COMP9318 Assignment 1

Due Date: 23:59 11 Sept, 2016 (SUN)

Note

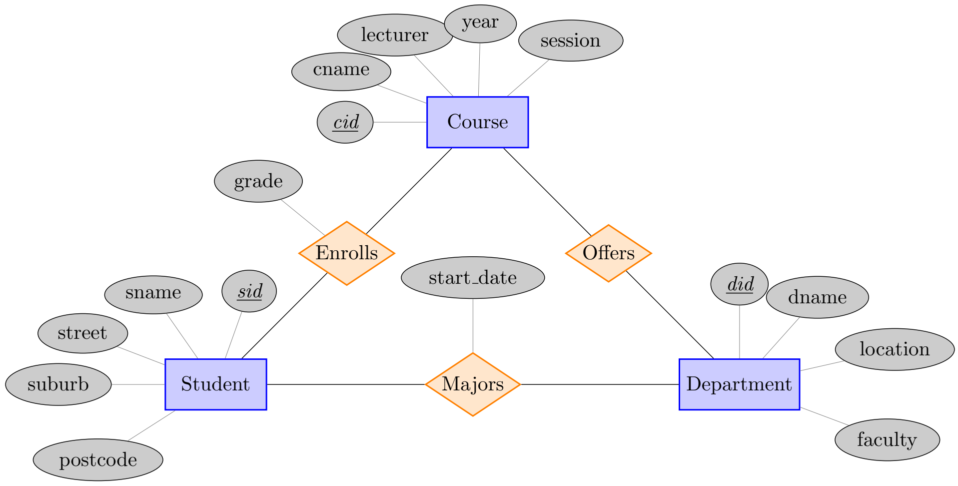
Modified parts are marked in this style.

DESCRIPTION

Q1

(40%)

Consider the following ER diagram and data.[1]



DEPARTMENT

did	*dname*	*location*	*faculty*
CSE	School of Computer Sci. & Eng.	K17	Engineering
EE	School of Electronic Engineering	M17	Engineering

STUDENT

sid	*sname*	*street*	*suburb*	*postcode*
111	John Doe	11 Barker Street	Kingsford	2033
112	John Smith	12 Barker Street	Kingsford	2033

COURSE

cid	*cname*	*lecturer*	*year*	*session*
101	COMP6714	Wei Wang	2015	2
102	COMP6714	Wei Wang	2016	2
103	COMP9318	Wei Wang	2016	2

OFFERS

cid	*did*
101	CSE
102	CSE
103	CSE

ENROLLS

sid	*cid*	*grade*
111	101	71
111	103	83
112	103	86
112	102	69

MAJORS

sid	*did*	*start_date*
111	CSE	1-Jan-2014
112	EE	3-Jan-2014

[1] Note that this is a badly designed ER diagram (think why?). It was made so to keep the complexity of the question low.

Answer the following questions:

1. Construct a star schema for the above database to facilitate a variety of analyses on students. The resulting schema should be able to provide analysis such as "what is the average grade of first year Engineering students", "what are the percentage of Engineering students taking courses from other Faculties". You need to clear specify the fact table and measures.

2. Show the contents of the tables in your star schema.

3. Write an MDX query that shows, for every department and year, the average mark of its student on Engineering courses (You can assume the default aggregate function is `AVERAGE`).

Q2

(40%) Consider the following base cuboid *Sales* with *four* tuples and the aggregate function SUM:

Location	*Time*	*Item*	*Quantity*
Sydney	2005	PS2	1400
Sydney	2006	PS2	1500
Sydney	2006	Wii	500
Melbourne	2005	XBox 360	1700

Location, *Time*, and *Item* are dimensions and *Quantity* is the measure. Suppose the system has built-in support for the value **ALL**.

1. List the tuples in the complete data cube of R in a tabular form with 4 attributes, i.e., $Location, Time, Item, \text{SUM}(Quantity)$?

2. Write down an equivalent SQL statement that computes the same result (i.e., the cube). You can *only* use standard SQL constructs, i.e., no **CUBE BY** clause.

3. Consider the following *ice-berg cube* query:

```
SELECT  Location, Time, Item, SUM(Quantity)
FROM    Sales
CUBE BY Location, Time, Item
HAVING  COUNT(*) > 1
```

   Draw the result of the query in a tabular form.

4. Assume that we adopt a MOLAP architecture to store the full data cube of R, with the following mapping functions:

$$f_{Location}(x) = \begin{cases} 1 & \text{if } x = \text{'Sydney'}, \\ 2 & \text{if } x = \text{'Melbourne'}, \\ 0 & \text{if } x = \textbf{ALL}. \end{cases}$$

$$f_{Time}(x) = \begin{cases} 1 & \text{if } x = 2005, \\ 2 & \text{if } x = 2006, \\ 0 & \text{if } x = \textbf{ALL}. \end{cases}$$

$$f_{Item}(x) = \begin{cases} 1 & \text{if } x = \text{`PS2'}, \\ 2 & \text{if } x = \text{`XBox 360'}, \\ 3 & \text{if } x = \text{`Wii'}, \\ 0 & \text{if } x = \textbf{ALL}. \end{cases}$$

Draw the MOLAP cube (i.e., sparse multi-dimensional array) in a tabular form of $(ArrayIndex, Value)$. You also need to write down the function you chose to map a multi-dimensional point to a one-dimensioinal point.

Q3

(20%)

Consider using multidimensional array to store the fact table. For simplicity, we only consider the case where there are only two dimensions (A and B), and one measure.

As we have seen in Q2, the key idea is to use a mapping *function* g to map the *logical address* (x, y)[2] to a physical address $g(x, y)$ (called *forward mapping*) and use its inverse *function* g^{-1} to map a physical address back to a logical address (which can then be decoded into tuples) (called *backward mapping*).

However, the method requires the cardinalities of all but the first dimension to be fixed. In practice, it is quite possible that a new value will be added to a dimension over the time (e.g., adding a new product or a new store). Given the huge size of the fact table, we cannot afford to reorganize the data upon every such dimension update.

One method to handle such case is to use multiple segments. An example is provided in Figure 1. If we focus on the logical address part:

- At timestamp 1, the 2D array is of size 2×3;
- At timestamp 2, a new value, 3, is added to dimension A, making the array of size 2×4;
- At timestamp 3, a new value, 2, is added to dimension B, making the array of size 3×4;
- At timestamp 4, a new value, 4, is added to dimension A, making the array of size 3×5;
- At timestamp 5, a new value, 5, is added to dimension A, making the array of size 3×6;
- At timestamp 6, a new value, 3, is added to dimension B, making the array of size 4×6;

In terms of physical addresses, every new "chunk" of logical addresses corresponds to a contiguous *segment* in an 1D space. For simplicity, you can assume the segments are contiguous in the physical address space.

Therefore, if we need to set the measure value of point $(4, 3)$ to the magic number 42, we can *somehow* find its physical adress is the 5th slot in the 6th segment, and we can associate this slot with the value 42. Similarly, given any slot in any segment, we should be able to find out its logical address, hence converting it back to a relational tuple.

Obviously, we need to keep additional information such that we can perform the forward and backward mappings in an *efficient* manner in terms of both space and time.

[2] x is the dimension value on Dimension A, and y is the dimension value on Dimension B.

1. Work out what information needs to be kept such that the mapping can be performed in constant time (i.e., the time complexities of your mapping algorithms do not depend on the number of segments).

2. Illustrate the algorithms to perform the following tasks using the given example in Figure 1.

   (a) When a new value 6 is appended to dimension A

   (b) map the logical address $(1, 3)$ to the physical address.

   (c) map the physical address of the last entry in the 4th segment to the logical address.

3. What are the worst time complexities for the above three operations?

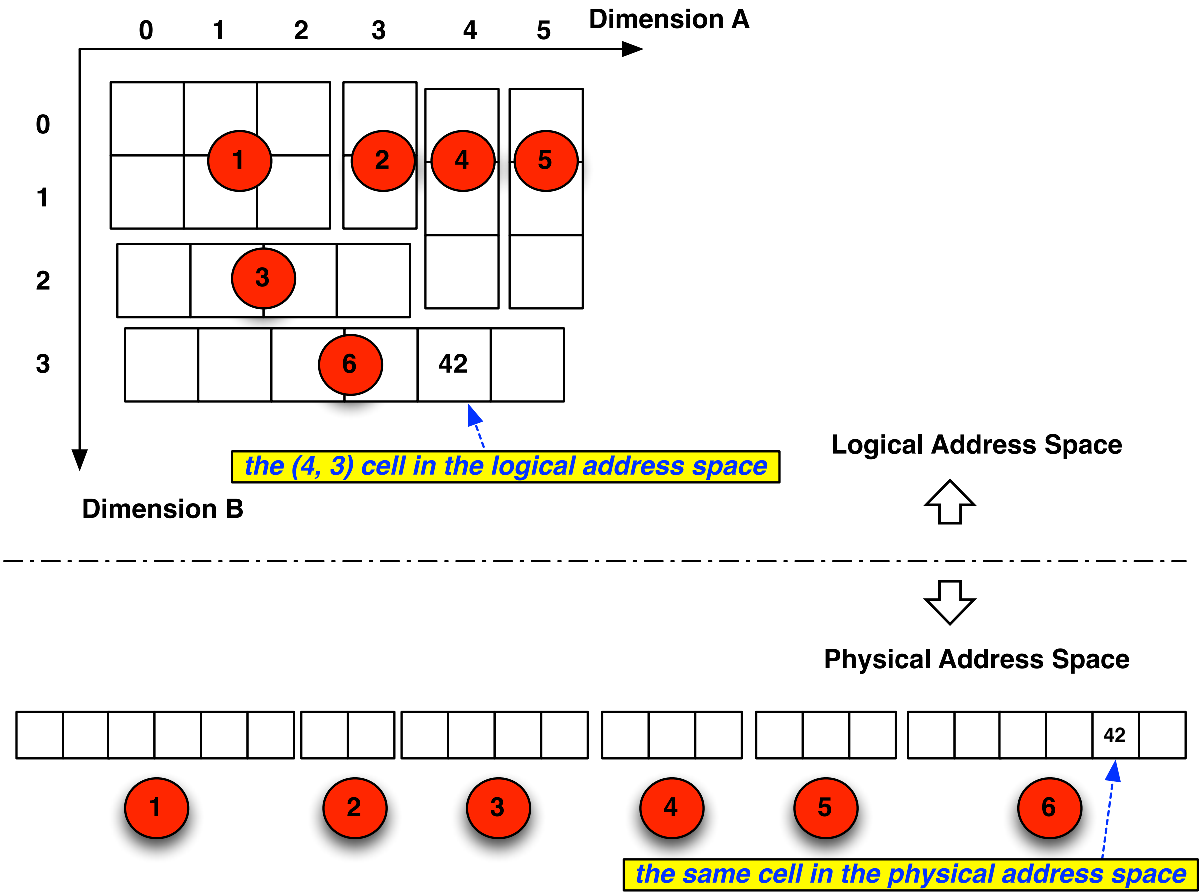


Figure 1: An Example of Dynamic Multidimensional Array. We deliberately leave some space between difference segments for illustration purpose only.

SUBMISSION DETAILS

1. This is an individual assignment.

2. You should submit a PDF document (in electronic form) named `ass1.pdf`.

3. Please submit the document via the **GIVE** system before the deadline. The command is:

```
give cs9318 ass1 ass1.pdf
```

4. Please also make sure all the documents can be opened and **PRINTED** correctly.
5. Late submission policy: **-10% per day** for the first two days, and **-20% per day** afterwards.
6. The size of the `ass1.pdf` file should not exceed 2MB.