Lecture 01: What is a DBMS?

- DB vs. IR
- Relational Databases
- ACID properties
- Two fundamental trade-offs
- OLTP vs OLAP
- Course outline
A few database examples

- Banking: supporting customer accounts, deposits and withdrawals
- University: students, past and present, marks, academic status
- Business: products, sales, suppliers
- Real Estate: properties, leases, owners, renters
- Aviation: flights, seat reservations, passenger info, prices, payments
- Aviation: Aircraft, maintenance history, parts suppliers, parts orders

Some observations about these DBMSs ...

- They contain highly structured data that has been engineered to model some restricted aspect of the real world
- They support the activity of an organization in an essential way
- They support concurrent access, both read and write
- They often outlive their designers
- Users need to know very little about the DBMS technology used
- Well designed database systems are nearly transparent, just part of our infrastructure
**Always ask What problem am I solving?**

<table>
<thead>
<tr>
<th>DBMS</th>
<th>IR system</th>
</tr>
</thead>
<tbody>
<tr>
<td>exact query results</td>
<td>fuzzy query results</td>
</tr>
<tr>
<td>optimized for concurrent updates</td>
<td>optimized for concurrent reads</td>
</tr>
<tr>
<td>data models a narrow domain</td>
<td>domain often open-ended</td>
</tr>
<tr>
<td>generates documents (reports)</td>
<td>search existing documents</td>
</tr>
<tr>
<td>increase control over information</td>
<td>reduce information overload</td>
</tr>
</tbody>
</table>

And of course there are many systems that combine elements of DB and IR.

---

**Still the dominant approach : Relational DBMSs**

- 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 an idealized world).
What “services” do applications expect from a DBMS?

Transactions — ACID properties

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
- Applications designers must exploit the DBMS's capabilities.

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

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

These concepts should be familiar from Concurrent Systems and Applications.

What constitutes a good DBMS application design?

At the very least, this diagram should commute!
- Does your database design support all required changes?
- Can an update corrupt the database?
Relational Database Design

Our tools

<table>
<thead>
<tr>
<th>Entity-Relationship (ER) modeling</th>
<th>high-level, diagram-based design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational modeling</td>
<td>formal model normal forms based</td>
</tr>
<tr>
<td>SQL implementation</td>
<td>on Functional Dependencies (FDs)</td>
</tr>
<tr>
<td></td>
<td>Where the rubber meets the road</td>
</tr>
</tbody>
</table>

The ER and FD approaches are complementary

- ER facilitates design by allowing communication with *domain experts* who may know little about database technology.
- FD allows us formally explore general design trade-offs. Such as — A *Fundamental Trade-off of Database Design*: the more we reduce data redundancy, the harder it is to enforce some types of data integrity. (An example of this is made precise when we look at 3NF vs. BCNF.)

---

ER Demo Diagram (Notation follows SKS book)¹

---

¹By Pável Calado,
http://www.example.net/likz/examples/entity-relationship-diagram
A Fundamental Trade-off of Database Implementation — Query response vs. update throughput

Redundancy is a Bad Thing.

- One of the main goals of ER and FD modeling is to reduce data redundancy. The seek normalized designs.
- A normalized database can support high update throughput and greatly facilitates the task of ensuring semantic consistency and data integrity.
- Update throughput is increased because in a normalized database a typical transaction need only lock a few data items — perhaps just one field of one row in a very large table.

Redundancy is a Good Thing.

- A de-normalized database most can greatly improve the response time of read-only queries.

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>Supports</th>
<th>OLAP</th>
<th>OLTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data is</td>
<td>analysis</td>
<td>day-to-day operations</td>
</tr>
<tr>
<td>Transactions mostly optimized for</td>
<td>historical</td>
<td>current</td>
</tr>
<tr>
<td>Normal Forms</td>
<td>query processing</td>
<td>updates</td>
</tr>
<tr>
<td></td>
<td>not important</td>
<td>updates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>important</td>
</tr>
</tbody>
</table>
Example: Data Warehouse (Decision support)

![Diagram of fast updates from Operational Database to Data Warehouse for business analysis queries.]

Example: Embedded databases

![Diagram of fast updates from Normalized Database to Read-optimized Embedded Database for table-driven applications.]

FIDO = Fetch Intensive Data Organization
Example: Hinxton Bio-informatics

NoSQL Movement

Technologies
- Key-value store
- Directed Graph Databases
- Main memory stores
- Distributed hash tables

Applications
- Facebook
- Google
- iMDB
- ...

Always remember to ask: What problem am I solving?
Term Outline

Lecture 01  **What is a DBMS?** Course overview. DB vs IR. ACID properties of DBMSs. Schema design. Fundamental trade-offs.

Lecture 02  **Mathematical relations and SQL tables.** Relations, attributes, tuples, and relational schema. Implementing these in SQL.

Lecture 03  **Relational Query Languages.** Relational algebra, relational calculi (tuple and domain). Examples of SQL constructs that mix and match these models.

Lecture 04  **Entity-Relationship (ER) Modeling** Entities, Attributes, and Relationships. Their “implementation” using mathematical relations and integrity constraints. Their implementation using SQL, Foreign Keys, Referential Integrity.

Lecture 05  **More on ER Modeling** N-ary relations.

Lecture 06  **Making the diagram commute.** Update anomalies. Evils of data redundancy. More on integrity constraints.

Lecture 07  **Functional Dependencies (FDs).** Implied functional dependencies, logical closure. Reasoning about functional dependencies.

Term Outline


Lecture 12 OLAP The extreme case: “read only” databases, data warehousing, data-cubes, and OLAP vs OLTP.

Recommended Reading

Textbooks

Chapters 1 (DBMSs)
2 (Entity-Relationship Model)
3 (Relational Model)
4.1 – 4.7 (basic SQL)
6.1 – 6.4 (integrity constraints)
7 (functional dependencies and normal forms)
22 (OLAP)


CJD Date, C.J. (2004). An introduction to database systems. Addison-Wesley (8th ed.).
Reading for the fun of it ...

Research Papers (Google for them)


Lecture 02 : Relations, SQL Tables, Simple Queries

- Mathematical relations and relational schema
- Using SQL to implement a relational schema
- Keys
- Database query languages
- The Relational Algebra
- The Relational Calculi (tuple and domain)
- a bit of SQL
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

$$r \subseteq S_1 \times S_2 \times \cdots \times S_n = \{(s_1, s_2, \ldots, s_n) \mid s_i \in S_i\}$$

Relational Schema

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 $u.A_1 = v.A_1 \land \cdots \land u.A_k = v.A_k$.
- $\bar{X}$ represents some (unspecified) ordering of the attribute names, $A_1, A_2, \ldots, A_k$.
Mathematical vs. database relations

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

$$R \subseteq \{(A_1, s_1), (A_2, s_2), \ldots, (A_n, s_n)\} \quad s_i \in S_i$$

Example

A relational schema

**Students**(name: string, sid: string, age: integer)

A relational instance of this schema

```
Students = 
{(name, Fatima), (sid, fm21), (age, 20)},
{(name, Eva), (sid, ev77), (age, 18)},
{(name, James), (sid, jj25), (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>
Key Concepts

Relational Key

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

$$u[Z] = v[Z] \implies u[X] = v[X],$$

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

Note that this is a semantic assertion, and that a relation can have multiple keys.

Creating Tables in SQL

```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
    ('230', 'James', 19),
    ('ev77', 'Fyo', 18);
```
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.

```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'
```

What is a (relational) database query language?

Input : a collection of relation instances

Output : a single relation instance

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

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)

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

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

Relational Calculi

The Tuple Relational Calculus (TRC)

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

The Domain Relational Calculus (DRC)

\[ Q = \{ (A_1 - v_1, A_2 - v_2, \ldots, A_k - v_k) | P(v_1, v_2, \ldots, v_k) \} \]
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
  - ...

---

Selection

\[
R \quad \quad \quad Q(R)
\]

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

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

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

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

SQL: \( \text{select * from } R \text{ where } R.A > 12 \)
Projection

\[ R \]

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

RA \quad Q = π_{B,C}(R)

TRC \quad Q = \{t | \exists u \in R \land t[B,C] = u[B,C]\}

DRC \quad Q = \{(B, b), (C, c) | \exists \{(A, a), (B, b), (C, c), (D, d)\} \in R\}

SQL \quad \text{select distinct } B, C \text{ from } R

---

Why the \textit{distinct} in the SQL?

The SQL query

\[
\text{select } B, C \text{ from } R
\]

will produce a bag (multiset)!

\[ R \]

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

SQL is actually based on multisets, not sets. We will look into this more in Lecture 11.
Lecture 03: More on Relational Query languages

Outline

- Constructing new tuples!
- Joins
- Limitations of Relational Algebra

Renaming

\[ R \rightarrow Q(R) \]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0</td>
<td>55</td>
<td>→</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>17</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>25</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>E</td>
<td>C</td>
<td>F</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0</td>
<td>55</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>77</td>
<td>25</td>
<td>4</td>
<td>0</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 \[ Q = \{ \langle (A, a), (E, b), (C, c), (F, d) \rangle \mid \{(A, a), (B, b), (C, c), (D, d)\} \in R \} \]

SQL \[ \text{select } A, 3 \text{ as } E, C, D \text{ as } F \text{ from } R \]
**Union**

\[
\begin{array}{|c|c|} \hline R & S \hline \end{array}
\begin{array}{|c|c|} \hline A & B \hline 20 & 10 \hline 11 & 10 \hline 4 & 99 \hline \end{array} \begin{array}{|c|c|} \hline A & B \hline 20 & 10 \hline 77 & 1000 \hline \end{array} \Rightarrow \begin{array}{|c|c|} \hline A & B \hline 20 & 10 \hline 11 & 10 \hline 4 & 99 \hline 77 & 1000 \hline \end{array}
\]

RA \quad Q = R \cup S

TRC \quad Q = \{ t \mid t \in R \lor t \in S \}

DRC \quad Q = \{ ((A, a), (B, b)) \mid ((A, a), (B, b)) \in R \lor ((A, a), (B, b)) \in S \}

SQL \quad (\text{select * from } R) \text{ union (select * from } S)

**Intersection**

\[
\begin{array}{|c|c|} \hline R & S \hline \end{array}
\begin{array}{|c|c|} \hline A & B \hline 20 & 10 \hline 11 & 10 \hline 4 & 99 \hline \end{array} \begin{array}{|c|c|} \hline A & B \hline 20 & 10 \hline 77 & 1000 \hline \end{array} \Rightarrow \begin{array}{|c|c|} \hline A & B \hline 20 & 10 \hline \end{array}
\]

RA \quad Q = R \cap S

TRC \quad Q = \{ t \mid t \in R \land t \in S \}

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

SQL \quad (\text{select * from } R) \text{ intersect (select * from } S)

T. Griffin (dl.cam.ac.uk) | Databases 2011 Lectures 01-03 | DB 2011 52/70
Wait, are we missing something?

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| Clarke   |
| James | 19  | jw25| Clare    |
+---------------------------
```
Put logically independent data in distinct tables?

<table>
<thead>
<tr>
<th>Students:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>age</td>
</tr>
<tr>
<td>+--------------------------+--------------------------</td>
<td></td>
</tr>
<tr>
<td>Eva</td>
<td>18</td>
</tr>
<tr>
<td>Fatima</td>
<td>20</td>
</tr>
<tr>
<td>James</td>
<td>19</td>
</tr>
<tr>
<td>+--------------------------+--------------------------</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Colleges:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cid</td>
<td>college_name</td>
</tr>
<tr>
<td>+--------------------------+--------------------------</td>
<td></td>
</tr>
<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>
<tr>
<td>...</td>
<td>.....</td>
</tr>
</tbody>
</table>

Products between the two tables above:

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q(R, S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>77</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

Note the automatic flattening:

RA Q = R × S

TRC Q = {t | \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)\} | \{(A, a), (B, b)\} \in R \land \{(C, c), (D, d)\} \in S}

SQL select A, a, C, d from R, S
Product is special!

\[ R \times \rho_{A \cdot B} \cdot C, B \cdot D(R) \]

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

- \( \times \) 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.

Natural Join

Given \( R(X, Y) \) and \( S(Y, Z) \), we define the natural join, denoted \( R \bowtie S \), as a relation over attributes \( X, Y, Z \) defined as

\[ R \times S = \{ t \mid \exists u \subseteq R, v \subseteq S, u[Y] = v[Y] \wedge t = u[X] \cup u[Y] \cup v[Z] \} \]

In the Relational Algebra:

\[ R \times S = \pi_{X,Y,Z}(\sigma_{Y=Y'}(R \times \rho_{Y'}(S))) \]
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>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \pi name, cname (Students \times Colleges) \]

\[ \rightarrow \]

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

The same in SQL

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

```
+----------+
| name     | cname  |
|-----------|
|          |       |
| Eva      | King's |
| Fatima   | Clare  |
| James    | Clare  |
+----------+
```
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

\[
R \div S \equiv \pi_X(R) - \pi_X((\pi_X(R) \times S) - R)
\]

Division

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)

\[
R \div S \equiv \{ x \mid \forall s \in S. x \cup s \in R \}.
\]

<table>
<thead>
<tr>
<th>name</th>
<th>award</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatima</td>
<td>writing</td>
</tr>
<tr>
<td>Fatima</td>
<td>music</td>
</tr>
<tr>
<td>Eva</td>
<td>music</td>
</tr>
<tr>
<td>Eva</td>
<td>writing</td>
</tr>
<tr>
<td>Eva</td>
<td>dance</td>
</tr>
<tr>
<td>James</td>
<td>dance</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c}
\text{award} & \text{name} \\
\hline
\text{music} & \text{Eva} \\
\text{writing} & \\
\text{dance} & \\
\end{array}
\]
Query Safety

A query like $Q = \{ t \mid t \in R \land t \neq S \}$ raises some interesting questions. Should we allow the following query?

$Q = \{ t \mid t \neq S \}$

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.

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