Database Usage (and Construction)

More SQL Queries and Relational Algebra

SELECT-FROM-WHERE

- Basic structure of an SQL query:

```
SELECT attributes
FROM   tables
WHERE  tests over rows
```

```
SELECT X
FROM T
WHERE C
```

$\Longleftrightarrow$

$$\pi_X(\sigma_C(T))$$

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 = 'VR';
```

COUNT can be applied to * count the number of rows.

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.

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!

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”
  - $\boldsymbol{\gamma}$ = 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	4	Rogardt Heldal	130
TDA590	2	Rogardt Heldal	70
TIN090	1	Devdatt Dubhashi	62

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

teacher	*AVG(nrSt.)*
Rogardt Heldal	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.

Summary – grouping and aggregation

- 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.

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	3	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;
```

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

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

Example

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

<u>code</u>	*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;
```

⟺

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

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;
```

GivenCourses

course	*per*	*teacher*	*nrSt*
TDA357	3	Niklas Broberg	130
TDA357	2	Graham Kemp	95
TIN090	1	Devdatt Dubhashi	62

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

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

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	3	Niklas Broberg	130
TDA357	Databases	TDA357	2	Graham Kemp	95
TDA357	Databases	TIN090	1	Devdatt Dubhashi	62
TIN090	Algorithms	TDA357	3	Niklas Broberg	130
TIN090	Algorithms	TDA357	2	Graham Kemp	95
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62

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

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	3	Niklas Broberg	130
TDA357	Databases	TDA357	2	Graham Kemp	95
TDA357	Databases	TIN090	1	Devdatt Dubhashi	62
TIN090	Algorithms	TDA357	3	Niklas Broberg	130
TIN090	Algorithms	TDA357	2	Graham Kemp	95
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62

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

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*
TDA357	Databases	TDA357	3	Niklas Broberg	130
TDA357	Databases	TDA357	2	Graham Kemp	95
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62

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

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*
TDA357	Databases	TDA357	3	Niklas Broberg	130
TDA357	Databases	TDA357	2	Graham Kemp	95
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi	62

AVG(nrSt)
112.5
62

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

Example:

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

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

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

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

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

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*	*AVG(nrSt)*
TDA357	Databases	112.5

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

Example:

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

code	*name*	*AVG(nrSt)*
TDA357	Databases	112.5

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

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

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

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

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

Relations as sets

- Relations are sets of tuples.
- Set theory has plenty to borrow from:
  - Some we've seen, like $\in$ (IN).
  - More operators:
    - $\cup$ (union)
    - $\cap$ (intersection)
    - $\setminus$ (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 course, period
 FROM   GivenCourses)
UNION
(SELECT code, NULL
 FROM   Courses
 WHERE  code NOT IN
      (SELECT course
       FROM GivenCourses));
```

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

course	*period*	*teacher*	*#students*
TDA357	3	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	70

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

course	*period*
TDA357	3
TDA357	4
TIN090	1
TDA590	2

U

code	*NULL*
TDA100	Null

Result

course	*period*
TDA357	3
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;
```

course	*period*	*teacher*	*#students*
TDA357	3	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

course	*period*	*teacher*	*#students*
TDA357	3	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

course	*period*	*teacher*	*#students*
TDA357	3	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

course	*period*	*teacher*	*#students*
TDA357	3	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	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 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.

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
TDA590	OOS
TDA357	Databases
TDA100	AI

course	*period*	*teacher*	*#students*
TDA357	3	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	70

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

code	*period*
TDA357	3
TDA357	4
TIN090	1
TDA590	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 ($\bowtie_C$, $\bowtie$, $\mathring{\bowtie}$)
  - 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 (6)

”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 Queries (8)

”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.

Next Lecture

More on Modifications and Table Creation

Assertions

Triggers