Lecture 8

# Database Usage (and Construction)

More SQL Queries and Relational Algebra

## 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.**

## 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 |

## 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.

## Capacity per campus?

| *name* | *capacity* | *campus* |
|---|---|---|
| HB2 | 186 | Johanneberg |
| HC1 | 105 | Johanneberg |
| HC2 | 115 | Johanneberg |
| Jupiter44 | 64 | Lindholmen |
| Svea239 | 60 | Lindholmen |
| VR | 300 | Neverland |

| *name* | *capacity* | *campus* |
|---|---|---|
| HB2 | 186 | Johanneberg |
| HC1 | 105 (406) | Johanneberg |
| HC2 | 115 | Johanneberg |
| Jupiter44 | 64 (124) | Lindholmen |
| Svea239 | 60 | Lindholmen |
| VR | 300 | Neverland |

| *SUM(capacity)* | *campus* |
|---|---|
| 406 | Johanneberg |
| 124 | Lindholmen |
| 300 | Neverland |

```
SELECT SUM(capacity), campus FROM Rooms GROUP BY campus;
```

## 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 $R_1 \ldots R_n$ such that all rows in $R_i$ have the same value on all attributes in X, and project X and G for each group.

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

```
SELECT   X,G
FROM     R
GROUP BY X;
```

  - ”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* | *per* | *teacher* | *nrSt.* |
|---|---|---|---|
| TDA357 | 2 | Mickey | 130 |
| DIT952 | 3 | Mickey | 70 |
| TIN090 | 1 | Tweety | 62 |

SQL?

Result?

Relational Algebra?

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

| course | per | teacher | nrSt. |
|---|---|---|---|
| TDA357 | 2 | Mickey | 130 |
| DIT952 | 3 | Mickey | 70 |
| TIN090 | 1 | Tweety | 62 |

SQL?

| teacher | AVG(nrSt.) |
|---|---|
| Mickey | 100 |
| Tweety | 62 |

Relational Algebra?

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

| course | per | teacher | nrSt. |
|---|---|---|---|
| TDA357 | 2 | Mickey | 130 |
| DIT952 | 3 | Mickey | 70 |
| TIN090 | 1 | Tweety | 62 |

```
SELECT teacher,
  AVG(nrStudents)
FROM GivenCourses
GROUP BY teacher;
```

| teacher | AVG(nrSt.) |
|---|---|
| Mickey | 100 |
| Tweety | 62 |

Relational Algebra?

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

| course | per | teacher | nrSt. |
|---|---|---|---|
| TDA357 | 2 | Mickey | 130 |
| DIT952 | 3 | Mickey | 70 |
| TIN090 | 1 | Tweety | 62 |

```
SELECT teacher,
  AVG(nrStudents)
FROM GivenCourses
GROUP BY teacher;
```

| teacher | AVG(nrSt.) |
|---|---|
| Mickey | 100 |
| Tweety | 62 |

$\gamma_{\text{teacher, AVG(nrStudents)}}(\text{GivenCourses})$

## Specialized renaming of attributes

- We've seen the general renaming operator already: $\rho_{A(X)}(R)$
  - Rename R to A and its attributes to X.
- Can be akward to use, so we are allowed an easier way to rename attributes:

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

  - E.g. $\gamma_{\text{teacher, AVG(nrStudents)}\rightarrow\text{avgStudents}}(\text{GivenCourses})$
  - Works in normal projection (π) as well.

## Tests on groups

- Aggregations can't be put in the WHERE clause – they're not functions on rows but on groups.
- Sometimes we want to perform tests on the result of an aggregation.
  - Example: List all teachers who have an average number of students of >100 in their courses.
- SQL allows us to put such tests in a special HAVING clause after GROUP BY.

## Example

```
SELECT   teacher
FROM     GivenCourses
GROUP BY teacher
HAVING   AVG(nrStudents) > 100;
```

| code | period | teacher | #students |
|---|---|---|---|
| TDA357 | 2 | Mickey | 130 |
| ~~TIN090~~ | ~~1~~ | ~~Tweety~~ | ~~95~~ |
| TDA357 | 3 | Donald | 135 |
| TDA283 | 2 | Donald | 70 |

| AVG(nrSt.) |
|---|
| 130 |
| 95 |
| 102.5 |

## Quiz!

- There is no correspondence in relational algebra to the HAVING clause of SQL. Why?
  - Because we can express it with an extra renaming and a selection. Example:

```
SELECT   teacher
FROM     GivenCourses
GROUP BY teacher
HAVING   AVG(nrStudents) > 100;
```

$\sigma_{avgSt > 100}(\gamma_{teacher,\ AVG(nrStudents) \rightarrow avgSt}(GivenCourses))$

## Sorting relations

- Relations are unordered by default.
- Operations could potentially change any existing ordering.

$\tau_X(R)$ `ORDER BY X [DESC]`

  - Sort relation R on attributes X.
  - Ordering only makes sense at the top level, or if only a given number of rows are sought, e.g. the top 5.
  - (For top 5: Append "LIMIT 5")
- $\tau$ = tau = greek letter t = sort (s is taken)

## Example

```
SELECT   *
FROM     Courses
ORDER BY name;
```

| *code* | *name* |
|---|---|
| TIN090 | Algorithms |
| TDA590 | Compiler Construction |
| TDA357 | Databases |

## SELECT-FROM-WHERE-GROUPBY-HAVING-ORDERBY

- Full structure of an SQL query:

```
SELECT   attributes
FROM     tables
WHERE    tests over rows
GROUP BY attributes
HAVING   tests over groups
ORDER BY attributes
```

Only the SELECT and FROM clauses must be included.

```
SELECT   X,G
FROM     T
WHERE    C
GROUP BY Y
HAVING   D
ORDER BY Z;
```

⇔ **What?**

## SELECT-FROM-WHERE-GROUPBY-HAVING-ORDERBY

- Full structure of an SQL query:

```
SELECT   attributes
FROM     tables
WHERE    tests over rows
GROUP BY attributes
HAVING   tests over groups
ORDER BY attributes
```

Only the SELECT and FROM clauses must be included.

```
SELECT   X,G
FROM     T
WHERE    C
GROUP BY Y
HAVING   D
ORDER BY Z;
```

⇔ $\tau_{Z'}(\pi_{X,G'}(\sigma_{D'}(\gamma_{Y,G'}(\sigma_C(T)))))$

X must be a subset of Y.
Primes ' mean we need some renaming.

## Example:

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

Courses

| *code* | *name* |
|---|---|
| TDA357 | Databases |
| TIN090 | Algorithms |

GivenCourses

| *course* | *per* | *teacher* | *nrSt* |
|---|---|---|---|
| TDA357 | 2 | Mickey | 130 |
| TDA357 | 3 | Donald | 95 |
| TIN090 | 1 | Tweety | 62 |

$\tau_{avSt}(\pi_{name,\ avSt}(\sigma_{avSt > 100}(\gamma_{code,\ name,\ AVG(nrStudents) \rightarrow avSt}(\sigma_{code = course}(Courses \times GivenCourses)))))$

## Example:

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

| code | name | course | per | teacher | nrSt |
|---|---|---|---|---|---|
| TDA357 | Databases | TDA357 | 2 | Mickey | 130 |
| TDA357 | Databases | TDA357 | 3 | Donald | 95 |
| TDA357 | Databases | TIN090 | 1 | Tweety | 62 |
| TIN090 | Algorithms | TDA357 | 2 | Mickey | 130 |
| TIN090 | Algorithms | TDA357 | 3 | Donald | 95 |
| TIN090 | Algorithms | TIN090 | 1 | Tweety | 62 |

$\tau_{avSt}(\pi_{name,avSt}(\sigma_{avSt>100}(\gamma_{code,name,AVG(nrStudents)\rightarrow avSt}(\sigma_{code=course}(\textbf{Courses x GivenCourses})))))$

## Example:

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

| code | name | course | per | teacher | nrSt |
|---|---|---|---|---|---|
| TDA357 | Databases | TDA357 | 2 | Mickey | 130 |
| code | name | course | per | teacher | nrSt |
| TDA357 | Databases | TDA357 | 2 | Mickey | 130 |
| TDA357 | Databases | TDA357 | 3 | Donald | 95 |
| TIN090 | Algorithms | TIN090 | 1 | Tweety | 62 |
| TIN090 | Algorithms | TIN090 | 1 | Tweety | 62 |

$\tau_{avSt}(\pi_{name,avSt}(\sigma_{avSt>100}(\gamma_{code,name,AVG(nrStudents)\rightarrow avSt}(\textbf{}\sigma_{code=course}(\text{Courses x GivenCourses})))))$

## Example:

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

| code | name | course | per | teacher | nrSt | AVG(nrSt) |
|---|---|---|---|---|---|---|
| TDA357 | Databases | TDA357 | 2 | Mickey | 130 | 112.5 |
| TDA357 | Databases | TDA357 | 3 | Donald | 95 | |
| TIN090 | Algorithms | TIN090 | 1 | Tweety | 62 | 62 |

$\tau_{avSt}(\pi_{name,avSt}(\sigma_{avSt>100}(\textbf{}\gamma_{code,name,AVG(nrStudents)\rightarrow avSt}(\sigma_{code=course}(\text{Courses x GivenCourses})))))$

## Example:

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

| code | name | AVG(nrSt) |
|---|---|---|
| TDA357 | Databases | 112.5 |
| TIN090 | Algorithms | 62 |

$\tau_{avSt}(\pi_{name,avSt}(\textbf{}\sigma_{avSt>100}(\gamma_{code,name,AVG(nrStudents)\rightarrow avSt}(\sigma_{code=course}(\text{Courses x GivenCourses})))))$

## Example:

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

| code | name | avSt | (nrSt) |
|---|---|---|---|
| TDA357 | Databases | 112.5 | 12.5 |

$\tau_{avSt}(\textbf{}\pi_{name,avSt}(\sigma_{avSt>100}(\gamma_{code,name,AVG(nrStudents)\rightarrow avSt}(\sigma_{code=course}(\text{Courses x GivenCourses})))))$

## Example:

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

| name | avSt |
|---|---|
| Databases | 112.5 |

$\textbf{}\tau_{avSt}(\pi_{name,avSt}(\sigma_{avSt>100}(\gamma_{code,name,AVG(nrStudents)\rightarrow avSt}(\sigma_{code=course}(\text{Courses x GivenCourses})))))$

# Break

## Why not simply this?

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course, avSt > 100
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

Because at the time of "WHERE",
aggregates have not been computed yet!

Remember: If "GROUP BY" is used, then aggregates are computed over each "GROUP BY" group, not over all entries

```
SELECT   name, AVG(nrStudents) AS avSt
FROM     Courses, GivenCourses
WHERE    code = course,
GROUP BY code, name
HAVING   AVG(nrStudents) > 100
ORDER BY avSt;
```

What about this then!?!?

## Lexical vs logical ordering

- Lexical order: the way it's written in SQL
- Logical order: the way the query executes

**Lexical order**

```
SELECT   attributes
FROM     tables
WHERE    tests over rows
GROUP BY attributes
HAVING   tests over groups
ORDER BY attributes
```

**Logical order**

```
FROM     tables
WHERE    tests over rows
GROUP BY attributes
HAVING   tests over groups
SELECT   "attributes"
ORDER BY attributes
```

## Available attributes in SELECT

- Aggregate functions "summarize" values per group
  - Without GROUP BY, the group is the entire table
- If aggregate functions are used, then only attributes can be selected that make sense in a grouping

```
SELECT   campus, MAX(capacity)
FROM     Rooms
```

**Invalid! Group = table, MAX returns 1 value, but 3 different campuses**

```
SELECT   MAX(capacity)
FROM     Rooms
```

**Valid! Group = table, MAX returns 1 value**

```
SELECT   campus, MAX(capacity)
FROM     Rooms
GROUP BY campus
```

**Valid! Grouped per campus, MAX returns 1 value per campus, there is 1 campus name per group**

# SQL Exercises

## Music Website

**Tracks**(trackId,title, length)
  length > 0

**Artists**(artistId, name)

**Albums**(albumId,title, yearReleased)

**TracksOnAlbum**(album,trackNr,track)
  album -> Albums.albumId
  track -> Tracks.trackId
  (album,track) unique
  trackNr > 0

**Participates**(track, artist)
  track -> Tracks.trackId
  artist -> Artists.artistId

**Users**(username, email, name)
  email unique

**Playlists**(user, playlistName)
  user -> Users.username

**InList**(user, playlist, number,track)
  (user, playlist) -> Playlists.(user, playlistName)
  track -> Tracks.trackId

**PlayLog**(user,time,track)
  user -> Users.username
  track -> Tracks.trackId
  (user,time) unique

## Music Website – Ex1

- Write an SQL query that lists all artists appearing on any album released from 2016

```
SELECT *
FROM Artists
WHERE artistId IN (
    SELECT artist
    FROM Participates WHERE track IN (
       SELECT track
       FROM TracksOnAlbum
       WHERE album IN (
              SELECT albumId
              FROM Albums
              WHERE yearReleased >= '1/1/2016')
    )
);
```

## Music Website – Ex2

- Write an SQL query that lists, for each user, how many playlists that user has.

```
SELECT username, COUNT(playlistname)
FROM PlayLists
GROUP BY (username);
```

## Music Website – Ex3

- Write an SQL query that lists, for each track, its ``trackId`` and title, together with the number of times that track has been played, and the number of distinct users that have played it.

```
SELECT trackId, title, COUNT(username) AS timesplayed,
   COUNT(DISTINCT username) AS differentUsers
FROM PlayLog
NATURAL JOIN Tracks
GROUP BY trackId, title;
```

## Next time, Lecture 9

More on SQL and Relational Algebra