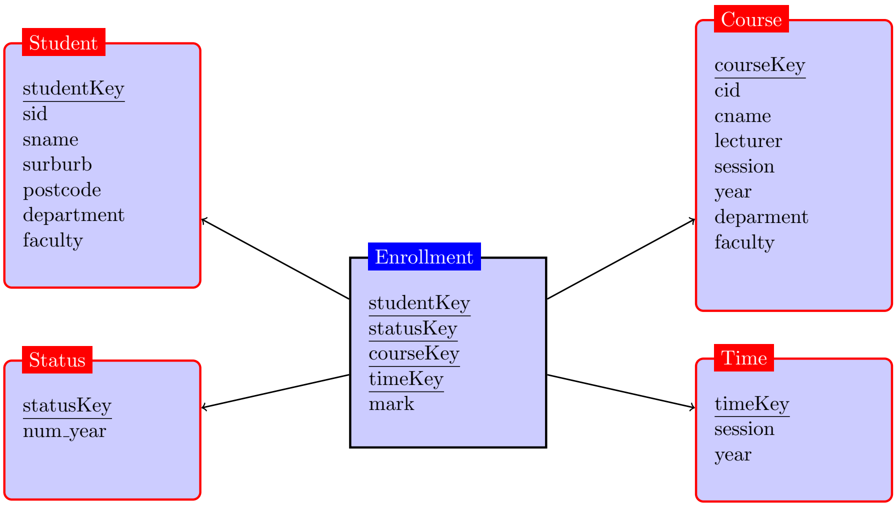
Solution to COMP9318 Assignment 1

Q1

1. See the figure below.



Note:

- In STATUS, it is possible to replace `num_year` with `start_date`. The current design is easy to query but need continuous update.
- The current design models each course offerings in COURSE. If we choose to model each course (by removing SESSION, YEAR, and LECTURER), then we need to have another dimension table named LECTURER.

2. The tables are shown below.

Student

studentKey	*sid*	*sname*	*suburb*	*postcode*	*dept*	*faculty*
1	111	John Doe	Kingsford	2033	CSE	Engineering
2	112	John Smith	Kingsford	2033	EE	Engineering

Status

statusKey	*num_year*
1	3
2	3

Course

courseKey	*cid*	*cname*	*lecturer*	*year*	*session*	*dept*	*faculty*
1	101	COMP6714	Wei Wang	2015	2	CSE	Engineering
2	102	COMP6714	Wei Wang	2016	2	CSE	Engineering
3	103	COMP9318	Wei Wang	2016	2	CSE	Engineering

Time

timeKey	*session*	*year*
1	2	2015
2	2	2016
3	2	2016

Enrollment

studentKey	*statusKey*	*courseKey*	*timeKey*	*mark*
1	1	1	1	71
1	1	3	3	83
2	2	3	3	86
2	2	2	2	69

3. MDX Query

```
SELECT {[Student].[Department].MEMBERS} on COLUMNS
        {[Time].[Year].MEMBERS}
FROM [Enrollment]
WHERE (Measures.[mark], [Course].[Faculty].[Engineering]})
```

Q2

1. See the following table.

cuboid	*Location*	*Time*	*Item*	*SUM(Quantity)*
LTI	Sydney	2005	PS2	1400
LTI	Sydney	2006	PS2	1500
LTI	Sydney	2006	Wii	500
LTI	Melbourne	2005	XBox 360	1700
LT	Sydney	2005	ALL	1400
LT	Sydney	2006	ALL	2000
LT	Melbourne	2005	ALL	1700
LI	Sydney	ALL	PS2	2900
LI	Sydney	ALL	Wii	500
LI	Melbourne	ALL	XBox 360	1700
TI	ALL	2005	PS2	1400
TI	ALL	2006	PS2	1500
TI	ALL	2006	Wii	500
TI	ALL	2005	XBox 360	1700
L	Sydney	ALL	ALL	3400
L	Melbourne	ALL	ALL	1700
T	ALL	2005	ALL	3100
T	ALL	2006	ALL	2000
I	ALL	ALL	PS2	2900
I	ALL	ALL	Wii	500
I	ALL	ALL	XBox 360	1700
	ALL	ALL	ALL	5100

2. See below.

```
SELECT   L, T, I, SUM(M)
FROM     R
GROUP BY L, T, I
UNION ALL
SELECT   L, T, ALL, SUM(M)
FROM     R
GROUP BY L, T
UNION ALL
SELECT   L, ALL, I, SUM(M)
FROM     R
GROUP BY L, I
UNION ALL
SELECT   ALL, T, I, SUM(M)
FROM     R
GROUP BY T, I
UNION ALL
SELECT   L, ALL, ALL, SUM(M)
FROM     R
GROUP BY L
UNION ALL
SELECT   ALL, T, ALL, SUM(M)
FROM     R
GROUP BY T
UNION ALL
SELECT   ALL, ALL, I, SUM(M)
FROM     R
GROUP BY I
UNION ALL
SELECT   ALL, ALL, ALL, SUM(M)
FROM     R
```

3. The iceberg cube is

cuboid	*Location*	*Time*	*Item*	$SUM(Quantity)$
LT	Sydney	2006	ALL	2000
LI	Sydney	ALL	PS2	2900
L	Sydney	ALL	ALL	3400
T	ALL	2005	ALL	3100
T	ALL	2006	ALL	2000
I	ALL	ALL	PS2	2900
	ALL	ALL	ALL	5100

4. The mapping function we choose should satisfy the property that it is a one-to-one function (such that we can always recover the original value even after the mapping). The simplest form is $h(L, T, I) = 12L + 4T + I$. Hence,

$Location$	$Time$	$Item$	$SUM(Quantity)$	$h(L, T, I)$
1	1	1	1400	17
1	2	1	1500	21
1	2	3	500	23
2	1	2	1700	30
1	1	0	1400	16
1	2	0	2000	20
2	1	0	1700	28
1	0	1	2900	13
1	0	3	500	15
2	0	2	1700	26
0	1	1	1400	5
0	2	1	1500	9
0	2	3	500	11
0	1	2	1700	6
1	0	0	3400	12
2	0	0	1700	24
0	1	0	3100	4
0	2	0	2000	8
0	0	1	2900	1
0	0	3	500	3
0	0	2	1700	2
0	0	0	5100	0

So the final result is:

index	*value*
17	1400
21	1500
23	500
30	1700
16	1400
20	2000
28	1700
13	2900
15	500
26	1700
5	1400
9	1500
11	500
6	1700
12	3400
24	1700
4	3100
8	2000
1	2900
3	500
2	1700
0	5100

Q3

1 For each chunk, we need to keep track of:

- the dimension that a new value is added (hence resulting in the current chunk)
- the size of all dimensions after the dimension value is added.

To enable fast mapping, for each value in each dimension, we also keep track of:

- the chunk ID for which this dimension value is associated with.

The following tables show the info kept.

Chunk	**Dimension**	**Dimension Sizes**
1	0	3×2
2	A	4×2
3	B	4×3
4	A	5×3
5	A	6×3
6	B	6×4

Dimension	**Value**	**Chunk**
A	0	1
A	1	1
A	2	1
A	3	2
A	4	4
A	5	5
B	0	1
B	1	1
B	2	3
B	3	6

2 We can then perform the mappings.

- A new chunk, 7, is created, of size 1×4. Two new entries are inserted to the above two tables, which are
  - $(7, A, 7 \times 4)$
  - $(A, 6, 7)$
- Given the logical address $(1, 3)$, we identify its chunk which is the larger chunk ID associated with dimension values. In this case, it is $\max(1, 6) = 6$. The chunk 6 is created when a new value is appended to Dimension B, hence, the offset within the chunk is based on its Dimension A value. Therefore, it is the 2nd entry in the chunk (as the index starts from 0).
- Given the physical address of the last entry (i.e., the 3rd entry) in Chunk 4, we first get Chunk 4's dimension sizes, which is 5×3, and we also know the new value is appended to Dimension A. Therefore, its logical address must have the new value, 4, associated with Dimension A, and its Dimension B value is the third value on Dimension B, which is 2.

3

- $O(1)$ for inserting expanding a dimension.
- $O(1)$ for $g()$, if the 2nd table is organized as a hash table mapping $(Dimension, Value)$ to $Chunk$.
- $O(1)$ for $g^{-1}()$.