Databases 2011
Lectures 08 — 12

Timothy G. Griffin

Computer Laboratory
University of Cambridge, UK

Databases, Easter 2011
Lectures 08 and 09: Top-Down vs Bottom-up Modeling

Outline
- Weak entities
- Using FDs and MVDs to refine ER models
- Another look at ternary relationships
Recall: a small change of scope ...

... changed this entity

into two entities and a relationship:

But is there something odd about the MovieRelease entity?
MovieRelease represents a **Weak** entity set

**Definition**

- Weak entity sets do not have a primary key.
- The existence of a weak entity depends on an identifying entity set through an **identifying relationship**.
- The primary key of the identifying entity together with the weak entities **discriminators** (dashed underline in diagram) identify each weak entity element.
Can FDs help us think about implementation?

\[
R(I, T, D, C)
\]

\[
I \rightarrow T
\]

\[
I = \text{MovieID} \\
T = \text{Title} \\
D = \text{Date} \\
C = \text{Country}
\]

Turn the decomposition crank to obtain

\[
R_1(I, T) \quad R_2(I, D, C) \\
\pi_I(R_2) \subseteq \pi_I(R_1)
\]
### Scope = UK

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austin Powers: International Man of Mystery</td>
<td>1997</td>
<td>15</td>
</tr>
<tr>
<td>Austin Powers: The Spy Who Shagged Me</td>
<td>1999</td>
<td>12</td>
</tr>
<tr>
<td>Dude, Where’s My Car?</td>
<td>2000</td>
<td>15</td>
</tr>
</tbody>
</table>

### Scope = Earth

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Country</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austin Powers: International Man of Mystery</td>
<td>1997</td>
<td>UK</td>
<td>15</td>
</tr>
<tr>
<td>Austin Powers: International Man of Mystery</td>
<td>1997</td>
<td>Malaysia</td>
<td>18SX</td>
</tr>
<tr>
<td>Austin Powers: International Man of Mystery</td>
<td>1997</td>
<td>Portugal</td>
<td>M/12</td>
</tr>
<tr>
<td>Austin Powers: International Man of Mystery</td>
<td>1997</td>
<td>USA</td>
<td>PG-13</td>
</tr>
<tr>
<td>Austin Powers: The Spy Who Shagged Me</td>
<td>1999</td>
<td>UK</td>
<td>12</td>
</tr>
<tr>
<td>Austin Powers: The Spy Who Shagged Me</td>
<td>1999</td>
<td>Portugal</td>
<td>M/12</td>
</tr>
<tr>
<td>Austin Powers: The Spy Who Shagged Me</td>
<td>1999</td>
<td>USA</td>
<td>PG-13</td>
</tr>
<tr>
<td>Dude, Where’s My Car?</td>
<td>2000</td>
<td>UK</td>
<td>15</td>
</tr>
<tr>
<td>Dude, Where’s My Car?</td>
<td>2000</td>
<td>USA</td>
<td>PG-13</td>
</tr>
<tr>
<td>Dude, Where’s My Car?</td>
<td>2000</td>
<td>Malaysia</td>
<td>18PL</td>
</tr>
</tbody>
</table>
Example of attribute migrating to strong entity set

From single-country scope,

```
MovieID  Title  Year  RatingReason
Movie    Rating
```

to multi-country scope:

```
MovieID  Title  Year  Reason  Country  Rating
Movie    Rated   RatingValue
Rating
```

T. Griffin (cl.cam.ac.uk)  Databases 2011 Lectures 08 — 12
Beware of FFDs = Faux Functional Dependencies

(US ratings)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Rating</th>
<th>RatingReason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stoned</td>
<td>2005</td>
<td>R</td>
<td>drug use</td>
</tr>
<tr>
<td>Wasted</td>
<td>2006</td>
<td>R</td>
<td>drug use</td>
</tr>
<tr>
<td>High Life</td>
<td>2009</td>
<td>R</td>
<td>drug use</td>
</tr>
<tr>
<td>Poppies: Odyssey of an opium eater</td>
<td>2009</td>
<td>R</td>
<td>drug use</td>
</tr>
</tbody>
</table>

But

\[ \text{Title} \rightarrow \{ \text{Rating}, \text{RatingReason} \} \]

is not a functional dependency.

This is a mildly amusing illustration of a real and pervasive problem — deriving a functional dependency after the examination of a limited set of data (or after talking to only a few domain experts).
Oh, but the real world is such a bother!

<table>
<thead>
<tr>
<th>Film Title</th>
<th>Country</th>
<th>Certificate</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Fast 2 Furious (2003)</td>
<td>Switzerland</td>
<td>16</td>
<td>(canton of Zurich)</td>
</tr>
<tr>
<td>28 Days (2000)</td>
<td>Canada</td>
<td>13+</td>
<td>(Quebec)</td>
</tr>
<tr>
<td>28 Days (2000)</td>
<td>Canada</td>
<td>AA</td>
<td>(Ontario)</td>
</tr>
<tr>
<td>28 Days (2000)</td>
<td>Canada</td>
<td>PA</td>
<td>(Manitoba)</td>
</tr>
<tr>
<td>28 Days (2000)</td>
<td>Canada</td>
<td>PG</td>
<td>(British Columbia)</td>
</tr>
</tbody>
</table>
Ternary or multiple binary relationships?

[Diagram of a ternary relationship between entities S, R, T, U, S, E, T, and U with relationships R1, R2, and R3.]
Ternary or multiple binary relationships?

[Diagram showing a ternary relationship with entities S, U, T and binary relationships R1 and R2]
Look again at ER Demo Diagram

How might this be refined using FDs or MVDs?

By Pável Calado,

http://www.texample.net/tikz/examples/entity-relationship-diagram
Lecture 10: Missing data and derived data in SQL

Outline

- NULL in SQL
- three-valued logic
- Multisets and aggregation in SQL
- Views
- General integrity constraints
What is **NULL** in SQL?

What if you don’t know Kim’s age?

```
mysql> select * from students;
+------+--------+------+
| sid  | name   | age  |
+------+--------+------+
| ev77  | Eva    | 18   |
| fm21  | Fatima | 20   |
| jj25  | James  | 19   |
| ks87  | Kim    | NULL |
+------+--------+------+
```
What is **NULL**?

- **NULL** is a place-holder, not a value!
- **NULL** is not a member of any domain (type),
- For records with **NULL** for **age**, an expression like `age > 20` must **unknown**!
- This means we need (at least) three-valued logic.

Let ⊥ represent **We don’t know!**

<table>
<thead>
<tr>
<th>∧</th>
<th>T</th>
<th>F</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>⊥</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>F</td>
<td>⊥</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∨</th>
<th>T</th>
<th>F</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>T</td>
<td>⊥</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∨</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>
NULL can lead to unexpected results

mysql> select * from students;
+----------------+
| sid | name | age |
+----------------+
| ev77 | Eva | 18 |
| fm21 | Fatima | 20 |
| jj25 | James | 19 |
| ks87 | Kim | NULL |
+----------------+

mysql> select * from students where age <> 19;
+----------------+
| sid | name | age |
+----------------+
| ev77 | Eva | 18 |
| fm21 | Fatima | 20 |
+----------------+
The ambiguity of **NULL**

**Possible interpretations of NULL**
- There is a value, but we don’t know what it is.
- No value is applicable.
- The value is known, but you are not allowed to see it.
- ...

A great deal of semantic muddle is created by conflating all of these interpretations into one non-value.

On the other hand, introducing distinct NULLs for each possible interpretation leads to very complex logics ...
Not everyone approves of NULL

C. J. Date [D2004], Chapter 19

“Before we go any further, we should make it very clear that in our opinion (and in that of many other writers too, we hasten to add), NULLs and 3VL are and always were a serious mistake and have no place in the relational model.”
The age column is guaranteed to go out of date! Let’s record dates of birth instead!

```sql
create table Students
    ( sid varchar(10) not NULL,
      name varchar(50) not NULL,
      birth_date date,
      cid varchar(3) not NULL,
      primary key (sid),
      constraint student_college foreign key (cid)
        references Colleges(cid) )
```
age is not a good attribute ...

```sql
mysql> select * from Students;
+------+---------+------------+-----+
| sid  | name    | birth_date | cid |
+------+---------+------------+-----+
| ev77  | Eva     | 1990-01-26 | k   |
| fm21  | Fatima  | 1988-07-20 | cl  |
| jj25  | James   | 1989-03-14 | cl  |
+------+---------+------------+-----+
```
Use a view to recover original table

(Note : the age calculation here is not correct!)

```sql
create view StudentsWithAge as
    select sid, name,
        (year(current_date()) - year(birth_date)) as age,
        cid
    from Students;
```

```sql
mysql> select * from StudentsWithAge;
+----------+---------+------+-----+
<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev77</td>
<td>Eva</td>
<td>19</td>
<td>k</td>
</tr>
<tr>
<td>fm21</td>
<td>Fatima</td>
<td>21</td>
<td>cl</td>
</tr>
<tr>
<td>jj25</td>
<td>James</td>
<td>20</td>
<td>cl</td>
</tr>
</tbody>
</table>
+----------+---------+------|-----+
```

Views are simply identifiers that represent a query. The view’s name can be used as if it were a stored table.
But that calculation is not correct ...

Clearly the calculation of age does not take into account the day and month of year.

From 2010 Database Contest (winner : Sebastian Probst Eide)

```sql
SELECT year(CURRENT_DATE()) - year(birth_date) -
    CASE WHEN month(CURRENT_DATE()) < month(birth_date)
      THEN 1
    ELSE
      CASE WHEN month(CURRENT_DATE()) = month(birth_date)
        THEN
          CASE WHEN day(CURRENT_DATE()) < day(birth_date)
            THEN 1
          ELSE 0
        END
      ELSE 0
    END
END
AS age FROM Students
```
mysql> select * from marks;

<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>
... of duplicates

```sql
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>

`group by`
### Visualizing group by

#### spellings

<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>

#### databases

<table>
<thead>
<tr>
<th>course</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>databases</td>
<td>92</td>
</tr>
<tr>
<td>databases</td>
<td>100</td>
</tr>
<tr>
<td>databases</td>
<td>92</td>
</tr>
<tr>
<td>databases</td>
<td>88</td>
</tr>
</tbody>
</table>

![Diagram](image)

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

<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       | 93.0000   |
+-----------+-----------+-----------+-----------+
```
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;
```

<table>
<thead>
<tr>
<th>course</th>
<th>minimum</th>
<th>maximum</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>databases</td>
<td>88</td>
<td>100</td>
<td>93.0000</td>
</tr>
</tbody>
</table>

T. Griffin (cl.cam.ac.uk)

Databases 2011 Lectures 08 — 12

DB 2011 43 / 58
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?
General integrity constraints

- Suppose that $C$ is some constraint we would like to enforce on our database.
- Let $Q_{\neg C}$ be a query that captures all violations of $C$.
- Enforce (somehow) that the assertion that is always $Q_{\neg C}$ empty.

Example

- $C = Z \rightarrow W$, and FD that was not preserved for relation $R(X)$,
- Let $Q_R$ be a join that reconstructs $R$,
- Let $Q'_R$ be this query with $X \mapsto X'$ and
- $Q_{\neg C} = \sigma_{W \neq W'}(\sigma_{Z = Z'}(Q_R \times Q'_R))$
create view C_violations as ....

create assertion check_C
    check not (exists C_violations)
Lecture 11 and 12: Relational Limitations and Alternatives

Outline

- Limits of SQL aggregation
- OLAP: Online Analytic Processing
- Data cubes
- Star schema
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.
OLAP vs. OLTP

- **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`
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.
Data Cube: A Relational Aggregation Operator
Generalizing Group-By, Cross-Tab, and Sub-Totals*

JIM GRAY
SURAJIT CHAUDHURI
ADAM BOSWORTH
ANDREW LAYMAN
DON REICHART
MURALI VENKATRAO
Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052

FRANK PELLOW
HAMID PIRAHESH
IBM Research, 500 Harry Road, San Jose, CA 95120

Gray@Microsoft.com
SurajitC@Microsoft.com
AdamB@Microsoft.com
AndrewL@Microsoft.com
DonRei@Microsoft.com
MuraliV@Microsoft.com
Pellow@vnet.IBM.com
Pirahesh@Almaden.IBM.com

Data Mining and Knowledge Discovery 1, 29–53 (1997)
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
```

```
   all
  /   \
Europe  \
  /     \
Germany  \
  /     \
   Frankfurt
     /   \
    L. Chan
   /   \
```

```
  /     \
Spain
  /     \
  /     \
```

```
Canada  \
  /     \
  /     \
```

```
North_America
  /     \
  /     \
Mexico
  /     \
```

```
  /     \
Vancouver
  /     \
  /     \
```

```
  /     \
Toronto
  /     \
```

```
  /     \
M. Wind
  /     \
```
Cube Operations

Example: computing sums

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>44</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>p2</td>
<td>12</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>56</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>p2</td>
<td>11</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sum</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>12</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

129

rollup

drill-down
The Star Schema as a design tool

- **time**
  - time_key
  - day
  - day_of_the_week
  - month
  - quarter
  - year

- **branch**
  - branch_key
  - branch_name
  - branch_type

- **item**
  - item_key
  - item_name
  - brand
  - type
  - supplier_type

- **location**
  - location_key
  - street
  - city
  - province_or_street
  - country

- **Sales Fact Table**
  - time_key
  - item_key
  - branch_key
  - location_key
  - units_sold
  - dollars_sold
  - avg_sales

- **Measures**
The End

(Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something? In a way -

Did you really name your son Robert? Drop table Students;-- ?

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.

(http://xkcd.com/327)