Database Usage (and Construction)

SQL Queries and Relational Algebra

Views

Which SQL definition for a room is most correct?

(A)

```
CREATE TABLE Rooms(
  name VARCHAR(10),
  capacity INTEGER,
  PRIMARY KEY(name)
);
```

(B)

```
CREATE TABLE Rooms(
  name VARCHAR(10),
  capacity INTEGER NOT NULL,
  PRIMARY KEY(name)
);
```

(C)

```
CREATE TABLE Rooms(
  name VARCHAR(10),
  capacity INTEGER CHECK(capacity > 0) NOT NULL,
  PRIMARY KEY(name)
);
```

(D)

```
CREATE TABLE Rooms(
  name VARCHAR(10),
  capacity INTEGER CHECK(capacity > 0),
  PRIMARY KEY(name)
);
```

Summary so far

- SQL is based on relational algebra.
  - Operations over relations
- Operations for:
  - Selection of rows (σ)
  - Projection of columns (π)
  - Combining tables
    - Cartesian product (x)
    - Join, natural join ($\bowtie_C$, $\bowtie$)

SELECT-FROM-WHERE

- Basic structure of an SQL query:

```
SELECT attributes
FROM   tables
WHERE  tests over rows
```

```
SELECT X
FROM   T
WHERE  C
```

⟺

$\pi_X(\sigma_C(T))$

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE  teacher = 'Mickey'
       AND code = course;
```

GivenCourses

course	*per*	*teacher*
TDA357	3	Mickey
TDA357	2	Tweety
TIN090	1	Pluto

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

$$\pi_{code,name,period}(\sigma_{teacher='Mickey' \,\&\, code = course}(\text{Courses x GivenCourses}))$$

Example:

```
SELECT code, name, period
FROM Courses, GivenCourses
WHERE  teacher = 'Mickey'
       AND code = course;
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Mickey
TDA357	Databases	TDA357	2	Tweety
TDA357	Databases	TIN090	1	Pluto
TIN090	Algorithms	TDA357	3	Mickey
TIN090	Algorithms	TDA357	2	Tweety
TIN090	Algorithms	TIN090	1	Pluto

$\pi_{code,name,period}(\sigma_{teacher='Mickey' \& code = course}(Courses \times GivenCourses))$

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE teacher = 'Mickey'
      AND code = course;
```

code	*name*	*course*	*per*	*Teacher*
TDA357	Databases	TDA357	3	Mickey
TDA357	Databases	TDA357	2	Tweety
TDA357	Databases			
TIN090	Algorithms			
TIN090	Algorithms	TDA357	2	Tweety
TIN090	Algorithms	TIN090	1	Pluto

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Mickey

$$\pi_{code,name,period}(\sigma_{teacher='Mickey' \ \& \ code\ =\ course}(Courses \times GivenCourses))$$

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE  teacher = 'Mickey'
       AND code = course;
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Mickey

code	*name*	*per*
TDA357	Databases	3

$\pi_{code,name,period}(\sigma_{teacher='Mickey' \ \& \ code = course}(Courses \times GivenCourses))$

Quiz!

What does the following relational algebra expression compute?

$$\sigma_{teacher='Mickey' \ \& \ code = course}(\pi_{code,name,period}(Courses \times GivenCourses))$$

The expression is invalid, since the result after the projection will not have attributes teacher and course to test.

More complex expressions

- So far we have only examples of the same simple structure:

$$\pi_X(\sigma_C(T))$$

- We can of course combine the operands and operators of relational algebra in (almost) any way imaginable.

$$\sigma_C(R_3 \bowtie_D \pi_X(R_1 \times R_2))$$

```
SELECT *
FROM   R3 JOIN (SELECT X FROM R1,R2) ON D
WHERE  C
```

Subqueries

- Subqueries is a term referring to a query used inside another query:

```
SELECT teacher
FROM   GivenCourses NATURAL JOIN
         (SELECT course, period
          FROM   Lectures
          WHERE  weekday = ’Mon’)
WHERE  period = 3;
```

- Beware the natural join!!
- ”List all teachers who have lectures on Mondays in period 3”
- SQL is a language where any query can be written in lots of different ways…

```
SELECT course, period
FROM   Lectures
WHERE  weekday = 'Mon'
```

course	*period*	*room*	*weekday*	*hour*
TDA357	3	HC1	Mon	13
TDA357	3	HC1	Thu	10
TDA357	2	VR	Tue	8
TDA357	2	HC1	Thu	13
TIN090	1	HA4	Mon	8
TIN090	1	HC3	Thu	13

```
SELECT course, period
FROM   Lectures
WHERE  weekday = 'Mon'
```

course	*period*	*room*	*weekday*	*hour*
TDA357	3	HC1	Mon	13
TIN090	1	HA4	Mon	8

```
SELECT teacher
FROM   GivenCourses NATURAL JOIN
          (SELECT course, period
           FROM   Lectures
           WHERE  weekday = 'Mon')
WHERE  period = 3;
```

course	*period*
TDA357	3
TIN090	1

course	*period*	*teacher*	*#students*
TDA357	3	Mickey	130
TDA357	2	Tweety	135
TIN090	1	Pluto	95

```sql
SELECT teacher
FROM   GivenCourses NATURAL JOIN
         (SELECT course, period
          FROM   Lectures
          WHERE  weekday = 'Mon')
WHERE  period = 3;
```

course	*period*	*teacher*	*#students*
TDA357	3	Mickey	130
TIN090	1	Pluto	95

Result

teacher
Mickey

Renaming attributes

- Sometimes we want to give new names to attributes in the result of a query.
  - To better understand what the result models
  - In some cases, to simplify queries

```
SELECT *
FROM   Courses NATURAL JOIN
         (SELECT course AS code, period, teacher
          FROM   GivenCourses);
```

Renaming relations

- Name the result of a subquery to be able to refer to the attributes in it.
- Alias existing relations (tables) to make referring to it simpler, or to disambiguate.

```
SELECT L.course, weekday, hour, room
FROM   Lectures L, GivenCourses G, Rooms
WHERE  L.course = G.course
       AND L.period = G.period
       AND room = name
       AND nrSeats < nrStudents;
```

What does this query mean?

Renaming relations

- Name the result of a subquery to be able to refer to the attributes in it.
- Alias existing relations (tables) to make referring to it simpler, or to disambiguate.

```
SELECT L.course, weekday, hour, room
FROM   Lectures L, GivenCourses G, Rooms
WHERE  L.course = G.course
       AND L.period = G.period
       AND room = name
       AND nrSeats < nrStudents;
```

List all lectures that are scheduled in rooms with too few seats.

Renaming in Relational Algebra

- Renaming = Given a relation, give a new name to it, and (possibly) to its attributes

$$\rho_{A(X)}(R)$$

  - Rename R to A, and the attributes of R to the names specified by X (must match the number of attributes).
  - Leaving out X means attribute names stay the same.
  - Renaming the relation is only necessary for subqueries.
  - **ρ** = rho = greek letter **r** = **r**ename

Sequencing

- Easier to handle subqueries separately when queries become complicated.
  - Example: $\pi_X(R_1 \bowtie_C R_2)$ could be written as

$$R_3 := R_1 \times R_2$$
$$R_4 := \sigma_C(R_3)$$
$$R := \pi_X(R_4)$$

  - In SQL:

```
WITH
 R3 AS (SELECT * FROM R1, R2),
 R4 AS (SELECT * FROM R3 WHERE C)
SELECT X FROM R4;
```

- Example:

```
WITH DBLectures AS
   (SELECT room, hour, weekday
    FROM   Lectures
    WHERE  course = 'TDA357'
           AND period = 3)
SELECT weekday
FROM   DBLectures
WHERE  room = 'HC1';
```

What does this query mean?

- Example:

```
WITH DBLectures AS
   (SELECT room, hour, weekday
    FROM   Lectures
    WHERE  course = 'TDA357'
           AND period = 3)
SELECT weekday
FROM   DBLectures
WHERE  room = 'HC1';
```

Lists the days when the Databases course has lectures in room HC1 during period 3.

Creating views

- A *view* is a ”virtual table”, or ”persistent query” – a relation defined in the database using data contained in other tables.

```
CREATE VIEW viewname AS query
```

- For purposes of querying, a view works just like a table.
- Depending on your DBMS, a view can be read-only, or allow modifications to the underlying table.

Example:

```
CREATE VIEW DBLectures AS
 SELECT room, hour, weekday
 FROM   Lectures
 WHERE  course = 'TDA357'
        AND period = 3;


SELECT weekday
FROM   DBLectures
WHERE  room = 'HC1';
```

BREAK!

Air Traffic Exercise

- Write an SQL query that shows the names of all cities together with the number of flights that depart/arrive from/to them

The WHERE clause

- Specify conditions *over rows*.
- Can involve
  - constants
  - attributes in the row
  - simple value functions (e.g. ABS, UPPER)
  - subqueries

- Lots of nice tests to make…

Testing for membership

- Test whether or not a tuple is a member of some relation.

```
tuple [NOT] IN subquery {or literal set}
```

```
SELECT course
FROM   GivenCourses
WHERE  period IN (1,4);
```

List all courses that take place in the first or fourth periods.

Quiz!

List all courses given by a teacher who also gives the Databases course (TDA357). (You must use IN…)

```
SELECT course
FROM   GivenCourses
WHERE  teacher IN
          (SELECT teacher
           FROM   GivenCourses
           WHERE  course = 'TDA357');
```

Testing for existance

- Test whether or not a relation is empty.

```
[NOT] EXISTS subquery
```

e.g. List all courses that have lectures.

```
SELECT code
FROM   Courses
WHERE  EXISTS
         (SELECT *
          FROM   Lectures
          WHERE  course = code);
```

Note that code is in scope here since it is an attribute in the row being tested in the outer ”WHERE” clause. This is called a correlated query.

Quiz!

List all courses that are not given in the third period. (You must use EXISTS…)

```
SELECT code
FROM   Courses
WHERE  NOT EXISTS
         (SELECT *
          FROM   GivenCourses
          WHERE  course = code
                 AND period = 3);
```

Ordinary comparisons

- Normal comparison operators like =, <, !=, but also the special BETWEEN.

```
value1 BETWEEN value2 AND value3
```

```
SELECT course
FROM   GivenCourses
WHERE  period BETWEEN 2 AND 3;
```

List all courses that take place in the second or third periods.

– Same thing as

```
value2 <= value1 AND value1 <= value3
```

Comparisons with many rows

- Two operators that let us compare with all the values in a relation at the same time.

```
tuple op ANY subquery {or literal set}
tuple op ALL subquery {or literal set}
```

```
SELECT course
FROM   GivenCourses
WHERE  period = ANY (ARRAY[1,4]);
```

List all courses that take place in the first or fourth periods.

Quiz!

List the course(s) with the fewest number of students (in any period). (You must use ANY or ALL…)

```
SELECT course
FROM   GivenCourses
WHERE  nrStudents <= ALL
         (SELECT nrStudents
          FROM   GivenCourses);
```

String comparisons

- Normal comparison operators like < use lexicographical order.
  – ’foo’ < ’fool’ < ’foul’
- Searching for patterns in strings:

```
string LIKE pattern
```

  – Two special pattern characters:
    - _ (underscore) matches any one character.
    - % matches any (possibly empty) sequence of characters.

Quiz!

List all courses that have anything to do with databases (i.e. have the word Database in their name).

```
SELECT *
FROM   Courses
WHERE  name LIKE '%Database%';
```

The NULL symbol

- Special symbol NULL means either
  - we have no value, or
  - we don’t know the value

- Use with care!
  - Comparisons and other operations won’t work.
  - May take up unnecessary space.

Comparing values with NULL

- The logic of SQL is a three-valued logic – TRUE, FALSE and UNKNOWN.
- Comparing any value with NULL results in UNKNOWN.
- A row is selected if all the conditions in the WHERE clause are TRUE for that row, i.e. not FALSE *nor UNKNOWN*.

Three-valued logic

- Rules for logic with unknowns:
  - true AND unknown = unknown
  - false AND unknown = false

  - true OR unknown = true
  - false OR unknown = unknown

  - unknown AND/OR unknown = unknown

Unintuitive result

```
SELECT *
FROM   Rooms
WHERE  nrSeats > 10
       OR nrSeats <= 10;
```

UNKNOWN

UNKNOWN

UNKNOWN

Rooms

<u>*name*</u>	*nrSeats*
VR	NULL

We don't know the value

Don’t expect the ”usual” results

- Laws of three-valued logic are not the same as those for two-valued logic.
- Some laws hold, like commutativity of AND and OR.
- Others do not:

  p OR NOT p = true

Select name of all rooms with capacity of 100 or more

(A) $\pi_{name}(\sigma_{capacity>=100}(Rooms))$

(B) $\sigma_{name}(\pi_{capacity>=100}(Rooms))$

(C) $\sigma_{name}(Rooms)$

(D) $\pi_{capacity>=100}(Rooms)$

Next time, Lecture 7

More Relational Algebra and SQL