
Databasesystemer, forår 2005
IT Universitetet i København

Forelæsning 2: Relationer og SQL

10. februar 2005

Forelæser: Rasmus Pagh

— Forelæsningen i dag —

- Praktisk information (fra kursushjemmeside).
- Hvad er en relationel datamodel?
- Hvad er en relation?
- Grundlæggende SQL forespørgsler.
- SQL forespørgsler der involverer flere relationer.
- Views i SQL.
- Aggregering i SQL.
- Opdatering af data i en relation i SQL.
- Skabelse af nye relationer i SQL.

— Hvad er den relationelle datamodel? —

En **datamodel** er en *præcis, konceptuel* beskrivelse af data gemt i en database.

En **relational data model** er en datamodel hvor al data er organiseret i relationer.

— Hvad er en relation? —

Matematisk set er en relation en **mængde** af **tupler** (også kaldet “rækker” eller “records”).

For at kunne referere til de enkelte komponenter (eller “søjler”) giver man dem navne, kaldet **attributter**.

— Hvordan visualiserer vi relationer? —

Relationer vises normalt som 2-dimensionelle tabeller, med attributterne som en speciel, første række, og tuplerne i de resterende rækker.

Eksempel:

<i>title</i>	<i>year</i>	<i>length</i>	<i>filmType</i>
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

— Ens betydende måder at opskrive relationer —

Rækkefølgen af rækkerne er uden betydning – kun *mængden* af rækker har betydning.

Søjlerne kan også byttes rundt uden at vi taler om en anden relation.

Eksempel: To måder at opskrive samme relation:

<i>title</i>	<i>year</i>	<i>length</i>	<i>filmType</i>
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

<i>title</i>	<i>filmType</i>	<i>length</i>	<i>year</i>
Wayne's World	color	95	1992
Star Wars	color	12	1977
Mighty Ducks	color	104	1991

Datatyper i SQL

I SQL er enhver attribut forbundet med en datatype. De datatyper, der er til rådighed afhænger desværre meget af, hvilken DBMS man bruger. Man kan dog komme langt med flg. udbredte datatyper:

- VARCHAR(x) for tekststrengene på højst x tegn.
- INT for heltal.
- FLOAT for reelle tal (“kommatal”).
- DATE for Gregorianske datoer.

Nyere DBMSer tillader ofte XML som datatype. XML kan være et godt valg til mindre strukturerede eller ad-hoc data i en relation.

Skemaer

I en relationel datamodel angives for hver relation et *skema* (schema), som består af:

- Relationens navn
- En liste af dens attributter, samt evt. deres datatyper

Eksempel: Relationen fra før kunne have flg. skema:

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

Et skema, der angiver datatyper kunne se sådan ud:

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

Det er skemaer af denne type, der bruges til at skabe nye relationer i SQL.

— Relationer, skemaer, og relationsinstanser —

Ordet “relation” bruges ofte i to betydninger:

- Datamodellen, dvs. hvad skemaet beskriver.
- En konkret mængde af tupler.

Hvis vi ønsker at skelne, bruger vi “skema” for den første betydning, og “instans” for den anden betydning.

— Problem session (5 minutter) —

Forklar hinanden flg. begreber:

- datamodel
- relationel datamodel
- tupel
- komponent i et tupel
- attribut
- datatype for en attribut
- relation
- skema
- instans af en relation

Identificer evt. uklarheder til videre diskussion.

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
```

(MDM Fig. 7-5 shown on slide.)

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 (10 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';`

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>R</i>	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, \dots) from R1 and a tuple (y_1, y_2, \dots) from R2 are combined by the join query, it results in the **concatenation** of the two tuples:

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

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;

	<i>a</i>	<i>b</i>	<i>c</i>
<i>R1</i>	7	9	13
	21	37	69
	21	9	0
	7	9	14

	<i>d</i>	<i>e</i>	<i>f</i>
<i>R2</i>	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.

	<i>a</i>	<i>b</i>	<i>c</i>		<i>b</i>	<i>a</i>	<i>d</i>
<i>R1</i>	7	9	13	<i>R2</i>	7	9	12
	21	37	69		21	31	41
	21	9	0		21	37	0
	7	9	14		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.b AND R1.b=R2.a;
```

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, \dots, t_k from R_1, R_2, \dots, R_k , respectively, include the concatenation of t_1, t_2, \dots, t_k in the result if $\langle \text{condition} \rangle$ 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> then  
        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;
```

	<i>a</i>	<i>b</i>	<i>d</i>
<i>R</i>	7	9	12
	21	31	41
	21	37	0
	21	37	5
	7	9	15

— Inner join syntax —

An alternative (SQL-92) syntax for joining two relations R1 and R2 is available in some DBMSs:

```
R1 INNER JOIN R2 ON <condition>
```

This syntax can only be used as *part* of a SELECT statement, e.g.

```
SELECT *  
FROM (R1 INNER JOIN R2 ON a=b)  
WHERE c>0;
```

Next: Views. Aggregation.

Views

Views are queries that have been given a name.

Syntax for declaring a view:

```
CREATE VIEW <name of view> AS <SQL query>
```

We may use the name of a view in SQL expressions, as a *shorthand* for the corresponding query. Potential benefits:

- Can be used to split complex queries into simpler parts.
- Can allow reuse of SQL code among different queries.
- A view may be rewritten without changing the queries that reference the view.

— Properties of views —

- Views are elements of the database schema, just like relation schemas.
- Sufficiently simple views can be modified, meaning that the the modifications are passed on to the underlying relations.
- Privileges to access a view are handled just like privileges for relations (more on privileges later).

— Materialized views —

Materialized views are views that are physically stored, i.e. stored relations that are results of queries.

Non-materialized views are sometimes called **dynamic views**.

Syntax for declaring a materialized view in Oracle:

```
CREATE MATERIALIZED VIEW <name of view>  
AS <SQL query>
```

Differences from an ordinary view:

- Allows faster access, as the query result is always computed.
- Needs to be updated when the underlying relations change.

Aggregates

Many analysis queries are about *aggregates* such as sums and averages. SQL can be used to specify aggregate queries.

Some examples:

```
SELECT SUM(price) FROM Sales;
```

```
SELECT dealer, AVG(price) FROM Sales  
GROUP BY dealer;
```

```
SELECT state, AVG(price)  
FROM Sales, Dealers  
WHERE dealer=name AND date>'2001-09-11'  
GROUP BY state;
```

— Conditions on groups —

In connection with a GROUP BY clause, one can specify a HAVING clause. This is similar to the WHERE clause, but can *only* be used to specify conditions on groups of tuples (as opposed to conditions on single tuples).

More precisely, it can refer to attributes in the GROUP BY clause, and to aggregates on any attribute.

Example:

```
SELECT state, AVG(price)
FROM Sales, Dealers
WHERE dealer=name AND date>'2001-09-11'
GROUP BY state
HAVING state='NY' OR COUNT(*)>2;
```

(MDM Fig. 7-8 shown on slide.)

Next: Inserting, deleting, and modifying tuples in a relation

Inserting

Inserting a tuple with value x_1 for attribute a_1 , value x_2 for attribute a_2 , etc:

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

If the **standard order** of the attributes is a_1, a_2, a_3, \dots this can also be written:

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

Deleting

Deletion is similar to selection, except that the tuples selected are permanently removed from the relation:

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

— Modifying —

Modification is similar to selection, except that the tuples selected are modified according to the SET clause.

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

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

Next: Creating new relations in SQL

— 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) – much more on this in the rest of the course.

— 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 define views, and compute simple aggregates.
- Know how to modify, insert, and delete tuples in SQL.
- Know how to create a new relation in SQL.

Næste gang

Næste gang tager vi hul på database design, og ser specielt på:

- Analyse og konceptuel modellering
- E-R modellering