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 (⋈_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 = 2;

What does this query mean?

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

course	period	room	weekday	hour
TDA357	2	room1	Mon	8
TDA357	2	room1	Thu	8
TDA357	4	room3	Tue	8
TDA357	4	room3	Thu	13
TIN090	1	room4	Mon	8
TIN090	1	room3	Thu	13

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

course	period	room	weekday	hour
TDA357	2	room1	Mon	8
TIN090	1	room4	Mon	8

SELECT teacher

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

WHERE period = 2;

course	period
TDA357	2
TIN090	1

course	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubashi	95



TDA357 2 Niklas Broberg 130 TIN090 1 Devdatt Dubashi 95	course	period	teacher	#students
TIN090 1 Devdatt Dubashi 95	TDA357	2	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 are.
 - 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 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
- ρ = rho = greek letter \mathbf{r} = \mathbf{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
AND A.period <> B.period;

Sequencing

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

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

- In SQL:

```
\rm R_3 AS (SELECT * FROM \rm R_1 , \rm R_2) , \rm R_4 AS (SELECT * FROM \rm R_3 WHERE C) SELECT X FROM \rm R_4 ;
```

```
· Example:
```

```
WITH DBLectures AS

(SELECT room, hour, weekday
FROM Lectures
WHERE course = 'TDA357'
AND period = 2)

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

What does this query mean?

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 = 2;

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

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...)

Testing for existence

• Test whether or not a relation is empty.

```
e.g. List all courses that have lectures.

SELECT code, name
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 second period. (You must use EXISTS...)

```
SELECT code
FROM Courses
WHERE NOT EXISTS
(SELECT *
FROM GivenCourses
WHERE course = code
AND period = 2);
```

Ordinary comparisons

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

```
      value1 BETWEEN value2 AND value3

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

      FROM GivenCourses
      second or third periods.

      WHERE period BETWEEN 2 AND 3;

      - Same thing as

      value2 <= value1 AND value1 <= value3</td>
```

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);
```

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 or 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 * UNKNOWN FROM Rooms WHERE nrSeats > 10 OR nrSeats <= 10; 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.

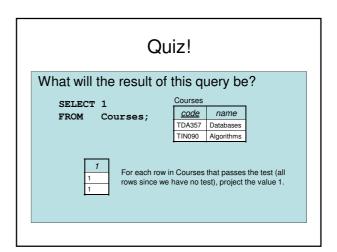
Constants

· Constants can be used in projections.

```
SELECT code, name,
    'Database course' as comment
FROM Courses
WHERE name LIKE '%Database%';

| code | name | comment |
| TDA357 | Databases | Databases course |
```

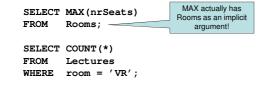
Beware of keywords...



Aggregation

• Aggregation functions are functions that produce a single value over a relation.

- SUM, MAX, MIN, AVG, COUNT...



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

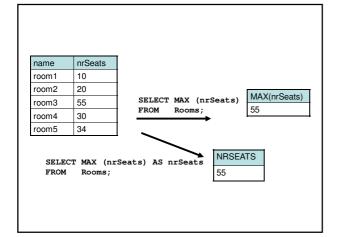
· Compare the following:

SELECT nrSeats FROM Rooms;

SELECT MAX(nrSeats) FROM Rooms;

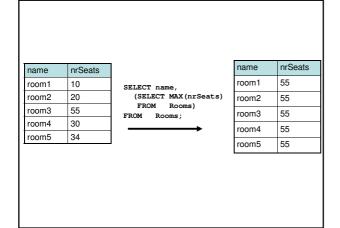
- The ordinary selection/projection results in a relation with a single attribute nrSeats, and one row for each row in Rooms.
- The aggregation results in a single value, not a relation.
- We can't mix both kinds in the same query! (almost...more on this later)

coom2 20 coom3 55 coom4 30 SELECT nrSeats 55 FROM Rooms; 55 30	name	nrSeats		nrSeats
oom3 55 FROM Rooms; 55 30	room1	10		10
oom4 30 30	room2	20	SELECT nrSeats	20
	room3	55	FROM Rooms;	55
oom5 34 34	room4	30	─	30
	room5	34		34



Quiz! New attempt List the room(s) with the highest number of seats, and its number of seats. SELECT name. (SELECT MAX(nrSeats) 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...



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.

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.

Grouping

- Grouping intuitively means to partition a relation into several groups, based on the value of some attribute(s).
 - "All courses with this teacher go in this group, all courses with that teacher go in that group, ..."
- Each group is a sub-relation, and aggregations can be computed over them.
- Within each group, all rows have the same value for the attribute(s) grouped on, and therefore we can project that value as well!

Grouping

Grouping = given a relation R, a set of attributes X, and a set of aggregation expressions G; partition R into groups R₁...R_n such that all rows in R_i have the same value on all attributes in X, and project X and G for each group.

- "For each X, compute G"
- $\gamma = gamma = greek letter$ **g**=**g**rouping

Example: List the average number of students that each teacher has on his or her courses.

course	<u>per</u>	teacher	nrSt.
TDA357	4	Rogardt Heldal	130
TDA590	2	Rogardt Heldal	70
TIN090	1	Devdatt Dubhashi	62

SQL?

Result?

Relational Algebra?

Specialized renaming of attributes

 General renaming operator, rename R to A and its attributes to X:

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

 More convenient alternative for grouping, rename the result of expression G to B:

$$\gamma_{X,G\to B}(R)$$

- e.g. $\gamma_{teacher, AVG(nrStudents) \rightarrow avgStudents}(Given Courses)$
- Works in normal projection (π) as well.

Summary – grouping and aggregation

- Aggregation functions: MAX, MIN, COUNT, AVG, SUM
 - Compute a single value over a whole relation, or a partition of a relation (i.e. a group).
 - If no grouping attributes are given, the aggregation affects the whole relation (and no ordinary attributes can be projected).
- Can't put aggregation directly in the WHERE clause (since it's not a function on values).
- Can't mix aggregation and normal projection!
 - If an aggregation function is used in the SELECT clause, then
 the only other things that may be used there are other
 aggregation functions, and attributes that are grouped on.

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