ISIT312 Big Data Management

SQL for Data Warehousing

Dr Fenghui Ren

School of Computing and Information Technology -
University of Wollongong

SQL for Data Warehousing

Outline

- SQL/OLAP Operations
- Window partitioning
- Window ordering
- Window framing

SQL/OLAP Operations

Consider the SALES fact table

To compute all possible aggregations along the dimensions Product and Customer we must scan the whole relational table SALES several times

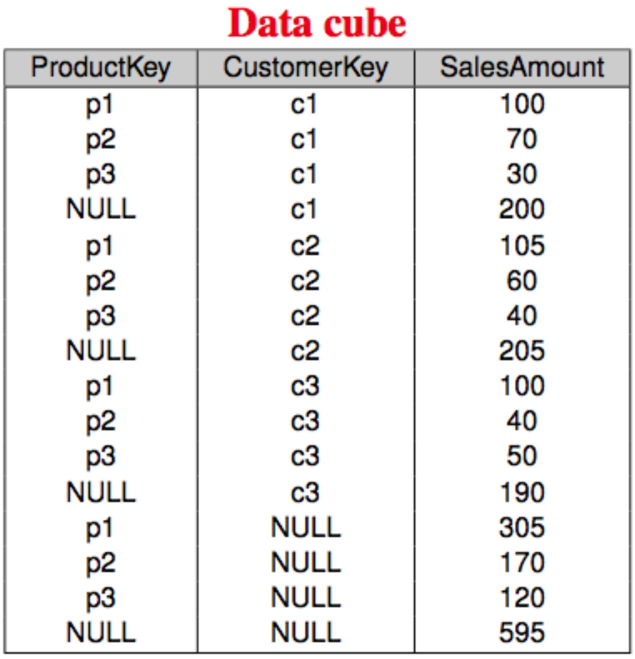
It can be implemented in SQL using NULL and UNION in the following way:

Finding aggregations along many dimensions

```
SELECT ProductKey, CustomerKey, SalesAmount
FROM Sales
  UNION
SELECT ProductKey, NULL, SUM(SalesAmount)
FROM Sales
GROUP BY ProductKey
  UNION
SELECT NULL, CustomerKey, SUM(SalesAmount)
FROM Sales
GROUP BY CustomerKey
  UNION
SELECT NULL, NULL, SUM(SalesAmount)
FROM Sales;
```

SQL/OLAP Operations

A data cube created through UNION of individual SELECT statements each one creating one combination of dimensions looks in the following way



Data cube

ProductKey	CustomerKey	SalesAmount
p1	c1	100
p2	c1	70
p3	c1	30
NULL	c1	200
p1	c2	105
p2	c2	60
p3	c2	40
NULL	c2	205
p1	c3	100
p2	c3	40
p3	c3	50
NULL	c3	190
p1	NULL	305
p2	NULL	170
p3	NULL	120
NULL	NULL	595

SQL/OLAP Operations

Computing a cube with n dimensions requires (2*2*2*... *2)(n times) `SELECT` statements with `GROUP BY` clause

SQL/OLAP extends the `GROUP BY` clause with the `ROLLUP` and `CUBE` operators

`ROLLUP` computes group subtotals in the order given by a list of attributes

`CUBE` computes all totals of such a list

Shorthands for a more powerful operator, `GROUPING SETS`

Equivalent queries

Sample application of ROLLUP operation

```
SELECT ProductKey, CustomerKey, SUM(SalesAmount)
FROM Sales
GROUP BY ROLLUP(ProductKey, CustomerKey);
```

Sample application of GROUPING SET operation

```
SELECT ProductKey, CustomerKey, SUM(SalesAmount)
FROM Sales
GROUP BY GROUPING SETS((ProductKey,CustomerKey),(ProductKey),());
```

SQL/OLAP Operations

Equivalent queries

Sample application of CUBE operation

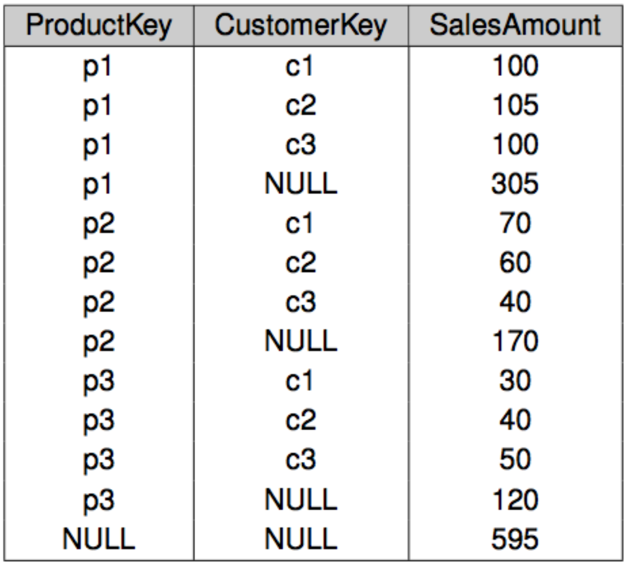
```sql
SELECT ProductKey, CustomerKey, SUM(SalesAmount)
FROM Sales
GROUP BY CUBE(ProductKey, CustomerKey);
```

Sample application of GROUPING SETS operation

```sql
SELECT ProductKey, CustomerKey, SUM(SalesAmount)
FROM Sales
GROUP BY GROUPING SETS((ProductKey, CustomerKey),(ProductKey),(CustomerKey),());
```

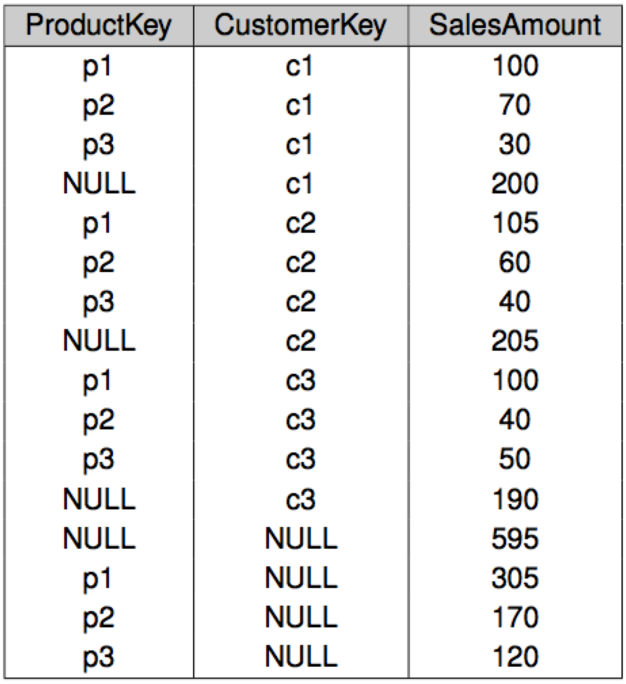
SQL/OLAP Operations

GROUP BY ROLLUP



ProductKey	CustomerKey	SalesAmount
p1	c1	100
p1	c2	105
p1	c3	100
p1	NULL	305
p2	c1	70
p2	c2	60
p2	c3	40
p2	NULL	170
p3	c1	30
p3	c2	40
p3	c3	50
p3	NULL	120
NULL	NULL	595

GROUP BY CUBE



ProductKey	CustomerKey	SalesAmount
p1	c1	100
p2	c1	70
p3	c1	30
NULL	c1	200
p1	c2	105
p2	c2	60
p3	c2	40
NULL	c2	205
p1	c3	100
p2	c3	40
p3	c3	50
NULL	c3	190
NULL	NULL	595
p1	NULL	305
p2	NULL	170
p3	NULL	120

SQL for Data Warehousing

Outline

- SQL/OLAP Operations
- Window partitioning
- Window ordering
- Window framing

Window partitioning

Allows to compare detailed data with aggregate values

For example, find a relevance of each customer with respect to the sales of the product

Sample window partitioning

```
SELECT ProductKey, CustomerKey, SalesAmount,
       MAX(SalesAmount) OVER (PARTITION BY ProductKey) AS MaxAmount
FROM SALES;
```

First three columns are obtained from the `Sales` table

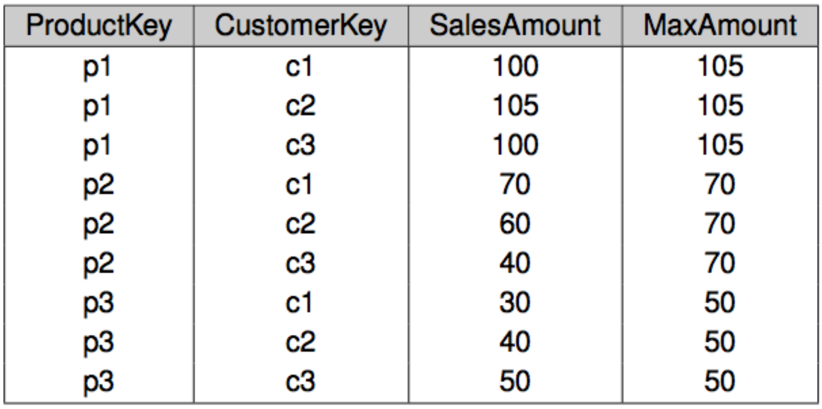
The fourth column is created in the following way

- Create a window called `partition` that contains all tuples of the same product
- `SalesAmount` is aggregated over this window using `MAX` function

Window partitioning

Sample window partitioning

```
SELECT ProductKey, CustomerKey, SalesAmount,
       MAX(SalesAmount) OVER (PARTITION BY ProductKey) AS MaxAmount
FROM SALES;
```



ProductKey	CustomerKey	SalesAmount	MaxAmount
p1	c1	100	105
p1	c2	105	105
p1	c3	100	105
p2	c1	70	70
p2	c2	60	70
p2	c3	40	70
p3	c1	30	50
p3	c2	40	50
p3	c3	50	50

SQL for Data Warehousing

Outline

- SQL/OLAP Operations
- Window partitioning
- Window ordering
- Window framing

Window ordering

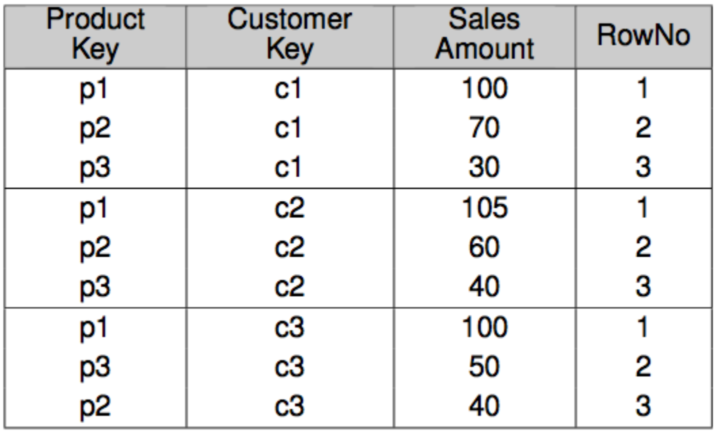
ORDER BY clause allows the rows within a partition to be ordered

It is useful to compute rankings, with a function RANK()

For example, how does each product rank in the sales of each customer

Sample window ordering

```
SELECT ProductKey, CustomerKey, SalesAmount,
       RANK() OVER (PARTITION BY CustomerKey ORDER BY SalesAmount DESC) AS RowNo
FROM Sales;
```



Product Key	Customer Key	Sales Amount	RowNo
p1	c1	100	1
p2	c1	70	2
p3	c1	30	3
p1	c2	105	1
p2	c2	60	2
p3	c2	40	3
p1	c3	100	1
p3	c3	50	2
p2	c3	40	3

SQL for Data Warehousing

Outline

- SQL/OLAP Operations
- Window partitioning
- Window ordering
- Window framing

Window framing

It is possible to define a size of a partition

It can be used to compute statistical functions over time series, like moving average

For example, three-month moving average of sales by product

Sample window framing

```
SELECT ProductKey, Year, Month, SalesAmount,
       AVG(SalesAmount) OVER (PARTITION BY ProductKey
                              ORDER BY Year, Month
                              ROWS 2 PRECEDING) AS MovAvg
FROM SALES;
```

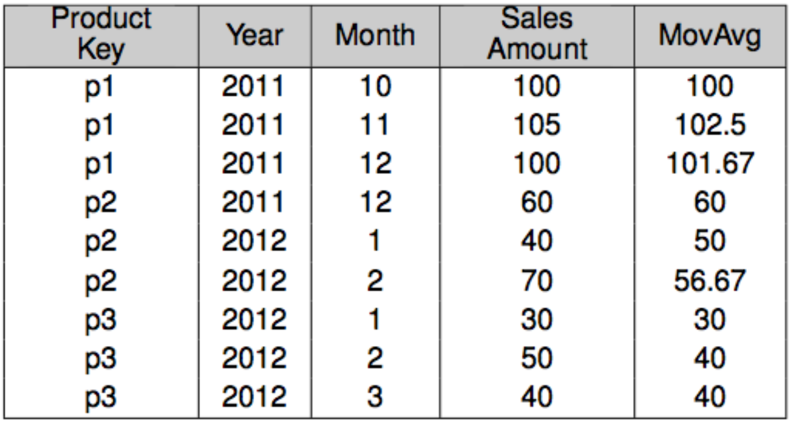
Processing of a query opens a window with the rows pertaining to the current product

Then, it orders the window by year and month and computes the average over the current row and the previous two ones if they exist

Window framing

Sample window framing

```
SELECT ProductKey, Year, Month, SalesAmount,
       AVG(SalesAmount) OVER (PARTITION BY ProductKey
                              ORDER BY Year, Month
                              ROWS 2 PRECEDING) AS MovAvg
FROM SALES;
```



Product Key	Year	Month	Sales Amount	MovAvg
p1	2011	10	100	100
p1	2011	11	105	102.5
p1	2011	12	100	101.67
p2	2011	12	60	60
p2	2012	1	40	50
p2	2012	2	70	56.67
p3	2012	1	30	30
p3	2012	2	50	40
p3	2012	3	40	40

Window framing

Another example, a year-to-date sum of sales by product

Sample window framing

```
SELECT ProductKey, Year, Month, SalesAmount,
       SUM(SalesAmount) OVER (PARTITION BY ProductKey, Year
                              ORDER BY Month
                              ROWS UNBOUNDED PRECEDING) AS YTD
FROM SALES;
```

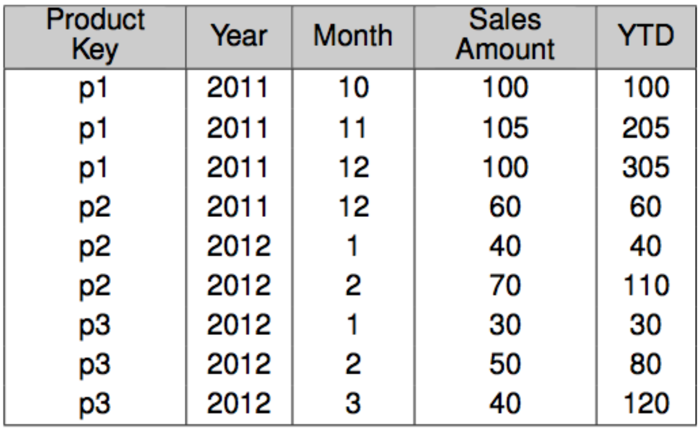
Processing of a query, opens a window with the tuples of the current product and year ordered by month

`AVG()` is applied to all rows before the current row (`ROWS UNBOUNDED PRECEDING`)

Window framing

Sample window framing

```
SELECT ProductKey, Year, Month, SalesAmount,
       SUM(SalesAmount) OVER (PARTITION BY ProductKey, Year
                              ORDER BY Month
                              ROWS UNBOUNDED PRECEDING) AS YTD
FROM SALES;
```



Product Key	Year	Month	Sales Amount	YTD
p1	2011	10	100	100
p1	2011	11	105	205
p1	2011	12	100	305
p2	2011	12	60	60
p2	2012	1	40	40
p2	2012	2	70	110
p3	2012	1	30	30
p3	2012	2	50	80
p3	2012	3	40	120

References

A. VAISMAN, E. ZIMANYI, Data Warehouse Systems: Design and Implementation, Chapter 5 Logical Data Warehouse Design, Springer Verlag, 2014