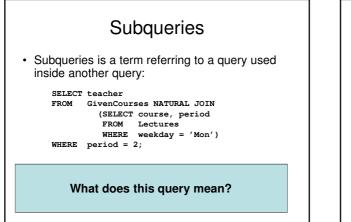
Database Usage (and Construction)

SQL Queries and Relational Algebra Views

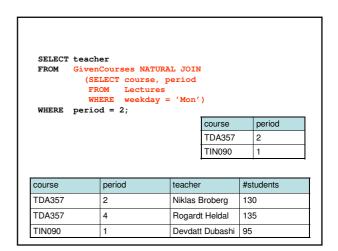
Summary so far

- 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 (🍂 , 🕅

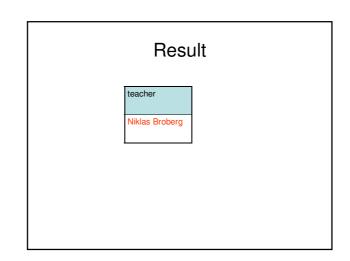


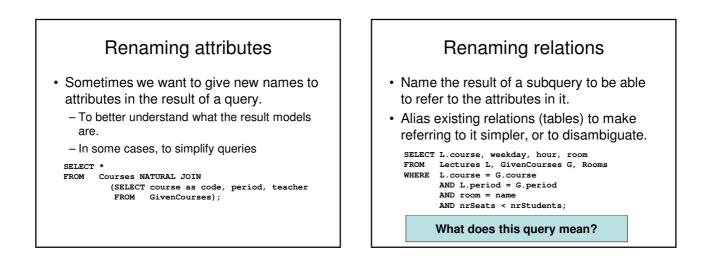
FROM					
course	period	room	weekday	hour	
TDA357	2	room1	Mon	8	
TDA357	2	room1	Thu	8	
TDA357	4	room3	Tue	8	
TDA357	4	room3	Thu	13	
TIN090	1	room4	Mon	8	
TIN090	1	room3	Thu	13	

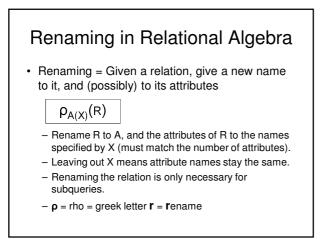
	ectures eekday = '	Mon'		
course	period	room	weekday	hour
TDA357	2	room1	Mon	8
TIN090	1	room4	Mon	8



FROM	FROM Lec	irse, period	
course	period	teacher	#students
course TDA357	period 2		#students 130









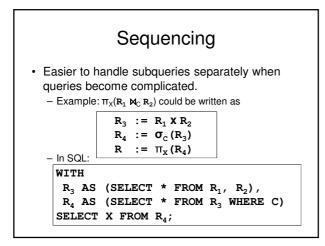
Write a query that lists all courses that are given in more than one period, with different teachers.

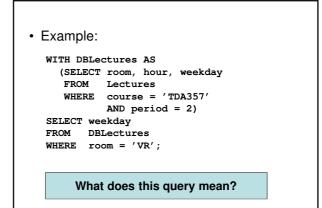
```
SELECT A.course

FROM GivenCourses A, GivenCourses B

WHERE A.course = B.course

AND A.teacher <> B.teacher;
```





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

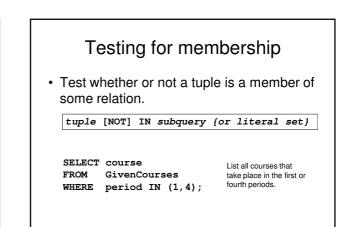
• For purposes of querying, a view works just like a table. The main difference is that you can't perform modifications on it – its contents is defined by other tables.

Example:

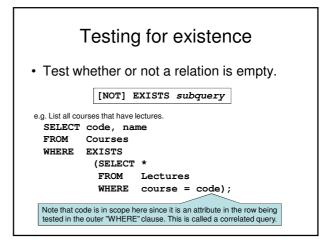
```
CREATE VIEW DBLectures AS
SELECT room, hour, weekday
FROM Lectures
WHERE course = 'TDA357'
AND period = 2;
SELECT weekday
FROM DBLectures
WHERE room = 'VR';
```

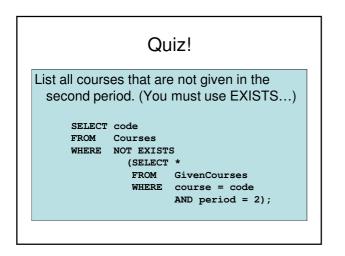
The WHERE clause

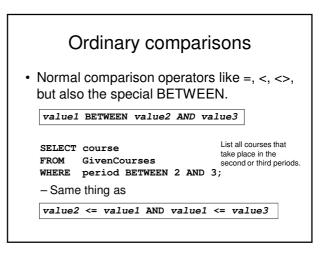
- Specify conditions over rows.
- Can involve
 - constants
 - attributes in the row
 - simple value functions (e.g. ABS, UPPER)
 - subqueries
- · Lots of nice tests to make...



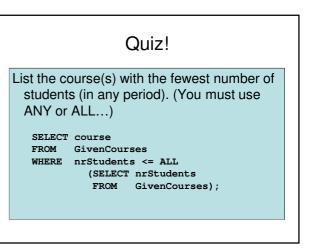
Quiz! List all courses given by a teacher who also gives the Databases course (TDA357). (You must use IN...) SELECT course FROM GivenCourses WHERE teacher IN (SELECT teacher FROM GivenCourses WHERE course = 'TDA357');







Comparisons with many rows • Two operators that let us compare with all the values in a relation at the same time. $tuple op ANY subquery \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ \{ or \ literal \ set \} \\tuple op ALL \ subquery \ subq$



String comparisons

- Normal comparison operators like < use lexicographical order.
 – 'foo' < 'fool' < 'foul'
- Searching for patterns in strings:
 - Two special pattern characters:
 - \bullet _ (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

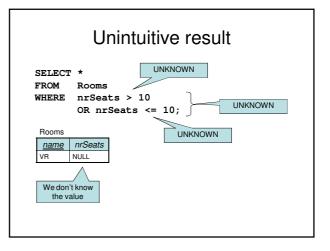
- Use with care!
 - Comparisons and other operations won't work.
 - May take up unnecessary space.

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

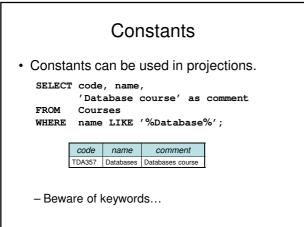


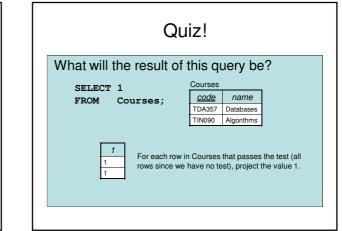
Don't expect the "usual" results

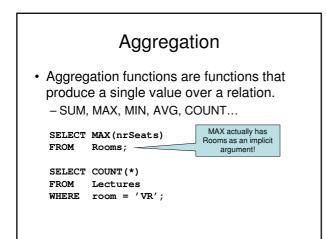
- Laws of three-valued logic are not the same as those for two-valued logic.
- Some laws hold, like commutativity of AND and OR.
- Others do not:
 p OR NOT p = true

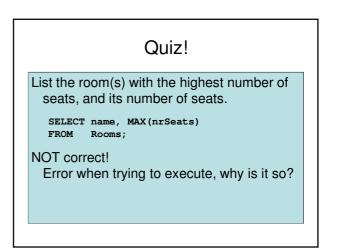
Arithmetic in queries • We allow arithmetic operations in queries. SELECT weekday, hour, room, course, nrSeats - nrStudents as nrFreeSeats FROM Rooms, (Lectures NATURAL JOIN GivenCourses) WHERE name = room; • Not just arithmetic, but rather any operations on values.

- Oracle has lots of pre-defined functions.









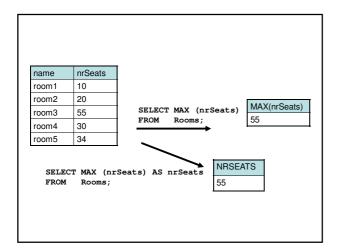
Aggregate functions are special

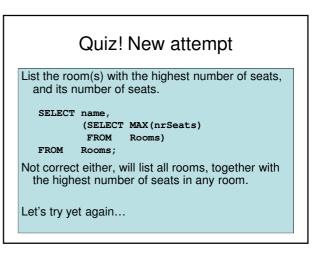
· Compare the following:

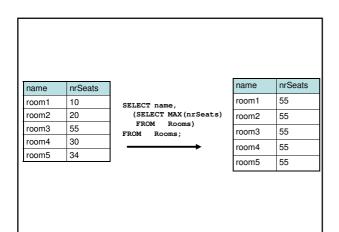
SELECT nrSeats FROM Rooms; SELECT MAX(nrSeats) FROM Rooms;

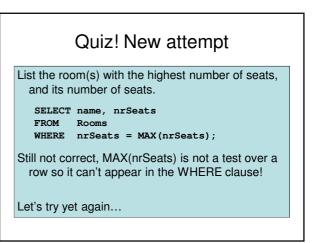
- The ordinary selection/projection results in a relation with a single attribute nrSeats, and one row for each row in Rooms.
- The aggregation results in a single value, not a relation.
- We can't mix both kinds in the same query!

name	nrSeats	SELECT nrSeats	nrSeats
room1	10		10
room2	20		20
room3	55	FROM Rooms;	55
room4	30	\longrightarrow	30
room5	34		34

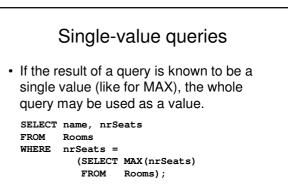








Quiz! List the room(s) with the highest number of seats, and its number of seats. SELECT name, nrSeats FROM Rooms WHERE nrSeats = (SELECT MAX(nrSeats) FROM Rooms); That's better!



• 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

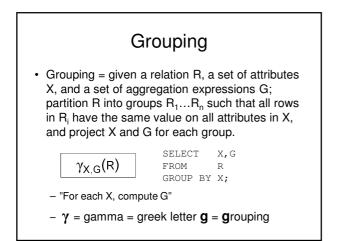
- Aggregation functions: MAX, MIN, COUNT, AVG, SUM
- Compute a single value over a whole relation.
- Can't put aggregation directly in the WHERE clause (since it's not a function on values).
- Can't mix aggregation and normal projection! ... well, not quite true...

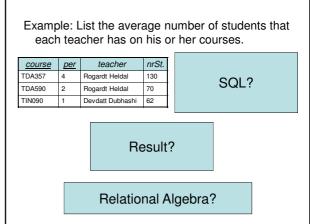
Not quite true?

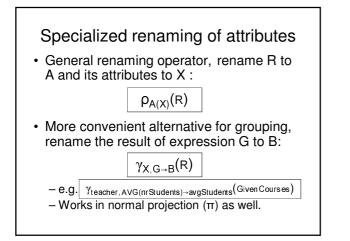
- Sometimes we want to compute an aggregation for every value of some other attribute.
 - Example: List the average number of students that each teacher has on his or her courses.
 - To write a query for this, we must compute the averaging aggregation for each value of teacher.

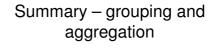
Grouping

- 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!









- 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