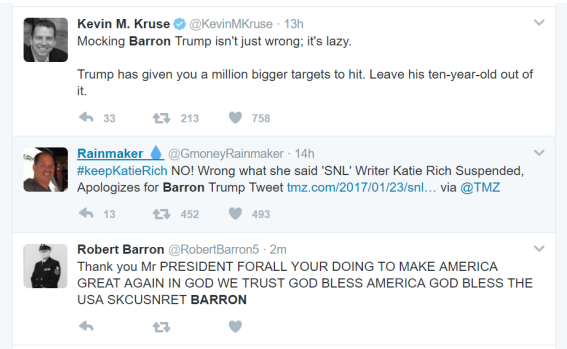
INFOH419 Data Warehousing January 2017
Toon Calders - Hatem Haddad

The exam is **open book**, so all books and notes can be used. "Open book" implies that you can refer to a specific slide, book page, or exercise. Verbatim copying lecturing material will not be rewarded. Stay focused on the question and avoid excessively long answers; succinct, to the point answers will be rewarded.
The questions in the exam should all be self-explanatory. In case of doubt, make a short note and solve the question to the best of your knowledge.
The maximal time to complete this exam is 3h and will be strictly observed.

---

1. (6 points) Consider the following requirement analysis: *We would like to store tweets of users in order to do market analysis. For that purpose, we capture the twitter stream, and store the information of interesting tweets in a data warehouse. For each tweet we store: time and date; the message itself; hashtags (for instance #DataWarehousing); mentions of users (for instance @tcalders); whether they are re-tweets and if so, of which tweet; and the user that sent the tweet. Furthermore, we perform natural language processing on the tweets resulting in a characterization of the sentiment of the tweet (euphoric, happy, neutral, sad, angry, and outrageous are the possible values.) Of all users we have the full name as registered by twitter. For some users, but not all, we have the demographic characteristics age category, gender, and nationality.*

   For example, for the following tweets on the left, the data is displayed on the right.



Tweet 1: user KevinMKruse full name Kevin M. Kruse
message sent 24/1/2017 at 2:43
sentiment: sad
no mentions, no hashtags

Tweet 2: user GmoneyRainmaker full name Rainmaker
message sent 24/1/2017 at 1:55
sentiment: angry
mentions @TMZ, hashtags #keepKatieRich

Tweet 3: user RobertBarron5 full name Robert Barron
message sent 20/11/2016 at 13:55
sentiment: euphoric
no mentions, no hashtags

   Each tweet has exactly one user and one sentiment associated to it, but may have 0 or more mentions, and 0 or more hashtags. A tweet can be a retweet of exactly one other tweet, or be an original tweet.

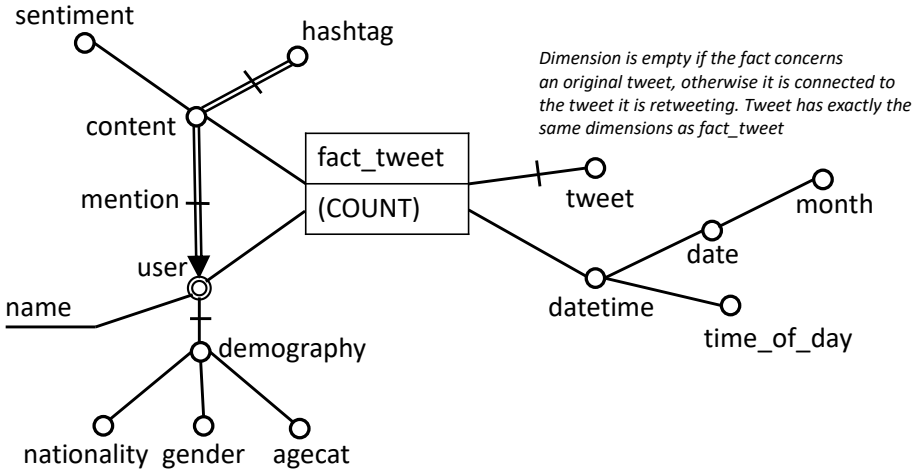
   *The datawarehouse storing the tweets will be used afterwards to answer questions **like** (non-exhaustive list!): for the month December 2016, give for each sentiment category the number of tweets which contained #trump; give per month the number of tweets which contain hashtag #barron; give per month how many times the tweets of user @realDonaldTrump are retweeted on average.*

   (a) Draw a dimensional fact model (DFM) for the data warehouse that will hold this information. 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.

Solution:

(a) The main difficulty in this assignment is how to deal with retweets; essentially a retweet is a tweet, and tweet is the logical choice for the fact. This leaves us with a number of options; either we completely copy the model part representing a tweet, once for the fact and once for the dimension. On the other hand we could also consider tweet as the single dimension, and make a "tweetevent" fact. But, in the translation this would result in a fact table with only one attribute referring to the identifier of the tweet. Also some intermediate solutions are possible, where a dimension tweet is introduced which contains relevant tweet information that is also of interest for the retweets, and other dimensions are directly connected to the fact. In the solution below we have opted for considering a retweet as a separate dimension which is structurally equivalent to the tweet fact. In the second part of the assignment we take this into account by storing them together in one table.

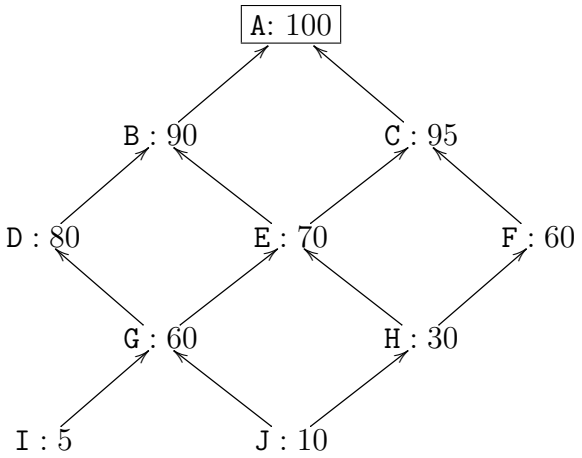


Notice that with this schema we can answer all proposed queries. For instance for the last one (number of retweets per month of tweets by @realDonaldTrump): count the number of tweets by month, that are retweets of a tweet by user @realDonaldTrump.

(b) For tweets and retweets the same table factTweet will be used. Therefore we have chosen to introduce a surrogate key into the fact table instead of using the combination of dimension keys as primary key. Depending on the solution chosen, however, other approaches may be correct as well. Primary keys are underlined, and foreign keys are identified by identical names; only if there is a foreign key dependency attribute names are reused in our solution. There is one additional foreign key dependency: ReTweetID in table factTweet refers to primary key TweetID in factTweet.

- dimDate(DID,day,month,year)
- dimTime(TID,hour,minute,second)
- hashtagGroup(HGID,hashtag)
- mentionGroup(MGID,UID)
- dimUser(UID,name,DemID)
- dimDemography(DemID,nationality,gender,agecat)
- dimContent(TweetID,text,sentiment,HGID,MGID)
- factTweet(TweetID,DID,TID,UID,ReTweetID)

2. (3 points) An important technique to speed up analytical queries is by pre-computing and materializing aggregations. Consider the following lattice of views that can be requested by the user, along with the size of each view.

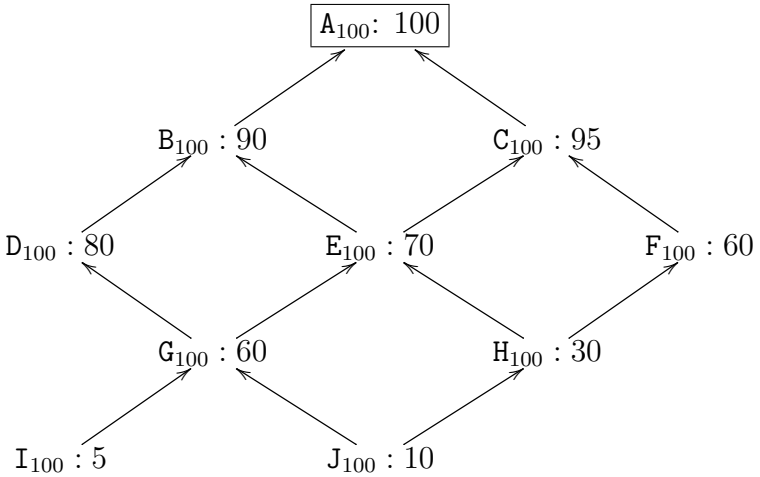


A is the view representing the base relation. The edges indicate the relation "can be computed from."

(a) Suppose that only the top-view A has been materialized. Select 4 additional views from the views B, C, D, E, F, G, H, I, and J to materialize. Apply the greedy method described by *Harinarayan, Rajaraman, and Ullman* in their seminal paper "Implementing Data Cubes Efficiently" (SIGMOD 1996) in order to optimize the overall query time.

(b) What is the gain under the cost model of *Harinarayan et al.* that you obtain by materializing these 4 views?

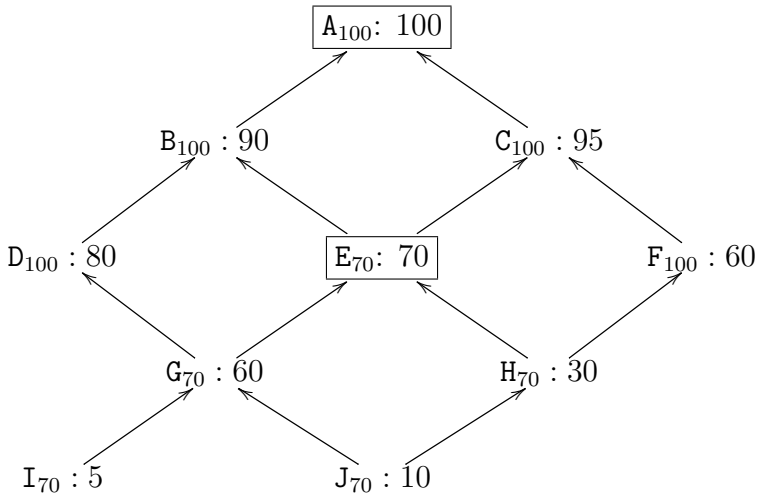
Solution: (a) As long as only A is materialized, the costs for computing the views is as follows (cost in subscript):



The benefits of materializing the different views are as follows:

View	benefit
B	$7 \times 10 = 70$
C	$7 \times 5 = 35$
D	$4 \times 20 = 80$
E	$\mathbf{5 \times 30 = 150}$
F	$3 \times 40 = 120$
G	$3 \times 40 = 120$
H	$2 \times 70 = 140$
I	$1 \times 95 = 95$
J	$1 \times 90 = 90$

Hence we first materialize E, resulting in a benefit of 150. The costs of the queries (in subscript) under the materialized views (in rectangles) now become:

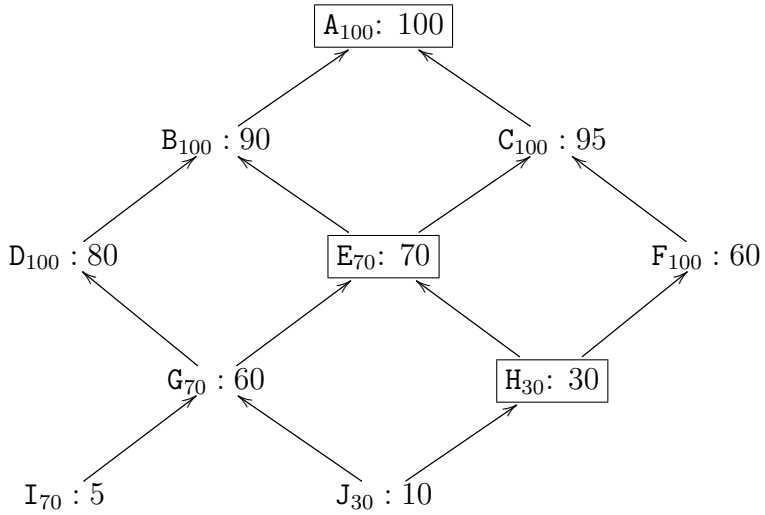


The benefits of materializing an additional view are as follows:

View	benefit
B	$2 \times 10 = 20$
C	$2 \times 5 = 10$
D	$1 \times 20 = 20$
F	$1 \times 40 + 2 \times 10 = 60$
G	$3 \times 10 = 30$
H	$\mathbf{2 \times 40} = 80$
I	$1 \times 65 = 65$
J	$1 \times 60 = 60$

Hence we materialize H, resulting in an additional benefit of 80. The costs of the queries

(in subscript) under the materialized views (in rectangles) now become:



The benefits of materializing a third view becomes:

View	benefit
B	$2 \times 10 = 20$
C	$2 \times 5 = 10$
D	$1 \times 20 = 20$
F	$1 \times 40 = 40$
G	$2 \times 10 = 20$
I	$\mathbf{1 \times 65 = 65}$
J	$1 \times 20 = 20$

Hence we materialize I, resulting in an additional benefit of 65. Only the cost of I itself decreases to 5.

The benefits of materializing a fourth view are now:

View	benefit
B	$2 \times 10 = 20$
C	$2 \times 5 = 10$
D	$1 \times 20 = 20$
F	$\mathbf{1 \times 40 = 40}$
G	$1 \times 10 = 10$
J	$1 \times 20 = 20$

The last materialized view is hence F resulting in an additional benefit of 40.

(b) The total benefit of materializing these 4 views is: 150+80+65+40=335. This is an improvement of 335/1000 = 33,5% over the original cost.

3. (2 points) Are the following statements true or false? Next to your answer, provide a justification in at most 140 characters.

   (a) A junk dimension combines several low-cardinality flags and attributes into a single dimension table rather than modeling them as separate dimensions. The main benefit of a junk dimension is that in this way the size of the fact table is reduced.
   Solution: True; for the combined junk dimensions now only one surrogate key needs to be stored in the fact table. *One could argue that this is not the main benefit, but that the main benefit comes from not having to perform the joins to get the values of the junk dimensions. In very specific situation this may be true; if the junk dimensions have only few values and are not degenerate (hence more than just flags; for instance also an explanation field and maybe a small hierarchy), and many queries specify many or all junk dimensions for slicing and rolling up, it could be the case that avoiding the joins gives some benefit. Notice, however, that the junk dimensions are small and surely fit in memory which would render the joins very cheap. The gain we get by having to use less bytes per fact will however vastly outweigh the benefit of having lesser joins in almost all cases.*

   (b) The lower the cardinality of an attribute, the more useful a bitmap index becomes.
   Solution: False; too low cardinality leads to poor query selectivity resulting in the index not being used.

   (c) For a rapidly-changing dimension where all changes are handled as type-1 changes, it does not make sense to introduce a mini-dimension.
   Solution: True; you would only pay the price of having to do additional joins without having the benefit of space and redundancy reduction.

   (d) View materialization as a query optimization technique for OLAP only makes sense if the measures are distributive.
   Solution: False; also for algebraic measures it makes sense and if the exact view is requested obviously materialization speeds up computations.

4. (2 points) Explain what is run-length encoding and why it is particularly effective in bitmap indices.
   Solution: Especially when the cardinality of an attribute grows, the bitmaps in the bitmap index will contain large streaks of 0's. Under the run-length encoding these streaks will be compressed by storing the length of the streak instead of the sequence of 0's. For instance, in case we chose for a run-length encoding of streaks of 0's only, the following bitmap:

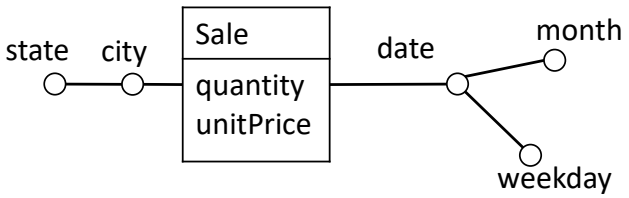
   $$000000000001100000001000001000000000000$$

   could alternative be stored as:

   $$11; 0; 7; 5; 12\ .$$

   This leads to huge space savings resulting in more bitmaps fitting into memory and/or in one disk block, significantly speeding up query processing. Important here is also to remark that the bitwise operations we need to perform on the bitmaps to resolve queries can be carried out directly on the run-length encoded bitmaps.

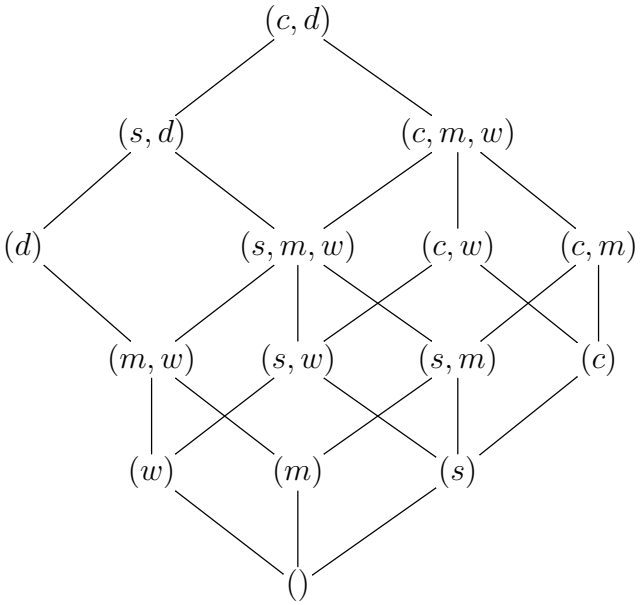
5. (4 points) Consider the following simple DFM



(a) Draw the roll-up lattice for the data cube modelled by this DFM.

(b) We want to completely materialize this data cube. Explain how this computation of the complete data cube could be optimized.

Solution: (a)

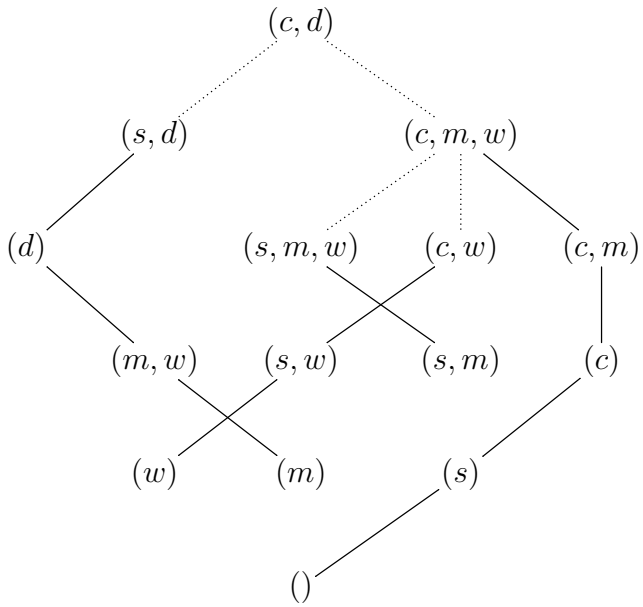


(b) The original table from which we need to compute all aggregations must contain all dimensional attributes which we need. That is, we have attributes (c, s, d, m, w). Given the attributes it is not unreasonable to assume that all views fit together into memory and hence we can use the hash-based aggregation algorithm. We maintain 15 hash-tables in memory, one for each cuboid, and compute the fully materialized data cube in one scan over the original relation.

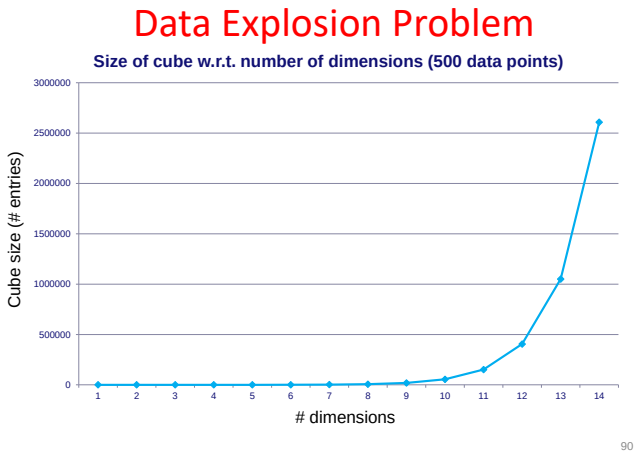
Alternatively, if the cuboids do not fit into memory, we could use the pipe-sorting algorithm. This algorithm is based on sort-based aggregation and organizes the attributes in such order that the output of one aggregation is immediately ordered correctly for computing a next cuboid. The execution can subsequently be pipelined (see slides on pipe-sort). The roll-up lattice in (a) can be decomposed in the following pipes:

- (s,c,m,w,d) → (s,c,m,w) → (s,c,m) → (s,c) → (s) → ()
- (m,w,d,s) → (m,w,d) → (m,w) → (m)
- (w,s,c) → (w,s) → (w)
- (s,m,w) → (s,m)

Hence, with only 4 sorting operations and 4 full table scans we can compute all 15 cuboids. The pipes can be visualized as follows (we maintained the original order of the attributes; dotted line means full table scan and sorting needed):



6. (3 points) When we studied the database explosion problem, the following graph was shown:



This graph illustrates how quickly the size of the fully materialized data cube grows in function of the number of dimensions. Explain the reasons behind the exponential growth in function of the number of dimensions and why it only occurs when the cube is sparse.
Solution: When we fully materialize a data cube, we have to compute all possible aggregations as well. Suppose that we have attributes $A_1,\ldots,A_n$ with active domain size $|dom(A_i)| = d_i$ for all $i = 1 \ldots n$. Then there exist $\Pi_{i=1}^{n} d_i$ possible non-aggregated values in the cube versus $\Pi_{i=1}^{n}(d_i + 1)$ total values (original values plus aggregations). The ratio between these two values is :

$$\frac{\Pi_{i=1}^{n}(d_i + 1)}{\Pi_{i=1}^{n} d_i} = \Pi_{i=1}^{n} \frac{(d_i + 1)}{d_i} \quad .$$

Even though this ratio is exponential in the number of dimensions, its growth rate is very modest (For instance: for domain sizes of 1000 or more the additional size is 2% for 20 dimensions). Hence, in cubes where almost all combinations of the dimensions are present (a dense cube), the growth will be relatively modest.

The story changes completely, however, when cubes are sparse. Even when only a limited number of possible dimension combinations exists, already a significant amount of aggregations may exist. Moreover, the probability that a certain aggregated value is present, increases exponentially with the number of dimensions that are aggregated over. For a mathematical elaboration see the additional material on the explosion problem that is available in the course notes. The graph is depicting this blow-up; in a situation with only few possible dimension combinations present (500 points out of 2^n with n the number of dimensions, the size of the fully materialized cube is plotted for an increasing number of dimensions. Here indeed we see an enormous exponential growth in the size of the data cube, which is called the database explosion problem. The database explosion problem can hence be defined as *the phenomenon of the fast exponential growth of the fully materialized cube size in function of the number of dimensions which manifests itself in sparse data cubes.*