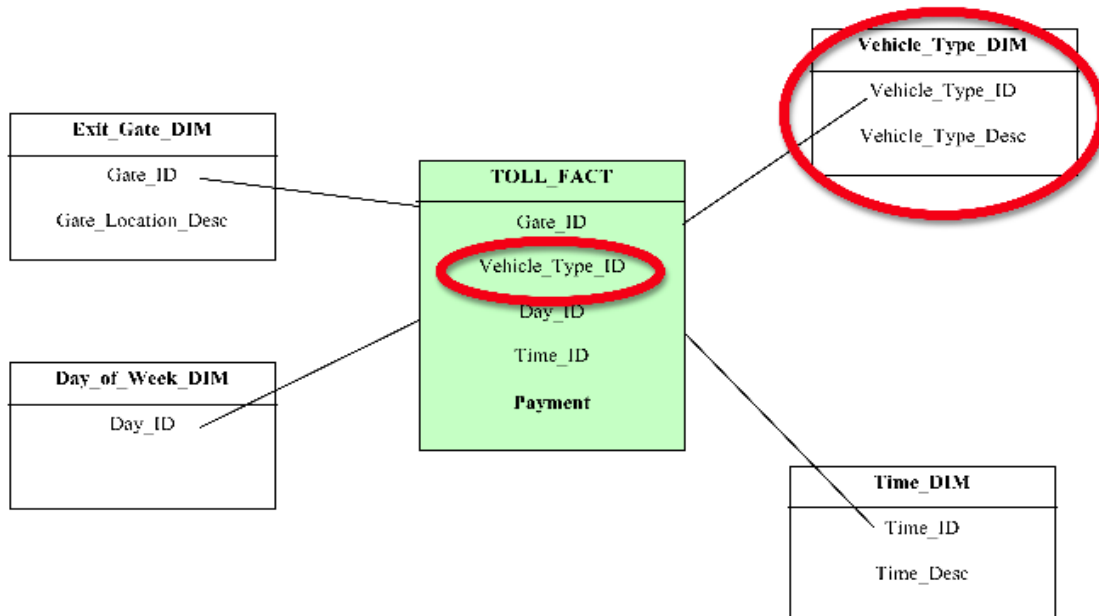
Level 2 – Highly aggregated. This is the most highly aggregated data where we keep “Total Payment” in the Fact table, and the Dimensions are based on Days of Week (not the actual date), and Time Period (not the actual travel time).



Continue your answer here:

Level 1– We add with an additional dimension, for example the Vehicle Type Dimension. In the Vehicle Type Dimension, we store the vehicle types, such as bus, truck, cars, etc. Note that we do not keep the “Registration Number” of the car. The granularity is at the vehicle type level, not at an individual car level.

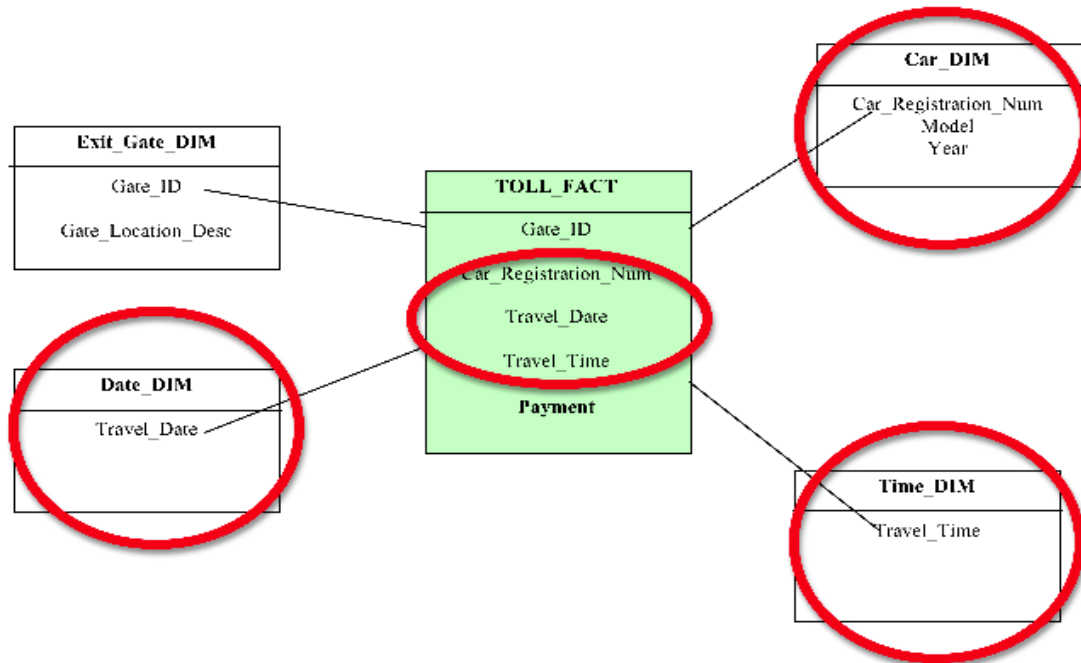
So, from level 2 to level 1, the total payment in level 2 is broken down by Vehicle Type. Other dimensions remain unchanged.



Continue your answer here:

Level 0– Detail Level

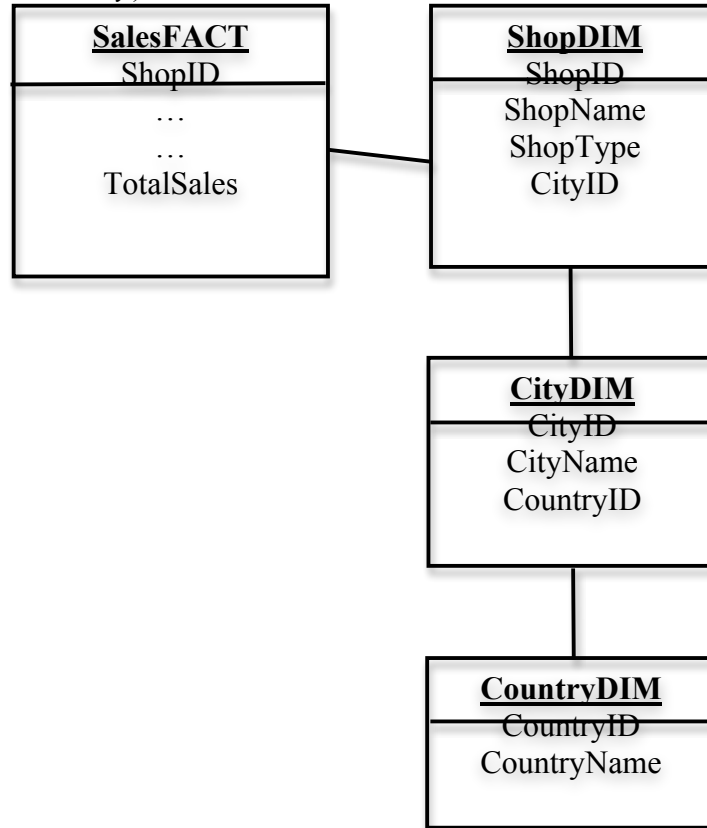
This is the most detail level as we have Date_DIM and Time_DIM to indicate the actual travel dates and travel time; as well as Car DIM to indicate the actual car (rather than just the type of vehicle)



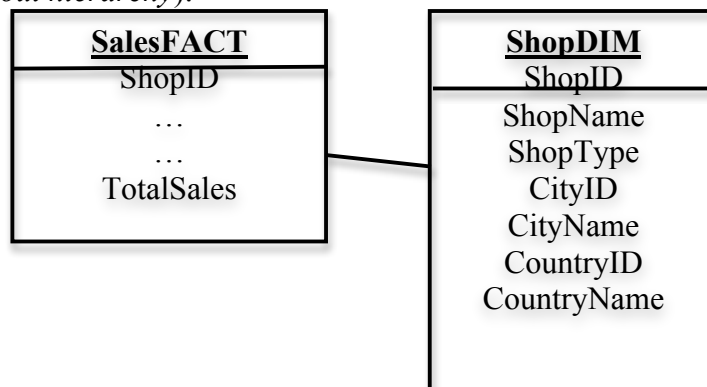
Question 3:

Consider the following star schemas. Star Schema-1 contains a hierarchy in the dimension, whereas Star Schema-2 collapses the hierarchy into one dimension.

Star Schema-1 (with hierarchy):



Star Schema-2 (without hierarchy):



Questions:

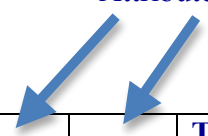
- Draw sample table contents of the fact and dimension tables of the two star schemas.
- Compare and contrast the two star schemas using the sample tables in question (a) above. Explain the pros and cons of each star schema.

Write your answers here:

(a) Star Schema-1

Attributes from other dimensions

ShopFACT



ShopID			TotalSales
1	\$1,500,000
2	\$2,750,000
3	\$1,800,000

ShopDIM

ShopID	ShopName	ShopType	CityID
1	Myer	DepartmentStore	C
2	Coles	GroceryStore	C
3	BigW	DepartmentStore	M

CityDIM

CityID	CityName	CountryID
C	Canberra	AD
M	Melbourne	AD

CountryDIM

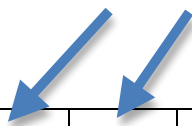
CountryID	CountryName
AD	Australia

Continue your answers here:

Star Schema-2

Attributes from other dimensions

ShopFACT



ShopID			TotalSales
1	\$1,500,000
2	\$2,750,000
3	\$1,800,000

ShopDIM

ShopID	ShopName	ShopType	CityID	CityName	CountryID	CountryName
1	Myer	DepartmentStore	C	Canberra	AD	Australia
2	Coles	GroceryStore	C	Canberra	AD	Australia
3	BigW	DepartmentStore	M	Melbourne	AD	Australia

(b)

Star Schema-1

Pros: normalized, minimized data duplication

Cons: When producing a report, we need to join the fact table with three dimension tables: shopDim, cityDim, and countryDim. Hence, we need three join operations.

Star Schema-2

Cons: unnormalized, has data duplication

Pros: When producing a report, we need to join the fact table with shopDim only. Hence, we need one join operation only.

Question 4:

Data cleaning is an important part in building a clean and correct data warehouse. Data cleaning is often needed, because there are mistakes and inconsistencies in the operational database. Before data cleaning is done, we need to do data exploration on the operational database in order to find out if there are any mistakes and inconsistencies in the operational database.

The following are the four tables in the operational database:

```
SQL> desc dw.uselog;
```

Name	Null?	Type
LOG_DATE	NOT NULL	DATE
LOG_TIME	NOT NULL	DATE
STUDENT_ID	NOT NULL	CHAR(11)
ACT		CHAR(1)

```
SQL> desc dw.student;
```

Name	Null?	Type
SEX		VARCHAR2(2)
FULL_PART		VARCHAR2(2)
TYPE		VARCHAR2(4)
CLASS_ID		VARCHAR2(6)
MAJOR_CODE		VARCHAR2(8)
STUDENT_ID	NOT NULL	CHAR(11)

```
SQL> desc dw.class;
```

Name	Null?	Type
CLASS_DESCRIPTION		CHAR(50)
CLASS_ID		VARCHAR2(6)

```
SQL> desc dw.major;
```

Name	Null?	Type
MAJOR_NAME		CHAR(35)
MAJOR_CODE		VARCHAR2(8)

Questions:

- Write the SQL command to find out if there are duplicate student records (**5 marks**)
- Write the SQL command to find out if there are records in dw.uselog whereby the Student_ID exists in dw.uselog actually do not exist in dw.student (**5 marks**)

Write your answers here:

(a) 5 marks (group by and count = 2 marks, having = 3 marks)

```
select student_id, count(*)
from dw.student
group by student_id
having count(*) > 1;
```

Notes for studying: students need to understand the difference between WHERE and HAVING

(b) 5 marks (not in = 3 marks, subquery = 2 marks)

```
select *
from dw.uselog
where student_id NOT IN
    (select student_id from dw.student);
```

Notes for studying: students need to understand nested queries

Question 5:

An established real estate agent in Melbourne has started their business many years ago and has implemented a very simple database system. The simple database system consists of one large table listed below.

Table Name: PROPERTY	
Field Name	Description
Key	Unique key
Date_offered	Date property offered to the public
Summary	Short description of the property
Adtext	Longer description of the property
Url	The URL of the advertisement
Address	Property address
Suburb	Property suburb name
Postcode	Property postcode
Longitude	Longitude of address
Latitude	Latitude of address
Category	'Residential' or 'Commercial'
Zoning	Commercial Zoning Type
Property_type	Residential Property Type: 'House', 'apartment', or 'lot'
Houseprice	Price of property
Num_bedrooms	Number of bedrooms
Lot_size	Size of the lot
Heating	'ducted', 'gas', 'open fireplace' or 'wood'
Garage	Type of garage
Ensuite	'yes' or 'no'
Balcony	'yes' or 'no'
Pool	'yes', 'no'
Tennis_court	'yes', 'no'
Spa	'yes', 'no'
Aspect_facing	'north', 'south', 'east', or 'west'
School_distance	Distance to nearest school – in km
Shops_distance	Distance to nearest shops – in km
Train_distance	Distance to nearest train station – in km
Bus_distance	Distance to nearest bus stop – in km
Hospital_distance	Distance to nearest hospital – in km
Major_road_distance	Distance to nearest major road – in km

Based on this information, the manager of the real estate agent requires a data warehouse for analysis purposes. The final star schema may include a junk dimension.

Questions:

- What is a junk dimension? Explain!!! Also, use a sample data to illustrate a junk dimension
- Design two versions of star schema for the above case study; one without a junk dimension, and the other with a junk dimension. Compare and contrast these two schemas; focusing on the junk dimension only

Write your answer here:

Solution a:

A junk dimension is a dimension where each of the non-key attributes contains only a small number of possible values (e.g. Yes/No, North/South/West/East, 1/2/3). Therefore the content of a junk dimension is a Cartesian product of the values of all its attributes.

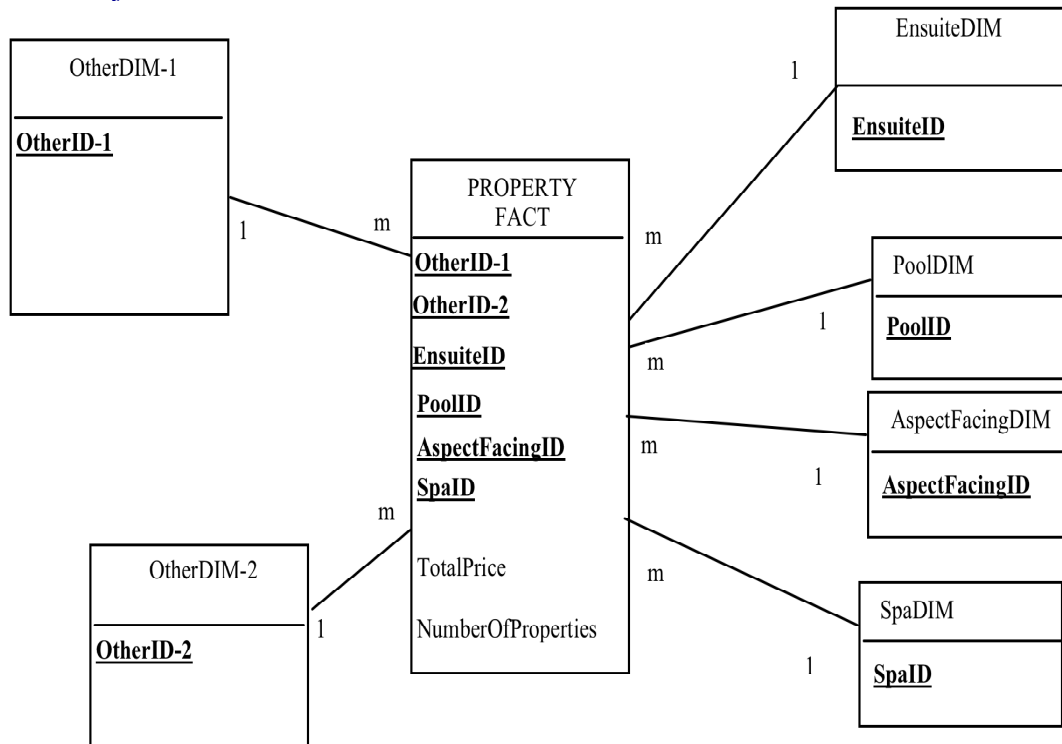
Sample data of a junk dimension:

JunkID	EnsuiteID	PoolID	AspectFacingID	SpaID
1	Yes	Yes	North	Yes
2	Yes	Yes	North	No
3	Yes	Yes	South	Yes
4	Yes	Yes	South	No
5	Yes	Yes	East	Yes
6	Yes	Yes	East	No
7	Yes	Yes	West	Yes
8	Yes	Yes	West	No
9	Yes	No	North	Yes
10	Yes	No	North	No
11	Yes	No	South	Yes
12	Yes	No	South	No
13	Yes	No	East	Yes
14	Yes	No	East	No
15	Yes	No	West	Yes
16	Yes	No	West	No
17	(repeat the above for No Ensuite)			
	...			
	...			
32	...			

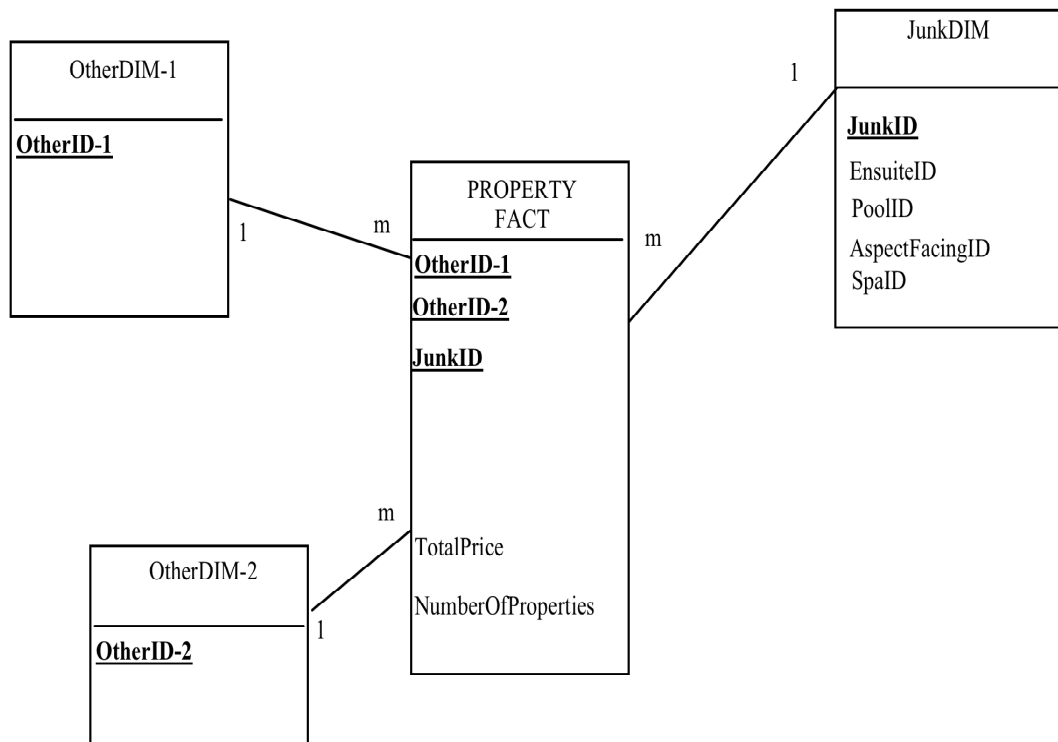
Continue your answer here:

Solution b:

Non-junk dimension star schema



Junk dimension star schema



The junk dimension basically combines (through a Cartesian product) all the small dimensions into one junk dimension.

Continue your answer here:

Non-junk:

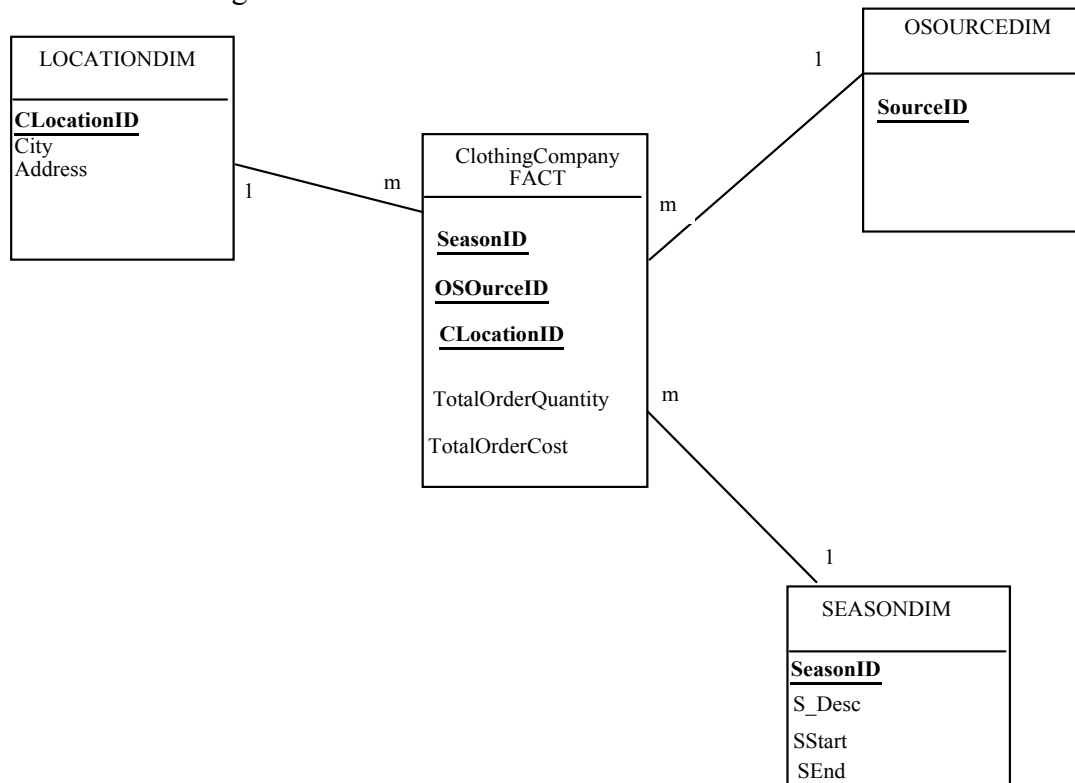
- If we need to create a report involving these four dimensions, we need to have three join operations.
- If we need to have one additional record to the dimension (e.g. aspect facing dimension has one more record called NorthEast, for example), then maintaining the dimension is rather straightforward.

Junk:

- We only need to join the fact and the junkdim. Having one join operation is always better than three join operations.
- If we need to add an additional record to the junk, it will be more difficult.
- Junk dimension is limited to handling small dimensions only. Larger dimensions are not advised to be included in the junk.
- Junk dimension is only suitable to handle a few small cardinality dimensions.

Question 6:

Given the following schema:



The tables (e.g. ClothingCompany fact and the three dimensions) have been created and populated with an adequate number of records.

The table names and attributes are shown in the above star schema. In the Fact table, the total order quantity and total order cost attributes are included.

Write the SQL for the following advanced OLAP queries:

- Perform a **CUBE** operation (use all dimensions). Display each TotalOrderCost and the subtotals.
- Like question (a) above, but now perform a **ROLLUP** operation.
- Perform a **CUMMULATIVE SUM** of the TotalOrderCost of all WEBSITE orders (use all dimensions).
- Like question (c) above, perform a CUMMULATIVE SUM of the TotalOrderCost but **PARTITIONED** based on the OSourceID, that is one partition for Phone orders, one partition for Fax orders, and one partition for Website orders.
- Show the total order costs of each source order, and **RANK** them.
- Display the source order that generates the highest total order cost.

Write your answer here:

a)

```
Select l.city, s.S_Desc, o.sourceid, sum(c.TotalOrderCost) as OrderCost
from LocationDim l, SeasonDim s, OsourceDim o, ClothingCompanyFact c
where l.CLocationID = c.CLocationID
and s.SeasonID = c.SeasonID
and o.sourceid = c.sourceID
group by cube (l.city, s.S_Desc, o.sourceid);
```

Notes:

1. In the above SQL, assume we use l.city (from LocationDIM), and s.s_desc (from SeasonDIM)
2. Alternatively, because OSourceDIM has only one attribute sourceID, we can use sourceID attribute in the fact. In the real-life, OSourceDIM has many more attributes, and therefore, a join with the OSourceDIM table is necessary
3. You can use DECODE and GROUPING for better formatting

```
Select l.city, s.S_Desc, c.sourceid, sum(c.TotalOrderCost) as OrderCost
from LocationDim l, SeasonDim s, ClothingCompanyFact c
where l.CLocationID = c.CLocationID
and s.SeasonID = c.SeasonID
group by cube (l.city, s.S_Desc, c.sourceid);
```

b)

same as (a), but change the cube with rollup

Continue your answer here:

```
c)
Select l.city, s.S_Desc, sum(c.TotalOrderCost),
      TO_CHAR(SUM(SUM(c.TotalOrderCost))
      OVER(ORDER BY l.city, s.S_Desc ROWS UNBOUNDED PRECEDING)),
      '9,999,999.99') AS Cumulative_Total_Order_Cost
from LocationDim l, SeasonDim s, OsourceDim o, ClothingCompanyFact c
where l.CLocationID = c.CLocationID
and s.SeasonID = c.SeasonID
and o.sourceid = c.sourceID
and o.SourceID = 'WEBSITE'
group by (l.city, s.S_Desc);
```

Notes:

1. The above will produce a listing sorted by City and then by Season (e.g. seasons within the same city will be sorted according to the season). The cumulative will be based on this ordering. This will only work if the query has ONE city. If it has multiple cities, it will not make sense that the cumulative total order cost for the second city will start from the first city the first season. Hence, the above query should be like this (e.g. to limit to one city only, such as Melbourne):

```
Select l.city, s.S_Desc, sum(c.TotalOrderCost),
      TO_CHAR(SUM(SUM(c.TotalOrderCost))
      OVER(ORDER BY l.city, s.S_Desc ROWS UNBOUNDED PRECEDING)),
      '9,999,999.99') AS Cumulative_Total_Order_Cost
from LocationDim l, SeasonDim s, OsourceDim o, ClothingCompanyFact c
where l.CLocationID = c.CLocationID
and s.SeasonID = c.SeasonID
and o.sourceid = c.sourceID
and o.SourceID = 'WEBSITE'
and l.city = 'MELBOURNE'
group by (l.city, s.S_Desc);
```

2. If we would like to include multiple cities in the report, the cumulative needs to have a PARTITION, based on city. Hence, for each city, there will be a separate cumulative:

```
Select l.city, s.S_Desc, sum(c.TotalOrderCost),
      TO_CHAR(SUM(SUM(c.TotalOrderCost))
      OVER(PARTITION BY l.city
      ORDER BY l.city, s.S_Desc ROWS UNBOUNDED PRECEDING)),
      '9,999,999.99') AS Cumulative_Total_Order_Cost
from LocationDim l, SeasonDim s, OsourceDim o, ClothingCompanyFact c
where l.CLocationID = c.CLocationID
and s.SeasonID = c.SeasonID
and o.sourceid = c.sourceID
and o.SourceID = 'WEBSITE'
group by (l.city, s.S_Desc);
```

Continue your answer here:

d) Note that we assume that we limit the query to ONE city only:

```
Select o.SourceID, l.city, s.S_Desc, sum(c.TotalOrderCost),
      TO_CHAR(SUM(SUM(c.TotalOrderCost))
      OVER(PARTITION BY o.SourceID
           ORDER BY o.SourceID, l.city, s.S_Desc ROWS UNBOUNDED PRECEDING),
           '9,999,999.99') AS Cumulative_Total_Order_Cost
from LocationDim l, SeasonDim s, OsourceDim o, ClothingCompanyFact c
where l.CLocationID = c.CLocationID
and s.SeasonID = c.SeasonID
and o.sourceid = c.sourceID
and l.city = 'MELBOURNE'
group by (o.SourceID, l.city, s.S_Desc);
```

Notes:

1. The above will produce a listing sorted by SourceID, and then City and then by Season. In this case, there is only one City, which is Melbourne.
2. Each SourceID will have it's own cumulative.
3. If you have multiple cities, you could have a separate cumulative for SourceID combined with City. It means SourceID 'Phone' and City 'Melbourne' will have one set of cumulative, whereas SourceID 'Phone' and City 'Sydney' will have a separate set of cumulative:

```
Select o.SourceID, l.city, s.S_Desc, sum(c.TotalOrderCost),
      TO_CHAR(SUM(SUM(c.TotalOrderCost))
      OVER(PARTITION BY o.SourceID, l.City
           ORDER BY o.SourceID, l.city, s.S_Desc ROWS UNBOUNDED PRECEDING),
           '9,999,999.99') AS Cumulative_Total_Order_Cost
from LocationDim l, SeasonDim s, OsourceDim o, ClothingCompanyFact c
where l.CLocationID = c.CLocationID
and s.SeasonID = c.SeasonID
and o.sourceid = c.sourceID
group by (o.SourceID, l.city, s.S_Desc);
```

Continue your answer here:

```
e)
select OsourceID,
       sum(TotalOrderCost)as OrderCost,
       rank() OVER(ORDER BY sum(TotalOrderCost) DESC)
from ClothingCompanyFact c
group by OsourceID;
```

```
f)
SELECT *
FROM (
  select OsourceID, sum(TotalOrderCost)as OrderCost,
  rank() OVER(ORDER BY sum(TotalOrderCost) DESC) as orderrank
  from ClothingCompanyFact c
  group by OsourceID;
)
WHERE orderrank <=1;
```

Notes:

1. Using MAX will be incorrect:

```
select OsourceID, max(TotalOrderCost)
from ClothingCompanyFact;
```

As you cannot mix an attribute (e.g. OSourceID) and an aggregation function (e.g. max) in one Select statement.

2. You can do this, but it will only show the maximum TotalOrderCost without the OSourceID:

```
select max(TotalOrderCost)
from ClothingCompanyFact;
```

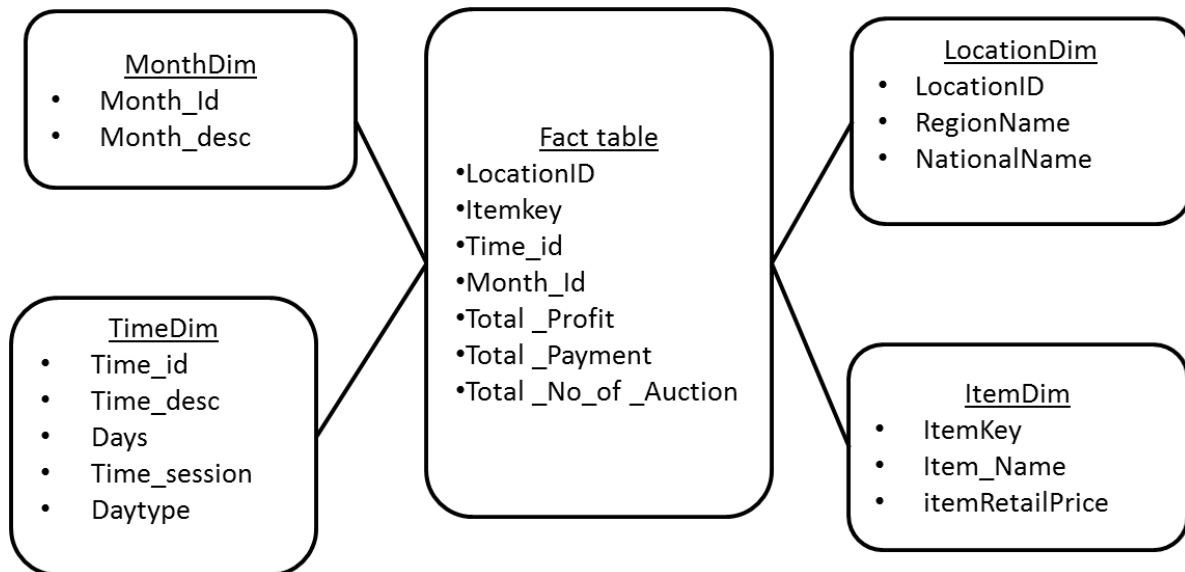
3. You cannot do a group by either:

```
select OsourceID, max(TotalOrderCost)
from ClothingCompanyFact
group by OsourceID;
```

because it will get the maximum TotalOrderCost for each sourceID, which is not what the query asks.

Question 7:

Given the following schema:



The tables (e.g. the fact and four dimensions) have been created and have also been populated with an adequate number of records.

Write the SQL statements for the following OLAP queries:

- Show the top 3 total number of auctions by time sessions.
- Show number of auctions (and subtotals) by month and region.
- Display the total profit, total payment and total number of auction with cumulative sum for each item.

Write your answer here:

A)

```
SELECT * FROM
  (SELECT
    t.time_session,
    sum(f.total_number_of_auctions),
    rank() over (order by sum(f.total_number_of_auctions) DESC) as Rank_top_3
  FROM online_auction_fact f, timeDim t
  WHERE f.time_id = t.time_id
  GROUP BY t.time_session)
WHERE Rank_top_3 <=3;
```

B)

```
SELECT
  DECODE(grouping(m.month_DESC),1,'All Month', m.month_DESC) AS MONTH,
  DECODE(grouping(l.r_name),1,'All Region', l.r_name) as Region,
  Sum(f.total_number_of_auctions) AS total_No_Auction
FROM monthDIM M ,LocationDim l, online_auction_fact f
WHERE f.Locationid=l.locationid
AND f.month_id = m.month_ID
GROUP BY ROLLUP(m.month_DESC,l.r_name);
```

Note: a GROUP BY CUBE will also be accepted, because the question does not specifically ask for a CUBE or ROLLUP.

Continue your answer here:

C)

```
SELECT
  i.i_name, f.month_id,
  sum(f.total_profit) as total_Profit,
  to_char (sum(sum(f.total_profit)) over (order by f.month_id
    rows unbounded Preceding), '9,999,999.99') as Cumulative_Profit,
  sum(f.total_payment) as total_payment,
  to_char(sum(sum(f.total_payment)) over(order by f.month_id
    rows unbounded preceding), '9,999,999,999') as cumulative_payment,
  sum(f.total_no_of_auctions)as total_number_of_Auction,
  to_char(sum(sum(f.total_number_of_auctions)) over(order by f.month_id
    rows unbounded preceding), '9,999,999,999') as cummulative_auctions
FROM online_auction_fact f,itemDim i
WHERE f.i_itemkey = i.i_itemkey
And i.i_name = 'ITEM1'
GROUP BY i.i_name, f.month_id;
```

Alternatively, use PARTITION BY i.i_name:

```
SELECT
  i.i_name, f.month_id,
  sum(f.total_profit) as total_Profit,
  to_char (sum(sum(f.total_profit)) over
    (PARTITION BY i.i_name order by f.month_id
    rows unbounded Preceding), '9,999,999.99') as Cumulative_Profit,
  sum(f.total_payment) as total_payment,
  to_char(sum(sum(f.total_payment)) over
    (PARTITION BY i.i_name order by f.month_id
    rows unbounded preceding), '9,999,999,999') as cumulative_payment,
  sum(f.total_no_of_auctions)as total_number_of_Auction,
  to_char(sum(sum(f.total_number_of_auctions)) over
    (PARTITION BY i.i_name order by f.month_id
    rows unbounded preceding), '9,999,999,999') as cummulative_auctions
FROM online_auction_fact f,itemDim i
WHERE f.i_itemkey = i.i_itemkey
GROUP BY i.i_name, f.month_id;
```


Question 8:

Given the following two tables: customer1 and order1,

```
SQL> desc customer1;
```

Name	Null?	Type
CUSTID		NUMBER(10)
LNAME		VARCHAR2(20)
FNAME		VARCHAR2(20)
ADDRESS		VARCHAR2(20)
PHONE		VARCHAR2(10)
CITY		VARCHAR2(20)

```
SQL> desc order1;
```

Name	Null?	Type
ORDERID		NUMBER(10)
ORDERDATE		DATE
PAYMETHOD		VARCHAR2(20)
ORDERSOURCE		VARCHAR2(20)
CUSTID		NUMBER(10)

table customer1 has the following Primary Key (PK):

```
Alter Table Customer1
Add Constraint Cust_ID_PK PRIMARY KEY (CustID);
```

The following that joins tables customer1 and order1 uses a USE_HASH hint:

```
Select /*+ USE_HASH (C) */ *
From Order1 O, Customer1 C
Where O.CustID = C.CustID;
```

Answer the following two questions:

- Will the execution plan of the above join query follow the USE_HASH hint or not. Give a reason for your answer. **(5 marks)**
- Draw the query tree of the above query **(5 marks)**

Write your answer here:

The query uses an incomplete hint (USE_HASH C). And therefore, the query will not obey the hint (the system will not obey the hint to use a HASH operation). On the other hand, because the customer1 table has PK, the system will opt for a SORT-MERGE operation, using the customer1 PK INDEX.

Drawing the execution table is not necessary, but this is given as a reference:

```
SQL> Explain Plan For
  2 Select /*+ USE_HASH (C) */ *
  3 From Order1 O, Customer1 C
  4 Where O.CustID = C.CustID;
```

Explained.

```
SQL> Select * From Table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 386756218

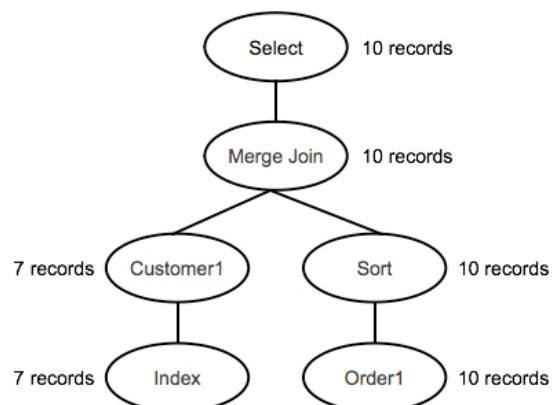
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10	780	6 (17)	00:00:01
1	MERGE JOIN		10	780	6 (17)	00:00:01
2	TABLE ACCESS BY INDEX ROWID	CUSTOMER1	7	343	2 (0)	00:00:01
3	INDEX FULL SCAN	CUST_ID_IDX	7		1 (0)	00:00:01
* 4	SORT JOIN		10	290	4 (25)	00:00:01
5	TABLE ACCESS FULL	ORDER1	10	290	3 (0)	00:00:01

Predicate Information (identified by operation id):

```
4 - access("O"."CUSTID"="C"."CUSTID")
    filter("O"."CUSTID"="C"."CUSTID")
```

18 rows selected.

SQL>



Question 9:

Given the following two tables: Customer1 and Order1:

```
SQL> desc customer1;
```

Name	Null?	Type
CUSTID		NUMBER(10)
LNAME		VARCHAR2(20)
FNAME		VARCHAR2(20)
ADDRESS		VARCHAR2(20)
PHONE		VARCHAR2(10)
CITY		VARCHAR2(20)

```
SQL> desc order1;
```

Name	Null?	Type
ORDERID		NUMBER(10)
ORDERDATE		DATE
PAYMETHOD		VARCHAR2(20)
ORDERSOURCE		VARCHAR2(20)
CUSTID		NUMBER(10)

Explain the execution plan of the following queries:

a.

```
Select * from
Customer1
Order By LName;
```

b.

```
Select Paymethod, Count(*)
From Order1
Group By Paymethod
Order By Paymethod DESC
```

Write your answer here:

a.

The Order By operation is basically a Sort operation.

It first reads all the records (for example: 7 records) from table Customer1, and then sorts the 7 records, and finally displays the 7 records. The SORT ORDER BY operation is a sorting operation, which in this example, the query sorts the records based on the attribute specified by the Order By clause in the SQL (i.e. LName).

Notes: the following table and execution plan are not required in the solution.

```
SQL> Select *
      2 From Customer1
      3 Order By LName;
```

CUSTID	LNAME	FNAME	ADDRESS	PHONE	CITY
181	Jane	Adam	229 Clayton Road	9543877	Clayton
183	Judy	Backhouse	122 Rose Street	9235345	Caulfield
179	Narayan	Ramesh	975 Fire Road	9456738	Carlton
107	Smith	John	731 Plenty Road	9231455	Clayton
154	Wallace	Jennifer	291 Berry Street	9234536	Preston
232	Wong	Franklin	638 Voss Street	9756945	Preston
133	Zelaya	Alicia	3321 Castle Ave	9867055	Balwyn

7 rows selected.

The explain plan for the Order By query is as follows:

```
SQL> Explain Plan For Select * from Customer1 Order By LName;
```

Explained.

```
SQL> Select * From Table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 1687277296

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		7	343	4 (25)	00:00:01
1	SORT ORDER BY		7	343	4 (25)	00:00:01
2	TABLE ACCESS FULL	CUSTOMER1	7	343	3 (0)	00:00:01

9 rows selected.

[Continue your answer here:](#)

b.

The method to use to process a query with Group By and Order By is **SORT GROUP BY**. The SORT GROUP BY operation is a combination between the HASH GROUP BY and the SORT ORDER BY operations. The SORT GROUP BY operation basically performs an aggregation function while sorting the records. As a result, the grouping as required by the Group By clause and the sorting as required by the Order By clause are achieved.

Notes: the following table and execution plan are not required in the solution.

```
SQL> Explain Plan For
2  Select Paymethod, Count(*)
3  From Order1
4  Group By Paymethod
5  Order By Paymethod DESC;
```

Explained.

```
SQL> Select * From Table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 1529597951

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2	12	4 (25)	00:00:01
1	SORT GROUP BY		2	12	4 (25)	00:00:01
2	TABLE ACCESS FULL	ORDER1	10	60	3 (0)	00:00:01

9 rows selected.

SQL>

Question 10:

The following are two tables, called customer1 and order1.

```
SQL> desc customer1;
```

Name	Null?	Type
CUSTID		NUMBER(10)
LNAME		VARCHAR2(20)
FNAME		VARCHAR2(20)
ADDRESS		VARCHAR2(20)
PHONE		VARCHAR2(10)
CITY		VARCHAR2(20)

```
SQL> desc order1;
```

Name	Null?	Type
ORDERID		NUMBER(10)
ORDERDATE		DATE
PAYMETHOD		VARCHAR2(20)
ORDERSOURCE		VARCHAR2(20)
CUSTID		NUMBER(10)

The two tables are now added with PK and FK, using the following Alter Table command:

```
Alter Table Customer1
```

```
Add Constraint Cust_ID_PK PRIMARY KEY (CustID);
```

```
Alter Table Order1
```

```
Add Constraint Cust_ID_FK FOREIGN KEY (CustID) REFERENCES Customer1(CustID);
```

Question:

When we do a join query between customer1 and order1 using the following SQL:

```
Select *
From Customer1 C, Order1 O
Where C.CustID = O.CustID;
```

Draw the query tree of the above query.

Write your answer here:

The execution plan shows that the FK does not give any impact at all. The execution plan still uses a SORT-MERGE JOIN, but table Order1 still needs to be sorted. This shows that FK is not implemented by an index, whereas PK is.

Notes: the following execution plan is not required. It is used here as a reference only.

```
SQL> Explain Plan For
```

```
2  Select *
3  From Customer1 C, Order1 O
4  Where C.CustID = O.CustID;
```

Explained.

```
SQL> Select * From Table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 386756218

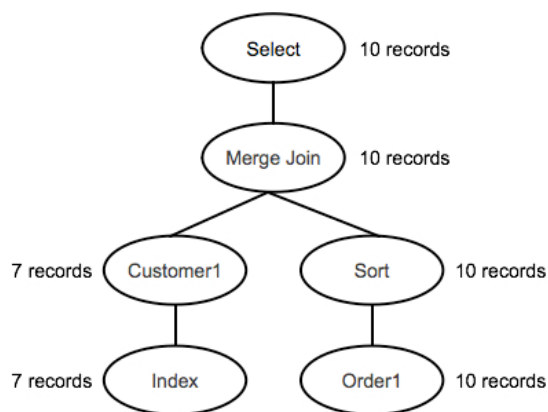
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10	780	6 (17)	00:00:01
1	MERGE JOIN		10	780	6 (17)	00:00:01
2	TABLE ACCESS BY INDEX ROWID	CUSTOMER1	7	343	2 (0)	00:00:01
3	INDEX FULL SCAN	CUST_ID_IDX	7		1 (0)	00:00:01
* 4	SORT JOIN		10	290	4 (25)	00:00:01
5	TABLE ACCESS FULL	ORDER1	10	290	3 (0)	00:00:01

Predicate Information (identified by operation id):

```
4 - access("C"."CUSTID"="O"."CUSTID")
    filter("C"."CUSTID"="O"."CUSTID")
```

18 rows selected.

```
SQL>
```



THE END