Introduction to Databases, Fall 2005
IT University of Copenhagen

Lecture 2: Relations and SQL

September 5, 2005

Lecturer: Rasmus Pagh

Today's lecture

- What, exactly, is the relational data model?
- What are the basic ways of writing expressions in SQL?
- How do you query data stored in multiple relations?
- How do you create relations, and modify their data?

What you should remember from last week

In this lecture I will assume that you remember:

- The intuitive concept of a relation as a two-dimensional table.
- How a subset of a relation `R` can be obtained using "`SELECT ... FROM R WHERE ...`".

What is the relational data model?

A **data model** is a *precise, conceptual* way of describing the data stored in a database.

The **relational data model** is a data model where all data takes the form of so-called relations.

Note: The term **data model** is also used when speaking about a concrete, conceptual description of a database.

What is a tuple?

A relation consists of so-called tuples.

A **tuple** is an ordered list of values.

Tuples are usually written in parentheses, with commas separating the values (or **components**), e.g.

`(Star Wars, 1977, 124, color)`

which contains the four values `Star Wars`, `1977`, `124`, and `color`.

Note:
Order is significant, e.g., the tuple `(Star Wars, 124, 1977, color)` is different from the tuple above.

What is a relation?

Mathematically, a relation is a **set** of **tuples**.

A relation is always defined on certain sets (or **domains**):

“the sets from which the values in the tuples come”.

Example: The tuple (`Star Wars, 1977, 124, color`) could be part of a relation defined on the sets:

- All text strings,
- the integers,
- the integers, and
- {'color', 'black and white'}.

Attributes

In order to be able to refer to the different components in a tuple, we will assign them names (called **attributes**).

Example:

For the tuple (`Star Wars, 1977, 124, color`) we might choose the attributes *title*, *year*, *length*, and *filmType*.

The attribute *length* would then refer to the third value of the tuple, `124`.

How do we write relations?

Relations are usually written as two-dimensional tables, with the attributes as a first, special row, and the tuples in the remaining rows.

Example:

title	*year*	*length*	*filmType*
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

Equivalent ways of writing relations

The order of the rows does not matter (just the **set** of rows).

We may freely reorder the columns, including the attributes.

Example: Two ways of writing the same relation:

title	*year*	*length*	*filmType*
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

title	*filmType*	*length*	*year*
Wayne's World	color	95	1992
Star Wars	color	12	1977
Mighty Ducks	color	104	1991

Data types in SQL

In SQL, any attribute must have a data type. The data types available depend on the particular DBMS. However, you can come a long way using the following widely available data types:

- `VARCHAR(x)` for text strings of at most `x` characters.
- `INT` for integers.
- `FLOAT` for real numbers.
- `DATE` for Gregorian dates.

Some newer DBMSs allow XML as a data type. XML may be a good choice for less structured or ad-hoc data in a relation.

Schemas

In the relational data model, a relation is described using a **schema** which consists of:

- The name of the relation, and
- a tuple with its attributes (+ sometimes also the attribute data types).

Example: The relation we saw before could have the schema:

```
Movies(title, year, length, filmType)
```

A schema that lists the data types could look like this:

```
Movies(title VARCHAR(20), year INT, length INT, filmType VARCHAR(13))
```

This is the kind of schema used when creating new relations in SQL.

Relations, schemas and relation instances

We use the word "relation" in two different meanings:

- The data model, i.e., what is described by a schema.
- The concrete set of data tuples.

When we want to be specific about the difference, we may refer to the former as the schema, and the latter as the **relation instance**.

Problem session (5 minutes)

Explain to each other the following terms (in the context of this lecture):

- data model
- relational data model
- tuple
- component in a tuple
- data type of a component
- attribute
- relation
- schema
- relation instance

Identify any unclarities about the terms to be discussed in class.

Next: Basic expressions in SQL

Projection

The choice of certain attributes in the `SELECT` part of `SELECT-FROM-WHERE` is referred to as **projection**.

SQL allows a general form of projection, where:

- It is possible to compute a new value from the attributes (not just copy the value of an attribute).
- Attributes in the result relation can be freely named.

Syntax (i.e., the way to write):

```
SELECT <expr1> [[AS] <name1>], <expr2> [[AS] <name2>], ...
```

optional optional

Selection

The choice of certain tuples in the `WHERE` part of `SELECT-FROM-WHERE` is referred to as **selection**.

SQL offers a variety of ways of forming conditional expressions

- Comparison operators (used between e.g. a pair of integers or strings):
  =, <, <=, >, >=, <>.
- Boolean operators (used to combine conditional expressions):
  `AND`, `OR`, `NOT`.
- The `LIKE` operator used to find strings that match a given pattern.
- Parentheses can be used to indicate the order of evaluation. If no parentheses are present, a standard order is used.

Truth values

Often we want to represent truth values (or **boolean values**) in relations.

Example: The `inColor` attribute of the `Movie` relation should contain the value "true" if a movie is in color, and "false" otherwise.

Many RDBMSs use integers to represent truth values (i.e., there is no special data type for that). Typically:

- `0` is used to represent the value "false", and
- `1` is used to represent the value "true".

Problem session (5 minutes)

How many tuples will be returned by each of the following selection queries:

1. `SELECT * FROM R WHERE a=1 OR b=1 OR (d='August');`
2. `SELECT * FROM R WHERE (a=1 AND NOT b=1) OR (NOT a=1 AND b=1);`
3. `SELECT * FROM R WHERE c>22 AND d<'November';`
4. `SELECT * FROM R WHERE d LIKE '%ber';`

R

a	*b*	*c*	*d*
0	0	11	August
0	1	22	September
1	0	33	October
1	1	44	November
0	0	55	December

Next: Querying data stored in multiple relations

Join in SQL

Combination of several relations is known as a **join**.

We first consider the case of *two* relations (call them R1 and R2).

SQL's SELECT-FROM-WHERE can be used to perform what is known as a **theta-join**: *Combine of all pairs of tuples that satisfy some condition*.

```
SELECT *
FROM R1, R2
WHERE <condition>
```

Often the condition will be the AND of one or more equalities that make sure that we join the pairs of tuples that "belong together".

How tuples are combined in the join

When a tuple $(x_1, x_2, \ldots)$ from `R1` and a tuple $(y_1, y_2, \ldots)$ from `R2` are combined by the join query, it results in the **concatenation** of the two tuples:

$$(x_1, x_2, \ldots, y_1, y_2, \ldots)$$

That is, one tuple is put after the other.

Unless the attributes are renamed, we keep the attributes of `R1` and `R2` in the resulting relation.

Problem session (5 minutes)

How many tuples will be returned by each of the following join queries:

1. ```
   SELECT *
   FROM R1, R2
   WHERE a=d AND b=e;
   ```

2. ```
   SELECT *
   FROM R1, R2
   WHERE a=d;
   ```

R1

a	*b*	*c*
7	9	13
21	37	69
21	9	0
7	9	14

R2

d	*e*	*f*
7	9	12
21	31	41
21	37	0
21	37	5
7	9	15

When attributes have the same name

Suppose we had attributes a and b in both R1 and R2.

R1

a	*b*	*c*
7	9	13
21	37	69
21	9	0
7	9	14

R2

a	*b*	*d*
7	9	12
21	31	41
21	37	0
21	37	5
7	9	15

We would then have to put the relation name in front of these attribute names in order to distinguish them. For example:

```
SELECT *
FROM R1, R2
WHERE R1.a=R2.a AND R1.b=R2.b;
```

If we want to join tuples with the same values in attributes of the same name, we can choose to instead use SQL's `NATURAL JOIN`.

Joining more than two relations

SQL may be used to join any number of relations:

```
SELECT *
FROM R1, R2,..., Rk
WHERE <condition>
```

How this is evaluated (the **semantics** of the query):

For every list of tuples $t_1, t_2, \ldots, t_k$ from `R1, R2,...,Rk`, respectively, include the concatenation of $t_1, t_2, \ldots, t_k$ in the result if `<condition>` is true.

Joining several relations – explanation 2

The semantics of

```
SELECT *
FROM R1, R2,..., Rk
WHERE <condition>
```

is specified by the following pseudocode:

```
for n1=1 to size(R1) do
 for n2=1 to size(R2) do
   ...
   for nk=1 to size(Rk) do
     if <condition on R1[n1],R2[n2],...,Rk[nk]>
       output R1[n1]+R2[n2]+...+Rk[nk]
```

Joining a relation with itself!

Sometimes you need to join a relation with itself:
"Find all pairs of tuples in R such that..."

This can be done using so-called **tuple variables** which can be thought of as representing *different copies* of the relation.

Joining R with itself using tuple variables r1 and r2 in the condition:

```
SELECT *
FROM R r1, R r2
WHERE <condition using r1 and r2>
```

Semantics:
Same as when joining relations r1 and r2 that are both identical to R.

Problem session (5 minutes)

How many tuples will be returned by the join query:

```
SELECT *
FROM R r1, R r2, R r3
WHERE r1.a=r2.a AND r2.a=r3.a;
```

R

a	b	d
7	9	12
21	31	41
21	37	0
21	37	5
7	9	15

Next: Creating a relation; Modifying its data

Inserting tuples

Inserting a tuple with value `x1` for attribute `a1`, value `x2` for attribute `a2`, etc:

```
INSERT INTO StarsIn(a1,a2,a3,...)
VALUES (x1,x2,x3,...);
```

If the **standard order** of the attributes is `a1,a2,a3,...` this can also be written:

```
INSERT INTO StarsIn
VALUES (x1,x2,x3,...);
```

Deleting tuples

Deletion is similar to selection, except that the tuples selected are permanently removed from the relation (and not returned as a result):

```
DELETE FROM R
WHERE <condition>;
```

Modifying tuples

Modification is similar to selection, except that the tuples selected are modified according to the `SET` clause (and not returned as a result).

Changing attribute `a` of all tuples in `R` that satisfy `<condition>`:

```
UPDATE R
SET a = <expression>
WHERE <condition>;
```

Creating a new relation

In SQL creating a relation called `NewRel` with attributes `a1,a2,a3,...` can be done as follows:

```
CREATE TABLE NewRel
(a1 <data type of a1>, a2 <data type of a2>,...);
```

The data types must be chosen from SQL's data types, e.g., INT, FLOAT, and VARCHAR.

A relation `UselessRel` can be permanently deleted using

```
DROP TABLE UselessRel;
```

Next: Some final points

Results are relations

As you have seen, the result of a query on one or more relations is itself a relation.

Later in the course, we will see that this is quite handy, using the so-called **subquery** capability of SQL.

Relations in DBMSs

There are differences between the mathematical formulation of a relation (as a set of tuples) and the representation in an RDBMS (which can be thought of as similar to the way we write relations as a table).

- Different terminology: Rows/records vs tuples, tables vs relations, attributes vs fields, . . .
- An RDBMS stores each tuple in a specific "standard" order, and stores the tuples as a list, rather than a set.
- An RDBMS may store the same tuple several times (i.e., we have a **bag** of tuples rather than a set).

It is important to understand both worlds (and their differences) – more on this in the rest of the course.

Outstanding point: NULL and UNKNOWN

`NULL` is a special value that may be used for any data type when no other value is applicable.

Evaluating expressions involving NULL:

- Arithmetic expressions involving NULL evaluate to NULL.
- Boolean expressions evaluate to NULL only when the correct value cannot be deduced.

Important point: Evaluation is done in a local, "step by step" way.

Some RDBMSs support a null value for booleans called UNKNOWN.

Most important points in this lecture

As a minimum, you should after this week:

- Understand what a relation is (mathematically and in an RDBMS).
- Know the basic ways of forming SQL expressions: Projection and renaming, selection using e.g. `AND`, `OR`, `LIKE`, <, <=, . . .
- Understand how to query (in SQL) information stored in multiple relations, including:
  - Dealing with identical attribute names.
  - Using tuple variables.
- Know how to modify, insert, and delete tuples in SQL.
- Know how to create a new relation in SQL.

Next time

Next week we will begin our study of database design:

- Conceptual modeling of a database using "E/R modeling".
- Conversion of an E/R model to a relational data model.