Lecture 7

Database Usage (and Construction)

More SQL Queries and Relational Algebra

Previously…

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';
```

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”
  - γ = 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	Niklas Broberg	130
DIT952	3	Niklas Broberg	70
TIN090	1	Devdatt Dubhashi	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	Niklas Broberg	130
DIT952	3	Niklas Broberg	70
TIN090	1	Devdatt Dubhashi	62

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

teacher	AVG(nrSt.)
Niklas Broberg	100
Devdatt Dubhashi	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	Niklas Broberg	130
~~TIN090~~	~~1~~	~~Devdatt Dubhashi~~	~~95~~
TDA357	3	Aarne Ranta	135
TDA283	2	Aarne Ranta	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;
```

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

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")
- τ = 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	Niklas Broberg	130
TDA357	3	Aarne Ranta	95
TIN090	1	Devdatt Dubhashi	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	Niklas Broberg	130
TDA357	Databases	TDA357	3	Aarne Ranta	95
TDA357	Databases	TIN090	1	Devdatt Dubhashi	62
TIN090	Algorithms	TDA357	2	Niklas Broberg	130
TIN090	Algorithms	TDA357	3	Aarne Ranta	95
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62

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

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	Niklas Broberg	130
TDA357	Databases	TDA357	3	Aarne Ranta	95
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62

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

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	AVG(nrSt)
TDA357	Databases	TDA357	2	Niklas Broberg	130	112.5
TDA357	Databases	TDA357	3	Aarne Ranta	95	
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62	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	AVG(nrSt)
TDA357	Databases	112.5
TIN090	Algorithms	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	avSt	AVG(nrSt)
TDA357	Databases	112.5	112.5

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

Example:

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

name	avSt
Databases	112.5

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

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

BREAK

Relations as sets

- Relations are sets of tuples.
- Set theory has plenty to borrow from:
  - Some we've seen, like ∈ (IN).
  - More operators:
    - ∪ (union)
    - ∩ (intersection)
    - ∖ (set difference)

Set operations

- Common set operations in SQL
  - UNION: Given two relations R_1 and R_2, add them together to form one relation $R_1 \cup R_2$.
  - INTERSECT: Given two relations R_1 and R_2, return all rows that appear in both of them, forming $R_1 \cap R_2$.
  - EXCEPT: Given two relations R_1 and R_2, return all rows that appear in R_1 but not in R_2, forming $R_1 \setminus R_2$.
- All three operations require that R_1 and R_2 have (almost) the same schema.
  - Attribute names may vary, but number, order and types must be the same.

Quiz!

List all courses and the periods they are given in. Courses that are not scheduled for any period should also be listed, but with NULL in the field for period. You must use a set operation.

```
(SELECT course, period
 FROM   GivenCourses)
UNION
(SELECT code, NULL
 FROM   Courses
 WHERE  code NOT IN
        (SELECT course
         FROM   GivenCourses));
```

```
(SELECT course, period
 FROM   GivenCourses)
UNION
(SELECT code, NULL
 FROM   Courses
 WHERE  code NOT IN
     (SELECT course
      FROM GivenCourses));
```

code	*name*
TIN090	Algorithms
TDA283	Compiler Construction
TDA357	Databases
TDA100	AI

course	*period*	*teacher*	*#students*
TDA357	2	Niklas Broberg	130
TDA357	3	Aarne Ranta	135
TIN090	1	Devdatt Dubhashi	95
TDA283	2	Aarne Ranta	70

```
(SELECT course, period
 FROM   GivenCourses)
UNION
(SELECT code, NULL
 FROM   Courses
 WHERE  code NOT IN
     (SELECT course
      FROM GivenCourses));
```

course	period
TDA357	2
TDA357	3
TIN090	1
TDA283	2

U

code	NULL
TDA100	Null

Result

course	period
TDA357	3
TDA357	4
TIN090	1
TDA283	2
TDA100	

Not sets but bags!

- In set theory, a set cannot contain duplicate values. Either a value is in the set, or it's not.
- In SQL, results of queries can contain the same tuples many times.
  - Done for efficiency, eliminating duplicates is costly.
- A set where duplicates may occur is called a *bag*, or *multiset*.

Controlling duplicates

- Queries return bags by default. If it is important that no duplicates exist in the set, one can add the keyword DISTINCT.
  - Example:

```
SELECT DISTINCT teacher
FROM   GivenCourses;
```

- DISTINCT can also be used with aggregation functions.
  - Example:

```
SELECT COUNT(DISTINCT teacher)
FROM   GivenCourses;
```

course	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	3	Aarne Ranta	135
TIN090	1	Devdatt Dubhashi	95
TDA283	2	Aarne Ranta	70

```
SELECT teacher
FROM   GivenCourses;
```

teacher
Niklas Broberg
Aarne Ranta
Devdatt Dubhashi
Aarne Ranta

course	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	3	Aarne Ranta	135
TIN090	1	Devdatt Dubhashi	95
TDA283	2	Aarne Ranta	70

```
SELECT DISTINCT teacher
FROM   GivenCourses;
```

teacher
Niklas Broberg
Aarne Ranta
Devdatt Dubhashi

course	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	3	Aarne Ranta	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Aarne Ranta	70

```
SELECT COUNT (teacher)
FROM   GivenCourses;
```

COUNT(teacher)
4

course	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	3	Aarne Ranta	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Aarne Ranta	70

```
SELECT COUNT (DISTINCT teacher)
FROM   GivenCourses;
```

COUNT (DISTINCT teacher)
3

Duplicate elimination

- Duplicate elimination = Given relation R, remove all duplicate rows.

  $\delta(R)$

  – Remove all duplicate rows from R.

```
SELECT DISTINCT X
FROM   R
WHERE  C;
```

$\delta(\pi_X(\sigma_C(R)))$

- δ = delta = greek letter d = duplicate elimination

Retaining duplicates

- Set operations eliminate duplicates by default.
  - For pragmatic reasons – to compute either intersection or set difference efficiently, the relations need to be sorted, and then eliminating duplicates comes for free.
- If it is important that duplicates are considered, one can add the keyword ALL.
  - Example:

```
(SELECT room
 FROM   Lectures)
EXCEPT ALL
(SELECT name
 FROM   Rooms);
```

All rooms appear once in Rooms. The set difference will remove each room once from the first set, thus leaving those rooms that have more than one lecture in them.

Common idiom

List all courses and the periods they are given in. Courses that are not scheduled for any period should also be listed, but with NULL in the field for period. You must use a set operation.

```
(SELECT code, period
 FROM   Courses, GivenCourses
 WHERE  code = course)
UNION
(SELECT code, NULL
 FROM   Courses
 WHERE  code NOT IN
          (SELECT course
           FROM   GivenCourses));
```

First compute those that fit in the join, then union with those that don't.

Outer join

- Compute the join as usual, but retain all tuples that don't fit in from either or both operands, padded with NULLs.

  $R_1 \overset{\circ}{\bowtie} R_2$

```
SELECT *
FROM
 R1 NATURAL FULL OUTER JOIN R2;
```

  – FULL means retain all tuples from both operands. LEFT or RIGHT retains only those from one of the operands.
  – Can be used with ordinary join as well.
    - R_1 LEFT OUTER JOIN R_2 ON C;

Quiz!

List all courses and the periods they are given in. Courses that are not scheduled for any period should also be listed, but with NULL in the field for period.

```
SELECT code, period
FROM   Courses LEFT OUTER JOIN GivenCourses
                 ON code = course;
```

```
SELECT code, period
FROM   Courses
     LEFT OUTER JOIN
       GivenCourses
      ON code = course;
```

code	*name*
TIN090	Algorithms
TDA283	Compiler Construction
TDA357	Databases
TDA100	AI

course	*period*	*teacher*	*#students*
TDA357	2	Niklas Broberg	130
TDA357	3	Aarne Ranta	135
TIN090	1	Devdatt Dubhashi	95
TDA283	2	Aarne Ranta	70

```
SELECT code, period
FROM   Courses
     LEFT OUTER JOIN
       GivenCourses
      ON code = course;
```

code	*period*
TDA357	3
TDA357	4
TIN090	1
TDA283	2
TDA100	Null

Summary
SQL and Relational Algebra

- SQL is based on relational algebra.
  - Operations over relations
- SELECT-FROM-WHERE-GROUPBY-HAVING-ORDERBY
- Operations for:
  - Selection of rows (σ)
  - Projection of columns (π)
  - Combining tables
    - Cartesian product (x)
    - Join, natural join, outer join ($\bowtie_C$, $\bowtie$, $\bowtie$)
  - Grouping and aggregation
    - Grouping (γ)
    - SUM, AVG, MIN, MAX, COUNT
  - Set operations
    - Union (∪)
    - Intersect (∩)
    - Set difference (\)
  - Miscellaneous
    - Renaming (ρ)
    - Duplicate elimination (δ)
    - Sorting (τ)
- Subqueries
  - Sequencing
  - (Views)

Next time, Lecture 8

More on Modifications and Table Creation
Assertions
Triggers