Solution Exam Data Warehouses January 2014

1. (4p) Carefully read the description stating the requirements for a data warehouse for a record label:
   A record label wants to keep track of all contracts they have with bands, records they are producing and the sales of these records. Currently they are only keeping data of their ongoing contracts; no historical information is kept. Therefore it is decided to construct a data warehouse for collecting and storing historical information. With the data warehouse the company wants to analyze its sales. Based on conversations with the managers of the company, you were able to compile the following description of the available data.

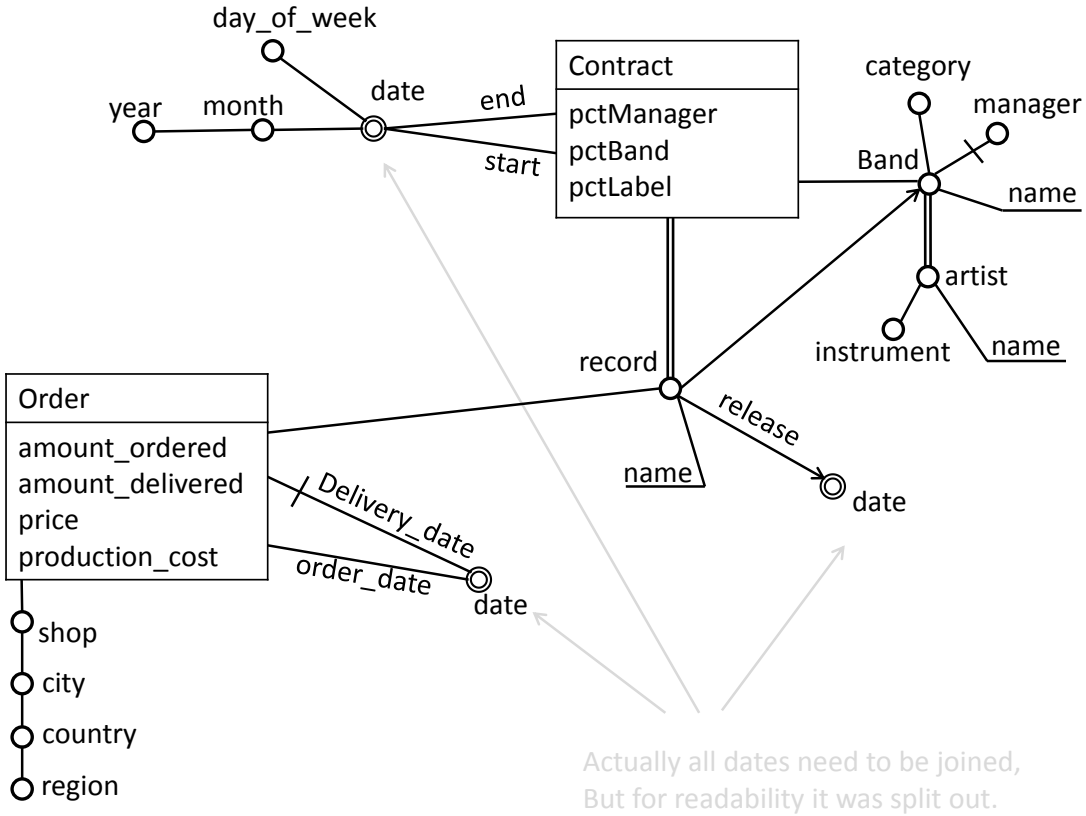
   The record label has several bands under contract. Each band has a name, a main category of music it plays, and one or more artists. A band may have a manager, but does not have to. Bands without a manager must have one of the artists as their main representative. An artist has one main instrument, which can also be his/her voice. Over time, bands can split and attract or replace group members. Bands make records which are physically produced and distributed by the record label. Records, including those that are not yet finished, have a title and are identified by a special international code. For every record the price to produce it and its release date are stored. The record label has contracts with the bands. For every contract start and end data are registered as well as for which records it holds, what percentage of sales goes to the manager of the group (if present), how much to the group, and how much to the record label. A contract can apply to multiple records, but every record falls under exactly one contract. Shops can order records at the record label. The date, amount, and agreed price of these orders are recorded, as well as the number of records that were sent to the shop in the end (sometimes demand outweighs supply) and the date on which the order was fulfilled. For every shop, the record label has information about the spatial location, including: city, country, and region of the shop.

   Some examples of the types of analyses the record label wants to perform based on the data warehouse are the following:

   - *Which record/group/artist has given the record label the highest/lowest gain/return on investment per week/month/year.*
   - *What seller/city/country has the highest number of demands that could not be met?*
   - *Which record/group/artist has the highest number of demands?*

   (a) Draw a dimensional fact model (DFM) for the data warehouse. Notice that there is not a single correct solution and that multiple facts may be needed. For every fact in your model explain succinctly what it represents. Not all information in the description is necessarily relevant for the dimensional fact model. Your model should represent the description as faithfully as possible.

   (b) Translate the DFM you constructed in (a) to an appropriate relational model. Clearly indicate primary and foreign keys in your tables. *In order to gain full points for (b) the model given in your answer to (a) must be reasonable. For instance: not answering (a) and giving an empty relational schema as a (correct) answer for (b) will obviously not gain you any points.*

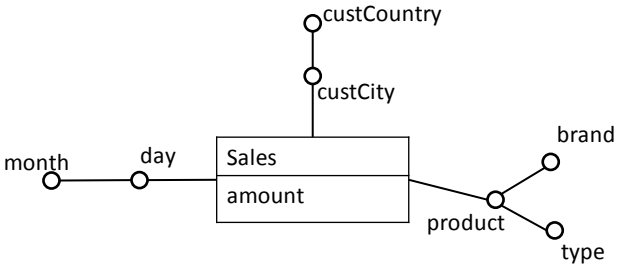


Common mistakes: dimensional attributes that are incorrectly placed into a hierarchy (for instance, roll-up from band to record); many-to-many relations modeled as one-to-many; models with only one fact usually did not work: contracts are not a property of sales, not vice versa. These are two different subjects in the data warehouse, hence requiring two facts. A third fact, for record, or the production of a record could be considered.

A possible relational schema:

- dimDate(DID, day, day_of_week, month, year)
- dimBand(BID, bandName, manager)
- dimArtist(AID, instrument, artistName)
- bridgeBandArtist(BID, AID)
- dimRecord(SpIntCode, releaseDID, recordName, BID)
- recordGroup(RGID, SpIntCode)
- factContract(startDID, endDID, RGID, BID, pctMan, pctBand, pctLabel)
- dimShop(SID, shopName, city, country, region)
- factOrder(deliveryDID, orderDID, SID, SpIntCode, amount_ordered, amount_delivered, price, production_cost)

2. (2.5p) Consider a data cube *Sales* with three dimensions *Customer*, *Date*, and *Product*. Dimension *Customer* has a hierarchy with levels *City* and *Country*. *City* is the lowest level of granularity in this dimension. Dimension *Date* has levels *Day* and *Month*. Dimension *Product* is stored at *product*-level and can be rolled up to level *Type* and level *Brand*. There is only one measure *Amount*. The DFM for the cube is hence as follows:



The users of the reports that are based on this data cube are complaining about degrading performance. So, you analyze the usage statistics in order to find out which queries have been asked how many times. All queries are of the form:

```
SELECT A1, ...,Ak, SUM(Amount)
FROM (Sales joined with the dimension tables)
GROUP BY A1, ..., Ak
```

`A1`, ..., `Ak` are dimensional attributes of the cube. We will represent every query by its set of grouping attributes. For instance: (Day, Brand) represents the query

```
SELECT Day, Brand, SUM(Amount)
FROM (Sales joined with the dimension tables)
GROUP BY Day, Brand
```

The cube is dense. That is: for every combination of a city c, date d, and product p, there is a tuple (c, d, p, a) in the base table with a non-zero amount a. There are 10 000 products of 100 brands and 20 product types, 50 cities from 10 countries, there are 1000 days and 36 months. Hence, in total there are $50 \times 1\,000 \times 10\,000 = 500M$ tuples in the base table. Furthermore, for every brand there is at least one product of each type. The following table summarizes the results.

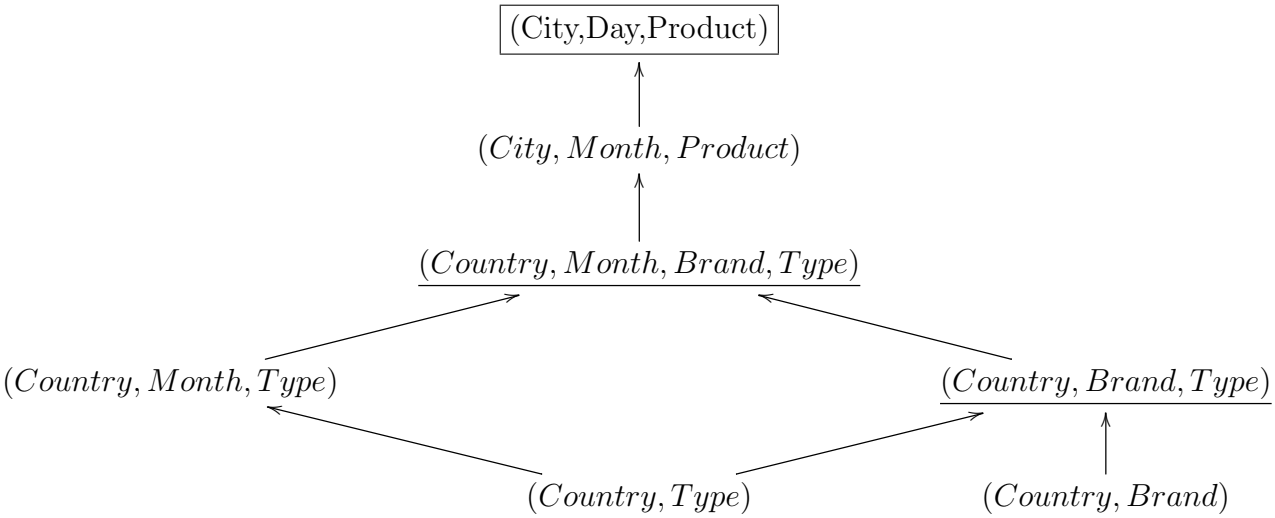
Query	Frequency of query	Output size
(City,Month,Product)	15%	18M
(Country,Brand)	40%	1K
(Country,Type)	20%	200
(Country,Month,Type)	25%	7.2K

No other queries were executed on the data warehouse.

(a) Apply the greedy method described by *Harinarayan, Rajaraman, and Ullman* in their seminal paper "Implementing Data Cubes Efficiently" (SIGMOD 1996) *(this is the greedy algorithm we saw in class)* to select two views to materialize.

(b) What benefit (in %) do you expect from materialization these two views?

Solution: As you will quickly notice if you try, constructing the complete lattice is very cumbersome and time-consuming; it is too large to completely draw it and to compute the benefit of every single view in it. Fortunately, we do not need the complete lattice. Since that there are only 4 views that are ever asked, we can restrict the views we need to consider. For instance, we do not need to consider the view (Country,Product); indeed: if we would materialize (Country, Product), of the 4 views only (Country,Brand) and (Country,Type) benefit. But, these 2 views would benefit even more if we materialize the smaller (Country,Brand,Type). Given the relation between the 4 views, it suffices to consider the following additional views: (Country,Brand,Type), and (Country,Month,Brand,Type). (*Convince yourself by trying out some other views; always one of the following six views will be even better.*) This gives us the following roll-up lattice:



Of these 6 views we now compute the benefit of materializing them:

View	Benefit
(City,Month,Product)	$100\% \times (500M - 18M)$
(Country,Month,Brand,Type)	$85\% \times (500M - 720K)$
(Country,Month,Type)	$45\% \times (500M - 7.2K)$
(Country,Brand,Type)	$60\% \times (500M - 20K)$
(Country,Type)	$20\% \times (500M - 200)$
(Country,Brand)	$40\% \times (500M - 1K)$

(City,Month,Product) gives the highest benefit, so this view is selected as the first to materialize. Then the benefits are computed again:

View	Benefit
(Country,Month,Brand,Type)	$85\% \times (18M - 720K)$
(Country,Month,Type)	$45\% \times (18M - 7.2K)$
(Country,Brand,Type)	$60\% \times (18M - 20K)$
(Country,Type)	$20\% \times (18M - 200)$
(Country,Brand)	$40\% \times (18M - 1K)$

This time (Country,Month,Brand,Type) gives the highest benefit and is selected.

The total benefit is: $(500M - 18M) + 85\% \times (18M - 720K)$. The relative benefit is hence:

$$\frac{(500M - 18M) + 85\% \times (18M - 720K)}{500M} \approx 0.99\%$$

3. (2.5p) An insurance company has a data warehouse holding all contracts of its customers. One of the dimensions is *dimCustomer* which is storing information about customers. This dimension holds the following properties for a customer:

   - passport number. The passport number uniquely identifies the customer.
   - first name, middle name, last name
   - date of birth and gender
   - * street, number, postal code, city, country
   - * marital status (single, married, divorced, widowed)
   - * number of children
   - * whether or not he or she is a home owner
   - * whether or not he or she owns a car
   - * whether or not he or she is a VIP client
   - * credit score (A+, A, B, C, or D)
   - * market segment (a number from 1 to 8)
   - * risk categorization as a car driver (number 1 to 10; starts at 5, decreases 1 per year, increases if person has an accident)
   - * health categorization (low risk, medium, or high risk)

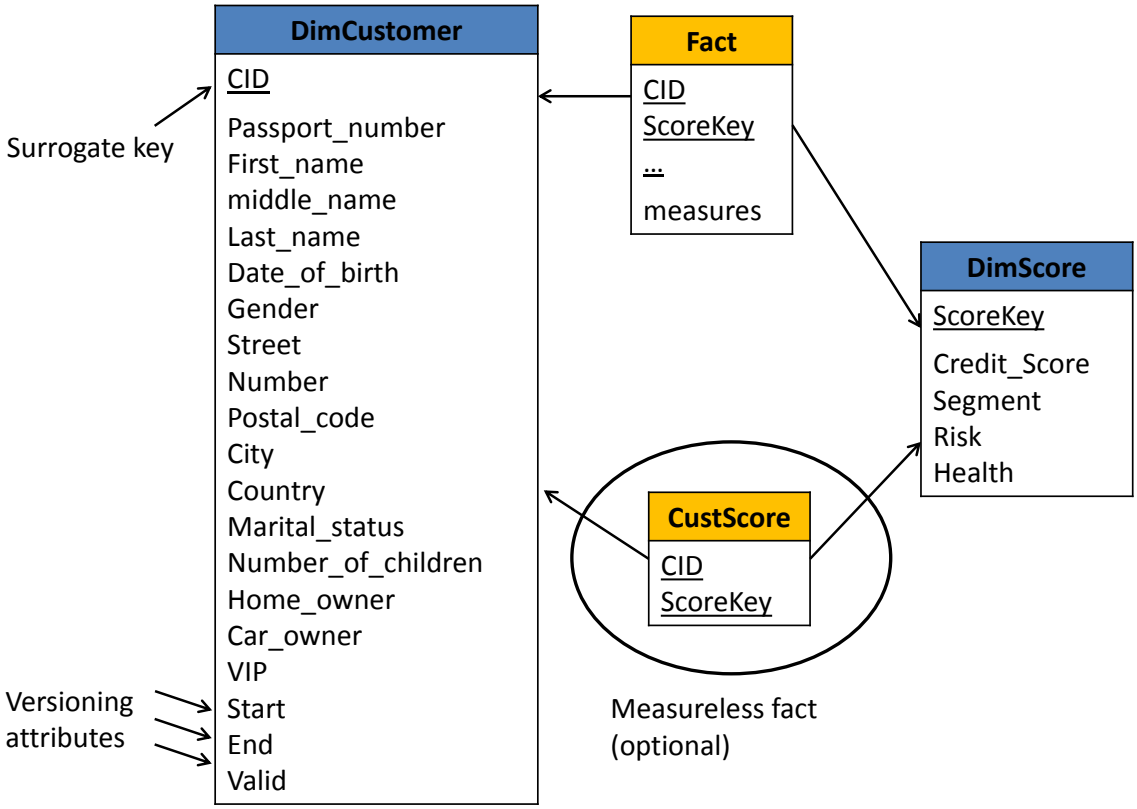
   The attributes with a star indicate attributes for which the history has to be maintained. The attributes without * are in principle not changing, except maybe for errors that may have to be corrected. You may assume that the address of a person, his or her marital status and number of children, whether he or she is a homeowner/car-owner, and whether he or she is a VIP client do not change very frequently. The credit score, market segment, risk and health categorizations, however, change frequently.

   (a) Give the relational table(s) with its (their) attributes for storing this dimension in a ROLAP solution.

   Solution: To keep track of the type-II changes, we need to introduce versioning of the tuples. Therefore we need at least a surrogate key. Furthermore, later when updating the database, we will need to be able to identify the current version of the tuple. As such we also need either an attribute "valid" or attributes "start" and "end" that indicate the valid time of the tuple.

   Furthermore, since some of the attributes are rapidly changing, it is better to split the dimension into two separate dimensions; one of them a so-called *mini-dimension.* The mini-dimension consists of the attributes credit score, market segment, risk, and health categorizations. These are placed in a separate dimensional table with its own surrogate key. The mini-dimension should not be versioned, and the key of the mini-dimension needs to be added in the fact table, not in the customer dimension table. Given that the mini-dimension contains only few attributes, each with only few different values, it is a good idea to completely materialize it right from the start. It does not give any benefit, on the contrary even, to further split up the customer dimension. If order to maintain the changes in the customer dimension, even if the customer does not generate new facts, one could add a measureless

fact-table with dimensions customer and the newly created mini-dimension. A fact in this table would then represent a change in the customers credit score, market segment, risk or health categorization. This complete solution is depicted below:



(b) Describe the process of updating this (these) dimension table(s) for the following updates (the attributes for which no value is specified are not important for this exercise). Clearly indicate which tuples are added in what table(s). Every update is assumed to take place during different data warehouse updates. That is: between every two changes in the source databases there is a data warehouse update.

i. A new customer with passport number 1234 and name Jan Janssens is added to the database. He is single, has no children and has a perfect credit score A+.
Solution: A new tuple is added to DimCustomer. A new surrogate key CID is generated for this tuple. Valid is set to true for this tuple; start is the current time, and end is null. One fact is added to the measureless fact table CustScore, connecting the CID of Jan Janssens with the right combination of scores.

ii. The customer with passport number 1234 gets married.
Solution: The current version of the tuple for 1234 (the tuple with Valid=true and end is null) gets updated: end is set to the current time, and Valid to false. A new version for 1234 is inserted with his updated marital status. A new fact is added in CustScore to connect this new version to the correct combination of scores.

iii. Customer 1234 buys a house. Obviously his address changes.

Solution: Similar as previous.

iv. The name of customer 1234 is corrected to Jan Jansens (one s is removed from the last name).
Solution: This is a type 1 change. All versions of the tuple for customer 1234 are updated.

v. Customer 1234 becomes the father of twins. His credit score drops to B.
Solution: The current version of the tuple for 1234 (the tuple with Valid=true and end is null) gets updated: end is set to the current time, and Valid to false. A new version for 1234 is inserted with his updated parenthood status. A new fact is added in CustScore to connect this new version to the updated combination of scores.

vi. Customer 1234's credit score raises again to A. He buys a car.
Solution: The current version of the tuple for 1234 (the tuple with Valid=true and end is null) gets updated: end is set to the current time, and Valid to false. A new version for 1234 is inserted with his updated car owner status. A new fact is added in CustScore to connect this new version to the updated combination of scores.

4. (1p) Consider the following relational model (this relational model was one of the solutions of the exercises in week 1):

   State(sName, region)
   District(dID, dName, sName)
   Congressman(cName, dID, sDate, party)
   Bill(bName, bDate, passed)
   Sponsors(cName, bName, bDate)
   Voted(cName, bName, bDate, vote)

   There are the following foreign key dependencies:

   - District(sName) references State
   - Congressman(dID) references District
   - Sponsors(cName) references Congressman and Sponsors(bName,bDate) references Bill
   - Voted(cName) references Congressman and Voted(bName,bDate) references Bill

   For this database, give an example of a bitmap-join index, and a query that benefits from this index.

Solution: Examples of a useful bitmap-join index are: for sName into table Sponsors (sName into Voted, for party into sponsors if in combination with other indices, for party into voted, if in combination with other indices). Example query for which this index is useful:

```
SELECT count(distinct bName, bDate)
FROM Sponsors S, Congressman C, District D
WHERE S.cName=C.cName AND C.dID=D.dId AND D.sName="Florida";
```

For this query we don't even have to do any of the joins, the result can directly be derived from the index.

(Common errors included: giving a bitmap index instead of a bitmap join index, or selecting an index that was insufficiently selective—unless used in combination with other bitmap/bitmap-join indices; stating a query that does not have a selection on the indexed attribute.)