Databases 2011
Lectures 01 – 03

Timothy G. Griffin

Computer Laboratory
University of Cambridge, UK

Databases, Lent 2011

Lecture 01 : What is a DBMS?

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

Example Database Management Systems (DBMSs)

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 contains 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

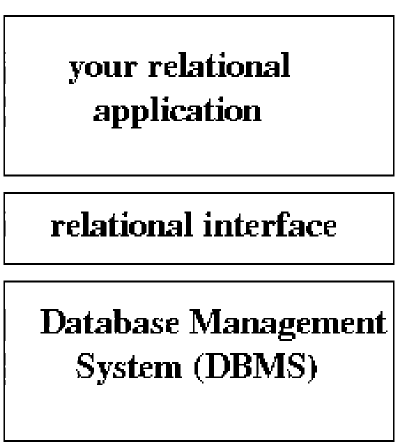
Databases vs Information Retrieval

Always ask What problem am I solving?

DBMS	IR system
exact query results	fuzzy query results
optimized for concurrent updates	optimized for concurrent reads
data models a narrow domain	domain often open-ended
generates documents (reports)	search existing documents
increase control over information	reduce information overload

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 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).

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

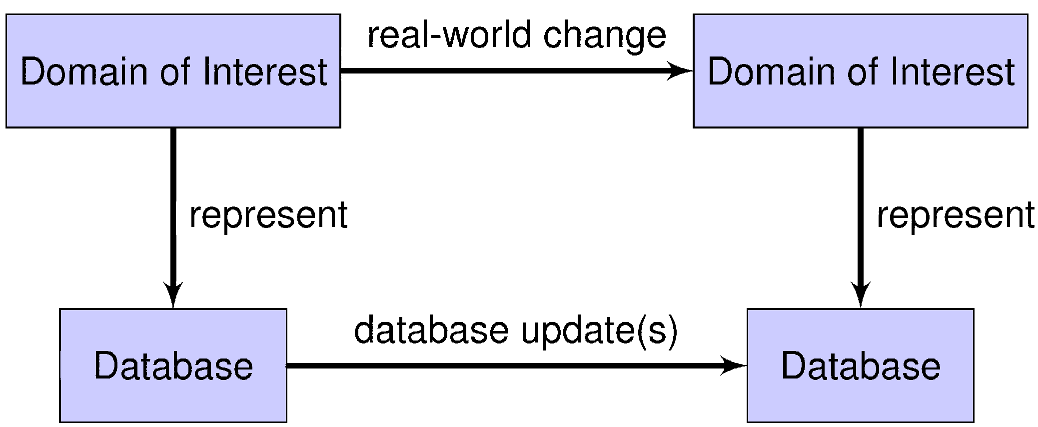
Consistency If each transaction is consistent, and the database is initially consistent, then it is left consistent
- **Applications designers must exploit the DBMS’s capabilities.**

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

Durability If a transactions 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

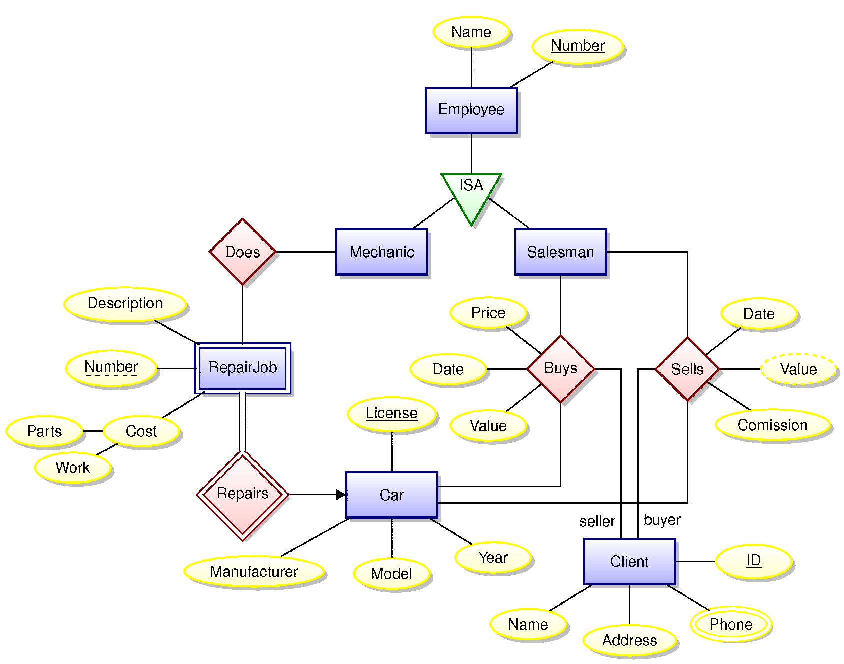
Our tools

Entity-Relationship (ER) modeling	high-level, diagram-based design
Relational modeling	formal model normal forms based on Functional Dependencies (FDs)
SQL implementation	Where the rubber meets the road

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]



[1] By Pável Calado, http://www.texample.net/tikz/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

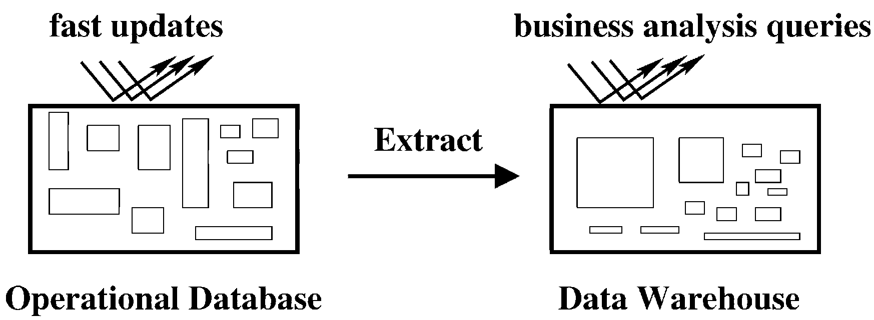
OLTP Online Transaction Processing

OLAP Online Analytical Processing

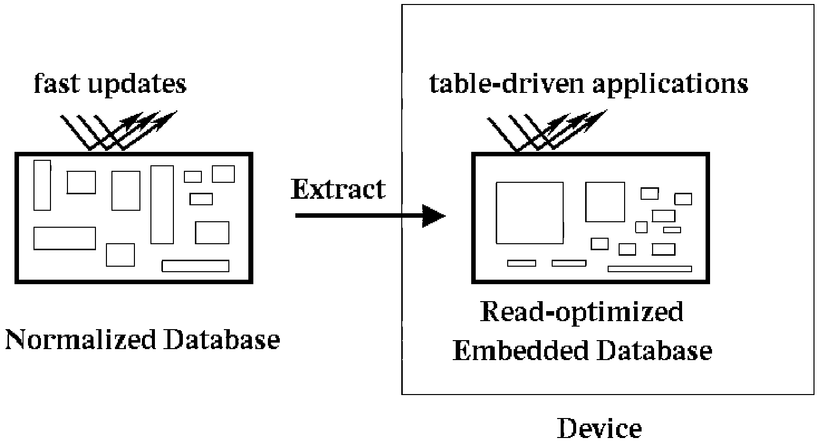
- Commonly associated with terms like Decision Support, Data Warehousing, etc.

	OLAP	OLTP
Supports	analysis	day-to-day operations
Data is	historical	current
Transactions mostly	reads	updates
optimized for	query processing	updates
Normal Forms	not important	important

Example : Data Warehouse (Decision support)

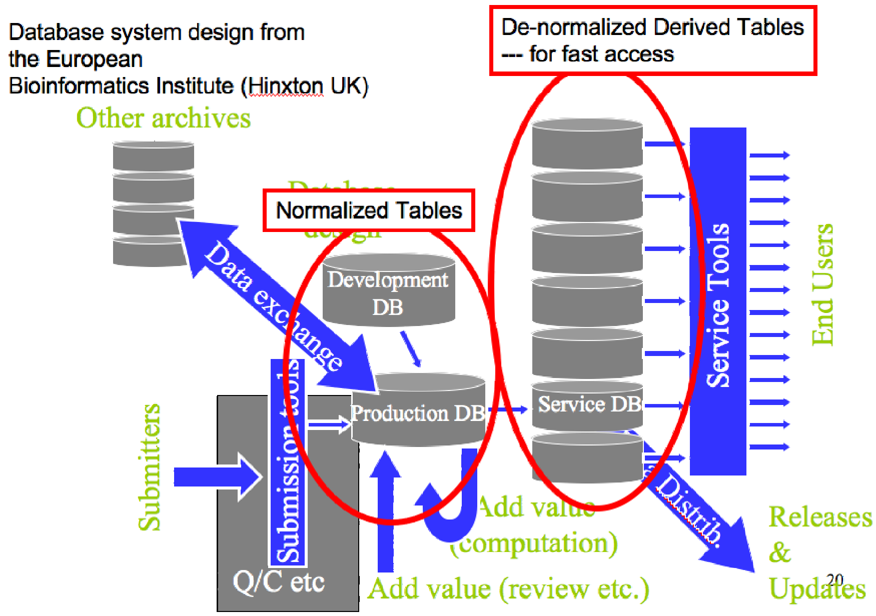


Example : Embedded databases



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 constrains. Their implementation using SQL, Foreign Keys, Referential Integrity.

Term Outline

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.

Lecture 08 **Normal Forms.** 3rd normal form. Boyce-Codd normal form. Decomposition examples. Multi-valued dependencies and Fourth normal form.

Term Outline

Lecture 09 **Schema Decomposition.** Schema decomposition. Lossless join decomposition. Dependency preservation.

Lecture 10 **Schema Evolution.** Scope and goals of database applications change over time. Integration of distinct databases. XML as a data exchange language. Schema integration.

Lecture 11 **Missing data and derived data in SQL** Null values (and three-valued logic). Inner and Outer Joins. Locking vs. update throughput. Indices are derived data! Aggregaton queries. Multi-set (bag) semantics. Database Views. Materialized views. Using views to implement complex integrity constraints. Selective de-normalization.

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

Recommended Reading

Textbooks

SKS **Silberschatz, A., Korth, H.F. and Sudarshan, S. (2002). Database system concepts. McGraw-Hill (4th edition).**

(Adjust accordingly for other editions)
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)

UW Ullman, J. and Widom, J. (1997). A first course in database systems. Prentice Hall.

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)

- **C1970** E.F. Codd, (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM.
- **F1977** Ronald Fagin (1977) Multivalued dependencies and a new normal form for relational databases. TODS 2 (3).
- **L2003** L. Libkin. Expressive power of SQL. TCS, 296 (2003).
- **C+1996** L. Colby et al. Algorithms for deferred view maintenance. SIGMOD 199.
- **G+1997** J. Gray et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals (1997) Data Mining and Knowledge Discovery.
- **H2001** A. Halevy. Answering queries using views: A survey. VLDB Journal. December 2001.

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 $\mathbf{X}$ be a set of k attribute names.

- We will often ignore domains (types) and say that $R(\mathbf{X})$ denotes a relational schema.
- When we write $R(\mathbf{Z},\ \mathbf{Y})$ we mean $R(\mathbf{Z} \cup \mathbf{Y})$ and $\mathbf{Z} \cap \mathbf{Y} = \phi$.
- $u.[\mathbf{X}] = v.[\mathbf{X}]$ abbreviates $u.A_1 = v.A_1 \wedge \cdots \wedge u.A_k = v.A_k$.
- $\vec{\mathbf{X}}$ 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

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

Example

A relational schema

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

A relational instance of this schema

$$\begin{array}{lcl}\textbf{Students} & = & \{ \\ & & \{(\textbf{name},\ \text{Fatima}), (\textbf{sid},\ \text{fm21}), (\textbf{age},\ 20)\}, \\ & & \{(\textbf{name},\ \text{Eva}), (\textbf{sid},\ \text{ev77}), (\textbf{age},\ 18)\}, \\ & & \{(\textbf{name},\ \text{James}), (\textbf{sid},\ \text{jj25}), (\textbf{age},\ 19)\} \\ & & \}\end{array}$$

A tabular presentation

name	sid	age
Fatima	fm21	20
Eva	ev77	18
James	jj25	19

Key Concepts

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

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

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 $R(\underline{\mathbf{Z}},\ \mathbf{Y})$ 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.

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);
```

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.

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

-- 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

$$R_1, R_2, \cdots, R_k \quad \Longrightarrow \quad Q(R_1, R_2, \cdots, R_k)$$

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)

$$
\begin{array}{rclr}
Q & ::= & R & \text{base relation} \\
& | & \sigma_p(Q) & \text{selection} \\
& | & \pi_{\mathbf{X}}(Q) & \text{projection} \\
& | & Q \times Q & \text{product} \\
& | & Q - Q & \text{difference} \\
& | & Q \cup Q & \text{union} \\
& | & Q \cap Q & \text{intersection} \\
& | & \rho_M(Q) & \text{renaming}
\end{array}
$$

- p is a simple boolean predicate over attributes values.
- $\mathbf{X} = \{A_1,\ A_2,\ \ldots,\ A_k\}$ 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)

$$Q = \{(A_1 = v_1,\ A_2 = v_2, \ldots, A_k = v_k) \mid P(v_1,\ v_2, \cdots,\ v_k)\}$$

The SQL standard

- Origins at IBM in early 1970's.
- SQL has grown and grown through many rounds of standardization :
  - ANSI: SQL-86
  - ANSI and ISO : SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008
- SQL is made up of many sub-languages :
  - Query Language
  - Data Definition Language
  - System Administration Language
  - ...

Selection

R

A	B	C	D
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

$\Longrightarrow$

$Q(R)$

A	B	C	D
20	10	0	55
77	25	4	0

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

TRC $Q = \{t \mid t \in R \wedge t.A > 12\}$

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

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

Projection

R

A	B	C	D
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

$\Longrightarrow$

$Q(R)$

B	C
10	0
99	17
25	4

RA $Q = \pi_{B,C}(R)$

TRC $Q = \{t \mid \exists u \in R \wedge t.[B, C] = u.[B, C]\}$

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

SQL `select distinct B, C from R`

Why the `distinct` in the SQL?

The SQL query

```
select B, C from R
```

will produce a bag (multiset)!

R

A	B	C	D
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

$\Longrightarrow$

$Q(R)$

B	C	
10	0	$\star\star\star$
10	0	$\star\star\star$
99	17	
25	4	

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

A	B	C	D
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

$\Longrightarrow$

$Q(R)$

A	E	C	F
20	10	0	55
11	10	0	7
4	99	17	2
77	25	4	0

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

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

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`

Union

R

A	B
20	10
11	10
4	99

S

A	B
20	10
77	1000

$\Longrightarrow$

$Q(R, S)$

A	B
20	10
11	10
4	99
77	1000

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 `(select * from R) union (select * from S)`

Intersection

R

A	B
20	10
11	10
4	99

S

A	B
20	10
77	1000

$\Longrightarrow$

$Q(R)$

A	B
20	10

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

`(select * from R) intersect (select * from S`

Difference

R

A	B
20	10
11	10
4	99

S

A	B
20	10
77	1000

$\Longrightarrow$

$Q(R)$

A	B
11	10
4	99

RA $Q = R - S$

TRC $Q = \{t \mid t \in R \wedge t \notin S\}$

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

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

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 | Clare  |
| James  |    19 | jj25 | Clare  |
+--------+-------+------+--------+
```

Put logically independent data in distinct tables?

```
Students : +--------+------+------+-----+
           | name   | age  | sid  | cid |
           +--------+------+------+-----+
           | Eva    |   18 | ev77 | k   |
           | Fatima |   20 | fm21 | cl  |
           | James  |   19 | jj25 | cl  |
           +--------+------+------+-----+

Colleges : +-----+----------------+
           | cid | college_name   |
           +-----+----------------+
           | k   | King's         |
           | cl  | Clare          |
           | sid | Sidney Sussex  |
           | q   | Queens'        |
             ...        .....
```


Product

R

A	B
20	10
11	10
4	99

S

C	D
14	99
77	100

$\Longrightarrow$

$Q(R,\ S)$

A	B	C	D
20	10	14	99
20	10	77	100
11	10	14	99
11	10	77	100
4	99	14	99
4	99	77	100

Note the automatic flattening

RA $Q = R \times S$

TRC $Q = \{t \mid \exists u \in R, v \in S,\ t.[A, B] = u.[A, B] \wedge t.[C, D] = v.[C, D]\}$

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

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

Product is special!

R

A	B
20	10
4	99

$\Longrightarrow$

$R \times \rho_{A \mapsto C,\ B \mapsto D}(R)$

A	B	C	D
20	10	20	10
20	10	4	99
4	99	20	10
4	99	4	99

- $\times$ 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

Natural Join

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

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

In the Relational Algebra:

$$R \bowtie S = \pi_{\mathbf{X},\mathbf{Y},\mathbf{Z}}(\sigma_{\mathbf{Y}=\mathbf{Y}'}(R \times \rho_{\vec{\mathbf{Y}} \mapsto \vec{\mathbf{Y}'}}(S)))$$

Join example

Students

name	sid	age	cid
Fatima	fm21	20	cl
Eva	ev77	18	k
James	jj25	19	cl

Colleges

cid	cname
k	King's
cl	Clare
q	Queens'
⋮	⋮

$$\pi_{\textbf{name},\textbf{cname}}(\text{Students} \bowtie \text{Colleges})$$

$\Longrightarrow$

name	cname
Fatima	Clare
Eva	King's
James	Clare

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, $R \div S \subseteq \pi_{\mathbf{X}}(R)$. So $R \div S = \pi_{\mathbf{X}}(R) - C$, where C represents counter examples to the division condition. That is, in the TRC,

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

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

Division in RA

$$R \div S \equiv \pi_{\mathbf{X}}(R) - \pi_{\mathbf{X}}((\pi_{\mathbf{X}}(R) \times S) - R)$$

Division

Given $R(\mathbf{X},\ \mathbf{Y})$ and $S(\mathbf{Y})$, the division of R by S, denoted $R \div S$, is the relation over attributes $\mathbf{X}$ defined as (in the TRC)

$$R \div S \equiv \{x \mid \forall s \in S,\ x \cup s \in R\}.$$

name	**award**
Fatima	writing
Fatima	music
Eva	music
Eva	writing
Eva	dance
James	dance

$\div$

award
music
writing
dance

$=$

name
Eva

Query Safety

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

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

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