IN3020/4020 – Database Systems Spring 2021, Week 3.1-4.1

RELATIONAL ALGEBRA (Parts 1-3)
Calculating with relations

Egor V. Kostylev

Based upon slides by E. Thorstensen and M. Naci Akkøk

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Relational Algebra

- Defines operations on relations (i.e., tables)
- Gives us a language to describe questions (queries) about the contents of relations
- Is a (more) **procedural language**: We say how the answer should be calculated. (The alternative is (more) **declarative** query languages like SQL where we only say what the answer will fulfill)
- It is the theoretical basis for SQL (mostly DDL, data definition language)

Why We Study Relational Algebra?

- It is the most important intermediate step for evaluating SQL queries:
  - First SQL query is first translated to RA expression
  - Then RA expression is compiled & optimized using laws of RA operators (next weeks)
- Translation is straightforward: the counterparts of RA operations exist in SQL (maybe with other names)

Relational Algebra Variants

- There are several variants of Relational Algebra, depending on
  - formalization of relations it works with (named vs. unnamed perspective, sets vs. bags)
  - set of SQL constructs it covers (full SQL is *Turing-complete*!)
- There is also Relational Calculus: mathematically equivalent to RA (Codd's theorem), closer to formal logic

Materials to Read

- Section 8 of the Book (Elmasri & Navathe, «Fundementals of Database Systems»)
- Part B of the Alice Book (Abiteboul, Hull & Vianu, «Foundation of Databases», available at http://webdam.inria.fr/Alice/)
- Many other places, including Wikipedia
- NOTE: I do not follow any of them line by line

Practical Counterparts

- We will use examples from w3resource (https://www.w3resource.com/) or w3schools (https://www.w3schools.com/sql/)
- **w3resource** has tutorials and examples for **2003 standard ANSI SQL** (use that primarily), as well as MySQL, PostgreSQL, Oracle etc., and for NoSQL, GraphQL and others that you will need later in this course (and in life)
- **w3schools** let you “Try it Yourself” that can help understand (the green button)
- There are very many SQL help & tutorials, also on each DBMS´ own site

Algebra (a.k.a. Algebraic Structure)

- **Domain D:** collection of values
- **Operators:** functions from $\mathbf{D}^k$ to **D** (k is arity of the function)
- **Expressions:**
  - **Atomic:** elements of **D**
  - **Complex:** operators applied to other expressions
  - Evaluate to elements of **D**

- Infix notation is often used for binary operators
  - for example, instead of +(a, b) we write a + b
  - brackets or conventions used: (a + b) * c vs. a + b * c

Example: Integer Algebra

- **Domain:** Integers (..., -3, -2, -1, 0, 1, 2, 3, ...)
- **Operators:** $+$, $-$, $\times$, $/$

- **Expression examples:**

  $2 + 5$

  $((2 - 4) \times 5) + (8 / 2)$

  $8 / 3$ (?)

Example: Regular Expressions

- **Domain:** Sets of strings over some alphabet Σ of letters
- **Operators:** Ø (empty set), ε (empty string),
  all a in Σ (letters), ◦ (or nothing, concatenation),
  | (alternation), * (Kleene star)

- **Expression examples:**
  ab*
  (a|b)*ab
- **Expressible operators:** +, ?

Relational Algebra Domain:
Relations (under named attributes perspective)

- **Relation (**or **table)** components:
  - **Relation name**
  - **Relation schema**: set of attribute names with associated datatypes
  - Set(!) of **Relation records:** tuples of elements conforming the schema

  - Example:

Tutorials

ID	site	tutorial	topic
1	w3schools	SQL_2003STD	Database
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Database
5	w3resource	MySQL	Database

Core Relational Algebra

- **Domain:** Relations
- **(Main) operators:**
  1. **(Set) Union**
  2. **(Set) Difference**
  3. **Projection**
  4. **Selection**
  5. **Cartesian product** (a.k.a., Cross Product and Cross Join)
  6. **Renaming**

- **Expressible operators:**
  1. (Set) Intersection
  2. Other joins (natural, equi-, left, etc.)
  3. Division
  4. etc.

Set Operations

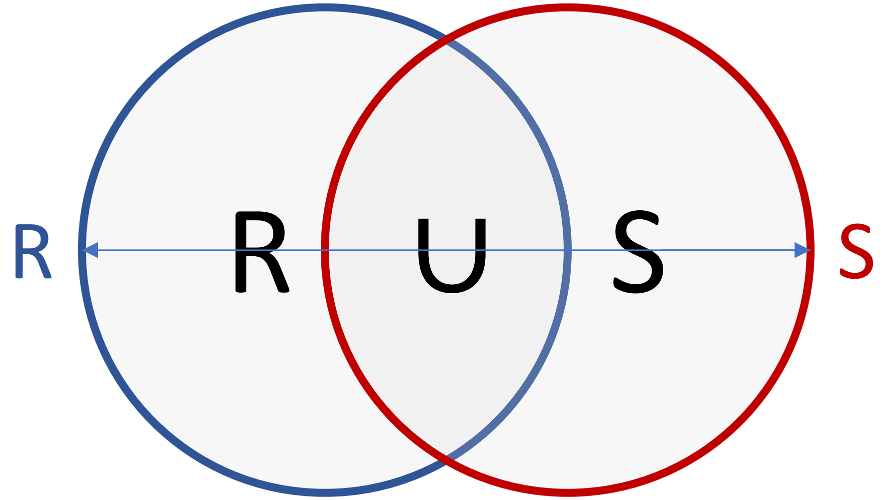
- **Union**: $R \cup S$
- **Difference**: $R - S$

- R and S must have same attributes
- Before performing the operation S are arranged so that the attributes are in the same order as in R

Set Operations: UNION

- R U S is a relation where
  - **All tuples** in R or in S or in both R and S are in R U S.
  - If *t* is in both R and S, is *t* still only once in in R U S (because a relation is a **set**)
  - No other tuples are in R U S

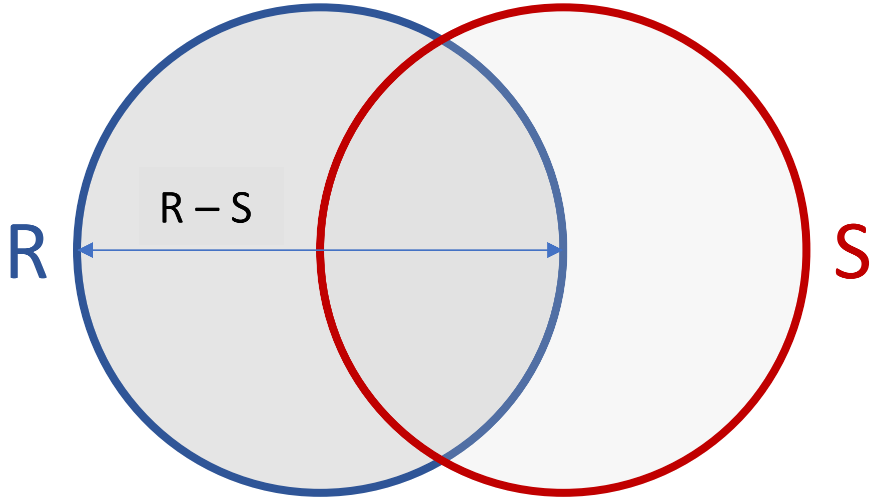
- **Example** of regular set union:
  {a, b, c} U {a, c, d} = {a, b, c, d}



Set Operations: DIFFERENCE

- R – S is a relation where
  - All tuples that are in R but not in S are in R – S
  - No other tuples appear in R – S

- Example of regular set difference:
  {a, b, c} – {a, c, d} = {b}



Operators that remove parts of a relation

- Selection: $\sigma_C(R)$
- Projection: $\pi_L(R)$

SELECTION (σ)

- σ_C(R) is the relation obtained from R by **selecting the tuples in R that satisfy the condition C**
- C is any Boolean expression made up of atoms, for example of the form $op_1\ \varphi\ op_2$, where
  - The operator φ is one of $=, \neq$ or domain specific (e.g., <, >, <=, LIKE)
  - Operands op_1 and op_2 are
    - either two attributes in R with same domain
    - or one attribute in R and a constant from the attribute domain
    - In (A LIKE *e*), *e* is a constant or a regular expression

SELECTION $\sigma_c(R)$ Example

Tutorials

ID	site	tutorial	topic
1	w3schools	SQL_2003STD	Database
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Database
5	w3resource	MySQL	Database

$\sigma_{topic = "Database"}(Tutorials)$

ID	site	tutorial	topic
1	w3schools	SQL_2003STD	Database
4	w3resource	SQL_2003STD	Database
5	w3resource	MySQL	Database

PROJECTION (π)

- $\pi_L(R)$, where R is a relation and L is a list of attributes in R, is the relation obtained from R is by selecting the columns of the attributes in L
- The relation has a schema with the attributes in L
- No tuples can occur more than once in $\pi_L(R)$

PROJECTION $\pi_L(R)$ Example

Tutorials

ID	site	tutorial	topic
1	w3schools	SQL_2003STD	Database
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Database
5	w3resource	MySQL	Database

$\pi_{\text{site, topic}}(\text{Tutorials})$

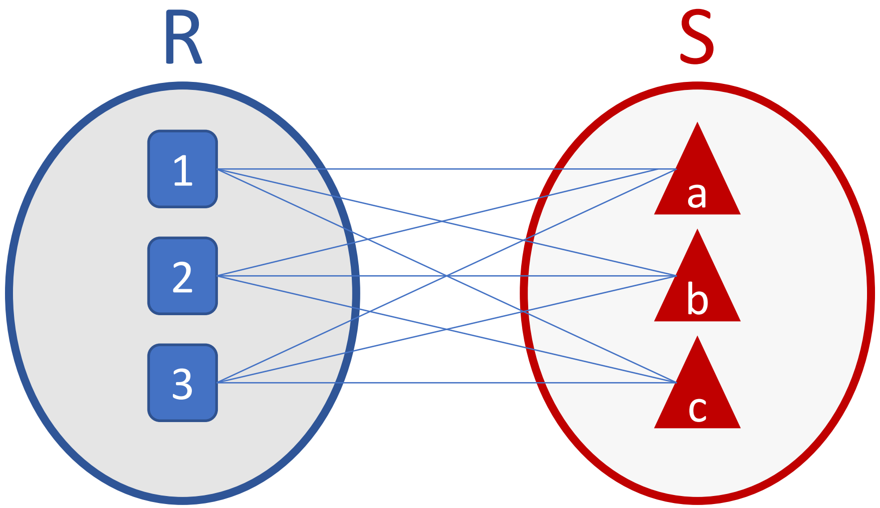
site	topic
w3schools	Database
w3schools	WebDev
w3schools	WebDev
w3resource	Database

Note only one copy of (w3resource, Database) in the result

Cartesian (Cross) Product R×S

- R×S is the relation obtained from R and S by forming all possible combinations of one tuple from R and one tuple from S
- We often say that one tuple *t* from R and one tuple *u* from S is **concatenated** into a tuple *v* = *tu* in R×S
- In the resulting schema, any name similarity between attributes in R and S is resolved by **qualifying** the names with the origin relation: R.A, S.A
- R and S cannot be the same, for self-joining one of them must first be renamed using renaming operation (see below)

Cartesian Product R×S Visual Example



EACH by EACH,
ALL TUPLES

(1, a)
(1, b)
(1, c)
(2, a)
(2, b)
(2, c)
(3, a)
(3, b)
(3, c)

Cartesian Product Example

S

ID	site	HQ
101	w3schools	USA
102	Udemy	UK
103	Folkeuniversitetet	NO

×

C

ID	topic
501	Database
503	Language

→

C × S

S.ID	site	HQ	C.ID	topic
101	w3schools	USA	501	Database
102	Udemy	UK	501	Database
103	Folkeuniversitetet	NO	501	Database
101	w3schools	USA	503	Language
102	Udemy	UK	503	Language
103	Folkeuniversitetet	NO	503	Language

- In SQL, the cartesian product is a **`CROSS JOIN`**
- NOTE: the result is often huge in practice and makes little sense

RENAMING (ρ)

- $\rho_{S(A1,A2,\ldots,An)}(R)$ renames R to a relation S with name S and attributes A1, A2, ..., An
- Shortcut: $\rho_S(R)$ renames R to a relation with name S Attribute names from R are kept as is
- In certain cases (operations on "self"), renaming the relation (giving it another name) may be necessary to avoid semantic misinterpretation

Self-join

- We want “names of all employees and each employee´s manager” given `Employee(Id, Name)`, `Manager(empId, mgrId)`.
- THIS IS WORNG:

```
SELECT e.Name, e.Name FROM Employee e
  JOIN Manager ON empId=Id AND mgrId=Id;
```

- There is obviously something very wrong with this query. We need TWO names!

Self-join continued

- Try again: "names of all employees and each employee´s manager" given `Employee(Id, Name)`, `Manager(empId, mgrId)`
- We need an extra copy of `Employee`:
- This is correct:

```
SELECT e.Name, s.Name FROM Employee e
JOIN Manager ON empId=e.Id
JOIN Employee s ON s.Id = mgrId;
```

Self-join in Relational Algebra

```
SELECT e.Name, s.Name FROM Employee e
JOIN Manager ON empId=e.Id
JOIN Employee s ON s.Id = mgrId;
```

$$\pi_{\texttt{e.Name, s.Name}}(\sigma_{\texttt{empId=e.Id \& s.Id=mgrId}}(\rho_{\texttt{e}}(\texttt{Employee}) \times \texttt{Manager} \times \rho_{\texttt{s}}(\texttt{Employee})))$$

This query can be directly translated using national joins (see below)

The minimal set of operators

- Operators in the set $\{\cup, -, \sigma, \pi, \times, \rho\}$ can not be expressed using the other operators in the set
- They are a minimal independent set of operators in our core relational algebra

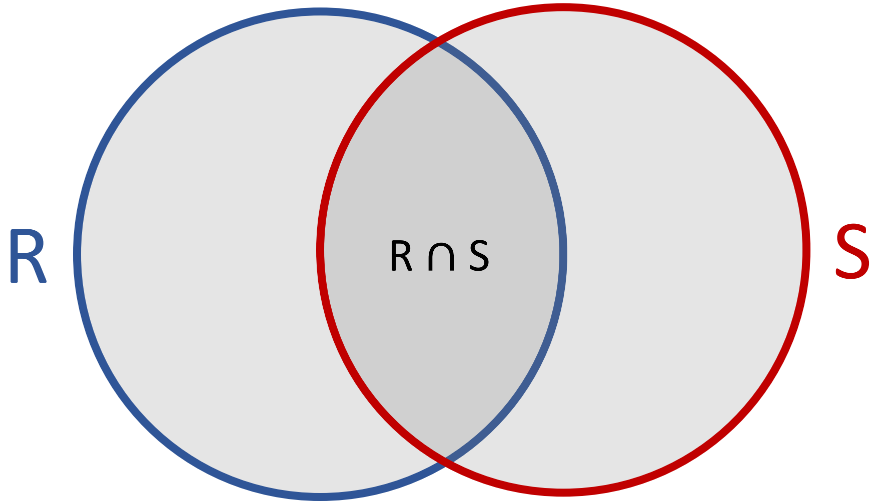
- We still wish to keep the other operators (considered next) because
  - there are effective algorithms for them and
  - it is often simpler to formulate queries using them

Set Operations: INTERSECTION

- R ∩ S is a relation where
  - Only those tuples that are in both R and S are in R ∩ S
  - No other tuples appear in R ∩ S

- Example of regular set intersection:
  {a, b, c} ∩ {a, c, d} = {a, c}

- R∩S = R – (R – S)
  - So we do not need ∩ in the core RA



Operators that combine tuples

- Cartesian product (cross product, cross join): $R \times S$

- Natural join: $R \bowtie S$
- Theta-join: $R \bowtie_{\theta} S$
- …

Natural Join

- R⋈S is the relation obtained from R and S by forming all possible mergers of one tuple from R with one from S where the **tuples are to match all attributes with matching names**
- Common attributes occur only once in the merged attributes
- The resulting schema has the attributes in R followed by those attributes in S that do not also occur in R
- Natural join is an EQUI-JOIN: join on equality

Dangling Tuple

- A **dangling tuple** is a tuple in one of the relations that has no matching tuple in the other relations
- Dangling tuples are not represented in the result relation after a natural join
- To keep them, use outer join (see below)

Natural Join R ⋈ S
Practical Example

foods

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

company

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

```
SELECT * FROM foods
NATURAL JOIN company;
```

Dangling?

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

NOTE that it is an "equi-join" on COMPANY_ID

Rewriting of Natural Join via Basic Operators

R ⋈ S is equivalent to $\pi_L(\rho_N(\sigma_C(R \times S)))$ where

- C is R.A1 = S.A1 AND … AND R.An = S.An for common attributes A1, …, An
- N renames all attributes R.Ai to Ai
- L is the set of all attributes Ai (but not S.Aj)

Similarly, we can rewrite R × S via ⋈ and ρ

Theta Join ($\bowtie_\theta$)

- Generalization of a natural join
- The relation $R \bowtie S$ where θ is a condition (Boolean expression) is calculated as follows:
  1. Calculate $R \bowtie S$
  2. Pick the tuples that satisfy the condition θ
- The constituents (atoms) in θ have the form A φ B where A and B are attributes in R and S, A and B respectively have the same domain, and $\varphi \in \{ =, \neq, <, >, <=, >= \}$ + LIKE ´*RegularExpression*´ in practice!
- AGAIN, NOTE that a theta join links tables based on a relationship other than «natural» equality, but the condition can use equality!

Theta Join ($\bowtie_\theta$) Example

Start with the Natural Join as suggested

```
SELECT * FROM foods
NATURAL JOIN company
WHERE ITEM_ID < 3
```

foods

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

company

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

Dangling?

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
~~15~~	~~6~~	~~Cheez-It~~	~~Pcs~~	~~Jack Hill Ltd~~	~~London~~
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
~~17~~	~~3~~	~~Mighty Munch~~	~~Pcs~~	~~Foodies.~~	~~London~~
~~15~~	~~4~~	~~Pot Rice~~	~~Pcs~~	~~Jack Hill Ltd~~	~~London~~
~~18~~	~~5~~	~~Jaffa Cakes~~	~~Pcs~~	~~Order All~~	~~Boston~~

Equi-join (special case of theta Join)

Special case of a theta-join $\bowtie_\theta$ where condition θ satisfies following requirements:

1. θ contains no other Boolean operators than AND, i.e., θ has the form θ_1 AND θ_2 AND ... AND θ_m
2. Where θ_k for $1 \leq k \leq m$ is in the form A = B there A is an attribute in R and B is an attribute in S with A and B having the same domain

(In other words: When theta join uses only "=")

Another Theta Join ($\bowtie_\theta$, an equi-join) Example

name	manufacturer
Efes Pilsen	Anadolu Gurubu
Ringnes Pils	Ringnes
Ringnes Lite	Ringnes

$\bowtie$

drinker	beer
Ada	Efes Pilsen
Naci	Efes Pilsen
Bjørn	Ringnes Lite

```
SELECT * FROM Beers B JOIN
Likes L ON B.name = L.beer;
```

name	manufacturer	drinker
Efes Pilsen	Anadolu Gurubu	Ada
Ringnes Pils	Ringnes	Naci
Ringnes Lite	Ringnes	Bjørn

Division

- Let R(A,B) and S(B) be two relations, and A, B be disjoint sets of attributes
- R **div** S is all tuples *t* from π_A(R) such that {*t*} × S is contained in R
- In other words: all *t* such that R contains a tuple *tu* for every *u* in S
- Division is the "inverse" of Cartesian product (R' × S') div S' = R'
- NOTE that the opposite is not valid (R div S) × S ≠ R

- Queries with "all" often indicate division
- No division operator in SQL STD 2003! You need a technique.

You can derive division using projection, Cartesian product, and difference

R(A,B) **div** S(B) is equivalent to

$$\pi_A(R) -$$
$$\pi_A((\pi_A(R) \times S) - R)$$

Note:

$(\pi_A(R) \times S) - R$ are those that do NOT satisfy the condition

Typical steps for computation of R(A,B) div S(B):

- Find out all possible combinations of S(B) with R(A) by computing R(A) × S(B); call it R1
- Subtract actual R(A,B) from R1; call it R2
- A in R2 are those that are not associated with any value in S(B); therefore R(A)-R2(A) gives us the A that are associated with all values in S
- Take a look at the examples here:
  - https://www.geeksforgeeks.org/sql-division/
  - https://www.studytonight.com/dbms/division-operator.php

Division: Example of use

- **Romutstyr (room equipment)** show the equipment that exists
- **Aktivitetskrav (for activity)** shows the kind of equipment needed for a given activity

Romutstyr	
rom	**utstyr**
BL211U71	lerret
BL211U71	lydanlegg
BL211U71	nettverk
BL211U71	tv-video
BL211U71	videoprojektor
BL21141	lerret
BL21141	lydanlegg
BL21141	piano
BL212U62	lysbildefremviser
BL212U62	tv-video
BL212U62	videoprojektor
BL21203	lerret
BL21203	lydanlegg
BL21203	videoprojektor

Aktivitetskrav	
aktivitet	**utstyr**
MUS1225-hørelære	lerret
MUS1225-hørelære	lydanlegg
MUS1225-musikkproduksjon	lerret
MUS1225-musikkproduksjon	videoprojektor
MUS1235-satslære	lerret
MUS1235-satslære	lydanlegg
MUS1235-satslære	piano

Room that covers all equipment needs

- Let R = Romutstyr (room equipment) and A = Aktivitetskrav (for activity)
- Room that covers <u>all</u> equipment requirements for MUS1225-hørelære (hearing training):

  R **div** $\pi_{\text{utstyr}}(\sigma_{\text{aktivitet = MUS1225-hørelære}}(A))$
- Room that covers <u>all</u> equipment requirements for MUS1225, i.e., both hørelære (hearing training) and musikkprosuksjon (music production):

  R **div** $\pi_{\text{utstyr}}(\sigma_{\text{aktivitet LIKE ´MUS1225\%´}}(A))$

Result of the division

Romutstyr	
rom	**utstyr**
BL211U71	lerret
BL211U71	lydanlegg
BL211U71	nettverk
BL211U71	tv-video
BL211U71	videoprojektor
BL21141	lerret
BL21141	lydanlegg
BL21141	piano
BL212U62	lysbildefremviser
BL212U62	tv-video
BL212U62	videoprojektor
BL21203	lerret
BL21203	lydanlegg
BL21203	videoprojektor

Aktivitetskrav	
aktivitet	**utstyr**
MUS1225-hørelære	lerret
MUS1225-hørelære	lydanlegg
MUS1225-musikkproduksjon	lerret
MUS1225-musikkproduksjon	videoprojektor
MUS1235-satslære	lerret
MUS1235-satslære	lydanlegg
MUS1235-satslære	piano

Romutstyr **div** $\pi_{utstyr}(\sigma_{aktivitet=MUS1225\text{-}hørelære}(Aktivitetskrav))$
rom
BL211U71
BL21141
BL21203

Romutstyr **div** $\pi_{utstyr}(\sigma_{aktivitet\ \mathbf{LIKE}\ 'MUS1225\%'}(Aktivitetskrav))$
rom
BL211U71
BL21203

Outer Join

- Outer join is used when you want to preserve dangling tuples from natural join **(Not expressible via other operators!)**
- $R \bowtie_O S$, outer join:
  - Start with $R \bowtie S$
  - Add dangling tuples from R and S
  - Missing attribute values are filled in with ⊥ (nil)
- $R \bowtie_{OL} S$ Left outer join: Only dangling tuples from R are added
- $R \bowtie_{OR} S$ Right outer join : Only dangling tuples from S are added

Outer Join Example

https://www.w3resource.com/sql/joins/perform-an-outer-join.php

company

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

foods

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	

COMPANY_NAME	COMPANY_ID	COMPANY_ID	ITEM_NAME	ITEM_UNIT
Akas Foods	16	16	Chex Mix	Pcs
Jack Hill Ltd	15	15	Cheez-It	Pcs
Jack Hill Ltd	15	15	BN Biscuit	Pcs
Foodies.	17	17	Mighty Munch	Pcs
Jack Hill Ltd	15	15	Pot Rice	Pcs
Order All	18	18	Jaffa Cakes	Pcs
sip-n-Bite.	19			

```
SELECT company.company_name,
company.company_id,
foods.company_id,
foods.item_name,
foods.item_unit
FROM company, foods
WHERE company.company_id =
foods.company_id(+);
```

Extended projection

- $\pi_L(R)$, extended: L is a list where each item can be
  i. A simple attribute in R
  ii. An expression A → B, where A is an attribute in R and B is an unused attribute name, renames A in R to B in the result relation (**Expressible in basic algebra)**
  iii. An expression E → B, where E is an expression built up of attributes in R, constants, arithmetic operators and string operators, and B is an unused attribute name (**Not Expressible)**

Extended projection – The result relation

The result relation $\pi_L(R)$ is obtained from R as follows:

- Consider each tuple in R separately
- Substitute the tuple´s values for the attribute names in L and calculate the expressions in L
- The result relation is a set with as many attributes as items in L, and with names as given in L

Bags

- Real-life DBMSs use **Bag** (multiset) and not Set as the basic type for realizing relations
  - Set (D):
    Each element in D occurs at most once. The order of the elements does not matter
    {a, b, c} = {a, c, b} = {a, a, b, c} = {c, a, b, a}
  - Bag (D):
    Each element in D can occur more than once. The order of the elements does not matter
    {a, b, c} = {a, c, b} ≠ {a, a, b, c} = {c, a, b, a}
- Every set is a bag

Why Bag and not Set?

- Bag provides more efficient union and projection calculations than Set
- In aggregation, we need Bag functionality

- But: Bag is more space consuming than Set

Relational operators on bags

- The definitions become slightly different
- Not all algebraic laws that hold for sets hold for bags
  Example: $(R \cup S) - T = (R - T) \cup (S - T)$ for sets but not for bags
- When we later in the lectures mention «bag relation», we mean a table as before <u>except</u> that tuples may repeat (are a bag)
- We need another relational algebra for bags
- In fact, DBMSs usually allow only sets as stored tables, but bags as query answers (we can ignore this for further exposition)

Bag Union

- Let R and S be bag relations

  If *t* is a tuple that occurs *n* times in R and *m* times in S, then *t* occurs *n* + *m* times in the bag relation R ∪ S

- Example of typical bag union:
  {a, a, b, c, c} ∪ {a, c, c, c, d} = {a, a, a, b, c, c, c, c, c, d}

Bag Intersection

- Let R and S be bag relations

  If *t* is a tuple that occurs *n* times in R and *m* times in S, then *t* occurs min(*n*, *m*) times in the bag relation R∩S

- Example of typical Bag intersection:
  {a, a, b, c, c} ∩ {a, c, c, c, d} = {a, c, c}

Bag Difference

- Let R and S be bag relations

  If *t* is a tuple that occurs *n* times in R and *m* times in S, then *t* occurs max (0, *n*-*m*) times in the bag relation R—S

- Example of typical Bag difference:
  {a, a, b, c, c} – {a, c, c, c, d} = {a, b}

Bag Selection

- If R is a bag relation, then σ_θ(R) is a bag relation obtained from R by applying θ to each tuple individually and selecting the tuples in R that satisfy the condition θ

Bag Projection

- If R is a bag relation and L is a (non-empty) list of attributes, then $\pi_L(R)$ is the bag relation obtained from R by selecting the columns of the attributes in L

- $\pi_L(R)$ has as many tuples as R

Cartesian Product of Bags

- R $\times$ S is the bag relation obtained from the bag relations R and S by forming all possible concatenations of one tuple from R and one tuple from S

- If R has *n* tuples and S has *m* tuples, there will be *nm* tuples in R $\times$ S

- Contrary to some previous operations, this is a proper generalization of set Cartesian product (applied to sets gives a set)

Natural Join of Bags

- If R and S are bag relations, then R⋈S is the bag relation obtained by merging matching tuples in R and S individually

- Expressible via other operators as before

Theta-Join of Bags

- Direct generalization of natural join, as before

- If R and S are bag relations, the bag relation $R \bowtie_{\theta} S$ where θ is a condition is formed as follows:

1. Calculate $R \bowtie S$ (natural join)
2. Select the tuples that satisfy the condition

Additional operators in bag relational algebra

- As before and omitted:
  - Outer Join
  - Extended projection

- Duplicate elimination
- Aggregation (with grouping)

- Sorting (beyond bags)

Duplicate elimination

- δ(R) removes multiple occurrences of tuples from the bag relation R
- The result is a set

Aggregation operations

- Used on bags of atomic values for an attribute A
- Used in combination with the grouping operator

Standard aggregation operations #1

- COUNT (A):
  - Counts the number of tuples in the relation with values in the column of A
  - Tuples where A is NULL are not counted

- MIN (A), MAX (A):
  - Selects the smallest / largest value in the column of A (The column must have at least one value)
  - The domain of A must have an *order* relation
  - For numeric values this is <
  - Lexicographic arrangement is used for strings

Standard aggregation operations #2

- SUM(A):
  - Sums all values in column A
  - A´s domain must be numeric values

- AVG(A):
  - Calculates the average of the values in column A
  - Assumes that the column has at least one value
  - A´s domain must be numeric values

Grouping (with aggregation)

- Used when we want to apply an aggregation operator to groups of values
- Form: $\gamma_L(R)$, where L is a list of items with all the items in the list different. The elements are in one of the following two forms:
- A
  - A is an attribute in R
  - A is called a grouping attribute
- AGG (A) → AggRes
  - AGG is an aggregation operator
  - AggRes is an unused attribute name
  - A is called an aggregation attribute

The resulting relation after grouping

Given $\gamma_L(R)$, the result relation is constructed as follows

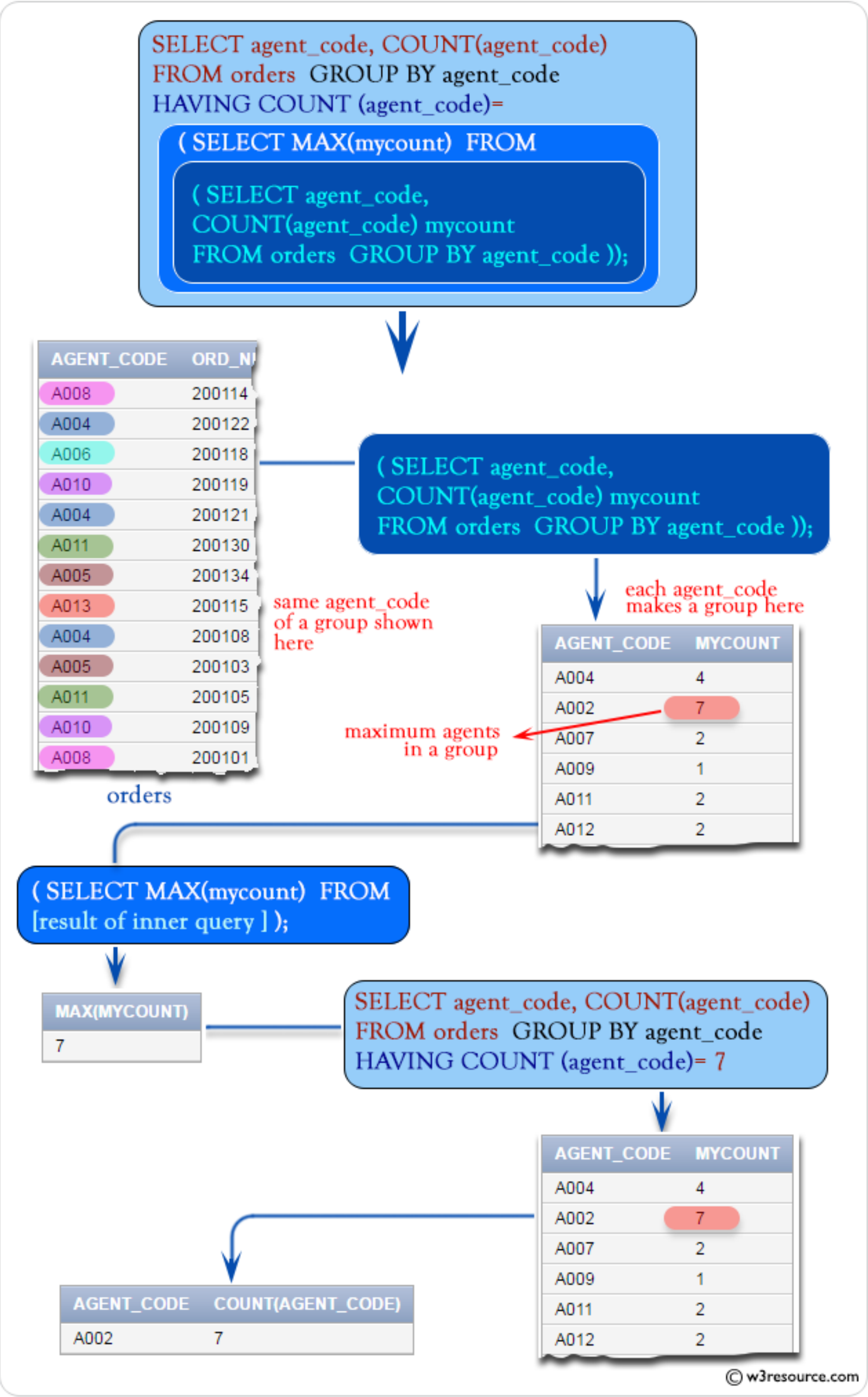
1. Partition R in groups, one group for each collection of tuples that are equal in all grouping attributes in L
2. For each group, produce a tuple consisting of
   i. The values of the grouping attributes in the group
   ii. For each aggregation attribute in L, the aggregation over all the tuples in the group

The result relation gets as many attributes as there are elements in L, and attribute names as specified by L. The result instance contains one tuple per group.

Grouping and aggregation use & example

```
SELECT MAX(mycount) FROM
  (SELECT
    agent_code, COUNT(agent_code) mycount
    FROM orders
    GROUP BY agent_code
  );
```

$$\gamma_{MAX(mycount) \rightarrow mmc}(\gamma_{agent\_code,\ COUNT(agent\_code) \rightarrow mycount}(orders))$$



Sorting

- $\tau_L(R)$, where R is a relation and L a list of attributes $A_1, A_2, ..., A_k$, results in a list of tuples sorted first by A_1, then by A_2 internally in each batch of equal A_1 values, etc.
- The attributes that are not included in the list are randomly arranged
- Result is a *list*, so the operation is meaningful only as a last, final operation on relations
- Beyond bag and set relational algebra!

Relations and rules of integrity

- We can express referential integrity, functional dependencies and multi-value dependencies - and also other classes of integrity rules - in relational algebra!

Examples of integrity rules in classical relational algebra

- If E is an expression in relational algebra, then $E=\emptyset$ is an integrity rule that says that E does not have any tuples
- If E_1 and E_2 are expressions in relational algebra, then $E_1 \subseteq E_2$ is an integrity rule that says that each tuple in E_1 shall also be in E_2
- Note that $E_1 \subseteq E_2$ and $E_1 - E_2 = \emptyset$ are equivalent.
  Also $E = \emptyset$ and $E \subseteq \emptyset$. Thus, only one of the forms above is sufficient
- Strictly speaking, $\emptyset$ is not a relational algebra expression. We could have written $R - R$ instead (for an arbitrary relation R with same schema as E)

Examples of integrity rules in classical relational algebra

- **Referential integrity**: ”A is foreign key for S”, where B is primary key in S:
  $\delta(\pi_A(R)) \subseteq \pi_B(S)$
- **FDs**: ”A1 A2 ... An→B1 B2 ... Bm” in R:
  $\sigma_\theta(\rho_{R1}(R) \times \rho_{R2}(R)) = \emptyset$
- where θ is the expression
  R1.A1 = R2.A1 AND ... AND R1.An = R2.An AND
  (R1.B1 ≠ R2.B1 OR ... OR R1.Bm ≠ R2.Bm)
- **Domain constraints**:
  $\sigma_{A \neq 'F' \text{ AND } A \neq 'M'}(R) = \emptyset$