

# Database Usage (and Construction)

SQL Queries and Relational Algebra  
Views

# 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:  
 $p \text{ OR } \text{NOT } p = \text{true}$



# 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%';
```

<i>code</i>	<i>name</i>	<i>comment</i>
TDA357	Databases	Database course

# Quiz!

What will the result of this query be?

```
SELECT 1
FROM    Courses;
```

Courses

<u>code</u>	<i>name</i>
TDA357	Databases
TIN090	Algorithms

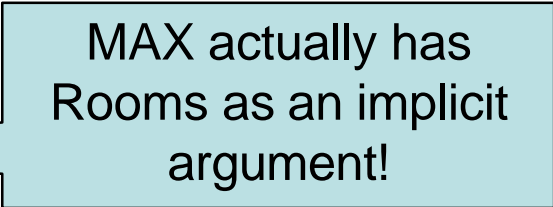
1
1
1

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)

<i>name</i>	<i>nrSeats</i>
HC1	105
HC2	115
VR	230
HA1	146
HA4	152

```
SELECT nrSeats  
FROM Rooms;
```



<i>nrSeats</i>
105
115
230
146
152

<i>name</i>	<i>nrSeats</i>
HC1	105
HC2	115
VR	230
HA1	146
HA4	152

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

<i>MAX(nrSeats)</i>
230

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

<i>nrSeats</i>
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...

<i>name</i>	<i>nrSeats</i>
HC1	105
HC2	115
VR	230
HA1	146
HA4	152

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



<i>name</i>	<i>nrSeats</i>
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.

Next time, Lecture 7

More Relational Algebra and SQL