# IN3020&4020 – Database Systems (2024)
# Part 1: Query Processing
# Lectures 5, 6: Relational Algebra

---

*29, 30 January*

**Egor V. Kostylev**
IFI, University of Oslo
egork@ifi.uio.no



# We start new part: Query Processing

Syllabus of the course:

- (Extended) Intro and SQL recap ✓

**Part 1. Query processing in relational databases**

Part 2. Transaction management in relational databases

Part 3. NoSQL DBMS

SQL query processing relies on two key ingredients:
Relational Algebra and Indexes

We study these ingredients first (almost in isolation)
and then put all things together

# Materials to read about Relational Algebra

1. *Fundamentals of Database Systems* by R. Elmasri and S.B. Navathe (7th edition): Chapter 8 (maybe, not the best)
2. *Foundation of Databases* by S. Abiteboul, R. Hull & V. Vianu: Part B (more details)
   available at `http://webdam.inria.fr/Alice/`
3. *Database Systems: the Complete Book* by H. Garcia-Molina, J. Ullman & J. Widom (2nd edition): Section 2.4
4. IN2090 – Databaser og datamodellering (Leif Harald Karlsen, Dimitru Roman): Lectures 3 (basics)

   `https://www.uio.no/studier/emner/matnat/ifi/IN2090/h21/timeplan/`
5. (English) Wikipedia has a good introduction
6. etc.

# Materials to Play With

- Many good places with interactive exercises:
  - w3resource (`https://www.w3resource.com/`)
  - w3schools (`https://www.w3schools.com/sql/`)
  - Web pages of systems (MySQL, PostgreSQL, Oracle, etc.)

- They use SQL rather than RA
- Should not be a problem: for simple examples
  - syntactic translation is direct
  - semantic differences are immaterial

# Lecture Plan

1. Role of RA in SQL engines
2. Algebras and algebraic laws
3. Relational Algebra intro
4. Core RA operations
5. Algebraic laws for core operations
6. (Some) expressible operations

# 1. Role of RA in SQL engines

# SQL query processor in action

```
SELECT DISTINCT title FROM StarsIn WHERE starName IN
  (SELECT name FROM MovieStar WHERE birthDate LIKE '%1960');
```

$\downarrow$ parsing

(parse tree)

$\downarrow$ preprocessing

$$\pi_{title}(\sigma_{starName=name}(\mathbf{StarIn} \times \pi_{name}(\sigma_{birthDate\ LIKE\ '\%1960'}(\mathbf{MovieStar}))))$$

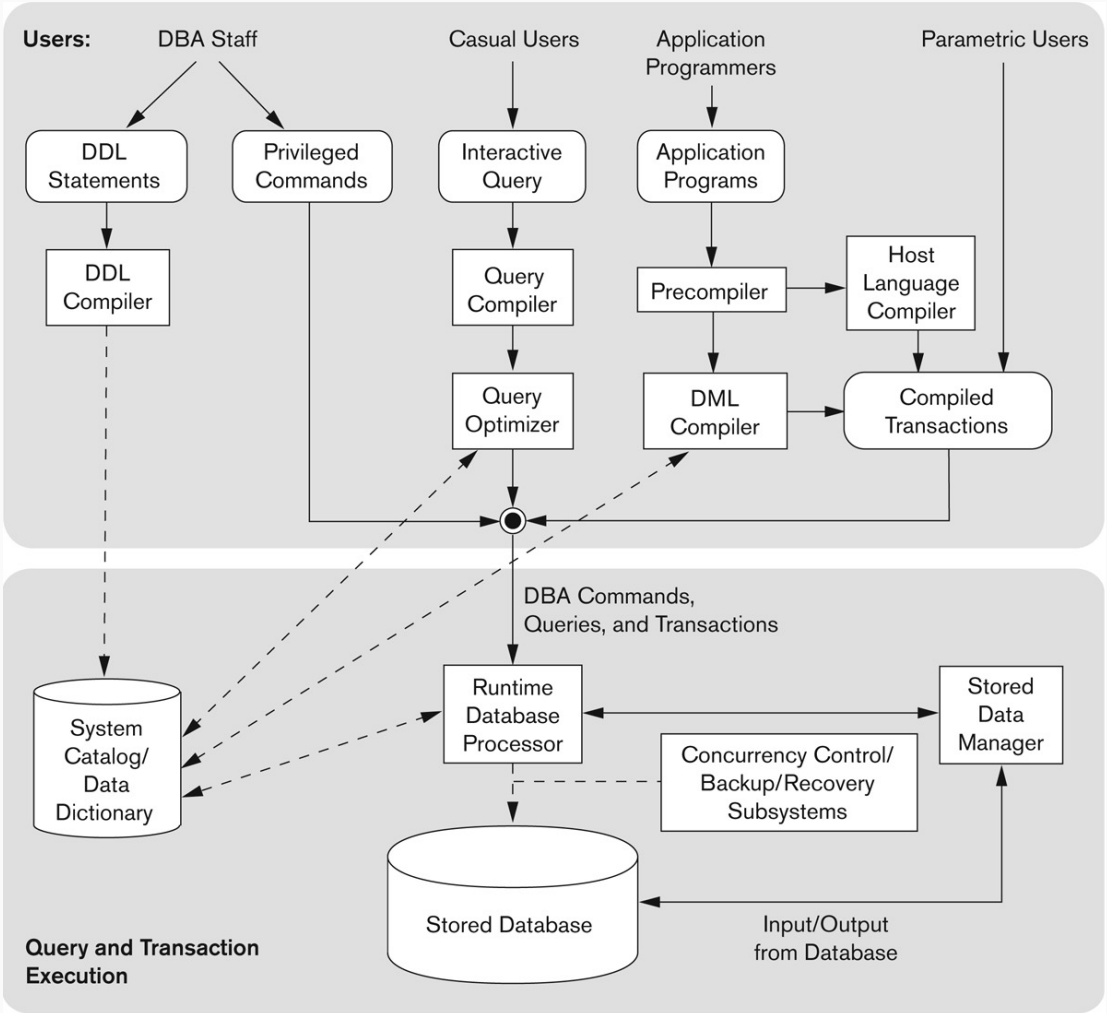
$\downarrow$ optimisation

$$\pi_{title}(\mathbf{StarIn} \bowtie_{starName=name} \pi_{name}(\sigma_{birthDate\ LIKE\ '\%1960'}(\mathbf{MovieStar})))$$

$\downarrow$ evaluation over tables (a lot of things)

(Result)

# The place of RA in DBMS architecture



# 2. Algebras and Algebraic Laws

# Algebra (a.k.a. algebraic structure)

## Definition (Algebra)

- Domain D: collection of values
- Operations: functions from $D^k$ to D ($k$ is arity of the function)

- Expressions:
  - Atomic: elements of D
  - Complex: operations applied to other expressions
    (e.g., $f(3, g(5))$)
- Expressions evaluate to elements of D

- Infix notation is often used for binary ($k = 2$) operations:
  - for example, instead of $+(3, 5)$ we write $3 + 5$
  - brackets or conventions used: $(3 + 5) \times 2$ vs. $3 + 5 \times 2$

# Example: Integer arithmetic

- Domain: Integers $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$
- Operations: usual $+$, $-$, $\times$, $/$ (all binary)

- Expression examples:

| | | |
|---|---|---|
| $2 + 5$ | evaluates to | $7$ |
| $((2 - 4) \times 5) + (8 \ / \ 2)$ | evaluates to | $-6$ |
| $8 \ / \ 3$ | evaluates to | undefined |

# Example: Regular expressions (the algebra of)

- Domain: Sets of strings over an alphabet (set of symbols) $\Sigma$
- Operations:

  binary $(k = 2)$: $\circ$ (or nothing, concatenation), $\mid$ (alternation)

  unary $(k = 1)$: $*$ (Kleene star)

- Expression examples (for $\Sigma = \{a, b\}$):

  $ab^*$ evaluates to $\{a, ab, abb, abbb, \ldots\}$

  $(a \mid b)^* ab$ evaluates to all words ending with $ab$

  (Note: here, $a$ and $b$ are abbreviations for $\{a\}$ and $\{b\}$)

- Expressible operations:

  $?$ $(e? := \epsilon \mid e$ for every domain element $e)$

  $+$ $(e^+ := ee^*$ for every domain element $e)$

# Algebraic Laws

**Definition (Algebra, reminder)**

- Domain D: collection of values
- operations: functions from $D^k$ to D ($k$ is arity of the function)

Algebraic Law: an identity (or equivalence) that holds in an algebra

Examples:

- $x + y = y + x$, $(x \times y) / y = x$ (for all integers $x, y$)
  are laws for Integer Algebra
- $(e_1 e_2) e_3 = e_1 (e_2 e_3)$, $e^+ = e e^*$ (for all sets of strings)
  are laws for RegExpr Algebra

# Named Algebraic Laws

Common laws may have names:

- binary op is commutative if $e_1 \text{ op } e_2 = e_2 \text{ op } e_1$ (for all $e_1, e_2$)
- binary op is associative if $(e_1 \text{ op } e_2) \text{ op } e_3 = e_1 \text{ op } (e_2 \text{ op } e_3)$
- binary op is idempotent if $e \text{ op } e = e$

  example: $\max(a, a) = a$ for integers

- unary op is idempotent if $\text{op}(\text{op}(e)) = \text{op}(e)$

  (not a coincidence, but no need for you to know why)

  example: $(e^*)^* = e^*$ for sets of words

- binary op is left-distributive over binary $\text{op}'$ if

  $e_1 \text{ op } (e_2 \text{ op}' e_3) = (e_1 \text{ op } e_2) \text{ op}' (e_1 \text{ op } e_3)$

  example: $a \times (b + c) = (a \times b) + (a \times c)$ for integers

  when op is commutative, we have just distributivity

# 3. Relational Algebra Intro

# Variants of Relational Algebra

- There are many RAs. Two RA may be
  - Equivalent: some operations are expressible via each other (e.g., $\times$ is expressible via $\bowtie$ and back in the presence of $\sigma$ and $\rho$)
  - One strictly more expressive than another: one has operations non-expressible in the other
  - Have different domains (e.g., named/unnamed, sets/bags)
- Full SQL requires a very expressive variant of RA for bags (Turing complete!)
- We start with a basic RA for sets under named perspective (as suggested by Codd)
- Why (except historic importance)?
  - every set is a bag
  - all set operations may be simulated by bag operations
  - fundamental core, which also covers most of real SQL queries

# (Our) Relational Algebra domain

Question: What is the domain of RA?

- Domain: (Set) relations (i.e., tables) up to their names
- Relation components (reminder):
  - Relation name: just a string (this we abstract away)
  - Signature: set of attribute names with domains (i.e., datatypes)
  - Set (no repetitions!) of relation tuples (a.k.a. records, rows): assignments of values to attribute names conforming signature
- Example:

**Tutorials**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

# (Our) Relational Algebra domain

Question: What is the domain of RA?

- Domain: (Set) relations (i.e., tables) up to their names
- Relation components (reminder):
  - Relation name: just a string (this we abstract away)
  - Signature: set of attribute names with domains (i.e., datatypes)
  - Set (no repetitions!) of relation tuples (a.k.a. records, rows): assignments of values to attribute names conforming signature
- Same example:

**R**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

# (Our) Relational Algebra operations

- Domain: Relations (up to their names)
- Core operations:
  - (set) union $\cup$
  - (set) difference $\setminus$
  - selection $\sigma_C$
  - projection $\pi_{Atts}$
  - Cartesian product $\times$ (a.k.a. cross product, cross join)
  - renaming $\varrho_N$ (technical, often not mentioned)
- Some unary, some binary, usually not always applicable
- Some parametrised: families of operations rather than single
- Expressible operations:
  - (set) intersection $\cap$
  - other joins: natural $\bowtie$, theta- $\bowtie_\theta$, semi- $\ltimes$ & $\rtimes$, etc.
  - division $\div$
  - ...

# 4. Core RA operations

# Union: binary, no parameters (example)

Set-theoretic union of tuples in same-structured relations:

**Tutorials1**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |

$\cup$

**Tutorials2**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

$=$

**Tutorials1** $\cup$ **Tutorials2**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

# Union: binary, no parameters (definition)

Set-theoretic union of tuples in same-structured relations

## Union (formal definition)

- Let $R$ and $S$ be two relations
- Union $R \cup S$ is the relation that
  - defined when $R$ and $S$ have the same signature (i.e., same attribute names and their datatypes)
  - has this signature
  - contains each tuple that is either in $R$ or in $S$ (or both)

Set-theoretic union: repetitions are removed!

# Difference: binary, no parameters (example)

Set-theoretic difference of tuples in same-structured relations:

**Tutorials1**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |

$\setminus$

**Tutorials2**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

$=$

**Tutorials1** $\setminus$ **Tutorials2**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |

# Difference: binary, no parameters (definition)

Set-theoretic difference of tuples in same-structured relations

## Difference (formal definition)

- Let $R$ and $S$ be two relations
- Difference $R \setminus S$ is the relation that
  - defined when $R$ and $S$ have the same signature (i.e., attribute names together with their datatypes)
  - has the same signature
  - contains each tuple that is in $R$ but not in $S$

No repetitions anyway (by construction)

# Selection: unary, one parameter (example)

Selects tuples from a relation satisfying condition in the parameter:

$\sigma_{ID>2 \ \& \ topic=\text{'Databases'}}$ (**Tutorials**:)

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

$=$

$\sigma_{ID>2 \ \& \ topic=\text{'Databases'}}$**(Tutorials)**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

# Selection: unary, one parameter (definition)

Selects tuples from a relation satisfying condition in the parameter

## Selection (formal definition)

- Let $R$ be a relation
- Let $C$ be a condition—that is, Boolean (i.e., using $\&$, $\vee$, $\neg$) combination of atoms
  - $e_1 = e_2$, $e_1 \neq e_2$, where $e_1$ and $e_2$ are attribute names of $R$ or some values (numbers, strings, etc.)
  - domain specific operations over attribute names and values, e.g., $e_1 > e_2$, $e$ `LIKE` *RegExp* (no subqueries (!!), etc.)
- Selection $\sigma_C(R)$ is the relation that
  - is defined when $C$ mentions only attribute names of $R$ and all operations in $C$ are well-typed
  - has the same signature as $R$
  - contains all tuples in $R$ satisfying $C$

# Projection: unary, one parameter (example)

Selects columns from a relation, as specified in the parameter

$\pi_{site,\ topic}$ **Tutorials**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

$=$

$\pi_{site,\ topic}$**(Tutorials)**:

| *site* | *topic* |
|---|---|
| w3schools | Databases |
| w3schools | WebDev |
| w3resource | Databases |

# Projection: unary, one parameter (definition)

Selects columns from a relation, as specified in the parameter

## Projection (formal definition)

- Let $R$ be a relation
- Let *Atts* be a list of attribute names
- Projection $\pi_{Atts}(R)$ is the relation that
  - is defined when *Atts* mentions only attribute names of $R$
  - has signature *Atts* (with domains from $R$)
  - contains projections (i.e., restrictions) of all tuples in $R$ to *Atts*

Repetitions are removed!

# Cartesian product: binary, no parameters (example)

Combines tuples from two relations in all possible ways:

**Tutorials**:

| *ID* | *site* | *tutorial* |
|---|---|---|
| 1 | w3schools | SQL_2003STD |
| 2 | w3schools | HTML_5 |
| 3 | w3schools | CSS_3 |
| 4 | w3resource | SQL_2003STD |
| 5 | w3resource | MySQL |

$\times$

**Topics**:

| *topicID* | *topic* |
|---|---|
| 1 | Databases |
| 2 | WebDev |

$=$

**Tutorials** $\times$ **Topics**:

| *ID* | *site* | *tutorial* | *topicID* | *topic* |
|---|---|---|---|---|
| 1 | w3schools | SQL_2003STD | 1 | Databases |
| 2 | w3schools | HTML_5 | 1 | Databases |
| 3 | w3schools | CSS_3 | 1 | Databases |
| 4 | w3resource | SQL_2003STD | 1 | Databases |
| 5 | w3resource | MySQL | 1 | Databases |
| 1 | w3schools | SQL_2003STD | 2 | WebDev |
| 2 | w3schools | HTML_5 | 2 | WebDev |
| 3 | w3schools | CSS_3 | 2 | WebDev |
| 4 | w3resource | SQL_2003STD | 2 | WebDev |
| 5 | w3resource | MySQL | 2 | WebDev |

# Cartesian product: binary, no parameters (definition)

Combines tuples from two relations in all possible ways

## Cartesian (or Cross) Product (formal definition)

- Let $R$ and $S$ be two relations
- Cartesian product $R \times S$ is the relation that
  - is defined when $R$ and $S$ have no(!) common attribute name
  - has the union of signatures of $R$ and $S$ as signature
  - contains all possible combinations of tuples in a tuple in $R$ with a tuple in $S$

- Note: first operation that has more in output than in input
- Maybe really large
- Makes little sense, not used in practice much
- Useful to present natural join $\bowtie$ (see below)

# Renaming: unary, one parameter

Renames attributes in a relation as specified in the parameter:

$\rho_{topic \to area}$ **ProjTutorials**:

| *site* | *topic* |
|---|---|
| w3schools | Databases |
| w3schools | WebDev |
| w3resource | Databases |

$=$ $\rho_{topic \to area}$**(ProjTutorials)**:

| *site* | *area* |
|---|---|
| w3schools | Databases |
| w3schools | WebDev |
| w3resource | Databases |

## Renaming (formal definition)

- Let $R$ be a relation
- Let $N$ be a list $A_1 \to A'_1, \ldots, A_n \to A'_n$, where $A_1, \ldots, A_n, A'_1, \ldots, A'_n$ are all different attribute names
- Renaming $\rho_N(R)$ is the relation that
  - is defined when $A_1, \ldots, A_n$ are attribute names of $R$ and $A'_1, \ldots, A'_n$ are not
  - is the same as $R$ except that $A_1, \ldots, A_n$ renamed to $A'_1, \ldots, A'_n$

# Independent set of operations

## Core Set RA

*Selection* $\sigma_C$, *Projection* $\pi_L$, *Cartesian Product* $\times$,
*Union* $\cup$, *Difference* $\setminus$, *Renaming* $\rho_N$

- Can express everything in Codd's algebra (but not all SQL!)
- These are independent operations: dropping any of them makes the language weaker
- There are other useful expressible operations: intersection $\cap$, natural join $\bowtie$, division $\div$, etc.
  - there are effective algorithms for them
  - it is often simpler to formulate queries using them
- We discuss them after algebraic laws for core operations

# 5. Algebraic laws for core RA operations

# RA Algebraic Laws: initial observations

The power of RA is largely in its rich set of algebraic laws

- allows to convert complex queries to equivalent but simpler ones (before evaluating them)
- 'simple' here means not 'short' or 'neat-looking', but 'expected to be evaluated quicker'
- we will discuss how to decide what is simpler (in this sense) than other later on
- there are (too) many laws, and we consider only most prominent examples
- start now with core operations, later touch expressible ones

# Laws for set-theoretic operations

Union is commutative, associative, and idempotent:

$$R \cup S = S \cup R \qquad (R \cup S) \cup T = R \cup (S \cup T) \qquad R \cup R = R$$

(for all relations $R, S, T$)

Difference is right-distributive over union:

$$(R \cup S) \setminus T = (R \setminus T) \cup (S \setminus T)$$

Question: Is it left-distributive: $R \setminus (S \cup T) \stackrel{?}{=} (R \setminus S) \cup (R \setminus T)$?

Answer: No (come to counter-example by yourself)

Other (unnamed) laws:

$$R \setminus (S \cup T) = (R \setminus S) \setminus T = (R \setminus T) \setminus S$$

...

# Laws for Cartesian product

Cartesian product is commutative and associative:

$$R \times S = S \times R \qquad (R \times S) \times T = R \times (S \times T)$$

(for all relations $R, S, T$)

Question: is it idempotent ($R \times R \stackrel{?}{=} R$)?

Answer: Of course no: it is never defined

Cartesian product is distributive over both union and difference:

$$(R \cup S) \times T = (R \times T) \cup (S \times T)$$
$$(R \setminus S) \times T = (R \times T) \setminus (S \times T)$$

Other (unnamed) laws:

...

# Laws for selection

Selection is idempotent:

$$\sigma_C(\sigma_C(R)) = \sigma_C(R) \quad \text{(for every relation } R \text{ and condition } C\text{)}$$

Conditions can be reshuffled and split:

$$\sigma_{C_1 \& C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R)) = \sigma_{C_2}(\sigma_{C_1}(R))$$
$$\sigma_{C_1 \vee C_2}(R) = \sigma_{C_1}(R) \cup \sigma_{C_2}(R)$$

Selection can be pushed through union, difference, and Cartesian product (not always!):

$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$
$$\sigma_C(R \setminus S) = \sigma_C(R) \setminus \sigma_C(S) \quad = \sigma_C(R) \setminus S$$
$$\sigma_C(R \times S) = \sigma_C(R) \times S \quad \text{if } C \text{ mentions only attributes of } R$$

... (other laws)

# Laws for projection

Projection is idempotent:

$\pi_{Atts}(\pi_{Atts}(R)) = \pi_{Atts}(R)$ (for relation $R$ and attributes $Atts$)

Idempotence can be generalised [corrected!]:

$\pi_{Atts}(R) = \pi_{Atts}(\pi_{Atts,Atts'}(R))$

Projection can be pushed through union and Cartesian product:

$\pi_{Atts}(R \cup S) = \pi_{Atts}(R) \cup \pi_{Atts}(S)$

$\pi_{Atts_R,Atts_S}(R \times S) = \pi_{Atts_R}(R) \times \pi_{Atts_S}(S)$

if $Atts_R$ are attributes of $R$ and $Atts_S$ are attributes of $S$

... (other laws)

# Laws for renaming

. . .

. . .

Homework: find a couple by yourself

# 6. (Some) expressible operations

# Intersection: binary, no parameters

Set-theoretic intersection of tuples in same-structured relations:

**Tutorials1**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | Databases |
| 2 | w3schools | HTML_5 | WebDev |
| 3 | w3schools | CSS_3 | WebDev |

$\cap$

**Tutorials2**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 3 | w3schools | CSS_3 | WebDev |
| 4 | w3resource | SQL_2003STD | Databases |
| 5 | w3resource | MySQL | Databases |

$=$

**Tutorials1 $\cap$ Tutorials2**:

| *ID* | *site* | *tutorial* | *topic* |
|---|---|---|---|
| 3 | w3schools | CSS_3 | WebDev |

## Intersection (formal definition)

*Intersection:* $R \cap S := R \setminus (R \setminus S)$ *for every* $R$ *and* $S$

*(including the undefined cases)*

# Natural Join: binary, no parameters (example)

Combines tuples from two relations in all possible ways that agree on common attributes:

**Tutorials**:

| *ID* | *site* | *tutorial* | *topicID* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | 1 |
| 2 | w3schools | HTML_5 | 2 |
| 3 | w3schools | CSS_3 | 2 |
| 4 | w3resource | SQL_2003STD | 1 |
| 5 | w3resource | MySQL | 1 |

$\bowtie$

**Topics**:

| *topicID* | *topic* |
|---|---|
| 1 | Databases |
| 2 | WebDev |
| 2 | WebDev1 |

$=$

**Tutorials** $\bowtie$ **Topics**:

| *ID* | *site* | *tutorial* | *topicID* | *topic* |
|---|---|---|---|---|
| 1 | w3schools | SQL_2003STD | 1 | Databases |
| 2 | w3schools | HTML_5 | 2 | WebDev |
| 2 | w3schools | HTML_5 | 2 | WebDev1 |
| 3 | w3schools | CSS_3 | 2 | WebDev |
| 3 | w3schools | CSS_3 | 2 | WebDev1 |
| 4 | w3resource | SQL_2003STD | 1 | Databases |
| 5 | w3resource | MySQL | 1 | Databases |

# Natural Join: binary, no parameters (definition)

## Natural Join (formal definition)

Let $R$ and $S$ be two relations:

- $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_m$ their attributes
- $\{B_1, \ldots, B_k\} = \{A_1, \ldots, A_n\} \cap \{A'_1, \ldots, A'_m\}$: common

Natural Join: $R \bowtie S := \rho_N(\pi_L(\sigma_C(\rho_{N_R}(R) \times \rho_{N_S}(S))))$ where

- $N_R$ and $N_S$ rename attributes of $R$ and $S$ apart:

$$\begin{aligned} N_R &= \{A_1 \to R.A_1, \ \ldots \ , A_n \to R.A_n\} \\ N_S &= \{A'_1 \to S.A'_1, \ \ldots \ , A_m \to S.A'_m\} \end{aligned}$$

- $C$ is the condition matching common attributes:

$$C = (R.B_1 = S.B_1 \ \& \ \cdots \ \& \ R.B_k = S.B_k)$$

- $L$ are all attributes of $R$ and $S$ except the repetitions:

$$L = R.A_1, \ \ldots \ , R.A_n, \ \ S.B'_1, \ \ldots \ , S.B'_\ell$$

where $\{B'_1, \ldots, B'_\ell\} = \{A'_1, \ldots, A'_m\} \setminus \{A_1, \ldots, A_n\}$

- $N$ renames attributes back:

$$N = \{R.A_1 \to A_1, \ldots, R.A_n \to A_n, S.B'_1 \to B'_1, \ldots, S.B'_\ell \to B'_\ell\}$$

# Natural Join (observations)

- Product $\times$ is expressible in the presence of natural join $\bowtie$
  - Question: Any ideas how?
  - Answer: $R \times S := R \bowtie S$ for every relations $R, S$ when $R \times S$ is defined

  (the difference is that $\bowtie$ is defined in more cases than $\times$)
- Another core set RA: selection $\sigma_C$, projection $\pi_L$, natural join $\bowtie$, union $\cup$, difference $\setminus$, renaming $\rho_N$

- Natural join is one of the most used operations in practice
- Implementing $\bowtie$ via $\times$ is extremely non-effective
- Native algorithms for $\bowtie$ are common (we will see)

# Theta Join: binary, one parameter (example)

Normal join + selection:

**Tutorials**:

| *ID* | *site* | *tutorial* | *topicID* |
|---|---|---|---|
| 1 | w3schools | SQL_2003STD | 1 |
| 2 | w3schools | HTML_5 | 2 |
| 3 | w3schools | CSS_3 | 2 |
| 4 | w3resource | SQL_2003STD | 1 |
| 5 | w3resource | MySQL | 1 |

$\bowtie_{ID > topicID}$

**Topics**:

| *topicID* | *topic* |
|---|---|
| 1 | Databases |
| 2 | WebDev |

$=$

**Tutorials** $\bowtie_{ID > topicID}$ **Topics**:

| *ID* | *site* | *tutorial* | *topicID* | *topic* |
|---|---|---|---|---|
| 3 | w3schools | CSS_3 | 2 | WebDev |
| 4 | w3resource | SQL_2003STD | 1 | Databases |
| 5 | w3resource | MySQL | 1 | Databases |

(This is not very meaningful query,
but the operation is very useful in practice)

# Theta Join: binary, one parameter (definition)

## Theta Join (formal definition)

Theta Join: $R \bowtie_\theta S := \sigma_\theta(R \bowtie S)$ for relations $R$ and $S$, and condition $\theta = RAtt_1\ \mathrm{op}_1\ SAtt_1\ \&\ \cdots\ \&\ RAtt_k\ \mathrm{op}_k\ SAtt_k$ where $RAtt_i$ and $SAtt_i$ are attributes of $R$ and $S$, and $\mathrm{op}_i \in \{=, \neq, >, <, \geq, \leq\}$ (including the undefined cases)

- Question: Why to bother? Why only these conditions?
- Answer:
  - one the one hand, very common in practice
  - one the other, we can design specialised efficient algorithms
- Equi-Join: theta join with $=$ as all $\mathrm{op}_i$ (e.g., $R \bowtie_{RAtt=SAtt} S$)
  - essentially, naturally join with incorporated renaming
  - even more common and own algorithms

# Semi-joins, anti-joins: binary, no parameters

## Semi-Joins (formal definition)

Left Semi-Join: $R \ltimes S := \pi_{RAtts}(R \bowtie S)$ for relations $R$ and $S$ where $RAtts$ are attributes of $R$

Right Semi-Join: $R \rtimes S := \pi_{SAtts}(R \bowtie S)$ for relations $R$ and $S$ where $SAtts$ are attributes of $S$ (symmetric: $R \rtimes S = S \ltimes R$)

- Left Semi-Join: all tuples in $R$ that join with tuples in $S$

## Anti-Joins (formal definition)

*Left Anti-Join:* $R \triangleright S := R \setminus (R \ltimes S)$ *for relations* $R$ *and* $S$

*Right Anti-Join:* $R \triangleleft S := \ldots$ *(symmetric:* $R \triangleleft S = S \triangleright R$*)*

- Left Anti-Join: all tuples in $R$ that do not join with tuples in $S$

Question: Where are Outer Joins? Answer: Next time: not in classical relational algebra, need NULLs

# Division: binary, no parameters (example 1)

'Inverse' of Cartesian Product:

**TutTimesTop**:

| *ID* | *site* | *tutorial* | *topicID* | *topic* |
|---|---|---|---|---|
| 1 | w3schools | SQL_2003STD | 1 | Databases |
| 2 | w3schools | HTML_5 | 1 | Databases |
| 3 | w3schools | CSS_3 | 1 | Databases |
| 4 | w3resource | SQL_2003STD | 1 | Databases |
| 5 | w3resource | MySQL | 1 | Databases |
| 1 | w3schools | SQL_2003STD | 2 | WebDev |
| 2 | w3schools | HTML_5 | 2 | WebDev |
| 3 | w3schools | CSS_3 | 2 | WebDev |
| 4 | w3resource | SQL_2003STD | 2 | WebDev |
| 5 | w3resource | MySQL | 2 | WebDev |

$\div$

**Topics**:

| *topicID* | *topic* |
|---|---|
| 1 | Databases |
| 2 | WebDev |

$=$

**TutTimesTop $\div$ Topics**:

| *ID* | *site* | *tutorial* |
|---|---|---|
| 1 | w3schools | SQL_2003STD |
| 2 | w3schools | HTML_5 |
| 3 | w3schools | CSS_3 |
| 4 | w3resource | SQL_2003STD |
| 5 | w3resource | MySQL |

# Division: binary, no parameters (example 2)

'Inverse' of Cartesian Product:

**TutTimesTop'**:

| *ID* | *site* | *tutorial* | *topicID* | *topic* |
|---|---|---|---|---|
| 1 | w3schools | SQL_2003STD | 1 | Databases |
| 3 | w3schools | CSS_3 | 1 | Databases |
| 4 | w3resource | SQL_2003STD | 1 | Databases |
| 5 | w3resource | MySQL | 1 | Databases |
| 1 | w3schools | SQL_2003STD | 2 | WebDev |
| 2 | w3schools | HTML_5 | 2 | WebDev |
| 4 | w3resource | PostgreSQL | 2 | WebDev |
| 5 | w3resource | MySQL | 2 | WebDev |
| 5 | w3resource | PostgreSQL | 2 | WevDev |

$\div$

**Topics**:

| *topicID* | *topic* |
|---|---|
| 1 | Databases |
| 2 | WebDev |

$=$

**TutTimesTop $\div$ Topics**:

| *ID* | *site* | *tutorial* |
|---|---|---|
| 1 | w3schools | SQL_2003STD |
| 5 | w3resource | MySQL |

# Division: binary, no parameters (definition)

## Division (formal definition)

Division: $R \div S := \pi_{Atts}(R) \setminus \pi_{Atts}((\pi_{Atts}(R) \times S) \setminus R)$ for relations $R$ and $S$, where $Atts$ is the attributes of $R$ not mentioned in $S$ (including the undefined cases)

- Defined only when each attribute of $S$ is an attribute of $R$ (follows from the definition)
- Equivalently, can be defined as:

  $R \div S$ is all tuples $t$ from $\pi_{Atts}(R)$ such that $\{t\} \times S \subseteq R$
- As expected, $(R \times S) \div S = R$ for every $R$ and $S$
- However, $(R \div S) \times S$ may be different from $R$
- (cf. Euclidian division, a.k.a. division with reminder for integers)

# (Some) algebraic laws for expressible operations

Our definitions of the expressible operations are essentially laws:

e.g., $R \div S = \pi_{Atts}(R) \setminus \pi_{Atts}((\pi_{Atts}(R) \times S) \setminus R)$

Intersection $\cap$ is commutative, associative, and idempotent, as well as distribute over union $\cup$

Natural join is commutative, associative, and idempotent:

$R \bowtie S = S \bowtie R \quad (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \quad R \bowtie R = R$

(the most important laws in practice)

Selection can be pushed through join (not always!):

$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$ if $C$ mentions only attributes of $R$

(also very important)

... (other laws)

# What have we learned?

Classic Relational Algebra:

core and expressible operations, algebraic laws