Databases

TDA357/DIT621– LP3 2023

Lecture 13

Ana Bove

(much of the material is based on material from both Thomas Hallgren and Jonas Duregård)

February 23rd 2023

Recall Last Lecture

- Transactions:
  - BEGIN/COMMIT or ROLLBACK;
  - ACID properties: atomicity, consistency, isolation and durability;
  - Interference problems: dirty read, non-repeatable read, phantom;
  - Isolation levels: serializable, repeatable read, read committed, read uncommitted;
- Authorisation and privileges:
  - Privileges can be granted and revoked to users/roles;
  - Most common privileges: SELECT, INSERT, UPDATE and DELETE;
- Indexes:
  - DBMS defines indexes on primary keys and unique attributes;
  - Users can define (and drop) other indexes: advantages and disadvantages on this!

Overview of Today's Lecture

- Relational algebra;
- Correspondence between **SQL** and relational algebra;
- A glance into query optimisation.

Recall (Lecture 6): Relations, Relation Schemas and Tables

- A *relation* R is a subset of the cartesian product of two or more sets $T_1, T_2, \dots, T_n$:
$$R \subseteq T_1 \times T_2 \times \cdots \times T_n$$
- A *relation schema* $R(a_1, \dots, a_n)$ can be augmented with the domain/type of each attribute $R(a_1 : T_1, \dots, a_n : T_n)$;
- The *relation signature* of the relation R is then the corresponding cartesian product $T_1 \times \cdots \times T_n$;
- Given a relation schema $R(a_1, \dots, a_n)$ with signature $T_1 \times \cdots \times T_n$:
  - A *table* for the schema $R(a_1, \dots, a_n)$ is a subset of the cartesian product $T_1 \times \cdots \times T_n$;
  - A *row* in the table is an element of the cartesian product $t \in T_1 \times \cdots \times T_n$.

Algebra

Definition: (Wikipedia) *Algebra* is the study of mathematical symbols and the rules for manipulating these symbols.

Broad field of mathematics, going from elementary equation solving (elementary algebra, linear algebra) to the study of abstractions such as groups, rings, lattices... (abstract algebra).

In an algebra we have set of values, operations on those values, and formulas built from the values and the operators.

Example: Natural numbers with addition and multiplication form an algebra.

Example: Boolean algebra consists of the two Boolean values and the operators $\vee$, $\wedge$, ...

Note: Observe that both the Natural numbers and Booleans are *closed* under those operations (the result of the operation is also in the set).

Relational Algebra

Definition: Relational algebra is a theory that uses algebraic structures for modeling data and defining queries on the data.

A concise mathematical notation in which we can express relations and queries.

Main advantages:

Reasoning: We can use mathematics to prove that our queries do what we intend them to do.

Simplification: Using known algebraic laws one can simplify complicated relational algebra expressions (queries).

Optimisation: Simplification can make queries faster.

SQL Query Processing

A DBMS processes a query in several steps:

Lexing: The input string is converted into a sequence of tokens.

Parsing: The sequence of tokens is converted into a(n **SQL**) syntax tree.

Type checking: The syntax tree is checked semantically.

Logical query plan generation: The syntax tree is converted into a relational algebra expression.

Optimisation: The relational algebra expression is converted into a more efficient relational algebra expression.

Which of two different queries solving the same problem is more efficient (partly) depends on what the DBMS does here.

Physical query plan generation: The more efficient expression is converted into a sequence of algorithm calls.

Execution: The physical query plan is executed and produces a result.

Back to Relational Algebra

What are the values and operations in this algebra?

The values are *relations* (the tables in the database).

This means the relation schema (name of the relation and its attributes) and the (labelled) tuples in the relation (rows in the table).

The operations are the different things we can do on a relation.

Example: Select, group by, order, ...

Each operation returns a new *relation*.

Note: When working with relational algebra, we focus on the schema, not the actual tuples.

Do we Work with Sets, Bags, Lists or Arrays?

	No order	Ordered
No duplicates	Sets	Ordered sets
Duplicates	Multisets/bags	Lists or arrays

In sets/relations: Order and duplication is irrelevant.

In tables: Order and duplication make a difference:

- **SQL** has DISTINCT and ORDER BY;
- Set operations UNION, INTERSECT and EXCEPT discard duplicates ...
- ... but UNION ALL, INTERSECT ALL and EXCEPT ALL preserve duplicates;
- Primary keys and unique constrains prevent duplicates ...
- ... but the result of a query can still contain duplicates.

Relational Algebra: Basic Notation and Correspondences

Concept	**Relational algebra**	**Set theory**	**SQL**
domain of attribute values		T	type
cartesian products of sets	$T_1 \times ... \times T_n$	$\{\langle t_1, ..., t_n \rangle \mid t_i \in T_i\}$	relation schema
relation	R	$R \subseteq T_1 \times ... \times T_n$	table
tuple	$\{a = t_1, ..., k = t_n\}$	$\langle t_1, ..., t_n \rangle$	row
label	a		attribute name
component	$t.a$	$\pi_{i(a)} t, t_{i(a)}$	value of attribute
		$\{t_i \mid \langle ..., t_i, ... \rangle \in R\}$	column

Relational Schema in this Lecture

- Countries and their currency values:

  Currencies (<u>code</u>, name, value)

  Countries (<u>name</u>, abbr , capital, area, population, continent, currency)
  currency → Currencies.code

- Countries with their capitals and currencies:

  Capitals (<u>country</u>, capital)
  country → Countries.name

  CurrencyCodes (<u>country</u>, currency)
  country → Countries.name

- Students and their grades:

  Students (<u>idnr</u>, name)

  Grades (<u>student</u>, <u>course</u>, grade)
  student → Students.idnr
  course → ...

From **SQL** to Relational Algebra: Basic Queries

Table names can be used directly.

SQL

```
SELECT projection
FROM Table
WHERE condition;
```

Relational Algebra

$$\pi_{\texttt{projection}}(\sigma_{\texttt{condition}}(\texttt{Table}))$$

π for projections of components

σ for selection of tuples/elements

Note: Here `condition` is a Boolean-expression, not an **SQL** sub-query/relational algebra expression!

Examples: Basic Queries in Relational Algebra

SQL	Relational Algebra
SELECT * FROM Countries;	$\texttt{Countries}$

Selecting rows:

SELECT * FROM Countries WHERE abbr = 'UY';	$\sigma_{\texttt{abbr}='\texttt{UY}'}(\texttt{Countries})$

Projection:

SELECT capital FROM Countries WHERE abbr = 'UY';	$\pi_{\texttt{capital}} \; (\sigma_{\texttt{abbr}='\texttt{UY}'}(\texttt{Countries}))$

Expressions:

SELECT capital, population/area FROM Countries WHERE abbr = 'UY';	$\pi_{\texttt{capital},\ \texttt{population}/\texttt{area}} \; (\sigma_{\texttt{abbr}='\texttt{UY}'}(\texttt{Countries}))$

Note: From now on, we might omit some (,) when we break into different lines.

From **SQL** to Relational Algebra: Renaming

SQL

Renaming columns:

```
SELECT name AS country,
       population/area AS density
FROM Countries
WHERE continent = 'EU';
```

Renaming tables:

```
SELECT A.abbr
FROM Countries AS A,
WHERE A.name = 'Sweden';
```

ρ can be also be used to give a new schema (renaming tables and attributes)

Relational Algebra

AS becomes $\rightarrow$:

$$\pi_{\texttt{name}\rightarrow\texttt{country},\ \texttt{population/area}\rightarrow\texttt{density}}\ \sigma_{\texttt{continent}='\texttt{EU}'}\ \texttt{Countries}$$

ρ for renaming tables:

$$\pi_{\texttt{A.abbr}}(\sigma_{\texttt{A.name}='\texttt{Sweden}'}(\rho_{\texttt{A}}\ \texttt{Countries}))$$

$$\pi_{\ldots,\texttt{A.a}_\texttt{i},\ldots}\ \sigma_{\ldots,\texttt{A.a}_\texttt{j}=\ldots}(\rho_{\texttt{A}<\texttt{a}_1,\texttt{a}_2,\ldots>}\ \texttt{Table})$$

From SQL to Relational Algebra: Sorting

SQL

Relational Algebra

Sorting on a selected attribute:

```
SELECT name, capital,
FROM Countries
ORDER BY name;
```

τ for sorting:

$\pi_{\texttt{name, capital}}\left(\tau_{\texttt{name}}\ \texttt{Countries}\right)$

$\tau_{\texttt{name}}\left(\pi_{\texttt{name, capital}}\ \texttt{Countries}\right)$

Descending:

```
SELECT name, capital,
FROM Countries
ORDER BY name DESC;
```

$\pi_{\texttt{name, capital}}\left(\tau_{-\texttt{name}}\ \texttt{Countries}\right)$

$\tau_{-\texttt{name}}\left(\pi_{\texttt{name, capital}}\ \texttt{Countries}\right)$

Sorting on a non-selected attribute:

```
SELECT name, capital,
FROM Countries
ORDER BY area;
```

One cannot sort by an attribute that has already been discarded by a projection:

$\pi_{\texttt{name, capital}}\left(\tau_{\texttt{area}}\ \texttt{Countries}\right)$

From **SQL** to Relational Algebra: Duplicates

Recall: set vs. bag semantics!

SQL		**Relational Algebra**
SELECT currency, FROM Countries;	⇒	Might contain duplicates: $\pi_{\texttt{currency}}\ \texttt{Countries}$
SELECT DISTINCT currency, FROM Countries;	⇒	δ for removing duplicates: $\delta(\pi_{\texttt{currency}}\ \texttt{Countries})$

From **SQL** to Relational Algebra: Grouping and Aggregations

SQL

```
SELECT currency, COUNT(name),
FROM Countries
GROUP BY currency;
```

Relational Algebra

γ combines SELECT and GROUP BY:

$\gamma_{\texttt{currency},\ \texttt{COUNT}(\texttt{name})}\ \texttt{Countries}$

Aggregations are done in γ!
Rename to use the result elsewhere.

```
SELECT currency, SUM(population),
FROM Countries
GROUP BY currency
HAVING COUNT(name) > 1;
```

$$\pi_{\texttt{currency},\ \texttt{sum\_pop}}\ \sigma_{\texttt{cnt}>1}\ \gamma_{\texttt{currency},\ \texttt{SUM}(\texttt{population})\to\texttt{sum\_pop},\ \texttt{COUNT}(\texttt{name})\to\texttt{cnt}}\ \texttt{Countries}$$

Note: Both WHERE cond and HAVING cond become $\sigma_{\texttt{cond}}$.

From SQL to Relational Algebra: Cartesian Products

SQL

Relational Algebra

Full cartesian product:

```
SELECT Countries.name,
       Currencies.name,
FROM Countries, Currencies;
```

Not often what we need...

$$\pi_{\texttt{Countries.name},\ \texttt{Currencies.name}} \; \texttt{Countries} \times \texttt{Currencies}$$

Theta join:

```
SELECT Countries.name,
       Currencies.name,
FROM Countries, Currencies
WHERE currency = code;
```

$$\pi_{\texttt{Countries.name},\ \texttt{Currencies.name}} \; \sigma_{\texttt{currency}=\texttt{code}} \; \texttt{Countries} \times \texttt{Currencies}$$

From **SQL** to Relational Algebra: Natural and Inner Joins

SQL

Relational Algebra

Natural join:

```
SELECT capital, currency,
FROM Capitals
NATURAL JOIN CurrencyCodes;
```

$$\pi_{\texttt{capital, currency}}\ \texttt{Capitals} \bowtie \texttt{CurrencyCodes}$$

Inner join:

```
SELECT Countries.name,
       Currencies.name,
FROM Countries JOIN Currencies
ON currency = code;
```

$$\pi_{\texttt{Countries.name, Currencies.name}}\ \texttt{Countries} \bowtie_{\texttt{currency}=\texttt{code}} \texttt{Currencies}$$

From SQL to Relational Algebra: Outer Joins

SQL

Relational Algebra

```
SELECT Countries.name,
       Currencies.name,
FROM Countries
RIGHT OUTER JOIN Currencies
ON currency = code;
```

Right outer join:

$$\pi_{\texttt{Countries.name},\ \texttt{Currencies.name}}\ \texttt{Countries} \bowtie^{OR}_{\texttt{currency}=\texttt{code}} \texttt{Currencies}$$

```
SELECT Countries.name,
       Currencies.name,
FROM Countries
LEFT OUTER JOIN Currencies
ON currency = code;
```

Left outer join:

$$\pi_{\texttt{Countries.name},\ \texttt{Currencies.name}}\ \texttt{Countries} \bowtie^{OL}_{\texttt{currency}=\texttt{code}} \texttt{Currencies}$$

```
SELECT Countries.name,
       Currencies.name,
FROM Countries
FULL OUTER JOIN Currencies
ON currency = code;
```

Full outer join:

$$\pi_{\texttt{Countries.name},\ \texttt{Currencies.name}}\ \texttt{Countries} \bowtie^{O}_{\texttt{currency}=\texttt{code}} \texttt{Currencies}$$

What about Correlated Queries?

Consider the query:

```
SELECT name
FROM Students AS S
WHERE 4 < (SELECT AVG(grade) FROM Grades WHERE student = S.idnr)
```

The correlation needs to be replaced with a join (or cartesian product and corresponding select).

$$\pi_{\texttt{name}}(\sigma_{4<\texttt{average}}(\gamma_{\texttt{student},\texttt{name},\texttt{AVG}(\texttt{grade})\to\texttt{average}}(\texttt{Grades} \bowtie_{\texttt{idnr}=\texttt{student}} \texttt{Students})))$$

From the join, group by students and name, and compute the average of the grades of each student, select those with an average of at least 4, finally project the name.

Back to Grouping in Relational Algebra

Are these the same query?

Groups by student id *and* name; name available for projection

$$\pi_{\texttt{name},\texttt{passed}}\ \gamma_{\texttt{student},\texttt{name},\texttt{COUNT}(*)\to\texttt{passed}}\ \sigma_{\texttt{grade}\geqslant 3\ \land\ \texttt{idnr}=\texttt{student}}(\texttt{Students} \times \texttt{Grades})$$

Groups *only* by student; needs a join to project name

$$\pi_{\texttt{name},\texttt{passed}}\ (\gamma_{\texttt{student},\texttt{COUNT}(*)\to\texttt{passed}}\ (\sigma_{\texttt{grade}\geqslant 3\ \land\ \texttt{idnr}=\texttt{student}}(\texttt{Students} \times \texttt{Grades}))\ \bowtie_{\texttt{student}=\texttt{idnr}}\ \texttt{Students})$$

Note: Since we have the FD student $\to$ name, these queries are the same. Otherwise they might not be since the result of grouping by $(\texttt{student}, \texttt{name})$ might be different than grouping by just $\texttt{student}$!

What about NOT IN or NOT EXISTS?

One needs to try to understand the query in terms of sets, and use set operations instead.

Example: Consider the query that selects students that have no grades:

```
SELECT idnr, name
FROM Students
WHERE idnr NOT IN (SELECT student FROM Grades)
```

We can use set difference in relational algebra to obtain this result:

$$\pi_{\texttt{idnr,name}}(\texttt{Students} \bowtie_{\texttt{idnr=st}} (\rho_{\texttt{NoGrades}\langle\texttt{st}\rangle}(\pi_{\texttt{idnr}}\,\texttt{Students} - \pi_{\texttt{student}}\,\texttt{Grades})))$$

The result of ($\pi_{\texttt{idnr}}\,\texttt{Students} - \pi_{\texttt{student}}\,\texttt{Grades}$) is a relation consisting of "1-tuples". We give a new name to the information (table and attribute).

We join to retrieve the rest of the information of the students to project their name.

Relation Algebra: Set Operation

Concept	Relational algebra	Set theory	SQL
cartesian products of relations	$S \times R$	$\{\langle x, .., u, ..\rangle \mid \langle x, ..\rangle \in S, \langle u, ..\rangle \in R\}$	FROM S, R
union with duplicates	$S \cup R$	$\{t \mid t \in S \text{ or } t \in R\}$ OBS: bags!	S UNION ALL R
union	$\delta(S \cup R)$	$\{t \mid t \in S \text{ or } t \in R\}$	S UNION R
intersection with duplicates	$S \cap R$	$\{t \mid t \in S \text{ and } t \in R\}$ OBS: bags!	S INTERSECT ALL R
intersection	$\delta(S \cap R)$	$\{t \mid t \in S \text{ and } t \in R\}$	S INTERSECT R
difference with duplicates	$S - R$	$\{t \mid t \in S \text{ and } t \notin R\}$ OBS: bags!	S EXCEPT ALL R
difference	$\delta(S - R)$	$\{t \mid t \in S \text{ and } t \notin R\}$	S EXCEPT R

Note: Recall “schemas” need to be compatible for some of the set operations to work!

Relation Algebra: Summary of Correspondences

Concept	Rel algebra	Set theory	SQL
projection	$\pi_{\mathtt{a,b,\ldots,k}}\ R$	$\langle t.a, t.b, ..., t.k \mid t \in R \rangle$	SELECT a, b, .., k
selection	$\sigma_{\mathtt{C}}\ R$ $\sigma_{\mathtt{C}}\ R$	$\{t \in R \mid C\}$	WHERE C HAVING C
theta join	$S \bowtie_C R = \sigma_C(S \times R)$	$\{t \in S \times R \mid C\}$	S INNER JOIN R ON C
outer join	$S \bowtie^{O}_{C} R$ $S \bowtie^{OL}_{C} R$ $S \bowtie^{OR}_{C} R$	... ...	S FULL OUTER JOIN R ON C LEFT OUTER JOIN RIGHT OUTER JOIN
natural join	$S \bowtie R$	...	S NATURAL JOIN R
renaming	$\mathtt{a} \rightarrow \mathtt{b}$ $\rho_{\mathtt{A}}\ R$	—	AS
new schema	$\rho_{\mathtt{A<a,\ldots,k>}}\ R$	—	—
removing duplicates	$\delta\ R$	—	DISTINCT
sorting	$\tau_a\ R$	—	ORDER BY a
grouping	$\gamma_a\ R$	—	GROUP BY a

Example: From Problem to Relational Algebra

Select the name of all students that have passed at least 2 courses.

Students (<u>idnr</u>, name)
Grades (<u>student</u>, <u>course</u>, grade)
student → Students.idnr
course → ...

- Group first, join later: select passed courses in Grades, group by students and count passed course per student, now do the join, select entries where at least 2 courses are passed and project student names.

$$\pi_{\texttt{name}}(\sigma_{\texttt{passed}\geqslant 2\,\wedge\,\texttt{idnr}=\texttt{student}}(\texttt{Students}\times \gamma_{\texttt{student},\texttt{COUNT}(*)\to\texttt{passed}}(\sigma_{\texttt{grade}\geqslant 3}\;\texttt{Grades})))$$

- Join first, group later: from the join, select passed courses, group by students and names, and count passed courses per student, select entries where at least 2 courses are passed and project student names.

$$\pi_{\texttt{name}}(\sigma_{\texttt{passed}\geqslant 2}(\gamma_{\texttt{student},\texttt{name},\texttt{COUNT}(*)\to\texttt{passed}}(\sigma_{\texttt{grade}\geqslant 3\,\wedge\,\texttt{idnr}=\texttt{student}}(\texttt{Students}\times\texttt{Grades}))))$$

Example: From **SQL** to Relational Algebra

A query with almost everything:

```
SELECT a1, MAX(a2) AS max
FROM T1, T2
WHERE a3 = 5
GROUP BY a1, a3
HAVING COUNT(*) > 10
ORDER BY a1 DESC;
```

A relational algebra expression for it:

$$\boxed{\tau_{-\texttt{a1}}(\pi_{\texttt{a1},\texttt{max}}(\sigma_{\texttt{cnt}>10}(\gamma_{\texttt{a1},\texttt{a3},\texttt{MAX}(\texttt{a2})\rightarrow\texttt{max},\texttt{COUNT}(*)\rightarrow\texttt{cnt}}(\sigma_{\texttt{a3}=5}(\texttt{T1}\times\texttt{T2})))))}$$

From the join, select entries where $\texttt{a3} = 5$.
Group by $\texttt{a1}, \texttt{a3}$ and compute number of elements and $\texttt{MAX}(\texttt{a2})$ per group.
Select entries where the count is at least 10.
Project $\texttt{a1}$ and $\texttt{MAX}(\texttt{a2})$ for those entries.
Sort the result descending in $\texttt{a1}$.

Sanity Check (1)!

Given this schema, is the relational algebra query below correct?

Students (<u>idnr</u>, name)
Grades (<u>student</u>, <u>course</u>, grade)
  student → Students.idnr
  course → ...

$$\pi_{\texttt{idnr}}(\sigma_{\texttt{passed} \geqslant 2 \wedge \texttt{idnr}=\texttt{student}}(\texttt{Students} \times \gamma_{\texttt{student},\texttt{COUNT}(*) \to \texttt{passed}}(\sigma_{\texttt{grade} \geqslant 3}\ \texttt{Grades})))$$

Let us *sanity check* the expression by computing the schema bit by bit!

- $\sigma_{\texttt{grade} \geqslant 3}\ \texttt{Grades} : (\texttt{student}, \texttt{course}, \texttt{grade})$
- $\gamma_{\texttt{student},\texttt{COUNT}(*) \to \texttt{passed}}(\sigma_{\texttt{grade} \geqslant 3}\ \texttt{Grades}) : (\texttt{student}, \texttt{passed})$
- $\texttt{Students} \times \gamma_{\texttt{student},\texttt{COUNT}(*) \to \texttt{passed}}(\sigma_{\texttt{grade} \geqslant 3}\ \texttt{Grades}) : (\texttt{idnr}, \texttt{name}, \texttt{student}, \texttt{passed})$
- $\sigma_{\texttt{passed} \geqslant 2 \wedge \texttt{idnr}=\texttt{student}}(\texttt{Students} \times \gamma_{\texttt{student},\texttt{COUNT}(*) \to \texttt{passed}}(\sigma_{\texttt{grade} \geqslant 3}\ \texttt{Grades}))$: $(\texttt{idnr}, \texttt{name}, \texttt{student}, \texttt{passed})$
- $\pi_{\texttt{idnr}}(\sigma_{\texttt{passed} \geqslant 2 \wedge \texttt{idnr}=\texttt{student}}(\texttt{Students} \times \gamma_{\texttt{student},\texttt{COUNT}(*) \to \texttt{passed}}(\sigma_{\texttt{grade} \geqslant 3}\ \texttt{Grades})))$: $(\texttt{idnr})$

Sanity Check (2)!

Given this schema, is the relational algebra query below correct?

Students ($\underline{\text{idnr}}$, name)
Grades ($\underline{\text{student}}$, $\underline{\text{course}}$, grade)
  student → Students.idnr
  course → ...

$$\pi_{\texttt{idnr}}(\sigma_{\texttt{cnt}\geqslant 2 \,\wedge\, \texttt{idnr}=\texttt{student} \,\wedge\, \texttt{grade}\geqslant 3}(\texttt{Students} \times \gamma_{\texttt{student},\texttt{COUNT}(*)\to\texttt{cnt}}\ \texttt{Grades}))$$

Let us *sanity check* the expression by computing the schema bit by bit!

- $\gamma_{\texttt{student},\texttt{COUNT}(*)\to\texttt{cnt}}\ \texttt{Grades} : (\texttt{student}, \texttt{cnt})$
- $\texttt{Students} \times \gamma_{\texttt{student},\texttt{COUNT}(*)\to\texttt{cnt}}\ \texttt{Grades} : (\texttt{idnr}, \texttt{name}, \texttt{student}, \texttt{cnt})$
- $\sigma_{\texttt{cnt}\geqslant 2 \,\wedge\, \texttt{idnr}=\texttt{student} \,\wedge\, \texttt{grade}\geqslant 3}(\texttt{Students} \times \gamma_{\texttt{student},\texttt{COUNT}(*)\to\texttt{cnt}}\ \texttt{Grades})$: **ERROR!**
  There is no grade in the schema of the expression to be used by the σ operator in order to evaluate the condition!

Note: Make sure your expression is correct by performing a sanity check on it!

Some Algebraic Laws

These (and other) algebraic laws can be used for query simplification and/or optimisation.

Some laws generate a potentially infinite number of equivalent expressions for a query. Query optimization tries to find the best of those.

Some laws work with sets but not with bags!

Set-theoretic laws:

$$\boxed{\begin{array}{l} R \bowtie S = S \bowtie R \\ R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T \end{array}}$$

If applicable, associativity, commutativity, distributivity, idempotence of unions, intersections, products and joins.

Repeated projection:

$$\boxed{\pi_{a_1,..,a_n}(\pi_{b_1,\ldots,b_m} R) = \pi_{a_1,\ldots,a_n} R}$$

$\mathtt{b}_1, \ldots, \mathtt{b}_\mathtt{m}$ should be a plain projection, will not work if there is a rename that it is later projected.

$\mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n}$ should be a subset of $\mathtt{b}_1, \ldots, \mathtt{b}_\mathtt{m}$ for the expression to be correct.

Some Algebraic Laws (Cont.)

Some laws can dramatically reduce the number of rows in the results!

Repeated selection: $\boxed{\sigma_{C_1}(\sigma_{C_2} R) = \sigma_{C_1 \wedge C_2} R}$

Pushing duplicate elimination inside:

$$\boxed{\begin{array}{l} \delta(\sigma_C R) = \sigma_C(\delta R) \\ \delta(R \times S) = \delta(R) \times \delta(S) \end{array}}$$

Pushing selection inside cartesian products:

If `C` only uses attributes in R_1: $\boxed{\sigma_C (R_1 \times R_2) = (\sigma_C R_1) \times R_2}$

If $\mathtt{C}_1$ only in R_1 and $\mathtt{C}_2$ only in R_2: $\boxed{\sigma_{C_1 \wedge C_2} (R_1 \times R_2) = (\sigma_{C_1} R_1) \times (\sigma_{C_2} R_2)}$

Note: See section 2 in chapter 16 of the book for more algebraic laws for improving queries!

Presentation

To make large relation algebra expressions more readable:

- Write them over several lines and use indentation to show the structure:

$$
\begin{array}{l}
\pi_{\texttt{A.name}} \\
\quad (\sigma_{\texttt{A.name}=\texttt{B.capital}} \\
\quad\quad (\rho_{\texttt{A}}\ \texttt{Countries} \times \rho_{\texttt{B}}\ \texttt{Countries}))
\end{array}
$$

- Split them in to several named parts:

$$
\begin{array}{l}
R_1 = \rho_{\texttt{A}}\ \texttt{Countries} \times \rho_{\texttt{B}}\ \texttt{Countries} \\
R_2 = \sigma_{\texttt{A.name}=\texttt{B.capital}} R_1 \\
\text{Result} = \pi_{\texttt{A.name}} R_2
\end{array}
$$

Final Remarks

- A basic **SQL** query (performing some projection, selection and cartesian product) is done "altogether" and produces a single result. In relational algebra, each operation results is a *new* relation.

- It is then important to keep track of what information is available after each step (see slide 28!).

  Recall the expression ($\pi_{\texttt{idnr}}\,\texttt{Students} - \pi_{\texttt{student}}\,\texttt{Grades}$) in slide 22: it results in a relation with student's id as single information; to retrieve the rest of the information of students we needed to perform a join!

- The abstract syntax tree of a relational algebra expression can help you understanding the expression.

- Translating **SQL** to relational algebra, simplifying the expression and then translating it back to **SQL** can give a not too compact query!

Overview of Next Lecture

- Quiz with recap of all course!
  (Good if you have a separate device to answer the quiz than that you use to see the questions if you are on zoom.)