Database Usage (and Construction)

More SQL Queries and Relational Algebra

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.

Quiz!

List all teachers who have an average number of students of >100 in their courses.

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

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	4	Rogardt Heldal	135
TDA590	2	Rogardt Heldal	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;
```

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

Sorting relations

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

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

  - 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.
  - Oracle: Use the implicit attribute `rownum` to limit how many rows should be used.
- τ = tau = greek letter t = sort (s is taken)

Example

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

code	name
TIN090	Algorithms
TDA357	Databases
TDA590	OOSD

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;
```

$\Leftrightarrow \quad \tau_{Z'}(\pi_{X,G'}(\sigma_{D'}(\gamma_{Y,G'}(\sigma_{C}(T)))))$

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	4	Rogardt Heldal	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	4	Rogardt Heldal	95
TDA357	Databases	TIN090	1	Devdatt Dubhashi	62
TIN090	Algorithms	TDA357	2	Niklas Broberg	130
TIN090	Algorithms	TDA357	4	Rogardt Heldal	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	4	Rogardt Heldal	95
TDA357	Databases	TIN090	1	Devdatt Dubhashi	62
TIN090	Algorithms	TDA357	2	Niklas Broberg	130
TIN090	Algorithms	TDA357	4	Rogardt Heldal	95
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62

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

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	4	Rogardt Heldal	95	
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62	62

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

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

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

code	name	AVG(nrSt)
TDA357	Databases	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;
```

code	name	AVG(nrSt)
TDA357	Databases	112.5

name	avSt
Databases	112.5

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

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

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

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 $R_1 \setminus R_2$.
    - Oracle calls this operation MINUS.
- 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 code, period
 FROM   GivenCourses)
UNION
(SELECT code, NULL
 FROM   Courses
 WHERE  code NOT IN
     (SELECT code
      FROM   GivenCourses));
```

code	*name*
TIN090	Algorithms
TDA590	OOS
TDA357	Databases
TDA100	AI

code	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	70

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

code	period
TDA357	2
TDA357	4
TIN090	1
TDA590	2

U

code	*NULL*
TDA100	Null

Result

code	period
TDA357	2
TDA357	4
TIN090	1
TDA590	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;
```

code	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	70

↓

```
SELECT teacher
FROM   GivenCourses;
```

teacher
Niklas Broberg
Rogardt Heldal
Devdatt Dubhashi
Rogardt Heldal

code	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	70

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

teacher
Niklas Broberg
Rogardt Heldal
Devdatt Dubhashi

code	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	70

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

COUNT(teacher)
4

```
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 duplicates 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:

Doesn't work in Oracle, there ALL only works for UNION.

```
(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.

Summary – relations as sets

- Set operations can be used on relations
  - Requires the operands to have the same arity (number of attributes) and types must match.
    - UNION
    - INTERSECT
    - EXCEPT (MINUS)
- Relations are treated as bags in most queries, but as sets in the result of a set operation.
  - To eliminate duplicates, use DISTINCT.
  - To retain duplicates for set operations, use ALL.

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.

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 ($\bowtie_C$, $\bowtie$, ⟗)
  - 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)

Course Objectives – Usage

When the course is through, you should

- Know how to query a database for relevant data using SQL
- Know how to change the contents of a database using SQL

"Add a course 'Databases' with course code 'TDA357', given by …"
"Give me all info regarding the course 'TDA357'"

Exam – Relational Algebra

"Here is a schema for a database over persons and their employments. …"

- What does this relational-algebraic expression compute? …
- Translate this relational-algebraic expression to SQL.
- Write a relational-algebraic expression that computes … .
- Translate this SQL query to a relational-algebraic expression.

Exam – SQL DML

"The grocery store wants your help in getting proper information from their database. …"

- Write a query that finds the total value of the entire inventory of the store.
- List all products with their current price, i.e. the discount price where such exists, otherwise the base price.

Database Construction (and Usage)

More on Modifications and Table Creation
Assertions
Triggers

Summary – Modifications

- Modifying the contents of a database:
  - Insertions

```
INSERT INTO tablename VALUES tuple
```

  - Deletions

```
DELETE FROM tablename WHERE test over rows
```

  - Updates

```
UPDATE tablename
SET    attribute = value
WHERE  test over rows
```

Insertions with queries

- The values to be inserted could be taken from the result of a query:

```
INSERT INTO tablename (query)
```

– Example:

```
INSERT INTO GivenCourses
 (SELECT course, period + 2, teacher, NULL
  FROM   GivenCourses
  WHERE  period <= 2);
```

All courses that are given in periods one and two are also scheduled to be given two periods later, with the same teacher.

Explicit attribute lists

- Attribute order could be given explicit when inserting.
  – Example:

```
INSERT INTO
 GivenCourses(course, period, teacher, nrStudents)
 (SELECT course, period + 2, teacher, NULL
  FROM   GivenCourses
  WHERE  period <= 2);
```

Perhaps the teacher and nrStudents attributes were listed in the other order in the definition of the table? Doesn't matter anymore since they are explicitly listed.

Quiz

What will the following insertion result in?

```
INSERT INTO
 GivenCourses(course, period, teacher)
 VALUES ('TDA357', 3, 'Niklas Broberg');
```

– Attribute lists can be partial. Any attributes not mentioned will be given the value a default value, which by default is NULL.

Default values

- Attributes can be given default values.
  – Specified when a table is defined using the DEFAULT keyword.
  – Example:

```
CREATE TABLE GivenCourses (
  course      CHAR(6),
  period      INT,
  teacher     VARCHAR(50),
  nrStudents  INT DEFAULT 0,
  … constraints …
);
```

  – Default default value is NULL.

Insertion with default values

- Leaving out an attribute in an insertion with explicitly named attributes gives that row the default value for that attribute:

```
INSERT INTO
 GivenCourses(course, period, teacher)
 VALUES ('TDA357', 3, 'Niklas Broberg');
```

- When no attribute list is given, the same effect can be achieved using the DEFAULT keyword:

```
INSERT INTO GivenCourses
 VALUES ('TDA357', 3, 'Niklas Broberg', DEFAULT);
```