Relational Algebra

- After completing this chapter, you should be able to
  - ▷ enumerate and explain the operations of **relational algebra** (there is a **core** of 5 relational algebra operators),
  - ▷ write relational algebra queries of the type **join–select–project,**
  - ▷ discuss **correctness and equivalence** of given relational algebra queries.

Relational Algebra

Overview

Example Database (recap)

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Relational Algebra (1)

- **Relational algebra (RA)** is a **query language** for the relational model with a solid theoretical foundation.
- Relational algebra is *not* visible at the user interface level (not in any commercial RDBMS, at least).
- However, almost any RDBMS uses RA to **represent queries internally** (for **query optimization** and **execution**).
- Knowledge of relational algebra will help in understanding SQL and relational database systems in general.

Relational Algebra (2)

- In mathematics, an **algebra** is a
  - ▷ **set** (the carrier), and
  - ▷ **operations** that are closed with respect to the set.
- Example: $(\mathbb{N}, \{*, +\})$ forms an algebra.
- In case of RA,
  - ▷ the carrier is the **set of all finite relations.**
- We will get to know the operations of RA in the sequel (one such operation is, for example, $\cup$).

Relational Algebra (3)

- Another operation of relational algebra is **selection.**

  In contrast to operations like $+$ in $\mathbb{N}$, the selection σ is **parameterized** by a simple predicate.

- For example, the operation $\sigma_{\texttt{SID}=101}$ selects all tuples in the input relation which have the value 101 in column SID.

Relational algebra: selection

$\sigma_{\texttt{SID}=101}$

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

$=$

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12

Relational Algebra (4)

- Since the output of any RA operation is some relation R again, R may be the input for another RA operation.

  The operations of RA nest to arbitrary depth such that complex queries can be evaluated. The final results will always be a relation.

- A **query** is a **term** (or expression) in this relational algebra.

A query

$$\pi_{\texttt{FIRST,LAST}}(\texttt{STUDENTS} \bowtie \sigma_{\texttt{CAT='M'}}(\texttt{RESULTS}))$$

Relational Algebra (5)

- There are some difference between the two **query languages** RA and SQL:
  - ▷ Null values are usually excluded in the definition of relational algebra, except when operations like **outer join** are defined.
  - ▷ Relational algebra treats **relations as sets**, *i.e.*, **duplicate tuples will never occur** in the input/output relations of an RA operator.

    Remember: In SQL, relations are **multisets** (bags) and may contain duplicates. Duplicate elimination is explicit in SQL (`SELECT DISTINCT`).

Relational Algebra (6)

- Relational algebra is *the* query language when it comes to the study of relational query languages (DB Theory):
  - ▷ The **semantics** of RA is *much* simpler than that of SQL. RA features five basic operations (and can be completely defined on a single page, if you will).
  - ▷ RA is also a yardstick for measuring the **expressiveness** of query languages. If a query language QL can express all possible RA queries, then QL is said to be **relationally complete.**

    SQL is relationally complete. Vice versa, every SQL query (without null values, aggregation, and duplicates) can also be written in RA.

Selection (1)

Selection

The **selection** σ_φ selects a subset of the tuples of a relation, namely those which satisfy **predicate** φ. Selections acts like a filter on a set.

Selection

$$\sigma_{A=1}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 5 \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ \hline\end{array}$$

Selection (2)

- A simple **selection predicate** φ has the form

$$\langle \mathit{Term} \rangle \ \langle \mathit{ComparisonOperator} \rangle \ \langle \mathit{Term} \rangle .$$

- $\langle \mathit{Term} \rangle$ is an expression that can be **evaluated to a data value** for a given tuple:
  - an **attribute** name,
  - a **constant value**,
  - an **expression** built from attributes, constants, and data type operations like $+, -, *, /$.

Selection (3)

- $\langle \mathit{ComparisonOperator} \rangle$ is
  - $=$ (equals), $\neq$ (not equals),
  - $<$ (less than), $>$ (greater than), $\leqslant$, $\geqslant$,
  - or other data type-dependent predicates (*e.g.*, `LIKE`).
- Examples for simple selection predicates:
  - `LAST` $=$ `'Smith'`
  - `POINTS` $\geqslant$ `8`
  - `POINTS` $=$ `MAXPT`.

Selection (4)

- $\sigma_\varphi(R)$ may be imlemented as:

"Naive" selection

```
create a new temporary relation T;
foreach t ∈ R do
  p ← φ(t);
  if p then
    insert t into T;
  fi
od
return T;
```

- If **index structures** are present (*e.g.*, a B-tree index), it is possible to evaluate $\sigma_\varphi(R)$ without reading every tuple of R.

Selection (5)

A few corner cases

$$\sigma_{C=1}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 5 \\ \hline\end{array}\right) = \lightning \quad \text{(schema error)}$$

$$\sigma_{A=A}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 5 \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 5 \\ \hline\end{array}$$

$$\sigma_{1=2}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 5 \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline A & B \\ \hline\end{array}$$

Selection (6)

- $\sigma_\varphi(R)$ corresponds to the following SQL query:

```
SELECT  *
FROM    R
WHERE   φ
```

- A different relational algebra operation called **projection** corresponds to the `SELECT` clause. Source of confusion.

Selection (7)

- More **complex selection predicates** may be performed using the **Boolean connectives:**

  ▷ $\varphi_1 \wedge \varphi_2$ ("and"), $\varphi_1 \vee \varphi_2$ ("or"), $\neg\varphi_1$ ("not").

- Note: $\sigma_{\varphi_1 \wedge \varphi_2}(R) = \sigma_{\varphi_1}(\sigma_{\varphi_2}(R))$.

$\vee$ and $\neg$

Are the Boolean connectives $\vee, \neg$ strictly needed?

- The selection predicate must permit evaluation for each input tuple in **isolation.**

  Thus, *exists* ($\exists$) and *for all* ($\forall$) or nested relational algebra queries are not permitted in selection predicates. Actually, such predicates **do not add to the expressiveness** of RA.

Projection (1)

Projection

The **projection** π_L eliminates all attributes (columns) of the input relation but those mentioned in the **projection list** L.

Projection

$$\pi_{A,C}\left(\begin{array}{|c|c|c|}\hline A & B & C \\ \hline 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 3 & 9 \\ \hline\end{array}$$

Projection (2)

- The projection $\pi_{A_{i_1},\ldots,A_{i_k}}(R)$ produces for each input tuple $(A_1 : d_1, \ldots, A_n : d_n)$ an output tuple $(A_{i_1} : d_{i_1}, \ldots, A_{i_k} : d_{i_k})$.

- π may be used to **reorder columns.**

"σ **discards rows,** π **discards columns.**"

- DB slang: "All attributes not in L are **projected away.**"

Projection (3)

- In general, the **cardinalities** of the input and output relations **are not equal**.

Projection eliminates duplicates

$$\pi_B \left(\begin{array}{|c|c|} \hline A & B \\ \hline 1 & 4 \\ 2 & 5 \\ 3 & 4 \\ \hline \end{array} \right) = \begin{array}{|c|} \hline B \\ \hline 4 \\ 5 \\ \hline \end{array}$$

Projection (4)

- $\pi_{A_{i_1},\ldots,A_{i_k}}(R)$ may be imlemented as:

"Naive" projection

create a new temporary relation T;
`foreach` $t = (A_1 : d_1, \ldots, A_n : d_n) \in R$ `do`
  $u \leftarrow (A_{i_1} : d_{i_1}, \ldots, A_{i_k} : d_{i_k})$;
  insert u into T;
`od`
eliminate duplicate tuples in T;
return T;

- The necessary duplicate elimination makes π_L one of the more costly operations in RDBMSs. Thus, query optimizers try hard to "prove" that the duplicate eliminaton step is not necessary.

Projection (5)

- If RA is used to formalize the semantics of SQL, the format of the projection list is often generalized:
  - ▷ **Attribute renaming:**

$$\pi_{B_1 \leftarrow A_{i_1}, \ldots, B_k \leftarrow A_{i_k}}(R) \ .$$

  - ▷ **Computations** (*e.g.*, string concatenation via `||`) to derive the value in **new columns**, *e.g.*:

$$\pi_{\texttt{SID,NAME} \leftarrow \texttt{FIRST || ' ' || LAST}}(\texttt{STUDENTS}) \ .$$

- Such generalized π operators are also referred to as **map** operators (as in functional programming languages).

Projection (6)

- $\pi_{A_1,\ldots,A_k}(R)$ corresponds to the SQL query:

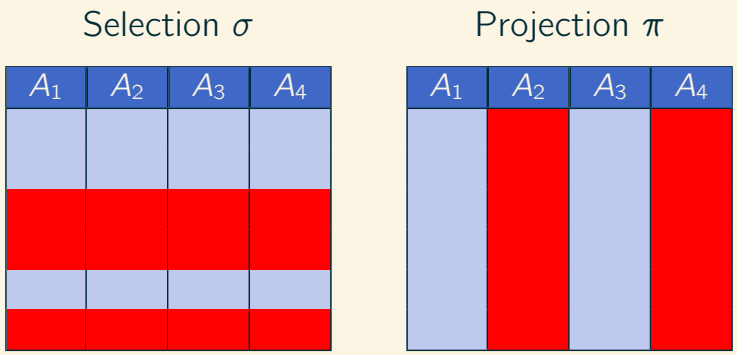
```
SELECT DISTINCT  A1, ..., Ak
FROM             R
```

- $\pi_{B_1 \leftarrow A_1, \ldots, B_k \leftarrow A_k}(R)$ is equivalent to the SQL query:

```
SELECT DISTINCT  A1 [AS] B1, ..., Ak [AS] Bk
FROM             R
```

Selection vs. Projection

Selection vs. Projection



Combining Operations (1)

- Since the result of any relational algebra operation is a relation again, this intermediate result may be the input of a subsequent RA operation.
- Example: retrieve the exercises solved by student with ID 102:

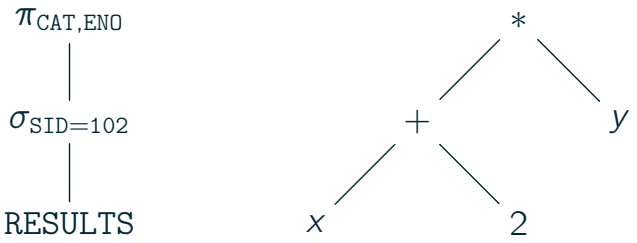
$$\pi_{\texttt{CAT,ENO}}(\sigma_{\texttt{SID=102}}(\texttt{RESULTS})) \ .$$

- We can think of the intermediate result to be stored in a **named temporary relation** (or as a macro definition):

$$\texttt{S102} \leftarrow \sigma_{\texttt{SID=102}}(\texttt{RESULTS});$$
$$\pi_{\texttt{CAT,ENO}}(\texttt{S102})$$

Combining Operations (2)

- Composite RA expressions are typically depicted as **operator trees:**



- In these trees, computation proceeds **bottom-up.** The evaluation order of sibling branches is not pre-determined.

Combining Operations (3)

- SQL-92 permits the **nesting of queries** (the result of a SQL query may be used in a place of a relation name):

Nested SQL Query

```
SELECT DISTINCT CAT, ENO
FROM            (SELECT *
                 FROM   RESULTS
                 WHERE  SID = 102) AS S102
```

- Note that this is *not* the typical style of SQL querying.

Combining Operations (4)

- Instead, a single SQL query is equivalent to an RA operator tree containing σ, π, and (multiple) $\times$ (see below):

SELECT-FROM-WHERE Block

```
SELECT DISTINCT CAT, ENO
FROM            RESULTS
WHERE           SID = 102
```

- Really complex queries may be constructed step-by-step (using SQL's **view** mechanism), S102 may be used like a relation:

SQL View Definition

```
CREATE VIEW S102
AS SELECT  *
   FROM    RESULTS
   WHERE   SID = 102
```


Relational Algebra

Overview

Cartesian Product (1)

- In general, queries need to combine information from **several tables.**
- In RA, such queries are formulated using $\times$, the **Cartesian product.**

Cartesian Product

The **Cartesian product** $R \times S$ of two relations R, S is computed by concatenating each tuple $t \in R$ with each tuple $u \in S$. ($\circ$ denotes tuple concatenation.)

Cartesian Product (2)

Cartesian Product

A	B
1	2
3	4

$\times$

C	D
6	7
8	9

$=$

A	B	C	D
1	2	6	7
1	2	8	9
3	4	6	7
3	4	8	9

- Since attribute names must be unique within a tuple, the Cartesian product may only be applied if R, S **do not share any attribute names.** (This is no real restriction because we have π.)

Cartesian Product (3)

- If $t = (A_1 : a_1, \ldots, A_n : a_n)$ and $u = (B_1 : b_1, \ldots, B_m : b_m)$, then $t \circ u = (A_1 : a_1, \ldots, A_n : a_n, B_1 : b_1, \ldots, B_m : b_m)$.

Cartesian Product: Nested Loops

```
create a new temporary relation T;
foreach t ∈ R do
  foreach u ∈ S do
    insert t ∘ u into T;
  od
od
return T;
```


Cartesian Product and Renaming

- $R \times S$ may be computed by the equivalent SQL query (SQL does not impose the unique column name restriction, a column A of relation R may uniquely be identified by $R.A$):

Cartesian Product in SQL

```
SELECT *
FROM   R, S
```

  - ▷ In RA, this is often formalized by means of of a **renaming operator** $\varrho_X(R)$. If $sch(R) = (A_1 : D_1, \ldots, A_n : D_n)$, then

$$\varrho_X(R) \quad \equiv \quad \pi_{X.A_1 \leftarrow A_1, \ldots, X.A_n \leftarrow A_n}(R) \ .$$

Join (1)

- The intermediate result generated by a Cartesian product may be quite large in general ($|R| = n, |S| = m \Rightarrow |R \times S| = n * m$).
- Since the combination of **Cartesian product and selection** in queries is common, a special operator **join** has been introduced.

Join

The **(theta-)join** $R \bowtie_\theta S$ between relations R, S is defined as

$$R \bowtie_\theta S \quad \equiv \quad \sigma_\theta(R \times S).$$

The **join predicate** θ may refer to attribute names of R and S.

Join (2)

ϱ_S(STUDENTS) $\bowtie_{S.\text{SID}=R.\text{SID}}$ ϱ_R(RESULTS)

S.SID	S.FIRST	S.LAST	S.EMAIL	R.SID	R.CAT	R.ENO	R.POINTS
101	Ann	Smith	...	101	H	1	10
101	Ann	Smith	...	101	H	2	8
101	Ann	Smith	...	101	M	1	12
102	Michael	Jones	(null)	102	H	1	9
102	Michael	Jones	(null)	102	H	2	9
102	Michael	Jones	(null)	102	M	1	10
103	Richard	Turner	...	103	H	1	5
103	Richard	Turner	...	103	M	1	7

- Note: student `Maria Brown` does not appear in the join result.

Join (3)

- $R \ltimes_\theta S$ can be evaluated by "folding" the procedures for σ, $\times$:

Nested Loop Join

```
create a new temporary relation T;
foreach t ∈ R do
  foreach u ∈ S do
    if θ(t ∘ u) then
      insert t ∘ u into T;
    fi
  od
od
return T;
```


Join (4)

- Join combines tuples from two relations **and** acts like a filter: tuples without join partner are removed.

  Note: if the join is used to **follow a foreign key relationship,** then **no tuples are filtered:**

Join follows a foreign key relationship (dereference)

$$\texttt{RESULTS} \ltimes_{\texttt{SID}=S.\texttt{SID}} \pi_{S.\texttt{SID} \leftarrow \texttt{SID},\texttt{FIRST},\texttt{LAST},\texttt{EMAIL}}(\texttt{STUDENTS})$$

- There are join variants which act like **filters only:** left and right **semijoin** ($\ltimes$, $\rtimes$):

$$R \ltimes_\theta S \quad \equiv \quad \pi_{sch(R)}(R \ltimes_\theta S) \ ,$$

  or **do not filter** at all: **outer-join** (see below).

Natural Join

- The **natural join** provides another useful abbreviation ("RA macro").

  In the natural join $R \bowtie S$, the join predicate θ is defined to be an **conjunctive equality comparison of attributes sharing the same name** in R, S.

  Natural join handles the necessary attribute renaming and projection.

Natural Join

Assume $R(A, B, C)$ and $S(B, C, D)$. Then:

$$R \bowtie S = \pi_{A,B,C,D}(\sigma_{B=B' \wedge C=C'}(R \times \pi_{B' \leftarrow B, C' \leftarrow C, D}(S)))$$

(**Note:** shared columns occur *once* in the result.)

Joins in SQL (1)

- In SQL, $R \bowtie_\theta S$ is normally written as

Join in SQL ("classic" and SQL-92)

```
SELECT  *                    SELECT  *
FROM    R,S      or          FROM    R JOIN S ON θ
WHERE   θ
```

- **Note:** this left query is **exactly** the SQL equivalent of $\sigma_\theta(R \times S)$ we have seen before.

  SQL is a **declarative language:** it is the task of the SQL optimizer to infer that this query may be evaluated using a join instead of a Cartesian product.

Algebraic Laws (1)

- The join satisfies the **associativity condition**

$$(R \bowtie S) \bowtie T \quad \equiv \quad R \bowtie (S \bowtie T) \ .$$

  In "join chains", parentheses are thus superfluous:

$$R \bowtie S \bowtie T \ .$$

- Join is **not commutative** unless it is followed by a projection, *i.e.*, a column reordering:

$$\pi_L(R \bowtie S) \quad \equiv \quad \pi_L(S \bowtie R) \ .$$

Algebraic Laws (2)

- A significant number of further **algebraic laws** hold, which are heavily utilized by the query optimizer.
- Example: **selection push-down.**

  If predicate φ **refers to attributes in** S **only,** then

$$\sigma_\varphi(R \bowtie S) \quad \equiv \quad R \bowtie \sigma_\varphi(S) \ .$$

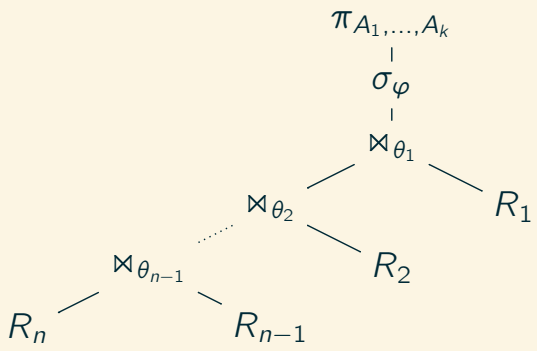
Selection push-down

Why is selection push-down considered one of the most significant algebraic optimizations?

- (Such effficiency considerations are the subject of "Datenbanken II.")

A Common Query Pattern (1)

- The following operator tree structure is **very** common:



① **Join** all tables needed to answer the query, ② **select** the relevant tuples, ③ **project** away all irrelevant columns.

A Common Query Pattern (2)

- The **select-project-join query**

$$\pi_{A_1,\ldots,A_k}(\sigma_\varphi(R_1 \bowtie_{\theta_1} R_2 \bowtie_{\theta_2} \cdots \bowtie_{\theta_{n-1}} R_n))$$

has the obvious SQL equivalent

```
SELECT DISTINCT  A_1, ..., A_k
FROM             R_1, ..., R_n
WHERE            φ
AND              θ_1 AND ··· AND θ_{n-1}
```

- It is a common source of errors to forget a join condition: think of the scenario $R(A, B), S(B, C), T(C, D)$ when attributes A, D are relevant for the query output.

Relational Algebra Quiz (Level: Novice)

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Relational Algebra Quiz (Level: Novice)

Formulate equivalent queries in RA

① Print all homework results for `Ann Smith` (print exercise number and points).

② Who has got the maximum possible number of points (`MAXPT`) for a homework? Print full name and homework number.

③ (Who has got the maximum number of points *for all* homework exercises?)

Self Joins (1)

- Sometimes it is necessary to refer to more than one tuple of the **same relation** at the **same time.**
  - ▷ Example: "Who got more points than the student with ID 101 for any of the exercises?"
  - ▷ Two answer this query, we need to compare two tuples t, u of the relation RESULTS:
    1. tuple t corresponding to the student with ID 101,
    2. tuple u, corresponding to the same exercise as the tuple t, in which $u.\texttt{POINTS} > t.\texttt{POINTS}$.

Self Joins (2)

- This requires a generalization of the select-project-join query pattern, in which **two instances of the same relation are joined** (the attributes in at least one instances must be renamed first):

$$S := \varrho_X(\texttt{RESULTS}) \underset{X.\texttt{CAT}=Y.\texttt{CAT} \wedge X.\texttt{ENO}=Y.\texttt{ENO}}{\bowtie} \varrho_Y(\texttt{RESULTS})$$

$$\pi_{X.\texttt{SID}}(\sigma_{X.\texttt{POINTS}>Y.\texttt{POINTS} \wedge Y.\texttt{SID}=101})(S)$$

- Such joins are commonly referred to as **self joins.**

Relational Algebra

Overview

1. Introduction; Selection, Projection
2. Cartesian Product, Join
3. Set Operations
4. Outer Join
5. Formal Definitions, A Bit of Theory

Set Operations (1)

- Relations are sets (of tuples). The "usual" family of binary **set operations** can also be applied to relations.
- It is a requirement that both input relations **have the same schema.**

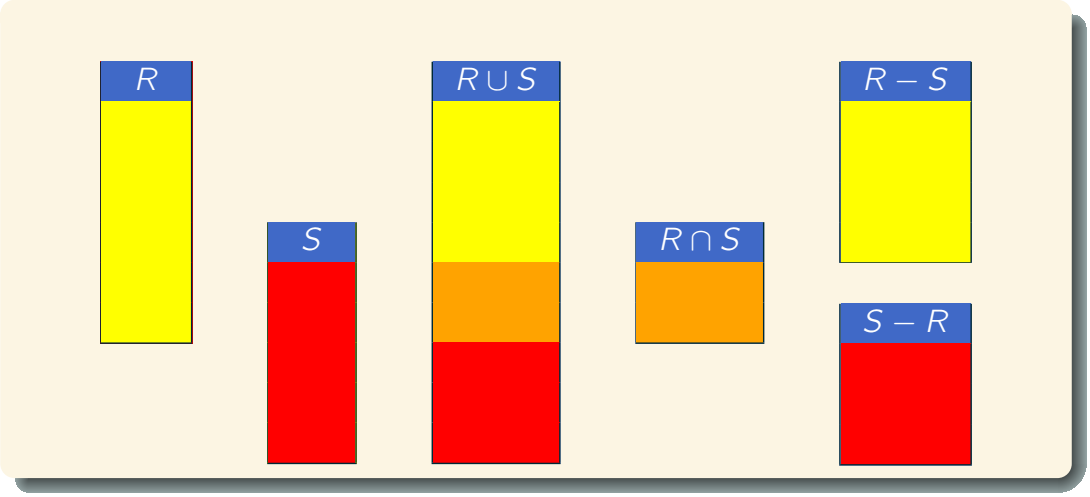
Set Operations

The **set operations** of relational algebra are $R \cup S$, $R \cap S$, and $R - S$ (**union**, **intersection**, **difference**).

A minimal set of operations

Which of these set operations is redundant (*i.e.*, may be derived using an alternative RA expression, just like $\bowtie$)?

Set Operations (2)



Set Operations (3)

- $R \cup S$ may be implemented as follows:

Union

create a new temporary relation T;
`foreach` $t \in R$ `do`
  insert t into T;
`od`
`foreach` $t \in S$ `do`
  insert t into T;
`od`
remove duplicates in T;
return T;

Set Operations (4)

- $R - S$ may be implemented as follows:

Difference

create a new temporary relation T;
`foreach` $t \in R$ `do`
  $remove \leftarrow false$;
  `foreach` $u \in S$ `do`
    $remove \leftarrow remove \vee (t = u)$;
  `od`
  `if` $\neg remove$ `then`
    insert t into T;
  `fi`
`od`
return T;

Union (1)

- In RA queries, a typical application for $\cup$ is **case analysis.**

Example: Grading

$$\texttt{MPOINTS} := \pi_{\texttt{SID,POINTS}}(\sigma_{\texttt{CAT='M'} \wedge \texttt{ENO=1}}(\texttt{RESULTS}))$$

$$\pi_{\texttt{SID,GRADE}\leftarrow\texttt{'A'}}(\sigma_{\texttt{POINTS} \geqslant 12}(\texttt{MPOINTS}))$$
$$\cup \quad \pi_{\texttt{SID,GRADE}\leftarrow\texttt{'B'}}(\sigma_{\texttt{POINTS} \geqslant 10 \ \wedge \ \texttt{POINTS} < 12}(\texttt{MPOINTS}))$$
$$\cup \quad \pi_{\texttt{SID,GRADE}\leftarrow\texttt{'C'}}(\sigma_{\texttt{POINTS} \geqslant 7 \ \wedge \ \texttt{POINTS} < 10}(\texttt{MPOINTS}))$$
$$\cup \quad \pi_{\texttt{SID,GRADE}\leftarrow\texttt{'F'}}(\sigma_{\texttt{POINTS} < 7}(\texttt{MPOINTS}))$$

Union (2)

- In SQL, $\cup$ is directly supported: keyword `UNION`.

  `UNION` may be placed between two `SELECT-FROM-WHERE` blocks:

SQL's `UNION`

```
  SELECT SID, 'A' AS GRADE
  FROM   RESULTS
  WHERE  CAT = 'M' AND ENO = '1' AND POINTS >= 12
UNION
  SELECT SID, 'B' AS GRADE
  FROM   RESULTS
  WHERE  CAT = 'M' AND ENO = '1'
  AND    POINTS >= 10 AND POINTS < 12
UNION
  ...
```


Set Difference (1)

- **Note:** the RA operators $\sigma, \pi, \times, \bowtie, \cup$ are **monotic** by definition, *e.g.*:

$$R \subseteq S \quad \Longrightarrow \quad \sigma_\varphi(R) \subseteq \sigma_\varphi(S) \ .$$

- Then it follows that every query Q that exclusively uses the above operators behaves monotonically:
  - ▷ Let I_1 be a database state, and let $I_2 = I_1 \cup \{t\}$ (database state after insertion of tuple t).
  - ▷ Then every tuple u contained in the answer to Q in state I_1 is also contained in the anser to Q in state I_2.

    Database insertion **never invalidates** a correct answer.

Set Difference (2)

- If we pose **non-monotonic** queries, *e.g.*,
  - ▷ "*Which student has not solved any exercise?*"
  - ▷ "*Who got the most points for Homework 1?*"
  - ▷ "*Who has solved all exercises in the database?*"

  then it is obvious that $\sigma, \pi, \times, \bowtie, \cup$ are **not sufficient** to formulate the query. Such queries require **set difference** $(-)$.

A non-monotonic query

"*Which student has not solved any exercise? (Print full name* (FIRST, LAST)."
(Example database tables repeated on next slide.)

Example Database (recap)

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Set Difference (3)

A correct solution?

$$\pi_{\texttt{FIRST,LAST}}(\texttt{STUDENTS} \bowtie_{\texttt{SID}\neq\texttt{SID2}} \pi_{\texttt{SID2}\leftarrow\texttt{SID}}(\texttt{RESULTS}))$$

A correct solution?

$$\pi_{\texttt{SID,FIRST,LAST}}(\texttt{STUDENTS} - \pi_{\texttt{SID}}(\texttt{RESULTS}))$$

Correct solution!

Set Difference (4)

- A typical RA query pattern involving set difference is the **anti-join**.
- Given $R(A, B)$ and $S(B, C)$, retrieve the tuples of R that **do not have a (natural) join partner** in S
  (Note: $sch(R) \cap sch(S) = \{B\}$):

$$R \bowtie (\pi_B(R) - \pi_B(S)) \quad .$$

  (The following is equivalent: $R - \pi_{sch(R)}(R \bowtie S)$.)

- There is no common symbol for this anti-join, but $R \overline{\ltimes} S$ seems appropriate (complemented semi-join).

Set Operations and Complex Selections

- Note that the availability of $\cup, -$ (and $\cap$) renders **complex selection predicates superfluous:**

Predicate Simplification Rules

$$\begin{aligned} \sigma_{\varphi_1 \wedge \varphi_2}(Q) &\stackrel{\rightarrow}{=} \sigma_{\varphi_1}(Q) \cap \sigma_{\varphi_2}(Q) \\ \sigma_{\varphi_1 \vee \varphi_2}(Q) &= \sigma_{\varphi_1}(Q) \cup \sigma_{\varphi_2}(Q) \\ \sigma_{\neg\varphi}(Q) &= Q - \sigma_{\varphi}(Q) \end{aligned}$$

RDBMS implement complex selection predicates anyway

Why?

Relational Algebra Quiz (Level: Intermediate)

- The "RA quiz" below refers to the Homework DB. Schema:

```
RESULTS (SID → STUDENTS,(CAT, ENO) → EXERCISES,
         POINTS)
STUDENTS (SID,FIRST,LAST,EMAIL)
EXERCISES (CAT,ENO,TOPIC,MAXPT)
```

Formulate equivalent queries in RA

① Who got the **most** points (of all students) for Homework 1?

② Which students solved **all** exercises in the database?

Union vs. Join

Find RA expressions that translate between the two

Two alternative representations of the homework, midterm exam, and final totals of the students are:

RESULTS_1

STUDENT	H	M	F
Jim Ford	95	60	75
Ann Smith	80	90	95

RESULTS_2

STUDENT	CAT	PCT
Jim Ford	H	95
Jim Ford	M	60
Jim Ford	F	75
Ann Smith	H	80
Ann Smith	M	90
Ann Smith	F	95

Summary

The five basic operations of relational algebra are:

① σ_φ **Selection**

② π_L **Projection**

③ $\times$ **Cartesian Product**

④ $\cup$ **Union**

⑤ $-$ **Difference**

- **Derived** (and thus redundant) **operations:** **Theta-Join** $\bowtie_\theta$, **Natural Join** $\bowtie$, **Semi-Join** $\ltimes$, **Renaming** ϱ, and **Intersection** $\cap$.

Relational Algebra

Overview

Outer Join (1)

- Join ($\bowtie$) **eliminates tuples without partner:**

A	B
a_1	b_1
a_1	b_2

$\bowtie$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_2	b_2	c_2

- The **left outer join** preserves all tuples in its **left** argument, even if a tuple does not team up with a partner in the join:

A	B
a_1	b_1
a_1	b_2

$\overset{\text{left}}{\bowtie}$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_1	b_1	(null)
a_2	b_2	c_2

Outer Join (2)

- The **right outer join** preserves all tuples in its **right** argument:

A	B
a_1	b_1
a_1	b_2

$\ltimes$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_2	b_2	c_2
(null)	b_3	c_3

- The **full outer join** preserves all tuples in **both** arguments:

A	B
a_1	b_1
a_1	b_2

$\rtimes$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_1	b_1	(null)
a_2	b_2	c_2
(null)	b_3	c_3

Outer Join (3)

$R \rtimes_\theta S$

create a new temporary relation T;
`foreach` $t \in R$ `do`
  haspartner $\leftarrow$ *false*;
  `foreach` $u \in S$ `do`
    `if` $\theta(t \circ u)$ `then`
      insert $t \circ u$ into T;
      haspartner $\leftarrow$ *true*;
    `fi`
  `od`
  `if` $\neg$*haspartner* `then`
    insert $t \circ \underbrace{(\text{null}, \ldots, \text{null})}_{\#\text{ attributes in } S}$ into T;
  `fi`
`od`
return T;

Outer Join (4)

- Example: Prepare a full homework results report, including those students who did not hand in any solution at all:

$$\texttt{STUDENTS} \;⟖_{\texttt{SID}=\texttt{SID}'}\; \pi_{\texttt{SID}'\leftarrow\texttt{SID},\texttt{ENO},\texttt{POINTS}}(\sigma_{\texttt{CAT}=\text{'H'}}(\texttt{RESULTS}))$$

SID	FIRST	LAST	EMAIL	SID'	ENO	POINTS
101	Ann	Smith	...	101	1	10
101	Ann	Smith	...	101	2	8
102	Michael	Jones	(null)	102	1	9
102	Michael	Jones	(null)	102	2	9
103	Richard	Turner	...	103	1	5
104	Maria	Brown	...	(null)	(null)	(null)

Outer Join (5)

Join vs. Outer Join

Is there any difference between $\texttt{STUDENTS} \bowtie \texttt{RESULTS}$ and $\texttt{STUDENTS} \ltimes \texttt{RESULTS}$?
(Can you tell without looking at the table states?)

- **Note:** Outer join is a **derived operation** (like $\bowtie, \cap$), *i.e.*, it can simulated using the five basic relational algebra operations.
- Consider $R(A, B)$ and $S(B, C)$. Then

$$R \;⟖\; S \quad \equiv \quad (R \bowtie S) \cup \big((R - \pi_{A,B}(R \bowtie S)) \times \{(C{:}\text{null})\}\big)$$

- SQL-92 provides $\{\texttt{FULL}, \texttt{LEFT}, \texttt{RIGHT}\}$ `OUTER JOIN`.

Relational Algebra

Overview

1. Introduction; Selection, Projection
2. Cartesian Product, Join
3. Set Operations
4. Outer Join
5. Formal Definitions, A Bit of Theory

Definitions: Syntax (1)

- Let the following be given:
  - ▷ A set $\mathcal{D}$ of **data type names** and for each $D \in \mathcal{D}$ a set $val(D)$ of **values.**
  - ▷ A set $\mathcal{A}$ of valid **attribute names** (identifiers).

Relational Database Schema

A **relational database schema** $\mathcal{S}$ consists of
- a **finite set of relation names** $\mathcal{R}$, and
- for every $R \in \mathcal{R}$ a **relation schema** $sch(R)$.

(We ill ignore constraints here.)

Definitions: Syntax (2)

- The set of **syntactically correct RA expressions** or queries is defined recursively, together with the **resulting schema** of each expression.

Syntax of RA (Base Cases)

① R (**relation name**)
For every $R \in \mathcal{R}$, R is an RA expression with schema $sch(R)$.

② $\{(A_1{:}d_1, \ldots, A_n{:}d_n)\}$ (**relation constant**)
A relation constant is an RA expression if $A_1, \ldots, A_n \in \mathcal{A}$, $d_i \in val(D_i)$ for $1 \leqslant i \leqslant n$ with $D_1, \ldots, D_n \in \mathcal{D}$. The schema of this expression is $(A_1{:}D_1, \ldots, A_n{:}D_n)$.

Definitions: Syntax (3)

- Let Q be an RA expr. with schema $s = (A_1{:}D_1, \ldots, A_n{:}D_n)$.

Syntax of RA (Recursive Cases)

③ $\sigma_{A_i=A_j}(Q)$
for $i, j \in \{1, \ldots, n\}$ is an RA expression with schema s.

④ $\sigma_{A_i=d}(Q)$
for $i \in \{1, \ldots, n\}$ and $d \in val(D_i)$ is an RA expression with schema s.

⑤ $\pi_{B_1 \leftarrow A_{i_1}, \ldots, B_m \leftarrow A_{i_m}}(Q)$
for $i_1, \ldots, i_m \in \{1, \ldots, n\}$ and $B_1, \ldots, B_m \in \mathcal{A}$ such that $B_j \neq B_k$ for $j \neq k$ is an RA expression with schema $(B_1{:}D_{i_1}, \ldots, B_m{:}D_{i_m})$.

Definitions: Syntax (4)

- Let Q_1, Q_2 be an RA expressions with the same schema s.

Syntax of RA (Recursive Cases)

⑥ $Q_1 \cup Q_2$ and ⑦ $Q_1 - Q_2$
are RA expressions with schema s.

- Let Q_1, Q_2 be RA expressions with schemas $(A_1{:}D_1, \ldots, A_n{:}D_n)$ and $(B_1{:}E_1, \ldots, B_m{:}E_m)$, respectively.

Syntax of RA (Recursive Cases)

⑧ $Q_1 \times Q_2$
is an RA expression with schema $(A_1{:}D_1, \ldots, A_n{:}D_n, B_1{:}E_1, \ldots, B_m{:}E_m)$ if $\{A_1, \ldots, A_n\} \cap \{B_1, \ldots, B_m\} = \emptyset$.

Definitions: Semantics (1)

Database State

A **database state** I (instance) defines a relation $I(R)$ for every relation name R in the database schema $\mathcal{S}$.

- The **result of a query** Q, *i.e.*, an RA expression, in a database state I is a relation. This relation is denoted by $I[Q]$ and defined recursively corresponding to the syntactic structure of Q.

Definition: Semantics (2)

$I[Q]$

- If Q is a relation name R, then $I[Q] := I(R)$.
- If Q is a constant relation $\{(A_1{:}d_1, \ldots, A_n{:}d_n)\}$, then $I[Q] := \{(d_1, \ldots, d_n)\}$.
- If Q has the form $\sigma_{A_i=A_j}(Q_1)$, then
$$I[Q] := \{(d_1, \ldots, d_n) \in I[Q_1] \mid d_i = d_j\}$$
- If Q has the form $\sigma_{A_i=d}(Q_1)$, then
$$I[Q] := \{(d_1, \ldots, d_n) \in I[Q_1] \mid d_i = d\}$$
- If Q has the form $\pi_{B_1 \leftarrow A_{i_1}, \ldots, B_m \leftarrow A_{i_m}}(Q_1)$, then
$$I[Q] := \{(d_{i_1}, \ldots, d_{i_m}) \mid (d_1, \ldots, d_n) \in I[Q_1]\}$$

Definition: Semantics (3)

$I[Q]$ (continued)

- If Q has the form $Q_1 \cup Q_2$, then
$$I[Q] := I[Q_1] \cup I[Q_2]$$
- If Q has the form $Q_1 - Q_2$, then
$$I[Q] := I[Q_1] - I[Q_2]$$
- If Q has the form $Q_1 \times Q_2$, then
$$I[Q] := \{\ (d_1, \ldots, d_n, e_1, \ldots, e_m) \mid (d_1, \ldots, d_n) \in I[Q_1], (e_1, \ldots, e_m) \in I[Q_2]\}\ .$$

Monotonicity

Smaller Database State

A database state I_1 is **smaller than (or equal to)** a database state I_2, written $I_1 \subseteq I_2$, iff $I_1(R) \subseteq I_2(R)$ for all relation names $R \in \mathcal{R}$ of schema $\mathcal{S}$.

Theorem: RA $\setminus\{-\}$ is monotonic

If an RA expression Q does not contain the $-$ (set difference) operator, then the following holds for all database states I_1, I_2:

$$I_1 \subseteq I_2 \quad \Longrightarrow \quad I_1[Q] \subseteq I_2[Q] \ .$$

Formulate proof by induction on syntactic structure of Q ("structural induction").

Equivalence (1)

Equivalence of RA Expressions

Two RA expressions Q_1 and Q_2 are **equivalent** iff they have the same (result) schema and **for all database states** I, the following holds:

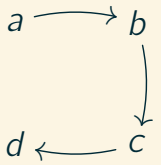
$$I[Q_1] \quad = \quad I[Q_2] \ .$$

- Examples:
  - ▷ $\sigma_{\varphi_1}(\sigma_{\varphi_2}(Q)) = \sigma_{\varphi_2}(\sigma_{\varphi_1}(Q))$
  - ▷ $(Q_1 \times Q_2) \times Q_3 = Q_1 \times (Q_2 \times Q_3)$
  - ▷ If A is an attribute in the result schema of Q_1, then $\sigma_{A=d}(Q_1 \times Q_2) = (\sigma_{A=d}(Q_1)) \times Q_2$.

  Theorem: The equivalence of (arbitrary) relational algebra expressions is undecidable.

Limitations of RA (1)

- Let R be a relation name and assume $sch(R) = (A{:}D, B{:}D)$, *i.e.*, both columns share the same data type D. Let $val(D)$ be infinite.
- The **transitive closure** of $I(R)$ is the set of all $(d, e) \in val(D) \times val(D)$ such that there are $n \in \mathbb{N}, n \geqslant 1$, and $d_0, d_1, \ldots, d_n \in val(D)$ with $d = d_0, e = d_n$ and $(d_{i-1}, d_i) \in I(R)$ for $i = 1, \ldots, n$.



R

from	*to*
a	b
b	c
c	d

Limitations of RA (2)

Theorem: There is no RA expression Q such that $I[Q]$ is the transitive closure of $I(R)$ for all database states I.

In the directed graph example, one self-join (of R with itself) is needed, to follow the edges in the graph:

$$\pi_{S.from, T.to}(\varrho_S(R) \bowtie_{S.to=T.from} \varrho_T(R))$$

- An n-fold self-join will find **all paths in the graph of length** $n + 1$**.** To compute the transitive closure for **arbitrary graphs**, *i.e.*, for all database states I, is impossible in RA.

Limitations of RA (3)

- This of course implies that relational algebra is **not computationally complete.**
  - ▷ There are functions from database states to relations (query results), for which we could write a program[4], but we will not be able to find an equivalent RA expression to do the same.
  - ▷ However, this would have been truly unexpected and actually unwanted, because want a guarantee that **query evaluation always terminates.** This is guaranteed for RA.

    Otherwise, we would have solved the halting problem.

[4] Pick your favourite programming language.

Limitations of RA (4)

- All RA queries can be evaluated in **time that is polynomically in the size of the database state.**
- This implies that certain "complex problems" cannot be formulated in relational algebra.

  For example, if you find a way to formulate the *Traveling Salesman* problem in RA, you have solved the famous $P = NP$ problem. (With a solution that nobody expects; contact me to collect your PhD.)
- As the transitive closure example shows, even not all problems of polynomial complexity can be formulated in "classical RA."

Expressive Power (1)

Relational Completeness

A query language $\mathcal{L}$ for the relational model is called **strong relationally complete** if, for every DB schema $\mathcal{S}$ and for every RA epxression Q_1 with respect to $\mathcal{S}$ there is a query $Q_2 \in \mathcal{L}$ such that for all database states I with respect to $\mathcal{S}$ the two queries produce the same results:

$$I[Q_1] = I[Q_2] \quad .$$

- Read as: "*It is possible to write an RA-to-$\mathcal{L}$ query compiler.*"

Expressive Power (2)

- SQL is strong relationally complete.
- If we can even write RA-to-$\mathcal{L}$ as well as $\mathcal{L}$-to-RA compilers, both query languages are **equivalent.**
  - ▷ SQL and RA are **not equivalent.** SQL contains concepts, *e.g.*, the aggregate `COUNT`, which cannot be simulated in RA.

Equivalent Query Languages

1. **Relational algebra,**
2. **SQL without aggregations and with mandatory duplicate elimination,**
3. **Tuple relational calculus,**
4. **Datalog** (a Prolog variant) **without recursion.**