Databases
Lectures 1 and 2

Timothy G. Griffin

Computer Laboratory
University of Cambridge, UK

Databases, Lent 2009

Re-ordered Syllabus

Note: All lecture slides have been written from scratch for Lent 2009 — please help me find the typos!

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 **Database updates.** Basic ACID properties. Serializability in multi-user database context. 2-phase commits. Locking vs. update throughput.

Re-ordered Syllabus

Lecture 05 **Redundancy is a Bad Thing.** Update anomalies. More redundancy implies more locking. Capturing redundancy with functional and multivalued dependencies.

Lecture 06 **Analysis of Redundancy.** Implied functional dependencies, logical closure. Reasoning about functional dependencies.

Lecture 07 **Eliminating Redundancy.** Schema decomposition. Lossless join decomposition. Dependency preservation. 3rd normal form. Boyce-Codd normal form.

Lecture 08 **Decomposition algorithms.** Decomposition examples. Multivalued dependencies and Fourth normal form.

Re-ordered Syllabus

Lecture 09 **Multisets, grouping, and aggregates.** Bag (multiset) algebra. Aggregates and grouping examples in SQL. More problems with null values.

Lecture 10 **Redundancy is a Good Thing!** The main issue: query response 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

UW1997 Ullman, J. and Widom, J. (1997). A first course in database systems. Prentice Hall.

D2004 Date, C.J. (2004). An introduction to database systems. Addison-Wesley (8th ed.).

SL2002 Silberschatz, A., Korth, H.F. and Sudarshan, S. (2002). Database system concepts. McGraw-Hill (4th ed.).

EN2000 Elmasri, R. and Navathe, S.B. (2000). Fundamentals of database systems. Addison-Wesley (3rd 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 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!

Edgar F. Codd

pgflastimage

- 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).

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:

```
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 `*('a, 'b, 'c)` it would make this implicit flattening more obvious.

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

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

A tabular presentation

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

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. We will get more precise in Lecture 06.

```
-- 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'
```

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 : +--------+------+------+-----+
           | 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'        |
             ...        .....
```

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 asses 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.

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

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

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)

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

- 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 09.

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`

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`

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 $Q = \{t \mid t \in R \vee t \in S\}$

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

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

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

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)`

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.

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

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

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

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