Databases

Timothy G. Griffin

Computer Laboratory
University of Cambridge, UK

Databases, Lent 2010
Outline

1. Lecture 01: Basic Concepts
2. Lecture 02: Query languages
3. Lecture 03: More on SQL
4. Lecture 04: Redundancy is a Bad Thing
5. Lecture 05: Analysis of Redundancy
6. Lecture 06: Eliminating Redundancy
7. Lecture 07: Schema Decomposition
8. Lecture 8, 9 and 10: Redundancy is a Good Thing!
Lecture 01 **Basic Concepts.** Relations, attributes, tuples, and relational schema. Tables in SQL.

Lecture 02 **Query languages.** Relational algebra, relational calculi (tuple and domain). Examples of SQL constructs that mix and match these models.

Lecture 03 **More on SQL.** Null values (and three-valued logic). Inner and Outer Joins. Views and integrity constraints.

Lecture 04 **Redundancy is a Bad Thing.** Update anomalies. Capturing redundancy with functional and multivalued dependencies.
Lecture 05 **Analysis of Redundancy.** Implied functional dependencies, logical closure. Reasoning about functional dependencies.


Lecture 07 **Schema Decomposition.** Decomposition examples. Multivalued dependencies and Fourth normal form.
Lectures 08, 09, 10  **Redundancy can be a Good Thing!** Database updates. The main issue: query response vs. update throughput. Locking vs. update throughput. Indices are derived data! Selective de-normalization. Materialized views. The extreme case: “read only” database, data warehousing, data-cubes, and OLAP vs OLTP.

**Lecture 11**  **Entity-Relationship Modeling.** High-level modeling. Entities and relationships. Representation in relational model. Reverse engineering as a common application.

**Lecture 12**  **What is a DBMS?** Different levels of abstraction, data independence. Other data models (Object-Oriented databases, Nested Relations). XML as a universal data exchange language.
## Recommended Reading

### Textbooks

<table>
<thead>
<tr>
<th>Code</th>
<th>Author and Year</th>
<th>Title and Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2004</td>
<td>Date, C.J. (2004).</td>
<td>An introduction to database systems. Addison-Wesley (8th ed.).</td>
</tr>
</tbody>
</table>
Reading for the **fun** of it ...

<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title and Details</th>
</tr>
</thead>
</table>
Lecture 01: Relations and Tables

Lecture Outline

- Relations, attributes, tuples, and relational schema
- Representation in SQL: Tables, columns, rows (records)
- Important: users should be able to create and manipulate relations (tables) without regard to implementation details!
The problem: in 1970 you could not write a database application without knowing a great deal about the low-level physical implementation of the data.

Codd’s radical idea [C1970]: give users a model of data and a language for manipulating that data which is completely independent of the details of its physical representation/implementation.

This decouples development of Database Management Systems (DBMSs) from the development of database applications (at least in a idealized world).
Let’s start with mathematical relations

Suppose that $S_1$ and $S_2$ are sets. The Cartesian product, $S_1 \times S_2$, is the set

$$S_1 \times S_2 = \{(s_1, s_2) \mid s_1 \in S_1, s_2 \in S_2\}$$

A (binary) relation over $S_1 \times S_2$ is any set $r$ with

$$r \subseteq S_1 \times S_2.$$ 

In a similar way, if we have $n$ sets,

$$S_1, S_2, \ldots, S_n,$$

then an $n$-ary relation $r$ is a set

$$r \subseteq S_1 \times S_2 \times \cdots \times S_n = \{(s_1, s_2, \ldots, s_n) \mid s_i \in S_i\}.$$
Did you notice the prestidigitation?

What do we really mean by this notation?

\[ S_1 \times S_2 \times \cdots \times S_n \]

Does it represent \(n - 1\) applications of a binary operator \(\times\)? NO!

If we wanted to be extremely careful we might write something like \(\times(S_1, S_2, \ldots, S_n)\).

We perform this kind of sleight of hand very often. Here’s an example from OCaml:

```ocaml
let flatten_left : (('a * 'b) * 'c) -> ('a * 'b * 'c) = function p ->
    (fst (fst p), snd (fst p), snd p)
```

Perhaps if we had the option of writing \(\ast(\texttt{a}, \texttt{b}, \texttt{c})\) it would make this implicit flattening more obvious.
Suppose we have an \( n \)-tuple \( t \in S_1 \times S_2 \times \cdots \times S_n \). Extracting the \( i \)-th component of \( t \), say as \( \pi_i(t) \), feels a bit low-level.

Solution: (1) Associate a name, \( A_i \) (called an attribute name) with each domain \( S_i \). (2) Instead of tuples, use records — sets of pairs each associating an attribute name \( A_i \) with a value in domain \( S_i \).

A database relation \( R \) over the schema
\[
A_1 : S_1 \times A_2 : S_2 \times \cdots \times A_n : S_n
\]
is a finite set
\[
R \subseteq \{(A_1, s_1), (A_2, s_2), \ldots, (A_n, s_n)\} \mid s_i \in S_i
\]
Example

A relational schema

\textbf{Students}(\texttt{name: string, sid: string, age : integer})

A relational instance of this schema

\texttt{Students} = \{ \\
\{\texttt{name}, Fatima\}, \{\texttt{sid}, fm21\}, \{\texttt{age}, 20\}\}, \\
\{\texttt{name}, Eva\}, \{\texttt{sid}, ev77\}, \{\texttt{age}, 18\}\}, \\
\{\texttt{name}, James\}, \{\texttt{sid}, jj25\}, \{\texttt{age}, 19\}\} \\
\}

A tabular presentation

<table>
<thead>
<tr>
<th>name</th>
<th>sid</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatima</td>
<td>fm21</td>
<td>20</td>
</tr>
<tr>
<td>Eva</td>
<td>ev77</td>
<td>18</td>
</tr>
<tr>
<td>James</td>
<td>jj25</td>
<td>19</td>
</tr>
</tbody>
</table>
Creating Tables in SQL

create table Students
    (sid varchar(10),
     name varchar(50),
     age int);

-- insert record with attribute names
insert into Students set
    name = 'Fatima', age = 20, sid = 'fm21';

-- or insert records with values in same order
-- as in create table
insert into Students values
    ('jj25', 'James', 19),
    ('ev77', 'Eva', 18);

T. Griffin  (cl.cam.ac.uk)
Listing a Table in SQL

-- list by attribute order of create table
mysql> select * from Students;
+--------+--------+------+
| sid    | name   | age  |
+--------+--------+------+
| ev77    | Eva    | 18   |
| fm21    | Fatima | 20   |
| jj25    | James  | 19   |
+--------+--------+------+
3 rows in set (0.00 sec)
Listing a Table in SQL

-- list by specified attribute order
mysql> select name, age, sid from Students;
+--------+------+------+
| name   | age  | sid  |
+--------+------+------+
| Eva    | 18   | ev77 |
| Fatima | 20   | fm21 |
| James  | 19   | jj25 |
+--------+------+------+
3 rows in set (0.00 sec)
Keys in SQL

A **key** is a set of attributes that will uniquely identify any record (row) in a table. We will get more precise in Lecture 06.

```sql
-- with this create table
create table Students
    (sid varchar(10),
     name varchar(50),
     age int,
     primary key (sid));
```

```sql
-- if we try to insert this (fourth) student ...
mysql> insert into Students set
    name = 'Flavia', age = 23, sid = 'fm21';
```

```
ERROR 1062 (23000): Duplicate entry 'fm21' for key 'PRIMARY'
```
Put all information in one big table?

Suppose we want to add information about college membership to our Student database. We could add an additional attribute for the college.

StudentsWithCollege:
+--------+------+------+--------+
| name   | age  | sid  | college|
+--------+------+------+--------+
| Eva    | 18   | ev77 | King’s |
| Fatima | 20   | fm21 | Clare  |
| James  | 19   | jj25 | Clare  |
+--------+------+------+--------+
Put logically independent data in distinct tables?

Students:

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>sid</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eva</td>
<td>18</td>
<td>ev77</td>
<td>k</td>
</tr>
<tr>
<td>Fatima</td>
<td>20</td>
<td>fm21</td>
<td>cl</td>
</tr>
<tr>
<td>James</td>
<td>19</td>
<td>jj25</td>
<td>cl</td>
</tr>
</tbody>
</table>

Colleges:

<table>
<thead>
<tr>
<th>cid</th>
<th>college_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>King’s</td>
</tr>
<tr>
<td>cl</td>
<td>Clare</td>
</tr>
<tr>
<td>sid</td>
<td>Sidney Sussex</td>
</tr>
<tr>
<td>q</td>
<td>Queens’</td>
</tr>
</tbody>
</table>

But how do we put them back together again?
The main themes of these lectures

- We will focus on databases from the perspective of an application writer.
  - We will not be looking at implementation details.
- The main question is this:
  - What criteria can we use to assess the quality of a database application?
- We will see that there is an inherent tradeoff between query response time and (concurrent) update throughput.
- Understanding this tradeoff will involve a careful analysis of the data redundancy implied by a database schema design.
Outline

1. Lecture 01: Basic Concepts
2. Lecture 02: Query languages
3. Lecture 03: More on SQL
4. Lecture 04: Redundancy is a Bad Thing
5. Lecture 05: Analysis of Redundancy
6. Lecture 06: Eliminating Redundancy
7. Lecture 07: Schema Decomposition
8. Lecture 8, 9 and 10: Redundancy is a Good Thing!
Lecture 02: Relational Expressions

Outline

- Database query languages
- The Relational Algebra
- The Relational Calculi (tuple and domain)
- SQL
What is a (relational) database query language?

Input: a collection of relation instances
Output: a single relation instance

\[ R_1, R_2, \ldots, R_k \implies Q(R_1, R_2, \ldots, R_k) \]

How can we express \( Q \)?

In order to meet Codd’s goals we want a query language that is high-level and independent of physical data representation.

There are many possibilities ...
The Relational Algebra (RA)

\[ Q ::= R \quad \text{base relation} \]
\[ \sigma_p(Q) \quad \text{selection} \]
\[ \pi_X(Q) \quad \text{projection} \]
\[ Q \times Q \quad \text{product} \]
\[ Q - Q \quad \text{difference} \]
\[ Q \cup Q \quad \text{union} \]
\[ Q \cap Q \quad \text{intersection} \]
\[ \rho_M(Q) \quad \text{renaming} \]

- \( p \) is a simple boolean predicate over attributes values.
- \( X = \{A_1, A_2, \ldots, A_k\} \) is a set of attributes.
- \( M = \{A_1 \mapsto B_1, A_2 \mapsto B_2, \ldots, A_k \mapsto B_k\} \) is a renaming map.
Relational Calculi

The Tuple Relational Calculus (TRC)

\[ Q = \{ t \mid P(t) \} \]

The Domain Relational Calculus (DRC)

\[ Q = \{ (A_1 = v_1, A_2 = v_2, \ldots, A_k = v_k) \mid P(v_1, v_2, \ldots, v_k) \} \]
The SQL standard

- Origins at IBM in early 1970’s.
- SQL has grown and grown through many rounds of standardization:
  - ANSI: SQL-86
- SQL is made up of many sub-languages:
  - Query Language
  - Data Definition Language
  - System Administration Language
  - ...

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

\[ R \]

\[
\begin{array}{cccc}
A & B & C & D \\
20 & 10 & 0 & 55 \\
11 & 10 & 0 & 7 \\
4  & 99 & 17 & 2 \\
77 & 25 & 4 & 0 \\
\end{array}
\]

\[ Q(R) \]

\[
\begin{array}{cccc}
A & B & C & D \\
20 & 10 & 0 & 55 \\
77 & 25 & 4 & 0 \\
\end{array}
\]

**RA** \( Q = \sigma_{A>12}(R) \)

**TRC** \( Q = \{ t | t \in R \land t.A > 12 \} \)

**DRC** \( Q = \{ \{(A, a), (B, b), (C, c), (D, d)\} | \\
\{(A, a), (B, b), (C, c), (D, d)\} \in R \land a > 12 \} \}

**SQL** select * from R where R.A > 12
Projection

\[ R \]

\[
\begin{array}{cccc}
A & B & C & D \\
20 & 10 & 0 & 55 \\
11 & 10 & 0 & 7 \\
4 & 99 & 17 & 2 \\
77 & 25 & 4 & 0 \\
\end{array}
\]

\[ Q(R) \]

\[
\begin{array}{cc}
B & C \\
10 & 0 \\
99 & 17 \\
25 & 4 \\
\end{array}
\]

\begin{itemize}
\item RA \quad Q = \pi_{B,C}(R)
\item TRC \quad Q = \{ t | \exists u \in R \land t.[B, C] = u.[B, C] \}
\item DRC \quad Q = \{ \{ (B, b), (C, c) \} | \exists\{ (A, a), (B, b), (C, c), (D, d) \} \in R \}
\item SQL \quad \text{select distinct } B, C \text{ from } R
\end{itemize}
Why the \texttt{distinct} in the SQL?

The SQL query

\begin{verbatim}
select B, C from R
\end{verbatim}

will produce a bag (multiset)!

\begin{align*}
R & & Q(R) \\
\begin{array}{cccc}
A & B & C & D \\
20 & 10 & 0 & 55 \\
11 & 10 & 0 & 7 \\
4 & 99 & 17 & 2 \\
77 & 25 & 4 & 0 \\
\end{array} & & \begin{array}{cc}
B & C \\
10 & 0 \\
10 & 0 \\
99 & 17 \\
25 & 4 \\
\end{array}
\end{align*}

SQL is actually based on multisets, not sets. We will look into this more in Lecture 09.
### Renaming

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>17</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>25</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\quad \Rightarrow \quad$

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>E</td>
<td>C</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>17</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>25</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**RA** $Q = \rho_{B\rightarrow E, D\rightarrow F}(R)$

**TRC** $Q = \{t \mid \exists u \in R \land t.A = u.A \land t.E = u.E \land t.C = u.C \land t.F = u.D\}$

**DRC** $Q = \{\{(A, a), (E, b), (C, c), (F, d)\} \mid \exists\{(A, a), (B, b), (C, c), (D, d)\} \in R\}$

**SQL** `select A, B as E, C, D as F from R`
### Note the automatic flattening

**RA** \( Q = R \times S \)

**TRC** \( Q = \{ t \mid \exists u \in R, v \in S, \ t.[A, B] = u.[A, B] \land t.[C, D] = v.[C, D] \} \)

**DRC** \( Q = \{(A, a), (B, b), (C, c), (D, d)\} \mid \\
\{(A, a), (B, b)\} \in R \land \{(C, c), (D, d)\} \in S\} \)

**SQL** `select A, B, C, D from R, S`
Union

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>77</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ Q(R, S) \]

RA \[ Q = R \cup S \]
TRC \[ Q = \{ t \mid t \in R \lor t \in S \} \]
DRC \[ Q = \{(A, a), (B, b)\} \mid (A, a), (B, b) \in R \lor (A, a), (B, b) \in S \} \]
SQL \( (\text{select } * \text{ from } R) \text{ union } (\text{select } * \text{ from } S) \)
### Intersection

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**R**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**S**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

**Q(R)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

**RA**

\[ Q = R \cap S \]

**TRC**

\[ Q = \{ t \mid t \in R \land t \in S \} \]

**DRC**

\[ Q = \{ \{(A, a), (B, b)\} \mid \{(A, a), (B, b)\} \in R \land \{(A, a), (B, b)\} \in S \} \]

**SQL**

\[ (\text{select } * \text{ from } R) \text{ intersect } (\text{select } * \text{ from } S) \]
### Difference

#### \( R \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
</tr>
</tbody>
</table>

#### \( S \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>77</td>
<td>1000</td>
</tr>
</tbody>
</table>

#### \( Q(R) \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
</tr>
</tbody>
</table>

#### Expressions

**RA** \( Q = R - S \)

**TRC** \( Q = \{ t \mid t \in R \land t \notin S \} \)

**DRC** \( Q = \{ \{(A, a), (B, b)\} \mid \{(A, a), (B, b)\} \in R \land \{(A, a), (B, b)\} \notin S \} \)

**SQL** `(select * from R) except (select * from S)`
Query Safety

A query like $Q = \{ t \mid t \in R \land t \not\in S \}$ raises some interesting questions. Should we allow the following query?

$$Q = \{ t \mid t \not\in S \}$$

We want our relations to be finite!

Safety

A (TRC) query

$$Q = \{ t \mid P(t) \}$$

is safe if it is always finite for any database instance.

- Problem: query safety is not decidable!
- Solution: define a restricted syntax that guarantees safety.

Safe queries can be represented in the Relational Algebra.
Given $R(X, Y)$ and $S(Y)$, the division of $R$ by $S$, denoted $R \div S$, is the relation over attributes $X$ defined as (in the TRC)

$$R \div S \equiv \{ x \mid \forall s \in S, \ x \cup s \in R \}.$$
Division in the Relational Algebra?

Clearly, \( R \div S \subseteq \pi_X(R) \). So \( R \div S = \pi_X(R) - C \), where \( C \) represents counter examples to the division condition. That is, in the TRC,

\[
C = \{ x \mid \exists s \in S, \ x \cup s \notin R \}.
\]

- \( U = \pi_X(R) \times S \) represents all possible \( x \cup s \) for \( x \in X(R) \) and \( s \in S \),
- so \( T = U - R \) represents all those \( x \cup s \) that are not in \( R \),
- so \( C = \pi_X(T) \) represents those records \( x \) that are counter examples.

Division in RA

\[
R \div S \equiv \pi_X(R) - \pi_X((\pi_X(R) \times S) - R)
\]
Limitations of simple relational query languages

- The expressive power of RA, TRC, and DRC are essentially the same.
  - None can express the transitive closure of a relation.
- We could extend RA to a more powerful languages (like Datalog).
- SQL has been extended with many features beyond the Relational Algebra.
  - stored procedures
  - recursive queries
  - ability to embed SQL in standard procedural languages
Outline

1. Lecture 01: Basic Concepts
2. Lecture 02: Query languages
3. Lecture 03: More on SQL
4. Lecture 04: Redundancy is a Bad Thing
5. Lecture 05: Analysis of Redundancy
6. Lecture 06: Eliminating Redundancy
7. Lecture 07: Schema Decomposition
8. Lecture 8, 9 and 10: Redundancy is a Good Thing!
Outline

- Joining Tables
- Foreign Keys
- What is `NULL` in SQL?
  - The need for three-valued logic (3VL).
- Views
Product is special!

\[
\begin{array}{|c|c|}
\hline
A & B \\
20 & 10 \\
4 & 99 \\
\hline
\end{array}
\quad \implies \quad
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
20 & 10 & 20 & 10 \\
20 & 10 & 4 & 99 \\
4 & 99 & 20 & 10 \\
4 & 99 & 4 & 99 \\
\hline
\end{array}
\]

- $\times$ is the only operation in the Relational Algebra that created new records (ignoring renaming),
- But $\times$ usually creates too many records!
- Joins are the typical way of using products in a constrained manner.
First, a wee bit of notation

Let $X$ be a set of $k$ attribute names.

- We will often ignore domains (types) and say that $R(X)$ denotes a relational schema.
- When we write $R(Z, Y)$ we mean $R(Z \cup Y)$ and $Z \cap Y = \emptyset$.
- $u.[X] = v.[X]$ abbreviates $u.A_1 = v.A_1 \land \cdots \land u.A_k = v.A_k$.
- $\vec{X}$ represents some (unspecified) ordering of the attribute names, $A_1, A_2, \ldots, A_k$.
- If $\vec{W} = B_1, B_2, \ldots, B_k$, then $X \mapsto W$ abbreviates $A_1 \mapsto B_1, \cdots A_k \mapsto B_k$. 
Equi-join

Given $R(\mathbf{X}, \mathbf{Y})$ and $S(\mathbf{Y}, \mathbf{Z})$, we define the equi-join, denoted $R \bowtie S$, as a relation over attributes $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ defined as

$$R \bowtie S \equiv \{ t \mid \exists u \in R, \ v \in S, \ u.[\mathbf{Y}] = v.[\mathbf{Y}] \land t = u.[\mathbf{X}] \cup u.[\mathbf{Y}] \cup v.[\mathbf{Z}] \}$$

In the Relational Algebra:

$$R \bowtie S = \pi_{\mathbf{X}, \mathbf{Y}, \mathbf{Z}}(\sigma_{\mathbf{Y} = \mathbf{Y}'}(R \times \rho_{\mathbf{Y} \leftarrow \overline{\mathbf{Y}}'}(S)))$$
## Join example

### Students

<table>
<thead>
<tr>
<th>name</th>
<th>sid</th>
<th>age</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatima</td>
<td>fm21</td>
<td>20</td>
<td>cl</td>
</tr>
<tr>
<td>Eva</td>
<td>ev77</td>
<td>18</td>
<td>k</td>
</tr>
<tr>
<td>James</td>
<td>jj25</td>
<td>19</td>
<td>cl</td>
</tr>
</tbody>
</table>

### Colleges

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>King’s</td>
</tr>
<tr>
<td>cl</td>
<td>Clare</td>
</tr>
<tr>
<td>q</td>
<td>Queens’</td>
</tr>
</tbody>
</table>

\[ \pi \text{name, cname}(\text{Students} \bowtie \text{Colleges}) \]

\[
\begin{array}{|c|c|}
\hline
\text{name} & \text{cname} \\
\hline
\text{Fatima} & \text{Clare} \\
\text{Eva}   & \text{King’s} \\
\text{James} & \text{Clare} \\
\hline
\end{array}
\]

\[\implies\]
The same in SQL

```sql
select name, cname
from Students, Colleges
where Students.cid = Colleges.cid
```

<table>
<thead>
<tr>
<th>name</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eva</td>
<td>King's</td>
</tr>
<tr>
<td>Fatima</td>
<td>Clare</td>
</tr>
<tr>
<td>James</td>
<td>Clare</td>
</tr>
</tbody>
</table>

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

Suppose \( R(\mathbf{X}) \) is a relational schema with \( \mathbf{Z} \subseteq \mathbf{X} \). If for any records \( u \) and \( v \) in any instance of \( R \) we have

\[
u.[\mathbf{Z}] = v.[\mathbf{Z}] \implies u.[\mathbf{X}] = v.[\mathbf{X}],\]

then \( \mathbf{Z} \) is a superkey for \( R \). If no proper subset of \( \mathbf{Z} \) is a superkey, then \( \mathbf{Z} \) is a key for \( R \). We write \( R(\mathbf{Z}, \mathbf{Y}) \) to indicate that \( \mathbf{Z} \) is a key for \( R(\mathbf{Z} \cup \mathbf{Y}) \).

Note that this is a semantic assertion, and that a relation can have multiple keys.
Foreign Keys and Referential Integrity

Foreign Key

Suppose we have $R(Z, Y)$. Furthermore, let $S(W)$ be a relational schema with $Z \subseteq W$. We say that $Z$ represents a Foreign Key in $S$ for $R$ if for any instance we have $\pi_Z(S) \subseteq \pi_Z(R)$. This is a semantic assertion.

Referential integrity

A database is said to have referential integrity when all foreign key constraints are satisfied.
create table Colleges
(  cid varchar(3) not NULL,
  cname varchar(50) not NULL,
  primary key (cid) )

create table Students
(  sid varchar(10) not NULL,
  name varchar(50) not NULL,
  age int,
  cid varchar(3) not NULL,
  primary key (sid),
  constraint student_college
    foreign key (cid)
    references Colleges(cid)  )
## An Example:Whatsamatta U

The entities of Whatsamatta U:

### Person

<table>
<thead>
<tr>
<th>name</th>
<th>pid</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatima</td>
<td>fm21</td>
<td><a href="mailto:ft@happy.com">ft@happy.com</a></td>
</tr>
<tr>
<td>Eva</td>
<td>ev77</td>
<td><a href="mailto:eva@funny.com">eva@funny.com</a></td>
</tr>
<tr>
<td>James</td>
<td>jj25</td>
<td><a href="mailto:jj@sad.com">jj@sad.com</a></td>
</tr>
<tr>
<td>Tim</td>
<td>tgg22</td>
<td><a href="mailto:tgg@glad.com">tgg@glad.com</a></td>
</tr>
</tbody>
</table>

### College

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>King’s</td>
</tr>
<tr>
<td>cl</td>
<td>Clare</td>
</tr>
<tr>
<td>q</td>
<td>Queens’</td>
</tr>
</tbody>
</table>

### Course

<table>
<thead>
<tr>
<th>csid</th>
<th>course_name</th>
<th>part</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Algorithms I</td>
<td>IA</td>
</tr>
<tr>
<td>a2</td>
<td>Algorithms II</td>
<td>IB</td>
</tr>
<tr>
<td>db</td>
<td>databases</td>
<td>IB</td>
</tr>
<tr>
<td>ds</td>
<td>Denotational Semantics</td>
<td>II</td>
</tr>
</tbody>
</table>

### Term

<table>
<thead>
<tr>
<th>tid</th>
<th>term_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>lt</td>
<td>Lent</td>
</tr>
<tr>
<td>ms</td>
<td>Michaelmas</td>
</tr>
<tr>
<td>er</td>
<td>Easter</td>
</tr>
</tbody>
</table>
An Example: Whatsamatta U

The relationships (more about this in Lecture 11) of Whatsamatta U:

InCollege

<table>
<thead>
<tr>
<th>pid</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>fm21</td>
<td>cl</td>
</tr>
<tr>
<td>ev77</td>
<td>k</td>
</tr>
<tr>
<td>ev77</td>
<td>q</td>
</tr>
<tr>
<td>jj25</td>
<td>cl</td>
</tr>
<tr>
<td>tgg22</td>
<td>k</td>
</tr>
</tbody>
</table>

Attends

<table>
<thead>
<tr>
<th>pid</th>
<th>csid</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev77</td>
<td>a2</td>
</tr>
<tr>
<td>ev77</td>
<td>db</td>
</tr>
<tr>
<td>jj25</td>
<td>a1</td>
</tr>
</tbody>
</table>

Lectures

<table>
<thead>
<tr>
<th>csid</th>
<th>tid</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>er</td>
</tr>
<tr>
<td>a2</td>
<td>ms</td>
</tr>
<tr>
<td>db</td>
<td>lt</td>
</tr>
<tr>
<td>ds</td>
<td>ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>csid</th>
<th>pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>fm21</td>
</tr>
<tr>
<td>a2</td>
<td>fm21</td>
</tr>
<tr>
<td>a2</td>
<td>tgg22</td>
</tr>
<tr>
<td>db</td>
<td>tgg22</td>
</tr>
</tbody>
</table>
Example query

All records of **name** and **term_name** associated with each lecturer and the terms in which they are lecturing.

\[ \pi_{\text{name}, \text{term\_name}}(\text{Person} \bowtie \text{Lectures} \bowtie \text{Course} \bowtie \text{OfferedIn} \bowtie \text{Term}) \]

<table>
<thead>
<tr>
<th>name</th>
<th>term_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatima</td>
<td>Michaelmas</td>
</tr>
<tr>
<td>Fatima</td>
<td>Easter</td>
</tr>
<tr>
<td>Tim</td>
<td>Lent</td>
</tr>
<tr>
<td>Tim</td>
<td>Michaelmas</td>
</tr>
</tbody>
</table>
What is **NULL** in SQL?

What if you don’t know Kim’s age?

```sql
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 be 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>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>⊥</td>
<td>T</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>T</td>
<td>⊥</td>
</tr>
</tbody>
</table>
**NULL can lead to unexpected results**

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

```sql
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 ...
"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."
**age** is not a good attribute ... 

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;
+-------+-------+---------------------+---+
<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>birth_date</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev77</td>
<td>Eva</td>
<td>1990-01-26</td>
<td>k</td>
</tr>
<tr>
<td>fm21</td>
<td>Fatima</td>
<td>1988-07-20</td>
<td>cl</td>
</tr>
<tr>
<td>jj25</td>
<td>James</td>
<td>1989-03-14</td>
<td>cl</td>
</tr>
</tbody>
</table>
+-------+-------+---------------------+---+
```
Use a view to recover original table

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

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

mysql> select * from StudentsWithAge;
+--------+---------+-----+-----+
| sid    | name    | age | cid |
+--------+---------+-----+-----+
| ev77   | Eva     | 19  | k   |
| fm21   | Fatima  | 21  | cl  |
| jj25   | James   | 20  | cl  |
+--------+---------+-----+-----+

Views are simply identifiers that represent a query. The view’s name

T. Griffin (cl.cam.ac.uk)  Databases  DB 2010
Clearly the calculation of age does not take into account the day and month of year. **Two prizes** will be awarded in lecture for

**SQL Contest**

- the **cleanest** correct solution using **standard SQL** (no vendor-specific hacks),
- the most **obfuscated** (yet still correct) solution
Outline

1. Lecture 01: Basic Concepts
2. Lecture 02: Query languages
3. Lecture 03: More on SQL
4. Lecture 04: Redundancy is a Bad Thing
5. Lecture 05: Analysis of Redundancy
6. Lecture 06: Eliminating Redundancy
7. Lecture 07: Schema Decomposition
8. Lecture 8, 9 and 10: Redundancy is a Good Thing!
Lecture 05: Functional Dependencies

Outline
- Update anomalies
- Functional Dependencies (FDs)
- Normal Forms, 1NF, 2NF, 3NF, and BCNF
Transactions from an application perspective

Main issues

- Avoid update anomalies
- Minimize locking to improve transaction throughput.
- Maintain integrity constraints.

These issues are related.
### Update anomalies

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>college</th>
<th>course</th>
<th>part</th>
<th>term_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>yy88</td>
<td>Yoni</td>
<td>New Hall</td>
<td>Algorithms I</td>
<td>IA</td>
<td>Easter</td>
</tr>
<tr>
<td>uu99</td>
<td>Uri</td>
<td>King’s</td>
<td>Algorithms I</td>
<td>IA</td>
<td>Easter</td>
</tr>
<tr>
<td>bb44</td>
<td>Bin</td>
<td>New Hall</td>
<td>Databases</td>
<td>IB</td>
<td>Lent</td>
</tr>
<tr>
<td>bb44</td>
<td>Bin</td>
<td>New Hall</td>
<td>Algorithms II</td>
<td>IB</td>
<td>Michaelmas</td>
</tr>
<tr>
<td>zz70</td>
<td>Zip</td>
<td>Trinity</td>
<td>Databases</td>
<td>IB</td>
<td>Lent</td>
</tr>
<tr>
<td>zz70</td>
<td>Zip</td>
<td>Trinity</td>
<td>Algorithms II</td>
<td>IB</td>
<td>Michaelmas</td>
</tr>
</tbody>
</table>

- How can we tell if an insert record is consistent with current records?
- Can we record data about a course before students enroll?
- Will we wipe out information about a college when last student associated with the college is deleted?
Redundancy implies more locking ...

... at least for correct transactions!

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>college</th>
<th>course</th>
<th>part</th>
<th>term_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>yy88</td>
<td>Yoni</td>
<td>New Hall</td>
<td>Algorithms I</td>
<td>IA</td>
<td>Easter</td>
</tr>
<tr>
<td>uu99</td>
<td>Uri</td>
<td>King’s</td>
<td>Algorithms I</td>
<td>IA</td>
<td>Easter</td>
</tr>
<tr>
<td>bb44</td>
<td>Bin</td>
<td>New Hall</td>
<td>Databases</td>
<td>IB</td>
<td>Lent</td>
</tr>
<tr>
<td>bb44</td>
<td>Bin</td>
<td>New Hall</td>
<td>Algorithms II</td>
<td>IB</td>
<td>Michaelmas</td>
</tr>
<tr>
<td>zz70</td>
<td>Zip</td>
<td>Trinity</td>
<td>Databases</td>
<td>IB</td>
<td>Lent</td>
</tr>
<tr>
<td>zz70</td>
<td>Zip</td>
<td>Trinity</td>
<td>Algorithms II</td>
<td>IB</td>
<td>Michaelmas</td>
</tr>
</tbody>
</table>

- Change **New Hall** to **Murray Edwards College**
  - Conceptually simple update
  - May require locking entire table.
Redundancy is the root of (almost) all database evils

- It may not be obvious, but redundancy is also the cause of update anomalies.
- By redundancy we do not mean that some values occur many times in the database!
  - A foreign key value may be have millions of copies!
- But then, what do we mean?
Functional Dependency

Let $R(X)$ be a relational schema and $Y \subseteq X$, $Z \subseteq X$ be two attribute sets. We say $Y$ functionally determines $Z$, written $Y \rightarrow Z$, if for any two tuples $u$ and $v$ in an instance of $R(X)$ we have

$$u.Y = v.Y \rightarrow u.Z = v.Z.$$ 

We call $Y \rightarrow Z$ a functional dependency.

A functional dependency is a semantic assertion. It represents a rule that should always hold in any instance of schema $R(X)$. 
### Example FDs

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>college</th>
<th>course</th>
<th>part</th>
<th>term_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>yy88</td>
<td>Yoni</td>
<td>New Hall</td>
<td>Algorithms I</td>
<td>IA</td>
<td>Easter</td>
</tr>
<tr>
<td>uu99</td>
<td>Uri</td>
<td>King’s</td>
<td>Algorithms I</td>
<td>IA</td>
<td>Easter</td>
</tr>
<tr>
<td>bb44</td>
<td>Bin</td>
<td>New Hall</td>
<td>Databases</td>
<td>IB</td>
<td>Lent</td>
</tr>
<tr>
<td>bb44</td>
<td>Bin</td>
<td>New Hall</td>
<td>Algorithms II</td>
<td>IB</td>
<td>Michaelmas</td>
</tr>
<tr>
<td>zz70</td>
<td>Zip</td>
<td>Trinity</td>
<td>Databases</td>
<td>IB</td>
<td>Lent</td>
</tr>
<tr>
<td>zz70</td>
<td>Zip</td>
<td>Trinity</td>
<td>Algorithms II</td>
<td>IB</td>
<td>Michaelmas</td>
</tr>
</tbody>
</table>

- $\text{sid} \rightarrow \text{name}$
- $\text{sid} \rightarrow \text{college}$
- $\text{course} \rightarrow \text{part}$
- $\text{course} \rightarrow \text{term\_name}$
Keys, revisited

Candidate Key

Let $R(X)$ be a relational schema and $Y \subseteq X$. $Y$ is a candidate key if

1. The FD $Y \rightarrow X$ holds, and
2. for no proper subset $Z \subset Y$ does $Z \rightarrow X$ hold.

Prime and Non-prime attributes

An attribute $A$ is prime for $R(X)$ if it is a member of some candidate key for $R$. Otherwise, $A$ is non-prime.

Database redundancy roughly means the existence of non-key functional dependencies!
First Normal Form (1NF)

We will assume every schema is in 1NF.

1NF

A schema $R(A_1 : S_1, A_2 : S_2, \cdots, A_n : S_n)$ is in First Normal Form (1NF) if the domains $S_1$ are elementary — their values are atomic.

<table>
<thead>
<tr>
<th>name</th>
<th>Timothy George Griffin</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_name</td>
<td>middle_name</td>
</tr>
<tr>
<td>Timothy</td>
<td>George</td>
</tr>
</tbody>
</table>
A relational schema $R$ is in 2NF if for every functional dependency $X \rightarrow A$ either
- $A \in X$, or
- $X$ is a superkey for $R$, or
- $A$ is a member of some key, or
- $X$ is not a proper subset of any key.
3NF and BCNF

Third Normal Form (3CNF)
A relational schema $R$ is in 3NF if for every functional dependency $X \rightarrow A$ either
- $A \in X$, or
- $X$ is a superkey for $R$, or
- $A$ is a member of some key.

Boyce-Codd Normal Form (BCNF)
A relational schema $R$ is in BCNF if for every functional dependency $X \rightarrow A$ either
- $A \in X$, or
- $X$ is a superkey for $R$. 
Inclusions

Clearly BCNF $\subseteq$ 3NF $\subseteq$ 2NF. These are proper inclusions:

In 2NF, but not 3NF

$R(A, B, C)$, with $F = \{A \rightarrow B, B \rightarrow C\}$.

In 3NF, but not BCNF

$R(A, B, C)$, with $F = \{A, B \rightarrow C, C \rightarrow B\}$.

- This is in 3NF since $AB$ and $AC$ are keys, so there are no non-prime attributes
- But not in BCNF since $C$ is not a key and we have $C \rightarrow B$. 
The Plan

Given a relational schema $R(X)$ with FDs $F$:

- Reason about FDs
  - Is $F$ missing FDs that are logically implied by those in $F$?
- Decompose each $R(X)$ into smaller $R_1(X_1)$, $R_2(X_2)$, $\cdots$ $R_k(X_k)$, where each $R_i(X_i)$ is in the desired Normal Form.

Are some decompositions better than others?
**Desired properties of any decomposition**

### Lossless-join decomposition

A decomposition of schema $R(X)$ to $S(Y \cup Z)$ and $T(Y \cup (X - Z))$ is a lossless-join decomposition if for every database instances we have $R = S \bowtie T$.

### Dependency preserving decomposition

A decomposition of schema $R(X)$ to $S(Y \cup Z)$ and $T(Y \cup (X - Z))$ is dependency preserving, if enforcing FDs on $S$ and $T$ individually has the same effect as enforcing all FDs on $S \bowtie T$.

We will see that it is not always possible to achieve both of these goals.
Outline

1. Lecture 01 : Basic Concepts
2. Lecture 02 : Query languages
3. Lecture 03 : More on SQL
4. Lecture 04 : Redundancy is a Bad Thing
5. Lecture 05 : Analysis of Redundancy
6. Lecture 06 : Eliminating Redundancy
7. Lecture 07 : Schema Decomposition
8. Lecture 8, 9 and 10 : Redundancy is a Good Thing!
Lecture 06: Reasoning about FDs

Outline

- Implied dependencies (closure)
- Armstrong’s Axioms
Semantic Closure

Notation

\[ F \models Y \rightarrow Z \]

means that any database instance that that satisfies every FD of \( F \),
must also satisfy \( Y \rightarrow Z \).

The semantic closure of \( F \), denoted \( F^+ \), is defined to be

\[ F^+ = \{ Y \rightarrow Z \mid Y \cup Z \subseteq \text{atts}(F) \text{ and } \land F \models Y \rightarrow Z \}. \]

The membership problem is to determine if \( Y \rightarrow Z \in F^+ \).
Reasoning about Functional Dependencies

We write $F \vdash Y \rightarrow Z$ when $Y \rightarrow Z$ can be derived from $F$ via the following rules.

**Armstrong’s Axioms**

- **Reflexivity** If $Z \subseteq Y$, then $F \vdash Y \rightarrow Z$.
- **Augmentation** If $F \vdash Y \rightarrow Z$ then $F \vdash Y, W \rightarrow Z, W$.
- **Transitivity** If $F \vdash Y \rightarrow Z$ and $F \models Z \rightarrow W$, then $F \vdash Y \rightarrow W$. 
Logical Closure (of a set of attributes)

Notation

\[
\text{closure}(F, \ X) = \{ A \mid F \vdash X \rightarrow A \}\]

Claim 1

If \( Y \rightarrow W \in F \) and \( Y \subseteq \text{closure}(F, \ X) \), then \( W \subseteq \text{closure}(F, \ X) \).

Claim 2

\( Y \rightarrow W \in F^+ \) if and only if \( W \subseteq \text{closure}(F, \ Y) \).
Soundness and Completeness

**Soundness**

\[ F \vdash f \implies f \in F^+ \]

**Completeness**

\[ f \in F^+ \implies F \vdash f \]
Proof of Completeness (soundness left as an exercise)

Show \( \neg(F \vdash f) \implies \neg(F \models f) \):

- Suppose \( \neg(F \vdash Y \rightarrow Z) \) for \( R(X) \).
- Let \( Y^+ = \text{closure}(F, Y) \).
- \( \exists B \in Z, \text{ with } B \not\in Y^+ \).
- Construct an instance of \( R \) with just two records, \( u \) and \( v \), that agree on \( Y^+ \) but not on \( X - Y^+ \).
- By construction, this instance does not satisfy \( Y \rightarrow Z \).
- But it does satisfy \( F \)!

\[ \text{Why?} \]

- let \( S \rightarrow T \) be any FD in \( F \), with \( u.[S] = v.[S] \).
- So \( S \subseteq Y^+ \). and so \( T \subseteq Y^+ \) by claim 1,
- and so \( u.[T] = v.[T] \)
Consequences of Armstrong’s Axioms

Union  If $F \models Y \rightarrow Z$ and $F \models Y \rightarrow W$, then $F \models Y \rightarrow W, Z$.

Pseudo-transitivity  If $F \models Y \rightarrow Z$ and $F \models U, Z \rightarrow W$, then $F \models Y, U \rightarrow W$.

Decomposition  If $F \models Y \rightarrow Z$ and $W \subseteq Z$, then $F \models Y \rightarrow W$.

Exercise : Prove these using Armstrong’s axioms!
Proof of the Union Rule

Suppose we have

\[ F \models Y \rightarrow Z, \]
\[ F \models Y \rightarrow W. \]

By augmentation we have

\[ F \models Y, Y \rightarrow Y, Z, \]

that is,

\[ F \models Y \rightarrow Y, Z. \]

Also using augmentation we obtain

\[ F \models Y, Z \rightarrow W, Z. \]

Therefore, by transitivity we obtain

\[ F \models Y \rightarrow W, Z. \]
Example application of functional reasoning.

Heath's Rule

Suppose $R(A, B, C)$ is a relational schema with functional dependency $A \rightarrow B$, then

$$R = \pi_{A,B}(R) \bowtie_A \pi_{A,C}(R).$$
Proof of Heath’s Rule

We first show that $R \subseteq \pi_{A,B}(R) \bowtie_A \pi_{A,C}(R)$.

- If $u = (a, b, c) \in R$, then $u_1 = (a, b) \in \pi_{A,B}(R)$ and $u_2 = (a, c) \in \pi_{A,C}(R)$.
- Since $\{ (a, b) \} \bowtie_A \{ (a, c) \} = \{ (a, b, c) \}$ we know $u \in \pi_{A,B}(R) \bowtie_A \pi_{A,C}(R)$.

In the other direction we must show $R' = \pi_{A,B}(R) \bowtie_A \pi_{A,C}(R) \subseteq R$.

- If $u = (a, b, c) \in R'$, then there must exist tuples $u_1 = (a, b) \in \pi_{A,B}(R)$ and $u_2 = (a, c) \in \pi_{A,C}(R)$.
- This means that there must exist a $u' = (a, b', c) \in R$ such that $u_2 = \pi_{A,C}(\{(a, b', c)\})$.
- However, the functional dependency tells us that $b = b'$, so $u = (a, b, c) \in R$. 

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

$R(A, B, C, D, D, F)$ with

\[
\begin{align*}
A, B & \rightarrow C \\
B, C & \rightarrow D \\
D & \rightarrow E \\
C, F & \rightarrow B
\end{align*}
\]

What is the closure of \{A, B\}? 

\[
\begin{align*}
\{A, B\} & \xrightarrow{A, B \rightarrow C} \{A, B, C\} \\
& \xrightarrow{B, C \rightarrow D} \{A, B, C, D\} \\
& \xrightarrow{D \rightarrow E} \{A, B, C, D, E\}
\end{align*}
\]

So $\{A, B\}^+ = \{A, B, C, D, E\}$ and $A, B \rightarrow C, D, E$. 

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

1. Lecture 01 : Basic Concepts
2. Lecture 02 : Query languages
3. Lecture 03 : More on SQL
4. Lecture 04 : Redundancy is a Bad Thing
5. Lecture 05 : Analysis of Redundancy
6. Lecture 06 : Eliminating Redundancy
7. Lecture 07 : Schema Decomposition
8. Lecture 8, 9 and 10 : Redundancy is a Good Thing!
Lecture 07: Decomposition to Normal Forms

Outline

- Attribute closure algorithm
- Schema decomposition methods
- Problems with obtaining both dependency preservation and lossless-join property
Closure

By soundness and completeness

\[
\text{closure}(F, X) = \{ A \mid F \vdash X \rightarrow A \} = \{ A \mid X \rightarrow A \in F^+ \}
\]

Claim 2 (from previous lecture)

\( Y \rightarrow W \in F^+ \) if and only if \( W \subseteq \text{closure}(F, Y) \).

If we had an algorithm for closure\((F, X)\), then we would have a (brute force!) algorithm for enumerating \( F^+ \):

\[ F^+ \]

- for every subset \( Y \subseteq \text{atts}(F) \)
  - for every subset \( Z \subseteq \text{closure}(F, Y) \),
    - output \( Y \rightarrow Z \)
Attribute Closure Algorithm

- Input: a set of FDs $F$ and a set of attributes $X$.
- Output: $Y = \text{closure}(F, X)$

1. $Y := X$
2. while there is some $S \rightarrow T \in F$ with $S \subseteq Y$ and $T \not\subseteq Y$, then $Y := Y \cup T$. 
An Example (UW1997, Exercise 3.6.1)

\[ R(A, B, C, D) \] with \( F \) made up of the FDs

\[
\begin{align*}
A, B & \rightarrow C \\
C & \rightarrow D \\
D & \rightarrow A
\end{align*}
\]

What is \( F^+ \)?

Brute force!

Let’s just consider all possible nonempty sets \( \mathbf{X} \) — there are only 15...
The only new dependency we get with a single attribute on the left is $C \rightarrow A$. 

For the single attributes we have:

- $\{A\}^+ = \{A\}$,
- $\{B\}^+ = \{B\}$,
- $\{C\}^+ = \{A, C, D\}$,
- $\{D\}^+ = \{A, D\}$.
Example (cont.)

\[ F = \{A, B \rightarrow C, \ C \rightarrow D, \ D \rightarrow A\} \]

Now consider pairs of attributes.

- \{A, B\}^+ = \{A, B, C, D\},
  - so \(A, B \rightarrow D\) is a new dependency
- \{A, C\}^+ = \{A, C, D\},
  - so \(A, C \rightarrow D\) is a new dependency
- \{A, D\}^+ = \{A, D\},
  - so nothing new.
- \{B, C\}^+ = \{A, B, C, D\},
  - so \(B, C \rightarrow A, D\) is a new dependency
- \{B, D\}^+ = \{A, B, C, D\},
  - so \(B, D \rightarrow A, C\) is a new dependency
- \{C, D\}^+ = \{A, C, D\},
  - so \(C, D \rightarrow A\) is a new dependency
Example (cont.)

\[ F = \{ A, B \rightarrow C, \ C \rightarrow D, \ D \rightarrow A \} \]

For the triples of attributes:

- \( \{ A, \ C, \ D \}^+ = \{ A, \ C, \ D \} \),
- \( \{ A, \ B, \ D \}^+ = \{ A, \ B, \ C, \ D \} \),
  - so \( A, B, D \rightarrow C \) is a new dependency
- \( \{ A, \ B, \ C \}^+ = \{ A, \ B, \ C, \ D \} \),
  - so \( A, B, C \rightarrow D \) is a new dependency
- \( \{ B, \ C, \ D \}^+ = \{ A, \ B, \ C, \ D \} \),
  - so \( B, C, D \rightarrow A \) is a new dependency

And since \( \{ A, \ B, \ C, \ D \}^+ = \{ A, \ B, \ C, \ D \} \), we get no new dependencies with four attributes.
Example (cont.)

We generated 11 new FDs:

\[
\begin{align*}
C & \rightarrow A \\
A, C & \rightarrow D \\
B, C & \rightarrow D \\
B, D & \rightarrow C \\
A, B, C & \rightarrow D \\
B, C, D & \rightarrow A \\
A, B & \rightarrow D \\
B, C & \rightarrow A \\
B, D & \rightarrow A \\
C, D & \rightarrow A \\
A, B, D & \rightarrow C
\end{align*}
\]

Can you see the Key?

\{A, B\}, \{B, C\}, and \{B, D\} are keys.

Note: this schema is already in 3NF! Why?
General Decomposition Method (GDM)

GDM

1. Understand your FDs $F$ (compute $F^+$),
2. find $R(X) = R(Z, W, Y)$ (sets $Z$, $W$ and $Y$ are disjoint) with FD $Z \rightarrow W \in F^+$ violating a condition of desired NF,
3. split $R$ into two tables $R_1(Z, W)$ and $R_2(Z, Y)$
4. wash, rinse, repeat

Reminder

For $Z \rightarrow W$, if we assume $Z \cap W = \emptyset$, then the conditions are

1. $Z$ is a superkey for $R$ (2NF, 3NF, BCNF)
2. $W$ is a subset of some key (2NF, 3NF)
3. $Z$ is not a proper subset of any key (2NF)
The lossless-join condition is guaranteed by GDM

- This method will produce a lossless-join decomposition because of (repeated applications of) Heath’s Rule!
- That is, each time we replace an $S$ by $S_1$ and $S_2$, we will always be able to recover $S$ as $S_1 \bowtie S_2$.
- Note that in GDM step 3, the FD $Z \rightarrow W$ may represent a key constraint for $R_1$.

But does the method always terminate? Please think about this ....
Return to Example — Decompose to BCNF

\( R(A, B, C, D) \)

\[ F = \{ A, B \rightarrow C, \; C \rightarrow D, \; D \rightarrow A \} \]

Which FDs in \( F^+ \) violate BCNF?

- \( C \rightarrow A \)
- \( C \rightarrow D \)
- \( D \rightarrow A \)
- \( A, C \rightarrow D \)
- \( C, D \rightarrow A \)
Decompose $R(A, B, C, D)$ to BCNF

Use $C \rightarrow D$ to obtain

- $R_1(C, D)$. This is in BCNF. Done.
- $R_2(A, B, C)$ This is not in BCNF. Why? $A, B$ and $B, C$ are the only keys, and $C \rightarrow A$ is a FD for $R_1$. So use $C \rightarrow A$ to obtain
  - $R_{2.1}(A, C)$. This is in BCNF. Done.
  - $R_{2.2}(B, C)$. This is in BCNF. Done.

Exercise: Try starting with any of the other BCNF violations and see where you end up.
The GDM **does not** always preserve dependencies!

<table>
<thead>
<tr>
<th>$R(A, \ B, \ C, \ D, \ E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A, B \rightarrow C$</td>
</tr>
<tr>
<td>$D, E \rightarrow C$</td>
</tr>
<tr>
<td>$B \rightarrow D$</td>
</tr>
</tbody>
</table>

- $\{A, \ B\}^+ = \{A, \ B, \ C, \ D\}$,
- so $A, B \rightarrow C, D$,
- and $\{A, \ B, \ E\}$ is a key.

- $\{B, \ E\}^+ = \{B, \ C, \ D, \ E\}$,
- so $B, E \rightarrow C, D$,
- and $\{A, \ B, \ E\}$ is a key (again)

Let’s try for a BCNF decomposition ...
Decompose $R(A, B, C, D, E)$ using $A, B \rightarrow C, D$:

- $R_1(A, B, C, D)$. Decompose this using $B \rightarrow D$:
  - $R_{1.1}(B, D)$. Done.
  - $R_{1.2}(A, B, C)$. Done.
- $R_2(A, B, E)$. Done.

But in this decomposition, how will we enforce this dependency?

$$D, E \rightarrow C$$
Decomposition 2

Decompose $R(A, B, C, D, E)$ using $B, E \rightarrow C, D$:

- $R_3(B, C, D, E)$. Decompose this using $D, E \rightarrow C$
  - $R_{3.1}(C, D, E)$. Done.
  - $R_{3.2}(B, D, E)$. Decompose this using $B \rightarrow D$:
    - $R_{3.2.1}(B, D)$. Done.
    - $R_{3.2.2}(B, E)$. Done.

- $R_4(A, B, E)$. Done.

But in this decomposition, how will we enforce this dependency?

$$A, B \rightarrow C$$
Summary

- It always is possible to obtain BCNF that has the lossless-join property (using GDM)
  - But the result may not preserve all dependencies.
- It is always possible to obtain 3NF that preserves dependencies and has the lossless-join property.
  - Using methods based on “minimal covers” (for example, see EN2000).
Outline

1. Lecture 01: Basic Concepts
2. Lecture 02: Query languages
3. Lecture 03: More on SQL
4. Lecture 04: Redundancy is a Bad Thing
5. Lecture 05: Analysis of Redundancy
6. Lecture 06: Eliminating Redundancy
7. Lecture 07: Schema Decomposition
8. Lecture 8, 9 and 10: Redundancy is a Good Thing!
Lecture 08: Multivalued Dependencies

Outline

- Multivalued Dependencies
- Fourth Normal Form (4NF)
- General integrity Constraints
Another look at Heath’s Rule

Given \( R(Z, W, Y) \) with FDs \( F \)

If \( Z \rightarrow W \in F^+ \), then

\[
R = \pi_{Z,W}(R) \Join \pi_{Z,Y}(R)
\]

What about an implication in the other direction? That is, suppose we have

\[
R = \pi_{Z,W}(R) \Join \pi_{Z,Y}(R).
\]

Q Can we conclude anything about FDs on \( R \)? In particular, is it true that \( Z \rightarrow W \) holds?

A No!
We just need one counter example ...

\[ R = \pi_{A,B}(R) \Join \pi_{A,C}(R) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b_1</td>
<td>c_1</td>
</tr>
<tr>
<td>a</td>
<td>b_2</td>
<td>c_2</td>
</tr>
<tr>
<td>a</td>
<td>b_1</td>
<td>c_2</td>
</tr>
<tr>
<td>a</td>
<td>b_2</td>
<td>c_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b_1</td>
</tr>
<tr>
<td>a</td>
<td>b_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c_1</td>
</tr>
<tr>
<td>a</td>
<td>c_2</td>
</tr>
</tbody>
</table>

Clearly \( A \rightarrow B \) is not an FD of \( R \).
### A concrete example

Assuming that texts and lecturers are assigned to courses independently, then a better representation would in two tables:

<table>
<thead>
<tr>
<th>course_name</th>
<th>lecturer</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>Tim</td>
<td>Ullman and Widom</td>
</tr>
<tr>
<td>Databases</td>
<td>Fatima</td>
<td>Date</td>
</tr>
<tr>
<td>Databases</td>
<td>Tim</td>
<td>Date</td>
</tr>
<tr>
<td>Databases</td>
<td>Fatima</td>
<td>Ullman and Widom</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>course_name</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>Tim</td>
</tr>
<tr>
<td>Databases</td>
<td>Fatima</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>course_name</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>Ullman and Widom</td>
</tr>
<tr>
<td>Databases</td>
<td>Date</td>
</tr>
</tbody>
</table>

T. Griffin (cl.cam.ac.uk)
Time for a definition!

**Multivalued Dependencies (MVDs)**

Let $R(Z, W, Y)$ be a relational schema. A multivalued dependency, denoted $Z \rightarrow W$, holds if whenever $t$ and $u$ are two records that agree on the attributes of $Z$, then there must be some tuple $v$ such that

1. $v$ agrees with both $t$ and $u$ on the attributes of $Z$,
2. $v$ agrees with $t$ on the attributes of $W$,
3. $v$ agrees with $u$ on the attributes of $Y$. 

T. Griffin (cl.cam.ac.uk)
A few observations

**Note 1**
Every functional dependency is multivalued dependency,

\[(Z \rightarrow W) \Rightarrow (Z \rightarrow W).\]

To see this, just let \(v = u\) in the above definition.

**Note 2**
Let \(R(Z, W, Y)\) be a relational schema, then

\[(Z \rightarrow W) \iff (Z \rightarrow Y),\]

by symmetry of the definition.
MVDs and lossless-join decompositions

Fun Fun Fact

Let $R(\mathbf{Z}, \mathbf{W}, \mathbf{Y})$ be a relational schema. The decomposition $R_1(\mathbf{Z}, \mathbf{W}), R_2(\mathbf{Z}, \mathbf{Y})$ is a lossless-join decomposition of $R$ if and only if the MVD $\mathbf{Z} \rightarrow \mathbf{W}$ holds.
Proof of Fun Fun Fact

Proof of \((Z \rightarrow W) \implies R = \pi_{Z,W}(R) \Join \pi_{Z,Y}(R)\):

- Suppose \(Z \rightarrow W\).
- We know (from proof of Heath’s rule) that \(R \subseteq \pi_{Z,W}(R) \Join \pi_{Z,Y}(R)\). So we only need to show \(\pi_{Z,W}(R) \Join \pi_{Z,Y}(R) \subseteq R\).
- Suppose \(r \in \pi_{Z,W}(R) \Join \pi_{Z,Y}(R)\).
- So there must be a \(t \in R\) and \(u \in R\) with \(\{r\} = \pi_{Z,W}(\{t\}) \Join \pi_{Z,Y}(\{u\})\).
- In other words, there must be a \(t \in R\) and \(u \in R\) with \(t.Z = u.Z\).
- So the MVD tells us that then there must be some tuple \(v \in R\) such that
  1. \(v\) agrees with both \(t\) and \(u\) on the attributes of \(Z\),
  2. \(v\) agrees with \(t\) on the attributes of \(W\),
  3. \(v\) agrees with \(u\) on the attributes of \(Y\).
- This \(v\) must be the same as \(r\), so \(r \in R\).
Proof of Fun Fun Fact (cont.)

Proof of \( R = \pi_{Z,W}(R) \bowtie \pi_{Z,Y}(R) \implies (Z \rightarrow W) \)

- Suppose \( R = \pi_{Z,W}(R) \bowtie \pi_{Z,Y}(R) \).
- Let \( t \) and \( u \) be any records in \( R \) with \( t.Z = u.Z \).
- Let \( v \) be defined by \( \{v\} = \pi_{Z,W}(\{t\}) \bowtie \pi_{Z,Y}(\{u\}) \) (and we know \( v \in R \) by the assumption).
- Note that by construction we have
  1. \( v.Z = t.Z = u.Z \),
  2. \( v.W = t.W \),
  3. \( v.Y = u.Y \).
- Therefore, \( Z \rightarrow W \) holds.
Fourth Normal Form

Trivial MVD
The MVD $Z \rightarrow W$ is trivial for relational schema $R(Z, W, Y)$ if
1. $Z \cap W \neq \{\}$, or
2. $Y = \{\}$.

4NF
A relational schema $R(Z, W, Y)$ is in 4NF if for every MVD $Z \rightarrow W$ either
- $Z \rightarrow W$ is a trivial MVD, or
- $Z$ is a superkey for $R$.

Note: $4NF \subset BCNF \subset 3NF \subset 2NF$
General Decomposition Method Revisited

**GDM++**

1. Understand your FDs and MVDs $F$ (compute $F^+$),
2. find $R(X) = R(Z, W, Y)$ (sets $Z$, $W$ and $Y$ are disjoint) with either FD $Z \rightarrow W \in F^+$ or MVD $Z \rightarrow W \in F^+$ violating a condition of desired NF,
3. split $R$ into two tables $R_1(Z, W)$ and $R_2(Z, Y)$
4. wash, rinse, repeat

T. Griffin (cl.cam.ac.uk)
We always want the lossless-join property. What are our options?

<table>
<thead>
<tr>
<th></th>
<th>3NF</th>
<th>BCNF</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserves FDs</td>
<td>Yes</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
<tr>
<td>Preserves MVDs</td>
<td>Maybe</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
<tr>
<td>Eliminates FD-redundancy</td>
<td>Maybe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Eliminates MVD-redundancy</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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
- $Q_{\neg C} = \sigma_{w \neq w'}(\sigma_{z = z'}(Q_R \times Q'_R))$
Assertions in SQL

create view C_violations as ....

create assertion check_C
check not (exists C_violations)
Outline

1. Lecture 01: Basic Concepts
2. Lecture 02: Query languages
3. Lecture 03: More on SQL
4. Lecture 04: Redundancy is a Bad Thing
5. Lecture 05: Analysis of Redundancy
6. Lecture 06: Eliminating Redundancy
7. Lecture 07: Schema Decomposition
8. Lecture 8, 9 and 10: Redundancy is a Good Thing!
Lecture 04: Database Updates

Outline

- Transactions
- Short review of ACID requirements
Transactions — ACID properties

Should be review from Concurrent Systems and Applications

**Atomicity** Either all actions are carried out, or none are
- logs needed to undo operations, if needed

**Consistency** If each transaction is consistent, and the database is initially consistent, then it is left consistent
- This is very much a part of applications design.

**Isolation** Transactions are isolated, or protected, from the effects of other scheduled transactions
- Serializability, 2-phase commit protocol

**Durability** If a transactions completes successfully, then its effects persist
- Logging and crash recovery
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**
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

Operational Database

Extract

Data Warehouse

fast updates

business analysis queries
Example: Embedded databases

- fast updates
- table-driven applications

Normalized Database

Read-optimized Embedded Database

Device
Example: Hinxton Bioinformatics

Database system design from the European Bioinformatics Institute (Hinxton UK)

Other archives

Normalized Tables

Development DB

Production DB

Service DB

Service Tools

End Users

De-normalized Derived Tables

Add value (computation)

Add value (review etc.)

Releases & Updates

Distrib

Submitter

Q/C etc

Submission tools

Data exchange

Data design

Other archives
Example: Data Warehouse (Decision support)

- Fast updates
  - Operational Database

- Business analysis queries
  - Extract
  - Data Warehouse
## 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>
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   |
+-------+-----------+------+
```

T. Griffin (cl.cam.ac.uk)
Databases
DB 2010 131 / 145
... 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;
```
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>

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

Databases

DB 2010 135 / 145
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>

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

<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>

\[
\text{course} | \text{mark} \\
\hline
\text{databases} | 92 \\
\text{databases} | 100 \\
\text{databases} | 92 \\
\text{databases} | 88 \\
\]
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 ...

```
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>

```
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
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
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
  /    
  ...  
  Spain
  /    
  ...  
North_America
  /    
Canada
 /      
  ...  
  Mexico
  /    
  ...  
Vancouver
 /      
  ...  
  Toronto
  /    
  ...  
L. Chan
  /    
  ...  
  M. Wind
```
Cube Operations

Example: computing sums

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

- **Measures**
  - units_sold
  - dollars_sold
  - avg_sales

- **Sales Fact Table**
  - time_key
  - item_key
  - branch_key
  - location_key

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