Lecture 09 and 10

**Two Themes** ...
- Redundancy can be a **GOOD** thing!
- Duplicates, aggregates, and **group by** in SQL, and evolution to “Data Cube”

**.... come together in OLAP**
- **OLTP**: Online Transaction Processing (traditional databases)
  - Data is normalized for the sake of updates.
- **OLAP**: Online Analytic Processing
  - These are (almost) read-only databases.
  - Data is de-normalized for the sake of queries!
  - Multi-dimensional data cube emerging as common data model.
  - This can be seen as a generalization of SQL’s **group by**
Materialized Views

- Suppose \( Q \) is a very expensive, and very frequent query.
- Why not de-normalize some data to speed up the evaluation of \( Q \)?
  - This might be a reasonable thing to do, or ...
  - ... it might be the first step to destroying the integrity of your data design.
- Why not store the value of \( Q \) in a table?
  - This is called a **materialized view**.
  - But now there is a problem: How often should this view be refreshed?

---

**FIDO = Fetch Intensive Data Organization**

```
fast updates

Normalized Database

Extract

fast reads

Read-optimized (NOT Normalized)
```
Example: Embedded databases

![Diagram showing fast updates, normalized database, table-driven applications, read-optimized embedded database, and device.]

Example: Hinxton Bioinformatics

![Diagram from the Hinxton Bioinformatics Institute showing database system design, normalized tables, de-normalized derived tables, and various services and tools.]
Example: Data Warehouse (Decision support)

OLAP vs. OLTP

**OLTP** Online Transaction Processing

**OLAP** Online Analytical Processing

- Commonly associated with terms like Decision Support, Data Warehousing, etc.

<table>
<thead>
<tr>
<th></th>
<th>OLAP</th>
<th>OLTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>analysis</td>
<td>day-to-day operations</td>
</tr>
<tr>
<td>Data is</td>
<td>historical</td>
<td>current</td>
</tr>
<tr>
<td>Transactions mostly</td>
<td>reads</td>
<td>updates</td>
</tr>
<tr>
<td>optimized for Normal Forms</td>
<td>query processing</td>
<td>updates</td>
</tr>
<tr>
<td></td>
<td>not important</td>
<td>important</td>
</tr>
</tbody>
</table>
OLAP Databases: Data Models and Design

The big question

Is the relational model and its associated query language (SQL) well suited for OLAP databases?

- Aggregation (sums, averages, totals, ...) are very common in OLAP queries
  - Problem: SQL aggregation quickly runs out of steam.
  - Solution: Data Cube and associated operations (spreadsheets on steroids)
- Relational design is obsessed with normalization
  - Problem: Need to organize data well since all analysis queries cannot be anticipated in advance.
  - Solution: Multi-dimensional fact tables, with hierarchy in dimensions, star-schema design.

Let's start by looking at aggregate queries in SQL...

An Example...

```sql
mysql> select * from marks;
+----------+----------+-------+
| sid      | course   | mark  |
|----------+----------+-------+
| ev77     | databases| 92    |
| ev77     | spelling | 99    |
| tgg22    | spelling | 3     |
| tgg22    | databases| 100   |
| fm21     | databases| 92    |
| fm21     | spelling | 100   |
| jj25     | databases| 88    |
| jj25     | spelling | 92    |
+----------+----------+-------+
```
... of duplicates

mysql> select mark from marks;
+------+
| mark |
+------+
| 92   |
| 99   |
| 3    |
| 100  |
| 92   |
| 100  |
| 88   |
| 92   |
+------+

Why Multisets?

Duplicates are important for aggregate functions.

mysql> select min(mark),
       max(mark),
       sum(mark),
       avg(mark)
       from marks;
+---------------+---------------+---------------+---------------+
| min(mark)     | max(mark)     | sum(mark)     | avg(mark)     |
+---------------+---------------+---------------+---------------+
| 3             | 100           | 666           | 83.2500      |
+---------------+---------------+---------------+---------------+
The **group by** clause

```sql
mysql> select course, 
                min(mark), 
                max(mark), 
                avg(mark) 
from marks 
group by course;
```

<table>
<thead>
<tr>
<th>course</th>
<th>min(mark)</th>
<th>max(mark)</th>
<th>avg(mark)</th>
</tr>
</thead>
<tbody>
<tr>
<td>databases</td>
<td>88</td>
<td>100</td>
<td>93.0000</td>
</tr>
<tr>
<td>spelling</td>
<td>3</td>
<td>100</td>
<td>73.5000</td>
</tr>
</tbody>
</table>

**Visualizing group by**

<table>
<thead>
<tr>
<th>sid</th>
<th>course</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev77</td>
<td>databases</td>
<td>92</td>
</tr>
<tr>
<td>ev77</td>
<td>spelling</td>
<td>99</td>
</tr>
<tr>
<td>tgg22</td>
<td>spelling</td>
<td>3</td>
</tr>
<tr>
<td>tgg22</td>
<td>databases</td>
<td>100</td>
</tr>
<tr>
<td>fm21</td>
<td>databases</td>
<td>92</td>
</tr>
<tr>
<td>fm21</td>
<td>spelling</td>
<td>100</td>
</tr>
<tr>
<td>jj25</td>
<td>databases</td>
<td>88</td>
</tr>
<tr>
<td>jj25</td>
<td>spelling</td>
<td>92</td>
</tr>
</tbody>
</table>
Visualizing group by

<table>
<thead>
<tr>
<th>course</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>spelling</td>
<td>99</td>
</tr>
<tr>
<td>spelling</td>
<td>3</td>
</tr>
<tr>
<td>spelling</td>
<td>100</td>
</tr>
<tr>
<td>spelling</td>
<td>92</td>
</tr>
</tbody>
</table>

\[ \text{min}(\text{mark}) \]

<table>
<thead>
<tr>
<th>course</th>
<th>min(mark)</th>
</tr>
</thead>
<tbody>
<tr>
<td>spelling</td>
<td>3</td>
</tr>
<tr>
<td>databases</td>
<td>88</td>
</tr>
</tbody>
</table>

The **having** clause

How can we select on the aggregated columns?

```sql
mysql> select course, 
       min(mark),
       max(mark),
       avg(mark)
from marks
group by course
having min(mark) > 60;
```

```
+-----------------+-------+-------+-------------+
| course          | min(mark) | max(mark) | avg(mark)   |
+-----------------+-------+-------+-------------+
| databases       | 88    | 100   | 93.00000    |
+-----------------+-------+-------+-------------+
```
Use renaming to make things nicer ...

```sql
mysql> select course,
       min(mark) as minimum,
       max(mark) as maximum,
       avg(mark) as average
       from marks
       group by course
       having minimum > 60;
```

```
+---------------------------------------------+
| course | minimum | maximum | average |
+---------------------------------------------+
| databases | 88 | 100 | 93.0000 |
+---------------------------------------------+
```

Limits of SQL aggregation

- Flat tables are great for processing, but hard for people to read and understand.
- Pivot tables and cross tabulations (spreadsheet terminology) are very useful for presenting data in ways that people can understand.
- SQL does not handle pivot tables and cross tabulations well.
Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals

JIM GRAY
SUARAJI CHAUDHURI
ADAM BOXWORTH
ANDREW LAYMAN
DON RECHT
MURALI VENKATRAO
Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052

FRANK FELLOW
HAJEM PIRAHESH
IBM Research, 500 Harry Road, San Jose, CA 95120

Cray@Microsoft.com
Sauraj@dMscrosoft.com
AdamB@Microsoft.com
AndrewL@Microsoft.com
Dorale@Microsoft.com
MuraliV@Microsoft.com
Pellow@vost.IBM.com
Pirahesh@Almadex.IBM.com

From aggregates to data cubes
The Data Cube

- Data modeled as an $n$-dimensional (hyper-) cube
- Each dimension is associated with a hierarchy
- Each “point” records facts
- Aggregation and cross-tabulation possible along all dimensions

Hierarchy for **Location** Dimension

- All
  - Region
    - Country
      - City
        - Office
  - Europe
  - North America
    - Canada
      - Canada
    - Mexico
      - L. Chan
      - M. Wind
  - Spain
  - Canada
    - Toronto
      - M. Wind
  - United States
    - New York
      - L. Chan
      - M. Wind
Cube Operations

Example: computing sums

rollup

drill-down

The Star Schema as a design tool