Database Usage
(and Construction)

SQL Queries and Relational Algebra

Views

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
```

⟺

$$\boxed{\pi_X(\sigma_C(T))}$$

Example:

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

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

GivenCourses

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

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

Example:

```
SELECT code, name, period
FROM Courses, GivenCourses
WHERE  teacher = ’Niklas Broberg’
       AND code = course;
```

code	name	course	per	teacher
TDA357	Databases	TDA357	3	Niklas Broberg
TDA357	Databases	TDA357	2	Graham Kemp
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	Algorithms	TDA357	3	Niklas Broberg
TIN090	Algorithms	TDA357	2	Graham Kemp
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

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

Example:

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

code	*name*	*course*	*per*	*Teacher*
TDA357	Databases	TDA357	3	Niklas Broberg
TDA357	Databases	TDA357	2	Graham Kemp
TDA357	Databas			
TIN090	Algorithms	TDA357	3	Niklas Broberg
TIN090	Algorithms	TDA357	2	Graham Kemp
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Niklas Broberg

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

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE  teacher = ’Niklas Broberg’
       AND code = course;
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Niklas Broberg

code	*name*	*per*
TDA357	Databases	3

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

Quiz!

What does the following relational algebra expression compute?

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

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:

$$\boxed{\pi_X(\sigma_C(T))}$$

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

$$\boxed{\sigma_C(R_3 \bowtie_D \pi_X(R_1 \text{ x } R_2))}$$

```
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	Niklas Broberg	130
TDA357	2	Graham Kemp	135
TIN090	1	Devdatt Dubashi	95

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

course	*period*	*teacher*	*#students*
TDA357	3	Niklas Broberg	130
TIN090	1	Devdatt Dubashi	95

Result

teacher
Niklas Broberg

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;
```

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

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

  - 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

Quiz!

Write a query that lists all courses that are given in more than one period, with different teachers.

```
SELECT A.course
FROM   GivenCourses A, GivenCourses B
WHERE  A.course = B.course
       AND A.teacher != B.teacher;
```

Sequencing

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

$$
\begin{aligned}
\mathbf{R_3} &\mathbf{:=} \mathbf{R_1 \; X \; R_2} \\
\mathbf{R_4} &\mathbf{:=} \boldsymbol{\sigma}_C(\mathbf{R_3}) \\
\mathbf{R} &\mathbf{:=} \pi_{\mathbf{X}}(\mathbf{R_4})
\end{aligned}
$$

  - In SQL:

WITH

 R_3 AS (SELECT * FROM R_1, R_2),

 R_4 AS (SELECT * FROM R_3 WHERE C)

SELECT X FROM R_4;

- 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. The main difference is that you can’t perform modifications on it – its contents is defined by other tables.

Example:

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


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

Lab Part III – Construction and Usage

- Implement your full schema from part II by creating tables in Oracle for your relations. Be sure to include all constraints.
- Create views that simplify key operations of the system and define an interface to the application.
- Fill your tables with data that stress-tests your implementation.

Lab Part III – Construction and Usage

- Hand in:
  - Your SQL code for creating the tables.
  - Your SQL code for creating the views.
  - Your SQL code for inserting data.
  - Motivations for the chosen data (plain text).
  - Your Oracle username and password.
- Submission deadline: Fri, Feb 14 (23:59)

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 (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

name	*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

Arithmetic in queries

- We allow arithmetic operations in queries.

```
SELECT weekday, hour, room, course,
       nrSeats – nrStudents AS nrFreeSeats
FROM   Rooms,
       (Lectures NATURAL JOIN GivenCourses)
WHERE  name = room;
```

- Not just arithmetic, but rather any operations on values.
  - Oracle has lots of pre-defined functions.

Constants

- Constants can be used in projections.

```
SELECT code, name,
       ’Database course’ AS comment
FROM   Courses
WHERE  name LIKE ’%Database%’;
```

code	*name*	*comment*
TDA357	Databases	Database course

  – Beware of keywords…

Quiz!

What will the result of this query be?

```
SELECT 1
FROM   Courses;
```

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

1
1
1

For each row in Courses that passes the test (all rows since we have no test), project the value 1.

Aggregation

- Aggregation functions are functions that produce a single value over a relation.
  - SUM, MAX, MIN, AVG, COUNT…

```
SELECT MAX(nrSeats)
FROM   Rooms;
```

MAX actually has Rooms as an implicit argument!

```
SELECT COUNT(*)
FROM   Lectures
WHERE  room = 'HC1';
```

Quiz!

List the room(s) with the highest number of seats, and its number of seats.

```
SELECT name, MAX(nrSeats)
FROM   Rooms;
```

NOT correct!

Error when trying to execute, why is it so?

Aggregate functions are special

- Compare the following:

```
SELECT nrSeats
FROM   Rooms;
```

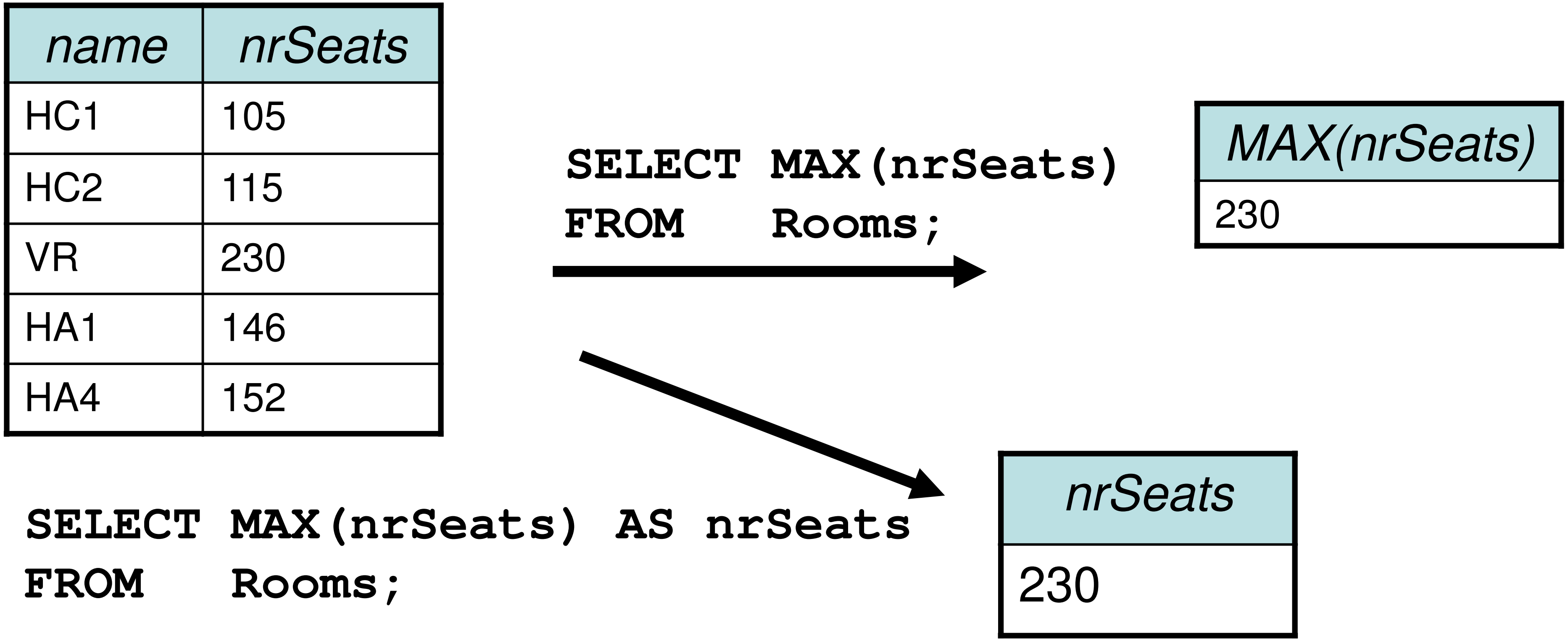
```
SELECT MAX(nrSeats)
FROM   Rooms;
```

  - The ordinary selection/projection results in a relation with a single attribute nrSeats, and one row for each row in Rooms.
  - The aggregation results in a single value, not a relation.
  - We can’t mix both kinds in the same query! (almost…more on this later)

name	*nrSeats*
HC1	105
HC2	115
VR	230
HA1	146
HA4	152

```
SELECT nrSeats
FROM   Rooms;
```

nrSeats
105
115
230
146
152



name	*nrSeats*
HC1	105
HC2	115
VR	230
HA1	146
HA4	152

```
SELECT MAX(nrSeats)
FROM   Rooms;
```

MAX(nrSeats)
230

```
SELECT MAX(nrSeats) AS nrSeats
FROM   Rooms;
```

nrSeats
230

Quiz! New attempt

List the room(s) with the highest number of seats, and its number of seats.

```
SELECT name,
        (SELECT MAX(nrSeats)
         FROM   Rooms)
FROM   Rooms;
```

Not correct either, will list all rooms, together with the highest number of seats in any room.

Let’s try yet again…

name	*nrSeats*
HC1	105
HC2	115
VR	230
HA1	146
HA4	152

```
SELECT name,
   (SELECT MAX(nrSeats)
    FROM   Rooms)
FROM   Rooms;
```

name	*nrSeats*
HC1	230
HC2	230
VR	230
HA1	230
HA4	230

Quiz! New attempt

List the room(s) with the highest number of seats, and its number of seats.

```
SELECT name, nrSeats
FROM   Rooms
WHERE  nrSeats = MAX(nrSeats);
```

Still not correct, MAX(nrSeats) is not a test over a row so it can’t appear in the WHERE clause!

Let’s try yet again…

Quiz!

List the room(s) with the highest number of seats, and its number of seats.

```
SELECT name, nrSeats
FROM   Rooms
WHERE  nrSeats =
          (SELECT MAX(nrSeats)
           FROM   Rooms);
```

That’s better!

Single-value queries

- If the result of a query is known to be a single value (like for MAX), the whole query may be used as a value.

```
SELECT name, nrSeats
FROM   Rooms
WHERE  nrSeats =
          (SELECT MAX(nrSeats)
           FROM   Rooms);
```

- Dynamic verification, so be careful…

NULL in aggregations

- NULL never contributes to a sum, average or count, and can never be the maximum or minimum value.
- If there are no non-null values, the result of the aggregation is NULL.

Summary – aggregation

- Aggregation functions: MAX, MIN, COUNT, AVG, SUM
- Compute a single value over a whole relation.
- Can’t put aggregation directly in the WHERE clause (since it’s not a function on values).
- Can’t mix aggregation and normal projection!
  - … well, not quite true…

Not quite true?

- Sometimes we want to compute an aggregation for every value of some other attribute.
  - Example: List the average number of students that each teacher has on his or her courses.
  - To write a query for this, we must compute the averaging aggregation *for each value of teacher*.

Summary

- Complex queries, involving subqueries
  - Renaming of relations and attributes
- Creating views
- Lots and lots of tests for the WHERE clause
  - IN, EXISTS, BETWEEN, ALL, ANY, LIKE
- Arithmetic and other functions, constant values
- Aggregation functions
  - more on these next time