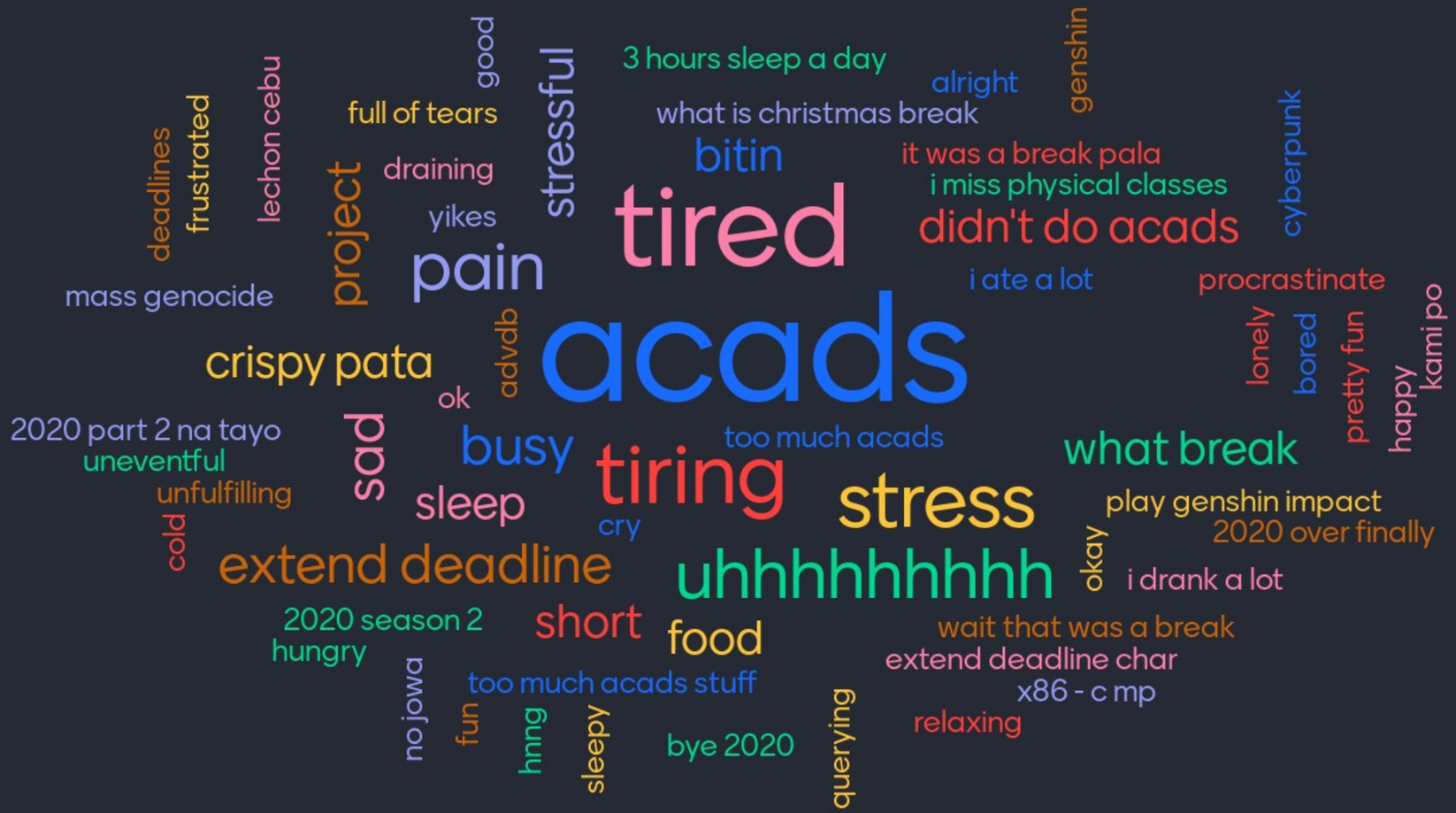
How was your Christmas break?

Mentimeter



55

OLAP Extensions to SQL

Advanced Database Systems

Outline

- >> Moving Data from Operational DB to Data Warehouse (extract, load, transform)
- >> SQL Extensions for OLAP
- >> Tools for Data Warehousing (OLTP, OLAP, Data Mining)

Data Warehouse Schema

- Used to represent data in multiple dimensions
- Defined by dimensions and facts
- Dimensions
  - The entities with respect to which an enterprise preserves the records
- Facts
  - Generated by events that occurred in the past.

http://www.tutorialspoint.com/dwh/dwh_schemas.htm

Motivation

- Limitations of SQL
  - Compute the percentage change in values between this month and a year ago, moving averages, cumulative sums, and other statistical functions
- Solutions
  - ANSI adopted a set of OLAP functions to enable these calculations as well as many others that used to be impossible or even impractical within SQL
  - IBM and Oracle jointly proposed these extensions early in 1999 and they now form part of the current SQL standard, namely SQL: 2011

OLAP Package

- Feature T431, 'Extended Grouping capabilities'
  - Aggregation is a fundamental part of OLAP
  - The SQL standard provides extensions to the GROUP BY clause such as the ROLLUP and CUBE functions
- Feature T611, 'Extended OLAP operators'
  - Ranking functions include cumulative distributions, percent rank, and N-tiles
  - Windowing allows the calculation of cumulative and moving aggregations using functions such as SUM, AVG, MIN, and COUNT

Extended Grouping Capabilities

- ROLLUP supports calculations using aggregations such as SUM, COUNT, MAX, MIN, and AVG at increasing levels of aggregation, from the most detailed up to a grand total
- CUBE enables a single statement to calculate all possible combinations of aggregations to generate the information needed in cross-tabulation reports with a single query
- ROLLUP and CUBE extensions specify exactly the groupings of interest in the GROUP BY clause and produces a single result set that is equivalent to a UNION ALL of differently grouped rows

ROLLUP Extension to GROUP BY

- Enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions
- ROLLUP appears in the GROUP BY clause in a SELECT statement using the following format:

```
SELECT ... GROUP BY ROLLUP(columnList)
```

- Algorithm:
  1. Calculate the aggregate values specified in the GROUP BY clause
  2. Create progressively higher level subtotals, moving from right to left through the column list until finally completing with a grand total

ROLLUP Example

- Show the totals for sales of *flats* or *houses* by branch offices located in Aberdeen, Edinburgh, or Glasgow for the months of August and September of 2013.

```
SELECT  propertyType, yearMonth, city,
        SUM(saleAmount) AS sales
FROM    Branch B, PropertyForSale P4S, PropertySale PS
WHERE   B.branchNo = PS.branchNo AND
        P4S.propertyNo = PS.propertyNo AND
        PS.yearMonth IN ('2013-08', '2013-09') AND
        B.city IN ('Aberdeen', 'Edinburgh', 'Glasgow')
GROUP BY ROLLUP(propertyType, yearMonth, city);
```

ROLLUP Example

ROLLUP creates subtotals at ***n + 1* levels** where *n* = number of grouping columns

propertyType	yearMonth	city	sales
flat	2013-08	Aberdeen	115432
flat	2013-08	Edinburgh	236573
flat	2013-08	Glasgow	7664
flat	2013-08		359669
flat	2013-09	Aberdeen	123780
flat	2013-09	Edinburgh	323100
flat	2013-09	Glasgow	8755
flat	2013-09		455635
flat			815304
house	2013-08	Aberdeen	77987
house	2013-08	Edinburgh	135670
house	2013-08	Glasgow	4765
house	2013-08		218422
house	2013-09	Aberdeen	76321
house	2013-09	Edinburgh	166503
house	2013-09	Glasgow	4889
house	2013-09		247713
house			466135
			1281439

subtotal per City

subtotal per Month

subtotal per propertyType

overall total

Connolly & Begg, 2015

GROUP BY with/out ROLLUP

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2001 | Finland | Phone      |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
+------+---------+------------+-------------+
```

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | India   | NULL       |        1350 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2000 | USA     | NULL       |        1575 |
| 2000 | NULL    | NULL       |        4525 |
| 2001 | Finland | Phone      |          10 |
| 2001 | Finland | NULL       |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
| 2001 | USA     | NULL       |        3000 |
| 2001 | NULL    | NULL       |        3010 |
| NULL | NULL    | NULL       |        7535 |
+------+---------+------------+-------------+
```

http://dev.mysql.com/doc/refman/5.7/en/group-by-modifiers.html

CUBE Extension to GROUP BY

- Takes a specified set of grouping columns and creates subtotals for all of the possible combinations
- CUBE appears in the GROUP BY clause in a SELECT statement using the following format:

```
SELECT ... GROUP BY CUBE(columnList)
```

- CUBE is most suitable in queries that use columns from multiple dimensions rather than columns representing different levels of a single dimension

CUBE Example

- Show all possible subtotals for sales of properties by branches offices in Aberdeen, Edinburgh, and Glasgow for the months of August and September of 2013.

```
SELECT  propertyType, yearMonth, city,
        SUM(saleAmount) AS sales
FROM    Branch B, PropertyForSale P4S, PropertySale PS
WHERE   B.branchNo = PS.branchNo AND
        P4S.propertyNo = PS.propertyNo AND
        PS.yearMonth IN ('2013-08', '2013-09') AND
        B.city IN ('Aberdeen', 'Edinburgh', 'Glasgow')
GROUP BY CUBE(propertyType, yearMonth, city);
```

CUBE Example

propertyType	yearMonth	city	sales
flat	2013-08	Aberdeen	115432
flat	2013-08	Edinburgh	236573
flat	2013-08	Glasgow	7664
flat	2013-08		359669
flat	2013-09	Aberdeen	123780
flat	2013-09	Edinburgh	323100
flat	2013-09	Glasgow	8755
flat	2013-09		455635
flat		Aberdeen	239212
flat		Edinburgh	559673
flat		Glasgow	16419
flat			815304
house	2013-08	Aberdeen	77987
house	2013-08	Edinburgh	135670
house	2013-08	Glasgow	4765
house	2013-08		218422
house	2013-09	Aberdeen	76321
house	2013-09	Edinburgh	166503
house	2013-09	Glasgow	4889
house	2013-09		247713
house		Aberdeen	154308
house		Edinburgh	302173
house		Glasgow	9654
house			466135
	2013-08	Aberdeen	193419
	2013-08	Edinburgh	372243
	2013-08	Glasgow	12429
	2013-08		578091
	2013-09	Aberdeen	200101
	2013-09	Edinburgh	489603
	2013-09	Glasgow	13644
	2013-09		703348
		Aberdeen	393520
		Edinburgh	861846
		Glasgow	26073
			1281439

subtotal per propertyType and city

CUBE creates subtotals that could be calculated for a data cube with the specified dimensions

subtotal per yearMonth per city

subtotal per city

Connolly & Begg, 2015

Elementary OLAP Operators

Ranking functions

- Computes the rank of a record compared to other records in the dataset based on the values of a set of measures
- Syntax for each ranking function

```
RANK( ) OVER (ORDER BY columnList)
DENSE_RANK( ) OVER (ORDER BY columnList)
```

- DENSE_RANK leaves no gaps in the sequential ranking sequence when there are ties for a ranking

Ranking Functions Example

- Rank the total sales of properties for branch offices in Edinburgh.

```
SELECT  branchNo, SUM(saleAmount) AS sales,
        RANK() OVER(ORDER BY SUM(saleAmount)) AS ranking
        DENSE_RANK() OVER(ORDER BY SUM(saleAmount)) AS
                    "dense ranking"
FROM    Branch B, PropertySale PS
WHERE   B.branchNo = PS.branchNo AND
        B.city = 'Edinburgh'
GROUP BY branchNo;
```

branchNo	sales	ranking	dense_ranking
B009	120,000,000	1	1
B018	92,000,000	2	2
B022	92,000,000	2	2
B028	92,000,000	2	2
B033	45,000,000	5	3
B046	42,000,000	6	4

Connolly & Begg, 2015

Elementary OLAP Operators

Windowing calculations

- Can be used to compute cumulative, moving, and centered aggregates
- Returns a value for each row in the table, which depends on other rows in the corresponding window
- Provide access to more than one row of a table without a self-join
- Can be used only in the SELECT and ORDER BY clauses of the query

Ranking Functions Example

- Show the monthly figures and 3-month moving averages and sums for property sales at branch office B003 for the first six months of 2013.

```
SELECT  yearMonth, SUM(saleAmount) AS monthlysales,
    AVG(SUM(saleAmount)
        OVER(ORDER BY yearMonth, ROWS 2 PRECEDING))
                    AS "3-month Moving Average",
    SUM(SUM(saleAmount)
        OVER(ORDER BY yearMonth, ROWS 2 PRECEDING))
                    AS "3-month Moving Sum",
FROM    PropertySale
WHERE   branchNo = "B003" AND
        yearMonth BETWEEN("2013-01" AND "2013-06")
GROUP BY yearMonth;
```

Connolly & Begg, 2015

Ranking Functions Example

yearMonth	monthlySales	3-Month Moving Avg	3-Month Moving Sum
2013-01	210000	210000	210000
2013-02	350000	280000	560000
2013-03	400000	320000	960000
2013-04	420000	390000	1170000
2013-05	440000	420000	1260000
2013-06	430000	430000	1290000

Connolly & Begg, 2015