Solutions of DM Test 2 (2022/23)

Question 1

a)

	OLTP	DW/OLAP
Users	Operational teams (e.g. clerks)	Analysts, Managers
Hardware utilization	Thousands of operations per second, constant utilization, small operations (read, insert, delete, and update)	Arbitrary use, no pattern, small number of queries, large amounts of operation (read)
Granularity	Higher granularity, very small operations	Diverse granularities, depending on the level of the organization

b)

i) Sometimes a dimension is defined that has no content except for its primary key. For example, when an invoice has multiple line items, the line item fact rows inherit all the descriptive dimension foreign keys of the invoice, and the invoice is left with no unique content. But the invoice number remains a valid dimension key for fact tables at the line item level. This *degenerate dimension* is placed in the fact table with the explicit acknowledgment that there is no associated dimension table. Degenerate dimensions are most common with transaction and accumulating snapshot fact tables (https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/degenerate-dimension/).

ii) Although most measurement events capture numerical results, it is possible that the event merely records a set of dimensional entities coming together at a moment in time. For example, an event of a student attending a class on a given day may not have a recorded numeric fact, but a fact row with foreign keys for calendar day, student, teacher, location, and class is well-defined. Likewise, customer communications are events, but there may be no associated metrics. *Factless fact tables* can also be used to analyze what didn't happen. These queries always have two parts: a factless coverage table that contains all the possibilities of events that might happen and an activity table that contains the events that did happen. When the activity is subtracted from the coverage, the result is the set of events that did not happen (https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/factless-fact-table/).

iii) Transactional business processes typically produce a number of miscellaneous, low-cardinality flags and indicators. Rather than making separate dimensions for each flag and attribute, you can create a single junk dimension combining them together. This dimension, frequently labeled as a transaction profile dimension in a schema, does not need to be the Cartesian product of all the attributes' possible values, but should only contain the combination of values that actually occur in the source data. (FROM https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/junk-dimension/)

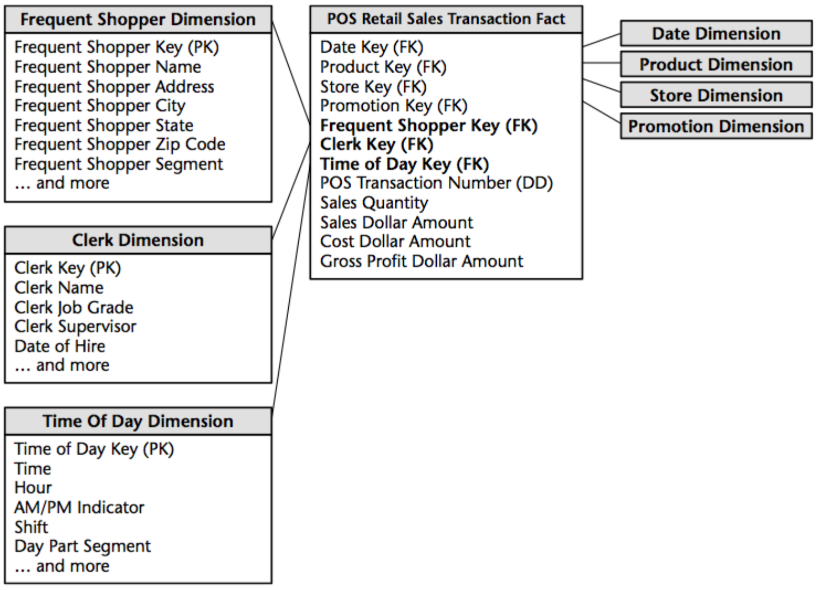
c) Slowly changing dimension *type 2* changes add a new row in the dimension with the updated attribute values. This requires generalizing the primary key of the dimension beyond the natural or durable key because there will potentially be multiple rows describing each member. When a new row is created for a dimension member, a new primary surrogate key is assigned and used as a foreign key in all fact tables from the moment of the update until a subsequent change creates a new dimension key and updated dimension row.
A minimum of three additional columns should be added to the dimension row with type 2 changes: 1) row effective date or date/time stamp; 2) row expiration date or date/time stamp; and 3) current row indicator (https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/type-2/)

The biggest advantage is that the historical records are kept with the correct information, regarding disadvantages queries may be more difficult, and the ETL process can be more demanding to keep the mappings of the natural keys to surrogate key. Can also impact the patterns of access to disk when querying fact tables. If the dimension changes a lot then, the dimension tables can also increase (see next question).

d) A mini-dimension is a subset of attributes from a large dimension that tend to change rapidly, causing the dimension to grow excessively if changes were tracked using the Type 2 technique. By extracting unique combinations of these attribute values into a separate dimension, and joining this new mini-dimension directly to the fact table, the combination of attributes that were in place when the fact occurred are tied directly to the fact record (https://www.kimballgroup.com/2010/09/design-tip-127-creating-and-managing-mini-dimensions/).

An exemple provided in the slides and classes, is the maintenance of demographic profiles.

e) The solution can be found in slide 53 of multidimensional design (see below)



Since the fact table has the highest granularity, then the facts will not be changed. We just have to add the appropriate foreign keys to the new dimensions (see below).

The Time Of Day dimension can be statically constructed with a single query that puts all the possible combinations in the dimension table (i.e., 24*60 = 1440 values). The appropriate foreign key should be determined from the time recorded by the OLTP system, using the degenerate dimension POS Transaction number.

The Clerk Dimension must be obtained from the OLTP system. The appropriate foreign key should be determined from the clerk responsible from the transaction recorded by the OLTP system, using the degenerate dimension POS Transaction number.

The frequent shopper program requires that the existing information should be registered by the user in their marketing campaigns. We have to create a row in the Frequent Dimension table (e.g. with key -1 – "Not a frequent shopper", the default key), and another one (with key 0 – "Sales before start of program"). In this way we can use the customer key to join with the frequent shopper key to determine if a sale in the sales fact is from a frequent shopper. Previous entries to the facts before of the start of the programme should be filled with 0 or -1 depending on the time that a customer joined the programme).

f)

i)

```
SELECT d.month, s.StoreKey, s.name, p.ProductKey, p.name,
m.key, m.campaign, SUM(SalesDollarAmmount), SUM(SalesQuantity)
FROM POSRetailSalesTransactionFact
    INNER JOIN DateDimension d USING (DateKey)
    INNER JOIN StoreDimension s USING (StoreKey)
    INNER JOIN ProductDimension p USING (ProductKey)
    INNER JOIN PromotionDimension m USING (PromotionKey)
GROUP BY d.month, s.StoreKey, s.name, p.ProductKey, p.name,
m.key, m.campaign;
```

ii)

```
SELECT d.day, p.ProductKey, p.name,
      SUM(SalesDollarAmmount), SUM(SalesQuantity)
FROM POSRetailSalesTransactionFact
    INNER JOIN DateDimension d USING (DateKey)
    INNER JOIN ProductDimension p USING (ProductKey)
GROUP BY d.day, p.ProductKey, p.name;
```

// To illustrate a different query to construct all the cubes, having has many rows as the previous rows plus one row for each product, one row for each day, and one extra row with the whole total.

```
SELECT d.day, p.ProductKey, p.name,
    SUM(SalesDollarAmmount), SUM(SalesQuantity)
FROM POSRetailSalesTransactionFact
  INNER JOIN DateDimension d USING (DateKey)
      INNER JOIN ProductDimension p USING (ProductKey)
   GROUP BY CUBE(d.day, p.ProductKey), p.name;
```

Question 2

a) Assuming that what is intended is the result presented in the figure, the corresponding query is:

```
SELECT District, Brand, SUM(DollarSalesAmount), SUM(QuantitySold)
FROM DailySalesFact f INNER JOIN DateDimension d USING (DateKey)
               INNER JOIN ProductDimension p USING (ProductKey)
               INNER JOIN StoreDimension s USING (StoreKey)
WHERE d.year = 2016 and quarter = 'Q1' AND
    p.category = 'Home Appliances' AND
    s.region = 'South Region'
GROUP BY District, Brand
```

If it is simple the total sales then:

```
SELECT SUM(DollarSalesAmount)
FROM DailySalesFact f INNER JOIN DateDimension d USING (DateKey)
               INNER JOIN ProductDimension p USING (ProductKey)
               INNER JOIN StoreDimension s USING (StoreKey)
WHERE d.year = 2016 and quarter = 'Q1' AND
    p.category = 'Home Appliances' AND
    s.region = 'South Region'
```

b) Assuming that what is intended is the result presented in the figure, the corresponding query is:

```
SELECT Month, Brand, SUM(DollarSalesAmount), SUM(QuantitySold)
FROM DailySalesFact f INNER JOIN DateDimension d USING (DateKey)
               INNER JOIN ProductDimension p USING (ProductKey)
               INNER JOIN StoreDimension s USING (StoreKey)
WHERE d.year = 2016 and d.quarter = 'Q1' AND
    p.category = 'Home Appliances' AND
    s.region = 'South Region' AND
    s.district = 'Belmont'
GROUP BY Month, Brand
```

We are slicing by the district attribute of the Store Dimension and drilling-down by the month attribute of the Date Dimension. We obtain 9 rows.

Question 3

a) The minimal entities are Sale Item and Sale Fee (no one to many relationships – leaf entities) and maximal entities are Period, Product Type, Fee Type, Location Type, State, and Customer type (no many to one relationships – root entities)

b) We cannot represent any fact regarding the average price since it is a ratio. See pages 53, and 55 of the slides OLTP2OLAP for a complete model. We can define a simplified star schema model just with the product, customer, and date dimensions, where the SaleItemFact table would contain only the columns:

SaleFact(Sale_date,Cust_id,Sum_of_Discount)
SaleItemFact(Sale_date,Cust_id,Prod_id, Sum_of_Qty, Sum_of_ItemCost)

The primary key of the fact tables are underlined. The dimension tables are as in the slides, except that we would not need Location, and LocationType.

The SQL code would be:

```
INSERT INTO SaleFact(Sale_date, Cust_id, Sum_of_Discount) AS
SELECT Sale_date, Cust_Id, SUM(Discount_Amt)
FROM Sale
GROUP BY Sale_date, Cust_id

INSERT INTO SaleItemFact(Sale_date, Cust_id, Prod_id, Sum_of_Qty, Sum_of_ItemCost) AS
SELECT Sale_date, Cust_Id, Prod_id, SUM(Qty) , SUM(Qty*UnitPrice)
FROM Sale INNER JOIN SaleItem USING (Sale_id)
GROUP BY Sale_date, Cust_id, Prod_id
```

A more difficult version would be to divide the discount proportionally by all products in the corresponding sale. Try it!

c) See solution in page 42, where we create a table for each transaction entity with the collapsed hierarchies. Note that we do not collapse transaction entities into transaction entities to avoid aggregation errors.