The Relational Algebra

COMPSCI 2DB3: Databases

Jelle Hellings   Holly Koponen

Department of Computing and Software
McMaster University

Winter 2024

How can we query relational databases?

SQL Used in practice, high-level, say *what* you want.
No direct relation to how to *compute* result.

How can we query relational databases?

SQL — Used in practice, high-level, say *what* you want.
No direct relation to how to *compute* result.

Relational Algebra — Say *how to compute* the result with basic operations.
Similar to how algebra can express mathematical computations.

How can we query relational databases?

SQL	Used in practice, high-level, say *what* you want. No direct relation to how to *compute* result.
Relational Algebra	Say *how to compute* the result with basic operations. Similar to how algebra can express mathematical computations.
Domain Calculus	Use *first-order logic* to describe the result. Useful tool to reason about queries *theoretically*.

How can we query relational databases?

SQL	Used in practice, high-level, say *what* you want. No direct relation to how to *compute* result.
Relational Algebra	Say *how to compute* the result with basic operations. Similar to how algebra can express mathematical computations.
Domain Calculus	Use *first-order logic* to describe the result. Useful tool to reason about queries *theoretically*.
Query-by-Example	A visual way to express queries. Very interesting idea: Microsoft Access supports a variant.

An example of the relational algebra and the domain calculus

Consider the following SQL query

```
SELECT S.name, C.title
FROM students S, enroll_in E, courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND
        S.sid NOT IN (SELECT fid FROM faculty);
```

An example of the relational algebra and the domain calculus

Consider the following SQL query

```
SELECT S.name, C.title
FROM students S, enroll_in E, courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND
        S.sid NOT IN (SELECT fid FROM faculty);
```

In the relational algebra

$$\pi_{\text{S.name,C.title}}(\sigma_{\text{S.sid=E.sid=X.sid}\wedge\text{E.cid=C.cid}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}) \times \rho_X(\pi_{\text{sid}}(\text{students}) \setminus \rho_{\text{fid}\mapsto\text{sid}}(\sigma_{\text{fid}}(\text{faculty})))))$$

An example of the relational algebra and the domain calculus

Consider the following SQL query

```
SELECT S.name, C.title
FROM students S, enroll_in E, courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND
      S.sid NOT IN (SELECT fid FROM faculty);
```

In the domain calculus

$$\{(\mathrm{sn}, \mathrm{cn}) \mid \exists \mathrm{si} \exists \mathrm{ci}\ (\mathrm{students}(\mathrm{si}, \mathrm{sn}) \wedge \mathrm{enroll\_in}(\mathrm{si}, \mathrm{ci}) \wedge \mathrm{courses}(\mathrm{ci}, \mathrm{cn}) \wedge \\ \forall \mathrm{fi} \forall \mathrm{fn} \forall \mathrm{fr}\ (\mathrm{faculty}(\mathrm{fi}, \mathrm{fn}, \mathrm{fr}) \implies \mathrm{fi} \neq \mathrm{si}))\}$$

The legacy of Query-by-Example

SQL *was* intended for novices …

Query-by-Example *works* for novices: graphical language!

The legacy of Query-by-Example

SQL *was* intended for novices …

Query-by-Example *works* for novices: graphical language!

Open problem

An easy way to make forms to input and edit tables and to perform basic queries.

- Web application frameworks such as Ruby-on-Rails and Django.
- ORM mappers such as Hibernate that map relational data to objects.
- Microsoft Access: easy-to-use 'database application creation' GUI.

Historical perspective–1

- Edgar F. Codd introduced the *relational model* in 1970.
- The relational model was a *revolution* for databases:
  from low-level systems resembling analog data management to high-level systems.
- To query relational data, Codd introduced two query languages:
  the *relational algebra* and the *domain calculus*.

See the paper by Codd at `https://doi.org/10.1145/362384.362685`.

Historical perspective–2

- Codd coined the term *relational completeness* of query languages:
  Any query language that can express the queries of the *domain calculus*.

- In 1978, both Bancilhon and Paredaens formalized relational completeness in a language-independent way.

- Early steps in the theoretical study of *databases* and *query languages*:
  Database theory studies *logic* on finite structures (database instances)—
  whereas logic in mathematics is often on infinite structures.

See E.F. Codd, "Relational Completeness of Data Base Sublanguages", 1971.

See the paper by Bancilhon at `https://doi.org/10.1007/3-540-08921-7_60` and the paper by Paredaens at `https://doi.org/10.1016/0020-0190(78)90055-8`.

What is the point of relational algebra?

Relational algebra queries are abstract and simple to manipulate.

What is the point of relational algebra?

Relational algebra queries are abstract and simple to manipulate.

Analogue in mathematics

The functions $f(x) = (x+2)(x-3)$ and $g(x) = x^2 - x - 6$ describe the same *function*.

What is the point of relational algebra?

Relational algebra queries are abstract and simple to manipulate.

Analogue in mathematics

The functions $f(x) = (x+2)(x-3)$ and $g(x) = x^2 - x - 6$ describe the same *function*.

Relational query languages are simple enough

Restricted *non-Turing complete language*: optimization is practical.
Many interesting question about the *properties* of a query are *decidable*.

What is the point of relational algebra?

Relational algebra queries are abstract and simple to manipulate.

Analogue in mathematics

The functions $f(x) = (x + 2)(x - 3)$ and $g(x) = x^2 - x - 6$ describe the same *function*.

Relational query languages are simple enough

Restricted *non-Turing complete language*: optimization is practical.
Many interesting question about the *properties* of a query are *decidable*.

Relational algebra versus SQL

Relational algebra is easy to formally define (2 pages versus thousands of pages for SQL).

What is the point of relational algebra?

Relational algebra queries are abstract and simple to manipulate
$\longrightarrow$ manipulating relational algebra queries is at the basis of efficient query answering.

Analogue in mathematics

The functions $f(x) = (x+2)(x-3)$ and $g(x) = x^2 - x - 6$ describe the same *function*.

Relational query languages are simple enough

Restricted *non-Turing complete language*: optimization is practical.
Many interesting question about the *properties* of a query are *decidable*.

Relational algebra versus SQL

Relational algebra is easy to formally define (2 pages versus thousands of pages for SQL).

A formal grammar of the relational algebra

A formal grammar of the relational algebra

$$e := T \qquad \text{(with } T \text{ a relation name ("table"))}$$

A formal grammar of the relational algebra

$$
\begin{aligned}
e := {} & T && \text{(with } T \text{ a relation name (“table”))} \\
\mid {} & \sigma_c(e) && \textit{selection} \text{ (with } c \text{ a } \textit{condition}\text{)}
\end{aligned}
$$

A formal grammar of the relational algebra

$$
\begin{aligned}
e := {} & T && \text{(with } T \text{ a relation name (“table”))} \\
\mid {} & \sigma_c(e) && \textit{selection} \text{ (with } c \text{ a } \textit{condition}\text{)} \\
\mid {} & \pi_D(e) && \textit{projection} \text{ (with } D \text{ a list of } \textit{columns}\text{)}
\end{aligned}
$$

A formal grammar of the relational algebra

$e := T$	(with T a relation name ("table"))
$\mid \sigma_c(e)$	*selection* (with c a *condition*)
$\mid \pi_D(e)$	*projection* (with D a list of *columns*)
$\mid e \cup e$	*union*
$\mid e \cap e$	*intersection*
$\mid e \setminus e$	*difference*

A formal grammar of the relational algebra

$e := T$	(with T a relation name ("table"))
$\mid \sigma_c(e)$	*selection* (with c a *condition*)
$\mid \pi_D(e)$	*projection* (with D a list of *columns*)
$\mid e \cup e$	*union*
$\mid e \cap e$	*intersection*
$\mid e \setminus e$	*difference*
$\mid \rho_R(e)$	*rename* (with R a *rename specification*)

A formal grammar of the relational algebra

$e := T$	(with T a relation name ("table"))
$\mid \sigma_c(e)$	*selection* (with c a *condition*)
$\mid \pi_D(e)$	*projection* (with D a list of *columns*)
$\mid e \cup e$	*union*
$\mid e \cap e$	*intersection*
$\mid e \setminus e$	*difference*
$\mid \rho_R(e)$	*rename* (with R a *rename specification*)
$\mid e \times e$	*cross product* (Cartesian product)

A formal grammar of the relational algebra

$e := T$	(with T a relation name ("table"))
$\mid \sigma_c(e)$	*selection* (with c a *condition*)
$\mid \pi_D(e)$	*projection* (with D a list of *columns*)
$\mid e \cup e$	*union*
$\mid e \cap e$	*intersection*
$\mid e \setminus e$	*difference*
$\mid \rho_R(e)$	*rename* (with R a *rename specification*)
$\mid e \times e$	*cross product* (Cartesian product)
$\mid e \bowtie_c e$	*join* (with c a *condition*)
$\mid e \bowtie e$	*natural join*

A formal grammar of the relational algebra

$$
\begin{aligned}
e := {} & T && \text{(with } T \text{ a relation name (“table”))} \\
\mid {} & \sigma_c(e) && \textit{selection} \text{ (with } c \text{ a } \textit{condition}\text{)} \\
\mid {} & \pi_D(e) && \textit{projection} \text{ (with } D \text{ a list of } \textit{columns}\text{)} \\
\mid {} & e \cup e && \textit{union} \\
\mid {} & e \cap e && \textit{intersection} \\
\mid {} & e \setminus e && \textit{difference} \\
\mid {} & \rho_R(e) && \textit{rename} \text{ (with } R \text{ a } \textit{rename specification}\text{)} \\
\mid {} & e \times e && \textit{cross product} \text{ (Cartesian product)} \\
\mid {} & e \bowtie_c e && \textit{join} \text{ (with } c \text{ a } \textit{condition}\text{)} \\
\mid {} & e \bowtie e. && \textit{natural join}
\end{aligned}
$$

A formal grammar of the relational algebra

$$
\begin{array}{lll}
e := T & & \text{(with } T \text{ a relation name ("table"))} \\
\quad \mid \sigma_c(e) & & \textit{selection} \text{ (with } c \text{ a } \textit{condition}\text{)} \\
\quad \mid \pi_D(e) & & \textit{projection} \text{ (with } D \text{ a list of } \textit{columns}\text{)} \\
\quad \mid e \cup e & & \textit{union} \\
\quad \mid e \cap e & & \textit{intersection} \\
\quad \mid e \setminus e & & \textit{difference} \\
\quad \mid \rho_R(e) & & \textit{rename} \text{ (with } R \text{ a } \textit{rename specification}\text{)} \\
\quad \mid e \times e & & \textit{cross product} \text{ (Cartesian product)} \\
\quad \mid e \bowtie_c e & & \textit{join} \text{ (with } c \text{ a } \textit{condition}\text{)} \\
\quad \mid e \bowtie e. & & \textit{natural join}
\end{array}
$$

Warning: Relational algebra and SQL

The relational algebra has *set semantics*, not *bag (multiset) semantics*.

Relation name atoms

Syntax of relation name atoms

$$T$$

T must be a valid relation name in the schema of the database D we are querying.

Relation name atoms

Syntax of relation name atoms

$$T$$

T must be a valid relation name in the schema of the database D we are querying.

Semantics of relation name atoms

Let $\mathrm{T} \in I$ be the table named T in instance I of D.
The expression T evaluated over I yields:

$$\mathrm{T}.$$

Relation name atoms: Example

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

courses

Relation name atoms: Example

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output	
cid	title
1	Programming
2	D. Mathematics
3	Databases

courses

The selection operator σ

Syntax of selection

$$\sigma_{\textit{condition } c}(\textit{expression } e)$$

The selection operator σ

Syntax of selection

$$\sigma_{\textit{condition } c}(\textit{expression } e)$$

Semantics of selection

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\sigma_c(e)$ evaluated over I yields:

$$\{r \in \tau \mid \text{condition } c \text{ holds on row } r\}.$$

The selection operator σ

Syntax of selection

$$\sigma_{\textit{condition } c}(\textit{expression } e)$$

Semantics of selection

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\sigma_c(e)$ evaluated over I yields:

$$\{r \in \tau \mid \text{condition } c \text{ holds on row } r\}.$$

Warning: Selection (σ) is not the **SELECT** clause in SQL

Selection does what the **WHERE** clause does in SQL!

The selection operator σ: Example

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$$\sigma_{(\mathrm{cid} \leq 1) \vee (\mathrm{title} = 'Databases')}(\mathrm{courses})$$

The selection operator σ: Example

courses	
<u>cid</u>	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
<u>cid</u>	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output	
cid	title
1	Programming
3	Databases

$$\sigma_{(\mathrm{cid} \leq 1) \vee (\mathrm{title} = '\mathit{Databases}')}(\mathrm{courses})$$

The projection operator π

Syntax of projection

$$\pi_{\mathit{columns}\ D_1,\ldots,D_m}(\mathit{expression}\ e)$$

The projection operator π

Syntax of projection

$$\pi_{\textit{columns } D_1,\ldots,D_m}(\textit{expression } e)$$

Semantics of projection

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I. The expression $\pi_{D_1,\ldots,D_m}(e)$ evaluated over I yields:

$$\{(r[D_1], \ldots, r[D_m]) \mid (r \in \tau) \wedge (D_1, \ldots, D_m \in \{C_1, \ldots, C_n\})\}.$$

We write $r[X]$ to get the value for attribute X in row r.

The projection operator π

Syntax of projection

$$\pi_{columns\ D_1, \ldots, D_m}(expression\ e)$$

Semantics of projection

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\pi_{D_1,\ldots,D_m}(e)$ evaluated over I yields:

$$\{(r[D_1], \ldots, r[D_m]) \mid (r \in \tau) \wedge (D_1, \ldots, D_m \in \{C_1, \ldots, C_n\})\}.$$

Projection and SQL

Projection does what the **SELECT** clause does in SQL!

We write $r[X]$ to get the value for attribute X in row r.

The projection operator π: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$$\pi_{\text{title}}(\text{courses})$$

The projection operator π: Examples

courses

<u>cid</u>	title
1	Programming
2	D. Mathematics
3	Databases

instructors

<u>cid</u>	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output

title
Programming
D. Mathematics
Databases

$$\pi_{\text{title}}(\text{courses})$$

The projection operator π: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output	
title	cid
Programming	1
D. Mathematics	2
Databases	3

$$\pi_{\text{title,cid}}(\text{courses})$$

The set operators ∪, ∩, and \

Syntax of set operators

$$\textit{expression } e_1 \otimes \textit{expression } e_2, \qquad \text{(with } \otimes \text{ a set operator } (\cup, \cap, \text{ or } \setminus))$$

The set operators $\cup$, $\cap$, and $\setminus$

Syntax of set operators

$$\textit{expression } e_1 \otimes \textit{expression } e_2, \qquad \text{(with } \otimes \text{ a set operator (} \cup, \cap, \text{ or } \setminus \text{))}$$

Semantics of set operators

Let $T_1(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $T_2(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \otimes e_2$ evaluated over I yields:

$$T_1 \otimes T_2.$$

The set operators $\cup$, $\cap$, and $\setminus$

Syntax of set operators

$$\textit{expression } e_1 \otimes \textit{expression } e_2, \qquad \text{(with } \otimes \text{ a set operator (}\cup, \cap, \text{ or } \setminus\text{))}$$

Semantics of set operators

Let $T_1(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $T_2(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \otimes e_2$ evaluated over I yields:

$$T_1 \otimes T_2.$$

Set operators and SQL

Relational algebra has no *bag (multiset) semantics*!

The set operators ∪, ∩, and \: Examples

courses	
<u>cid</u>	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
<u>cid</u>	name
2	Eva
3	Alicia
4	Bo

$$\pi_{\text{cid}}(\text{courses}) \cup \pi_{\text{cid}}(\text{instructors})$$

The set operators ∪, ∩, and \: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output
cid
1
2
3
4

$$\pi_{\text{cid}}(\text{courses}) \cup \pi_{\text{cid}}(\text{instructors})$$

The set operators $\cup$, $\cap$, and $\setminus$: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$$\pi_{\text{cid}}(\text{courses}) \cap \pi_{\text{cid}}(\text{instructors})$$

The set operators ∪, ∩, and \: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output
cid
2
3

$$\pi_{\text{cid}}(\text{courses}) \cap \pi_{\text{cid}}(\text{instructors})$$

The set operators ∪, ∩, and \: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$$\pi_{\text{cid}}(\text{courses}) \setminus \pi_{\text{cid}}(\text{instructors})$$

The set operators ∪, ∩, and \: Examples

courses

cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors

cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output

cid
1

$$\pi_{\text{cid}}(\text{courses}) \setminus \pi_{\text{cid}}(\text{instructors})$$

The rename operator ρ–attributes

Syntax of renaming

$$\rho_{\textit{rename specification}\ \mathsf{D} \mapsto \mathsf{D}', \ldots}(\textit{expression } e)$$

The rename operator ρ–attributes

Syntax of renaming

$$\rho_{\textit{rename specification}\ \mathsf{D} \mapsto \mathsf{D}', \ldots}(\textit{expression } e)$$

Semantics of renaming

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\rho_{\mathsf{D} \mapsto \mathsf{D}', \ldots}(e)$ evaluated over I yields:

$$\tau \text{ with column } \mathsf{D} \in \{C_1, \ldots, C_n\} \text{ renamed to } \mathsf{D}' \notin \{C_1, \ldots, C_n\}, \ldots.$$

The rename operator ρ–attributes

Syntax of renaming

$$\rho_{\textit{rename specification}\ \mathrm{D} \mapsto \mathrm{D}', \ldots}(\textit{expression } e)$$

Semantics of renaming

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\rho_{\mathrm{D} \mapsto \mathrm{D}', \ldots}(e)$ evaluated over I yields:

$$\tau \text{ with column } \mathrm{D} \in \{C_1, \ldots, C_n\} \text{ renamed to } \mathrm{D}' \notin \{C_1, \ldots, C_n\}, \ldots.$$

Renaming and SQL

Attribute renaming does what **AS** does in the **SELECT** clause.

The renaming operator ρ–attributes: Example

courses

cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors

cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output

title
Alicia
Bo
D. Mathematics
Databases
Eva
Programming

$$\pi_{\text{title}}(\text{courses}) \cup \pi_{\text{title}}(\rho_{\text{name}\mapsto\text{title}}(\text{instructors}))$$

The rename operator ρ–relations

Syntax of renaming

$$\rho_{\textit{rename specification } R, \ldots}(\textit{expression } e)$$

The rename operator ρ–relations

Syntax of renaming

$$\rho_{\textit{rename specification } R,\ldots}(\textit{expression } e)$$

Semantics of renaming

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\rho_R(e)$ evaluated over I yields:

τ with the table τ renamed to R.

The rename operator ρ–relations

Syntax of renaming

$$\rho_{\textit{rename specification } R,\ldots}(\textit{expression } e)$$

Semantics of renaming

Let $\tau(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\rho_R(e)$ evaluated over I yields:

τ with the table τ renamed to R.

Renaming and SQL

Relation renaming does what **AS** does in the **FROM** clause.

The rename operator ρ–shorthand notation

Syntax of renaming

$$\rho_{\textit{rename specification}}\, R(D_1, \ldots, D_n)\,(\textit{expression}\ e)$$

The rename operator ρ–shorthand notation

Syntax of renaming

$$\rho_{\textit{rename specification } R(D_1,\ldots,D_n)}(\textit{expression } e)$$

Semantics of renaming

Let $\tau(C_1,\ldots,C_n)$ be the n-ary table obtained from evaluating e over some instance I.
The expression $\rho_R(e)$ evaluated over I yields:

> τ with the table τ renamed to R and the columns $C_1,\ldots,C_n$ renamed to $D_1,\ldots,D_n$.

The rename operator ρ–shorthand notation

Syntax of renaming

$$\rho_{\textit{rename specification } R(D_1,\ldots,D_n)}(\textit{expression } e)$$

Semantics of renaming

Let $\tau(C_1, \ldots, C_n)$ be the table obtained from evaluating e over some instance I.
The expression $\rho_R(e)$ evaluated over I yields:

τ with the table τ renamed to R and the columns $C_1, \ldots, C_n$ renamed to $D_1, \ldots, D_n$.

Shorthand notation

$$\rho_{R(D_1,\ldots,D_n)}(e) \equiv \rho_{C_1 \mapsto D_1,\ldots,C_n \mapsto D_n}(\rho_R(e)).$$

The cross-product operator $\times$

Syntax of the cross-product operator

$$\textit{expression } e_1 \times \textit{expression } e_2$$

The cross-product operator $\times$

Syntax of the cross-product operator

$$\textit{expression } e_1 \times \textit{expression } e_2$$

Semantics of the cross-product operator

Let $T(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \dots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \times e_2$ evaluated over I yields:

$$\{(r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m]) \mid (r \in T) \land (s \in U)\}.$$

The cross-product operator ×: Examples

courses	
<u>cid</u>	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
<u>cid</u>	name
2	Eva
3	Alicia
4	Bo

$$\text{courses} \times \mathit{instructors}$$

The cross-product operator ×: Examples

courses	
<u>cid</u>	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
<u>cid</u>	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output			
cid	title	cid	name
1	Programming	2	Eva
1	Programming	3	Alicia
1	Programming	4	Bo
	$\vdots$		
3	Databases	4	Bo

$$\text{courses} \times \textit{instructors}$$

The cross-product operator ×: Examples

courses

cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors

cid	name
2	Eva
3	Alicia
4	Bo

⟶

Query output

cid	title	cid	name
1	Programming	2	Eva
1	Programming	3	Alicia
1	Programming	4	Bo
	⋮		
3	Databases	4	Bo

$$\text{courses} \times \textit{instructors}$$

Attribute name conflicts: refer to attributes via *table_name.attribute*.

The cross-product operator ×: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$$\sigma_{\text{courses.cid=instructors.cid}}(\text{courses} \times \textit{instructors})$$

Attribute name conflicts: refer to attributes via *table_name.attribute*.

The cross-product operator $\times$: Examples

courses

cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors

cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output

cid	title	cid	name
2	D. Mathematics	2	Eva
3	Databases	3	Alicia

$$\sigma_{\text{courses.cid}=\text{instructors.cid}}(\text{courses} \times \textit{instructors})$$

Attribute name conflicts: refer to attributes via *table_name.attribute*.

The cross-product operator ×: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output			
cid	title	cid	name
2	D. Mathematics	2	Eva
3	Databases	3	Alicia

$$\sigma_{\text{C.cid=I.cid}}(\rho_C(\text{courses}) \times \rho_I(\textit{instructors}))$$

Attribute name conflicts: refer to attributes via *table_name.attribute*.

The cross-product operator $\times$: Examples

courses	
<u>cid</u>	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
<u>cid</u>	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output	
title	name
D. Mathematics	Eva
Databases	Alicia

$$\pi_{\text{C.title},\text{I.name}}(\sigma_{\text{C.cid}=\text{I.cid}}(\rho_C(\text{courses}) \times \rho_I(\textit{instructors})))$$

Attribute name conflicts: refer to attributes via *table_name.attribute*.

The conditional join operator $\bowtie$

Syntax of the conditional join operator

$$\textit{expression } e_1 \bowtie_{\textit{condition } c} \textit{expression } e_2$$

The conditional join operator $\bowtie$

Syntax of the conditional join operator

$$\textit{expression } e_1 \bowtie_{\textit{condition } c} \textit{expression } e_2$$

Semantics of the conditional join operator

Let $T(C_1, \ldots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \ldots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \times e_2$ evaluated over I yields:

$$\{(r[C_1], \ldots, r[C_n], s[D_1], \ldots, s[D_m]) \mid (r \in T) \land (s \in U) \land \\ \text{condition } c \text{ holds on row } (r[C_1], \ldots, r[C_n], s[D_1], \ldots, s[D_m])\}.$$

The conditional join operator $\bowtie$

Syntax of the conditional join operator

$$\textit{expression } e_1 \bowtie_{\textit{condition } c} \textit{expression } e_2$$

Semantics of the conditional join operator

Let $T(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \dots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \times e_2$ evaluated over I yields:

$$\{(r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m]) \mid (r \in T) \wedge (s \in U) \wedge \\ \text{condition } c \text{ holds on row } (r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m])\}.$$

Shorthand notation

$$e_1 \bowtie_c e_2 \equiv \sigma_c(e_1 \times e_2).$$

The natural join operator $\bowtie$

Syntax of the natural join operator

$$\textit{expression}\ e_1 \bowtie \textit{expression}\ e_2$$

The natural join operator $\bowtie$

Syntax of the natural join operator

$$\textit{expression } e_1 \bowtie \textit{expression } e_2$$

Semantics of the conditional join operator

Let $T(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \dots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \times e_2$ evaluated over I yields:

$$\{(r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m]) \mid (r \in T) \land (s \in U) \land \\ r[E] = s[E] \text{ for all } E \in (\{C_1, \dots, C_n\} \cap \{D_1, \dots, D_m\})\}.$$

The natural join operator $\bowtie$

Syntax of the natural join operator

$$\textit{expression } e_1 \bowtie \textit{expression } e_2$$

Semantics of the conditional join operator

Let $T(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \dots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \times e_2$ evaluated over I yields:

$$\{(r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m]) \mid (r \in T) \land (s \in U) \land \\ r[E] = s[E] \text{ for all } E \in (\{C_1, \dots, C_n\} \cap \{D_1, \dots, D_m\})\}.$$

Shorthand notation

$e_1 \bowtie e_2 \equiv e_1 \bowtie_c e_2$ with c holding equalities between all shared columns.

The conditional join and natural join operators ⋈: Examples

courses

cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors

cid	name
2	Eva
3	Alicia
4	Bo

⟶

Query output

title	name
D. Mathematics	Eva
Databases	Alicia

$$\pi_{\text{C.title,I.name}}(\sigma_{\text{C.cid=I.cid}}(\rho_C(\text{courses}) \times \rho_I(\text{instructors})))$$

The conditional join and natural join operators ⋈: Examples

courses	
cid	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
cid	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output	
title	name
D. Mathematics	Eva
Databases	Alicia

$$\pi_{\text{C.title,I.name}}(\rho_C(\text{courses}) \bowtie_{\text{C.cid=I.cid}} \rho_I(\text{instructors}))$$

The conditional join and natural join operators ⋈: Examples

courses	
<u>cid</u>	title
1	Programming
2	D. Mathematics
3	Databases

instructors	
<u>cid</u>	name
2	Eva
3	Alicia
4	Bo

$\longrightarrow$

Query output	
title	name
D. Mathematics	Eva
Databases	Alicia

$$\pi_{\text{C.title,I.name}}(\rho_C(\text{courses}) \bowtie \rho_I(\text{instructors}))$$

The extended projection operator π

Projections can be used to express computations.

students		
sid	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

$$\pi_{\text{sid}+\text{year}\mapsto X,\text{name}}(\text{students}).$$

The extended projection operator π

Projections can be used to express computations.

students		
<u>sid</u>	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

$\longrightarrow$

Query output	
X	name
2021	Alicia
2021	Celeste
2023	Dafni

$$\pi_{\mathrm{sid+year} \mapsto X, \mathrm{name}}(\mathrm{students}).$$

Linear notation

Shorthand notation to denote long relational algebra expressions.

Linear notation

Shorthand notation to denote long relational algebra expressions.

Example

- $X := \rho_C(\text{courses}) \times \rho_I(\text{instructors});$
- $Y := \sigma_{\text{C.cid=I.cid}}(X);$
- $\pi_{\text{C.title,I.name}}(Y).$

Pattern: At-least n

courses

cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture at-least two courses.

Pattern: At-least n

courses		
cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture at-least two courses.

- $X := \rho_{C_1}(\text{courses}) \times \rho_{C_2}(\text{courses});$ (Combine pairs of courses (C_1, C_2))

Pattern: At-least n

courses		
cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture at-least two courses.

- $X := \rho_{C_1}(\text{courses}) \times \rho_{C_2}(\text{courses});$ (Combine pairs of courses (C_1, C_2))
- $Y := \sigma_{C_1.\text{cid} \neq C_2.\text{cid}}(X);$ (Keep courses from C_1 *different* from C_2)

Pattern: At-least *n*

courses

cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture at-least two courses.

- $X := \rho_{C_1}(\text{courses}) \times \rho_{C_2}(\text{courses});$ (Combine pairs of courses (C_1, C_2))
- $Y := \sigma_{C_1.\text{cid} \neq C_2.\text{cid}}(X);$ (Keep courses from C_1 *different* from C_2)
- $Z := \sigma_{C_1.\text{lecturer} = C_2.\text{lecturer}}(Y);$ (Only keep pairs with the *same* lecturer)

Pattern: At-least *n*

courses

cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

$\longrightarrow$

Query output

title
Databases
Advanced Databases

Return courses with lectures that lecture at-least two courses.

- $X := \rho_{C_1}(\text{courses}) \times \rho_{C_2}(\text{courses});$ (Combine pairs of courses (C_1, C_2))
- $Y := \sigma_{C_1.\text{cid} \neq C_2.\text{cid}}(X);$ (Keep courses from C_1 *different* from C_2)
- $Z := \sigma_{C_1.\text{lecturer} = C_2.\text{lecturer}}(Y);$ (Only keep pairs with the *same* lecturer)
- $\pi_{C_1.\text{title}}(Z).$

Pattern: Exact *n*

courses

cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture exactly one course.

Pattern: Exact n

courses		
cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture exactly one course.

- X := courses with lectures that lecture at-least two courses;

Pattern: Exact *n*

courses

cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture exactly one course.

- X := courses with lectures that lecture at-least two courses;
- Y := courses with lectures that lecture at-least one courses;

Pattern: Exact n

courses		
cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture exactly one course.

- X := courses with lectures that lecture at-least two courses;
- Y := courses with lectures that lecture at-least one courses;
- $Z := Y \setminus X$;

Pattern: Exact n

courses		
cid	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

Return courses with lectures that lecture exactly one course.

- X := courses with lectures that lecture at-least two courses;
- Y := courses with lectures that lecture at-least one courses;
- $Z := Y \setminus X$;
- $\pi_{\text{title}}(Z)$.

Pattern: Exact *n*

courses		
<u>cid</u>	title	lecturer
1	Programming	1
2	Discrete Mathematics	3
3	Databases	2
4	Advanced Databases	2

$\longrightarrow$

Query output
title
Programming
Discrete Mathematics

Return courses with lectures that lecture exactly one course.

- X := courses with lectures that lecture at-least two courses;
- Y := courses with lectures that lecture at-least one courses;
- $Z := Y \setminus X$;
- $\pi_{\text{title}}(Z)$.

Pattern: Largest value

students

<u>sid</u>	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

Return students from the latest year.

Pattern: Largest value

students		
sid	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

Return students from the latest year.

- $X := \rho_{S_1}(\text{students}) \times \rho_{S_2}(\text{students});$ (Combine pairs of students (S_1, S_2))

Pattern: Largest value

students		
sid	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

Return students from the latest year.

- $X := \rho_{S_1}(\text{students}) \times \rho_{S_2}(\text{students});$ (Combine pairs of students (S_1, S_2))
- $Y := \sigma_{S_1.\text{year} < S_2.\text{year}}(X);$ (Keep students from S_1 that are *not* from the latest year)

Pattern: Largest value

students		
sid	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

Return students from the latest year.

- $X := \rho_{S_1}(\text{students}) \times \rho_{S_2}(\text{students});$ (Combine pairs of students (S_1, S_2))
- $Y := \sigma_{S_1.\text{year} < S_2.\text{year}}(X);$ (Keep students from S_1 that are *not* from the latest year)
- $Z := \pi_{\text{sid}}(\text{students}) \setminus \pi_{S_1.\text{sid}}(Y);$ (Keep students *not* in S_1)

Pattern: Largest value

students

<u>sid</u>	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

Return students from the latest year.

- $X := \rho_{S_1}(\text{students}) \times \rho_{S_2}(\text{students});$ (Combine pairs of students (S_1, S_2))
- $Y := \sigma_{S_1.\text{year} < S_2.\text{year}}(X);$ (Keep students from S_1 that are *not* from the latest year)
- $Z := \pi_{\text{sid}}(\text{students}) \setminus \pi_{S_1.\text{sid}}(Y);$ (Keep students *not* in S_1)
- $\pi_{\text{name}}(\text{students} \bowtie Z)$

Pattern: Largest value

students		
sid	name	year
1	Alicia	2020
3	Celeste	2018
4	Dafni	2019

$\longrightarrow$

Query output
name
Alicia

Return students from the latest year.

- $X := \rho_{S_1}(\text{students}) \times \rho_{S_2}(\text{students});$ (Combine pairs of students (S_1, S_2))
- $Y := \sigma_{S_1.\text{year} < S_2.\text{year}}(X);$ (Keep students from S_1 that are *not* from the latest year)
- $Z := \pi_{\text{sid}}(\text{students}) \setminus \pi_{S_1.\text{sid}}(Y);$ (Keep students *not* in S_1)
- $\pi_{\text{name}}(\text{students} \bowtie Z)$

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

Semantics of the conditional join operator

Let table $T(C_1, \dots, C_n, D_1, \dots, D_m)$ be obtained from evaluating e_1 over some instance I.
Let table $U(D_1, \dots, D_m)$ be obtained from evaluating e_2 over some instance I.
The expression $e_1 \div e_2$ evaluated over I yields:

$$\{(r[C_1], \dots, r[C_n]) \mid (r \in T) \land \forall s\, ((s \in U) \implies ((r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m]) \in T))\}.$$

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

What?

Consider two tables:

- **enroll_in**(student, course)
  keeps track of all enrollments of students in courses.
- **core_courses**(course)
  list all core courses.

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

What?

Consider two tables:

- **enroll_in**(student, course)
  keeps track of all enrollments of students in courses.
- **core_courses**(course)
  list all core courses.

The query

$$\text{enroll\_in} \div \text{core\_courses}$$

returns only the students that enrolled in all core courses.

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

Expressing $e_1 \div e_2$.

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

Expressing $e_1 \div e_2$.

- $X := \pi_{C_1,\dots,C_n}(e_1) \times e_2$; (Pair each $(r[C_1], \dots, r[C_n]) \in T$ with all $s \in U$)

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

Expressing $e_1 \div e_2$.

- $X := \pi_{C_1,\dots,C_n}(e_1) \times e_2$; (Pair each $(r[C_1],\dots,r[C_n]) \in T$ with all $s \in U$)
- $Y := X \setminus e_1$; (Keep all $r \in T$ that are *not complete*)

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

Expressing $e_1 \div e_2$.

- $X := \pi_{C_1,\dots,C_n}(e_1) \times e_2$; (Pair each $(r[C_1], \dots, r[C_n]) \in T$ with all $s \in U$)
- $Y := X \setminus e_1$; (Keep all $r \in T$ that are *not complete*)
- $Z := \pi_{C_1,\dots,C_n}(Y)$; (Only keeps the columns $C_1, \dots C_n$ of incomplete $r \in T$).

Pattern: Division (or quotient)

Syntax of the division operator

$$\textit{expression } e_1 \div \textit{expression } e_2$$

Expressing $e_1 \div e_2$.

- $X := \pi_{C_1,\dots,C_n}(e_1) \times e_2$; (Pair each $(r[C_1], \dots, r[C_n]) \in T$ with all $s \in U$)
- $Y := X \setminus e_1$; (Keep all $r \in T$ that are *not complete*)
- $Z := \pi_{C_1,\dots,C_n}(Y)$; (Only keeps the columns $C_1, \dots C_n$ of incomplete $r \in T$).
- $\pi_{C_1,\dots,C_n}(e_1) \setminus Z$. ("Complement of Z")

  (Keep columns $C_1, \dots, C_n$ of all $r \in T$ that are *complete*)

Exercise: Unique course title

courses(<u>cid</u>, title, lecturer).

Write a relational algebra query that returns all non-unique course titles.

Exercise: Unique course title

courses(<u>cid</u>, title, lecturer).

Write a relational algebra query that returns all non-unique course titles.

At-least two courses with the same title

$$\pi_{C_1.\text{title}}(\sigma_{C_1.\text{cid} \neq C_2.\text{cid} \wedge C_1.\text{title} = C_2.\text{title}}(\rho_{C_1}(\mathit{courses}) \times \rho_{C_2}(\mathit{courses})))$$

Exercise: Unique course title

courses(<u>cid</u>, title, lecturer).

Write a relational algebra query that returns all non-unique course titles.

At-least two courses with the same title

$$\pi_{C_1.\text{title}}(\rho_{C_1}(\mathit{courses}) \bowtie_{C_1.\text{cid} \neq C_2.\text{cid} \wedge C_1.\text{title} = C_2.\text{title}} \rho_{C_2}(\mathit{courses}))$$

Exercise: Enrollment interval

enroll_in(<u>student</u>, <u>course</u>, year)

Return, for each student, the first and last year they enrolled.

Exercise: Enrollment interval

enroll_in(<u>student</u>, <u>course</u>, year)

Return, for each student, the first and last year they enrolled.

For each student, the first year they enrolled.

Exercise: Enrollment interval

enroll_in(<u>student</u>, <u>course</u>, year)

Return, for each student, the first and last year they enrolled.

For each student, the first year they enrolled.

- $X_{\text{not-first}} := \sigma_{E_1.\text{student}=E_2.\text{student} \wedge E_1.\text{year}>E_2.\text{year}}(\rho_{E_1}(\text{enroll\_in}) \times \rho_{E_2}(\text{enroll\_in}));$

Exercise: Enrollment interval

enroll_in(<u>student</u>, <u>course</u>, year)

Return, for each student, the first and last year they enrolled.

For each student, the first year they enrolled.

- $X_{\text{not-first}} := \sigma_{E_1.\text{student}=E_2.\text{student} \wedge E_1.\text{year}>E_2.\text{year}}(\rho_{E_1}(\text{enroll\_in}) \times \rho_{E_2}(\text{enroll\_in}));$
- $Y_{\text{first}} := \pi_{\text{student},\text{year}}(\mathit{enroll\_in}) \setminus \pi_{E_1.\text{student},E_1.\text{year}}(X_{\text{not-first}}).$

Exercise: Enrollment interval

enroll_in(<u>student</u>, <u>course</u>, year)

Return, for each student, the first and last year they enrolled.

For each student, the first year they enrolled.

- $X_{\text{not-first}} := \sigma_{E_1.\text{student}=E_2.\text{student} \land E_1.\text{year}>E_2.\text{year}}(\rho_{E_1}(\text{enroll\_in}) \times \rho_{E_2}(\text{enroll\_in}))$;
- $Y_{\text{first}} := \pi_{\text{student},\text{year}}(\mathit{enroll\_in}) \setminus \pi_{E_1.\text{student},E_1.\text{year}}(X_{\text{not-first}})$.

Y_{last} The *last* year they enrolled: swap E_1.year $>$ E_2.year for E_1.year $<$ E_2.year.

Exercise: Enrollment interval

enroll_in(student, course, year)

Return, for each student, the first and last year they enrolled.

For each student, the first year they enrolled.

- $X_{\text{not-first}} := \sigma_{E_1.\text{student}=E_2.\text{student} \wedge E_1.\text{year}>E_2.\text{year}}(\rho_{E_1}(\text{enroll\_in}) \times \rho_{E_2}(\text{enroll\_in}));$
- $Y_{\text{first}} := \pi_{\text{student},\text{year}}(\mathit{enroll\_in}) \setminus \pi_{E_1.\text{student},E_1.\text{year}}(X_{\text{not-first}}).$

Y_{last} The *last* year they enrolled: swap $E_1.\text{year} > E_2.\text{year}$ for $E_1.\text{year} < E_2.\text{year}$.

$$\pi_{F.\text{student},F.\text{year},L.\text{year}}(\sigma_{F.\text{student}=L.\text{student}}(\rho_F(Y_{\text{first}}) \times \rho_L(Y_{\text{last}})))$$

Exercise: Enrollment interval

enroll_in(<u>student</u>, <u>course</u>, year)

Return, for each student, the first and last year they enrolled.

For each student, the first year they enrolled.

- $X_{\text{not-first}} := \sigma_{E_1.\text{student}=E_2.\text{student} \land E_1.\text{year}>E_2.\text{year}}(\rho_{E_1}(\text{enroll\_in}) \times \rho_{E_2}(\text{enroll\_in}));$
- $Y_{\text{first}} := \pi_{\text{student},\text{year}}(\mathit{enroll\_in}) \setminus \pi_{E_1.\text{student},E_1.\text{year}}(X_{\text{not-first}}).$

Y_{last} The *last* year they enrolled: swap $E_1.\text{year} > E_2.\text{year}$ for $E_1.\text{year} < E_2.\text{year}$.

$$\rho_{R(\text{student},\text{first\_year},\text{last\_year})}(\pi_{F.\text{student},F.\text{year},L.\text{year}}(\sigma_{F.\text{student}=L.\text{student}}(\rho_F(Y_{\text{first}}) \times \rho_L(Y_{\text{last}}))))$$

Relational algebra and efficient query evaluation

Consider the following basic SQL query

```
SELECT C.title
FROM students S, enroll_in E, courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND S.name = 'Dafni';
```

Relational algebra and efficient query evaluation

Consider the following basic SQL query

```
SELECT C.title
FROM students S, enroll_in E, courses C
WHERE S.sid = E.sid AND E.cid = C.cid AND S.name = 'Dafni';
```

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{`}Dafni\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \land E.\text{cid}=C.\text{cid} \land S.\text{name}=\text{'}Dafni\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

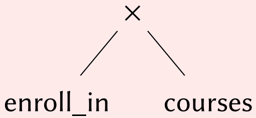
An abstract execution plan

Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid}\wedge E.\text{cid}=C.\text{cid}\wedge S.\text{name}=\text{`}\textit{Dafni}\text{'}}(\rho_S(\text{students})\times\rho_E(\text{enroll\_in})\times\rho_C(\text{courses}))).$$

An abstract execution plan

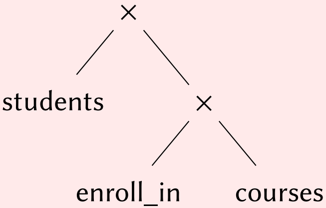


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{'}\textit{Dafni}\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

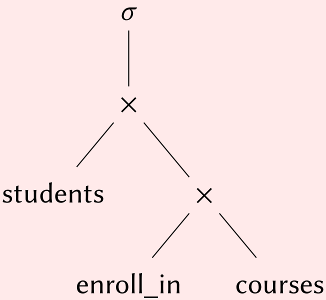


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{`}\mathit{Dafni}\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

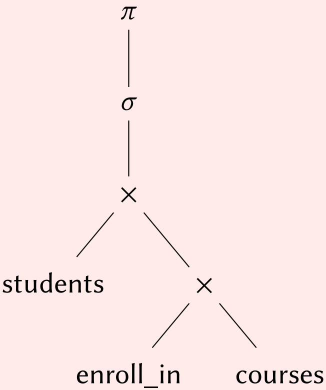


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{'}\mathit{Dafni}\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

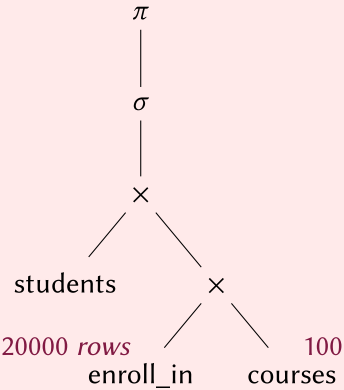


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=`\textit{Dafni}'}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan



20 courses/student, 1000 students, 20000 rows

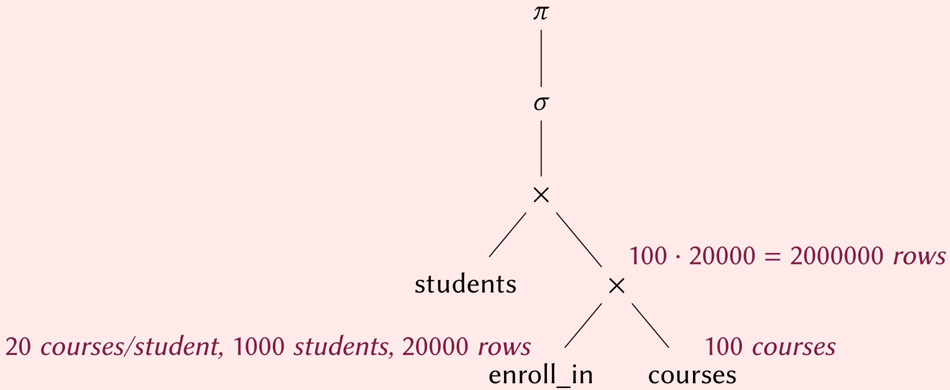
100 courses

Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{`}\mathit{Dafni}\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

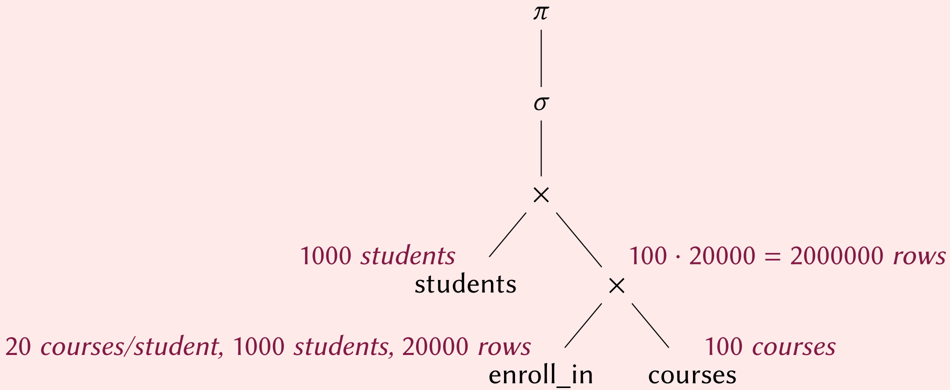


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid}\wedge E.\text{cid}=C.\text{cid}\wedge S.\text{name}=\text{‘}Dafni\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

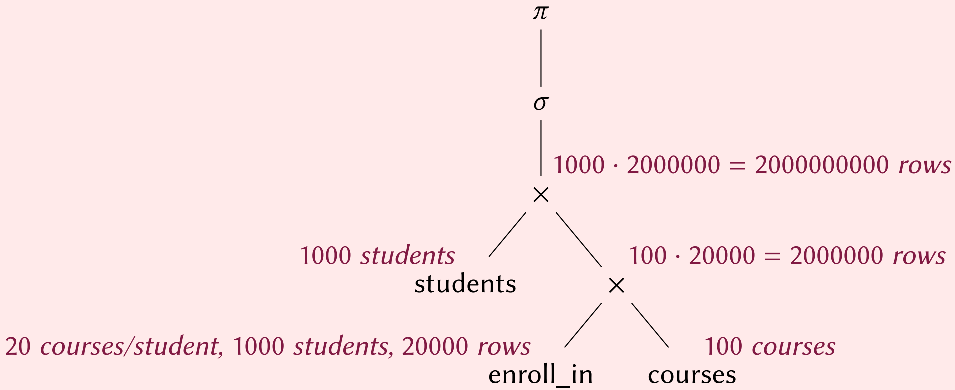


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{'}\mathit{Dafni}\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

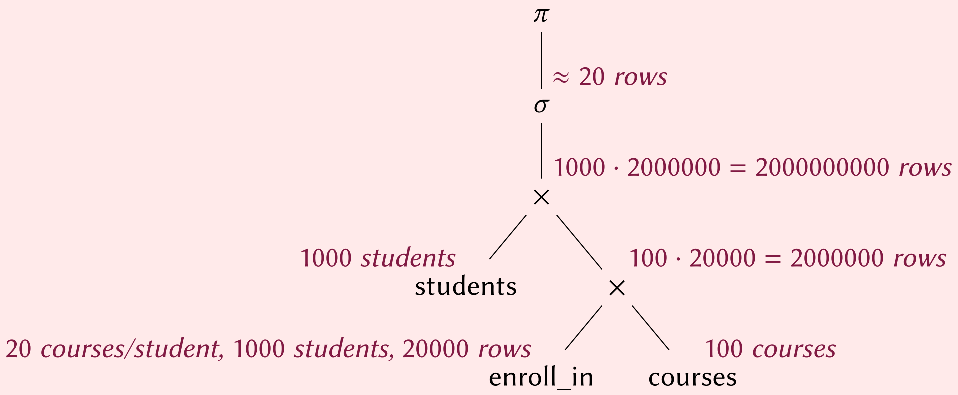


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \land E.\text{cid}=C.\text{cid} \land S.\text{name}=\text{‘}\mathit{Dafni}\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

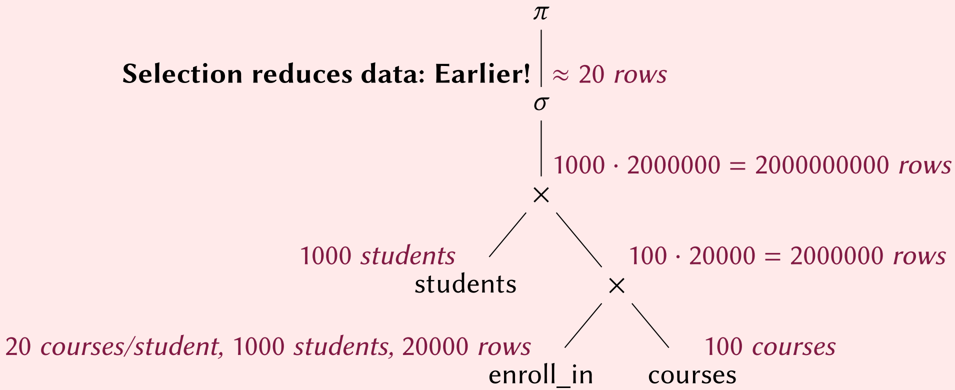


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \land E.\text{cid}=C.\text{cid} \land S.\text{name}=\text{‘}Dafni\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An abstract execution plan

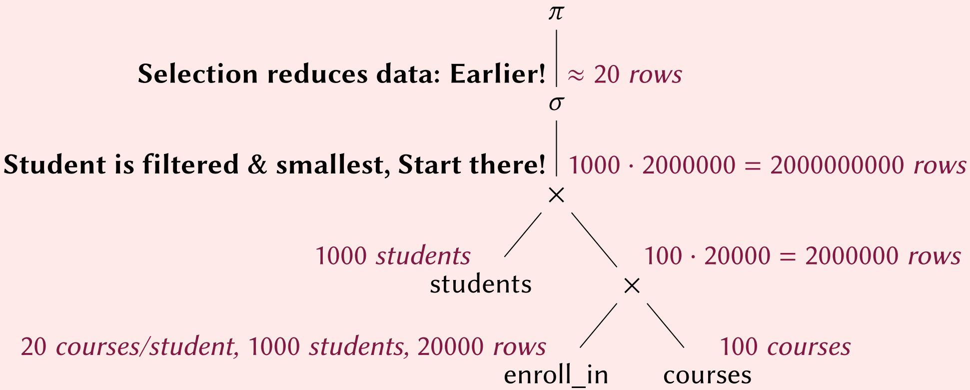


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid}\wedge E.\text{cid}=C.\text{cid}\wedge S.\text{name}=\text{`}\mathit{Dafni}\text{'}}(\rho_S(\text{students})\times\rho_E(\text{enroll\_in})\times\rho_C(\text{courses}))).$$

An abstract execution plan



Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid}\wedge E.\text{cid}=C.\text{cid}\wedge S.\text{name}=\text{`}\textit{Dafni}\text{'}}(\rho_S(\text{students})\times\rho_E(\text{enroll\_in})\times\rho_C(\text{courses}))).$$

An improved execution plan

1000 *students*

students

Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{‘}\mathit{Dafni}\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An improved execution plan

σ

1000 *students*

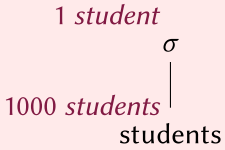
students

Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid}\wedge E.\text{cid}=C.\text{cid}\wedge S.\text{name}=\text{'}\mathit{Dafni}\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An improved execution plan

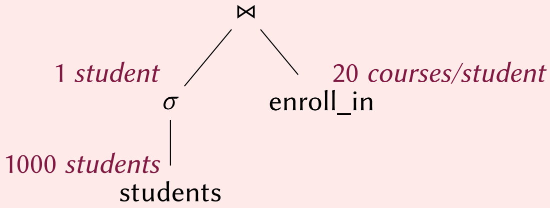


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{‘}\mathit{Dafni}\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An improved execution plan

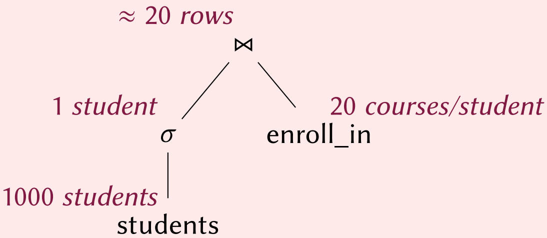


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{‘}\mathit{Dafni}\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An improved execution plan

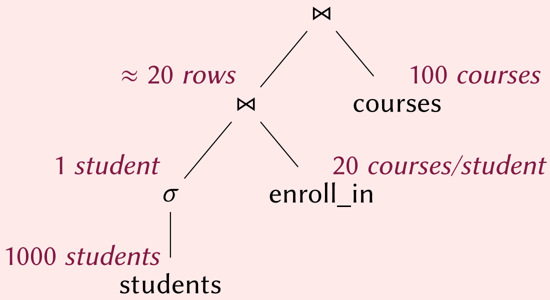


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid}\wedge E.\text{cid}=C.\text{cid}\wedge S.\text{name}=\text{`}\mathit{Dafni}\text{'}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An improved execution plan

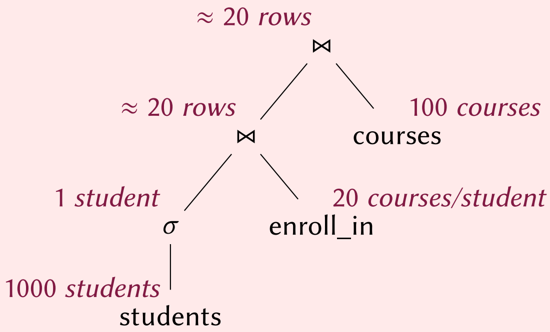


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{‘}Dafni\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An improved execution plan

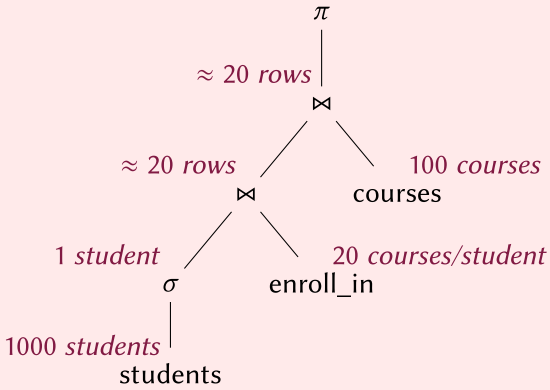


Relational algebra and efficient query evaluation

A basic query in relational algebra

$$\pi_{\text{title}}(\sigma_{S.\text{sid}=E.\text{sid} \wedge E.\text{cid}=C.\text{cid} \wedge S.\text{name}=\text{‘}\mathit{Dafni}\text{’}}(\rho_S(\text{students}) \times \rho_E(\text{enroll\_in}) \times \rho_C(\text{courses}))).$$

An improved execution plan



Optimizing query evaluation

- Basic always-valid rewrite rules: “push down selection” (& “push down projection”).
- Reordering joins: influences by guesstimates of input and output sizes.
- Choosing specific algorithms and indices: Huge impact on joins.
  E.g., materializing intermediate tables versus pipeline design.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$.
- $0 \leq |\pi_D(T)| \leq m$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$.
- $0 \leq |\pi_D(T)| \leq m$.
- $\max(m, n) \leq |T \cup U| \leq m + n$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$.
- $0 \leq |\pi_D(T)| \leq m$.
- $\max(m, n) \leq |T \cup U| \leq m + n$.
- $0 \leq |T \cap U| \leq \min(m, n)$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$.
- $0 \leq |\pi_D(T)| \leq m$.
- $\max(m, n) \leq |T \cup U| \leq m + n$.
- $0 \leq |T \cap U| \leq \min(m, n)$.
- $\max(m - n, 0) \leq |T \setminus U| \leq m$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$.
- $0 \leq |\pi_D(T)| \leq m$.
- $\max(m, n) \leq |T \cup U| \leq m + n$.
- $0 \leq |T \cap U| \leq \min(m, n)$.
- $\max(m - n, 0) \leq |T \setminus U| \leq m$.
- $0 \leq |T \bowtie_C U| \leq m \cdot n$.

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$. Typical databases *really* try to get a better guess!
- $0 \leq |\pi_D(T)| \leq m$.
- $\max(m, n) \leq |T \cup U| \leq m + n$.
- $0 \leq |T \cap U| \leq \min(m, n)$.
- $\max(m - n, 0) \leq |T \setminus U| \leq m$.
- $0 \leq |T \bowtie_C U| \leq m \cdot n$. Typical databases *really* try to get a better guess!

Optimization: Estimate query sizes

Let T, U be tables with $|T| = m$ and $|U| = n$.

Exact estimates

- $|T| = m$.
- $|\rho_R(T)| = m$.
- $|T \times U| = m \cdot n$.

Estimates

- $0 \leq |\sigma_C(T)| \leq m$. Typical databases *really* try to get a better guess!
- $0 \leq |\pi_D(T)| \leq m$. Bag (multiset) semantics: $|\pi_D(T)| = m$.
- $\max(m, n) \leq |T \cup U| \leq m + n$.
- $0 \leq |T \cap U| \leq \min(m, n)$.
- $\max(m - n, 0) \leq |T \setminus U| \leq m$.
- $0 \leq |T \bowtie_C U| \leq m \cdot n$. Typical databases *really* try to get a better guess!

A special type of joins: The semi-joins

Syntax of the semi-join operators

$\textit{expression } e_1 \otimes_{\textit{condition } c} \textit{expression } e_2$ (with $\otimes$ a semi-join operator ($\ltimes$ or $\rtimes$))

A special type of joins: The semi-joins

Syntax of the semi-join operators

expression $e_1 \otimes_{\textit{condition } c}$ *expression* e_2 (with $\otimes$ a semi-join operator ($\ltimes$ or $\rtimes$))

Semantics of the semi-join operator

Let $T(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \dots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \ltimes_c e_2$ evaluated over I yields:

$$\{r \mid (r \in T) \land (s \in U) \land \text{condition } c \text{ holds on row } (r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m])\}.$$

A special type of joins: The semi-joins

Syntax of the semi-join operators

$$\textit{expression } e_1 \otimes_{\textit{condition } c} \textit{expression } e_2 \qquad \text{(with } \otimes \text{ a semi-join operator } (\ltimes \text{ or } \rtimes))$$

Semantics of the semi-join operator

Let $T(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \dots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \rtimes_c e_2$ evaluated over I yields:

$$\{s \mid (r \in T) \land (s \in U) \land \text{condition } c \text{ holds on row } (r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m])\}.$$

A special type of joins: The semi-joins

Syntax of the semi-join operators

$\textit{expression } e_1 \otimes_{\textit{condition } c} \textit{expression } e_2$ (with $\otimes$ a semi-join operator ($\ltimes$ or $\rtimes$))

Semantics of the semi-join operator

Let $T(C_1, \dots, C_n)$ be the n-ary table obtained from evaluating e_1 over some instance I.
Let $U(D_1, \dots, D_m)$ be the m-ary table obtained from evaluating e_2 over some instance I.
The expression $e_1 \rtimes_c e_2$ evaluated over I yields:

$$\{s \mid (r \in T) \land (s \in U) \land \text{condition } c \text{ holds on row } (r[C_1], \dots, r[C_n], s[D_1], \dots, s[D_m])\}.$$

Semi-joins and SQL

Implement **IN** subqueries (and equivalent joins)!

Extending the relational algebra

SQL versus relational algebra

- Adding aggregation.
- Introducing `NULL` values.
- Set versus bag (multiset) semantics.

Extending the relational algebra

SQL versus relational algebra

- Adding aggregation.
- Introducing `NULL` values.
- Set versus bag (multiset) semantics.

We will use only the basic relational algebra for the assignment!

Extension: Aggregation

Syntax of the aggregation operators

$$\gamma_{\textit{group-by columns},\ \textit{aggregated columns}}(\textit{expression } e)$$

Extension: Aggregation

Syntax of the aggregation operators

$$\gamma_{\textit{group-by columns}, \textit{aggregated columns}}(\textit{expression } e)$$

Example

productreview

user	product	rating
Alicia	cheese	10
Alicia	phone	5
Eva	cheese	9
Eva	shoe	8
Bo	phone	3
Bo	shoe	5
Celeste	cheese	7

$$\gamma_{\text{product}, m:=\max(\text{rating}), n:=\min(\text{rating})}(\text{productreview})$$

Extension: Aggregation

Syntax of the aggregation operators

$$\gamma_{\textit{group-by columns}, \textit{aggregated columns}}(\textit{expression } e)$$

Example

$$\gamma_{\text{product}, m:=\max(\text{rating}), n:=\min(\text{rating})}(\text{productreview})$$

productreview

user	product	rating
Alicia	cheese	10
Alicia	phone	5
Eva	cheese	9
Eva	shoe	8
Bo	phone	3
Bo	shoe	5
Celeste	cheese	7

Query output

product	m	n
cheese	10	7
phone	5	3
shoe	8	5

Extension: Bag (multiset) semantics

Similar to SQL.

Extension: Bag (multiset) semantics

Similar to SQL.

Impact on operators

Unaffected	σ_c, ρ_R, $\times$, $\bowtie_C$, $\bowtie$.
Affected	$\cup$, $\cap$, $\setminus$, γ ($\ltimes$, $\rtimes$).
Strongly affected	π_D.

Extension: Bag (multiset) semantics

Similar to SQL.

Impact on operators

Unaffected	σ_c, ρ_R, $\times$, $\bowtie_C$, $\bowtie$.
Affected	$\cup$, $\cap$, $\setminus$, γ ($\ltimes$, $\rtimes$).
Strongly affected	π_D.

Syntax of the deduplication operator

$$\delta(\textit{expression } e)$$