Course 6. Relational Algebra

Preliminaries

Query languages allow manipulation and retrieval of data from a database. Having a strong formal foundation based on logic, relational model supports simple, powerful query languages

Query Languages are not usual programming languages, they:

- are not expected to be "*Turing complete*",
- are not intended to be used for complex calculations,
- support easy, efficient access to large data sets.

Two mathematical query languages form the basis for "real" languages (e.g. SQL), and for implementation:

- Relational Algebra: More operational, very useful for representing execution plans;
- Relational Calculus: Lets users describe what they want, rather than how to compute it (non-operational, declarative language).

In Relational Algebra a query is applied to *relation instances*, and the result of a query is also a *relation instance*.

The schema of *input* relations for a query are fixed (but the query will run regardless of instance). The schema for the *result* of a given query is also fixed, being determined by the definition of query language constructs.

Positional vs. named-field notation:

- positional notation is easier for formal definitions,
- named-field notation is more readable.

Both variants are used in SQL & Relational Algebra

Basic Relational Algebra Operations

Basic operations:

- <u>*Projection*</u> (π): Deletes unwanted columns from relation.
- <u>*Selection*</u> (σ): Selects a subset of rows from relation.
- <u>*Cross-product*</u> (X): Allows us to combine two relations.
- <u>*Set-difference*</u> (-): Tuples in reln. 1, but not in reln. 2.
- <u>*Union*</u> ($\cup$): Tuples in reln. 1 and in reln. 2.

Additional operations (not essential, but veryuseful):

- Intersection ($\cap$), join ($\otimes$), division (/), renaming (ρ)

Since each operation returns a relation, operations can be *composed* (relational algebra is "closed").

Projection

L =(a_1, …, a_n) is a list of attributes (i.e. *a list of columns*) of the relation R

Keeping vertical slices of a relation according to L

$$\pi_L(R) = \{\ t \mid t_1 \in R \wedge$$
$$t.a_1 = t_1.a_1 \wedge$$
$$\ldots \wedge$$
$$t.a_n = t_1.a_n \}$$

Projection sample:

$\pi_{cid,\ grade}$(Enrolled)

$\pi_{cid,\ grade}$(

sid	cid	grade
1234	Alg1	9
1235	Alg1	10
1234	DB1	10
1234	DB2	9
1236	DB1	7
1237	DB2	9`
1237	DB1	5
1237	Alg1	10

) =

cid	grade
Alg1	9
Alg1	10
DB1	10
DB2	9
DB1	7
DB1	5

Because relational algebra works with sets, there are no duplicates in the result. So, $\pi_{cid,\ grade}$(Enrolled) is equivalent with the following SQL query: SELECT <u>DISTINCT</u> cid, grade FROM Enrolled

Selection

Selecting the t-uples of a relation R verifying a condition *c* (*selection predicate*).

$$\sigma_c(R) = \{\ t \mid t \in R \wedge c\ \}$$

Selection sample:

$\sigma_{grade > 8}$(Enrolled) = {t | t $\in$ Enrolled $\wedge$ grade>8 }

$\sigma_{grade > 8}$(

sid	cid	grade
1234	Alg1	9
1235	Alg1	10
1234	DB2	9
1236	DB1	7
1237	DB1	5
1237	Alg1	6

) =

sid	cid	grade
1234	Alg1	9
1235	Alg1	10
1234	DB2	9

$\sigma_{grade > 8}$(Enrolled) is equivalent with SELECT DISTINCT * FROM Enrolled WHERE grade > 8

The selection condition have the following structure: **Term Op Term**, where **Term** is an attribute name a constant and **Op** is one of the following operators: <, >, =, ≠ etc. Also, **(C1 $\wedge$ C2)**, **(C1 $\vee$ C2)**, **($\neg$ C1)** are conditions, where C1 and C2 are conditions

The result of an operation is a relation, so it could be an input for another operation (composability):

$$\pi_{cid,\ grade}(\sigma_{\ grade > 8}(\text{Enrolled}))$$

$\pi_{cid,\ grade}(\sigma_{\ grade > 8}($

sid	cid	grade
1234	Alg1	9
1235	Alg1	10
1234	DB1	10
1234	DB2	9
1236	DB1	7
1237	DB2	9`
1237	DB1	5
1237	Alg1	10

$)) =$

cid	grade
Alg1	9
Alg1	10
DB1	10
DB2	9

Union, Intersection, Set-difference

$R_1 \cup R_2 = \{ t \mid t \in R_1 \vee t \in R_2 \}$

$R_1 \cap R_2 = \{ t \mid t \in R_1 \wedge t \in R_2 \}$

$R_1 - R_2 = \{ t \mid t \in R_1 \wedge t \notin R_2 \}$

The relations R_1 and R_2 must be union compatible:

- same number of attributes (same *arity*)
- corresponding attributes have *compatible* domains and the *same name*

$R_1 \cup R_2$

```
SELECT DISTINCT *
FROM R1
UNION
SELECT DISTINCT *
FROM R2
```

$R_1 \cap R_2$

```
SELECT DISTINCT *
FROM R1
```

INTERSECT

SELECT DISTINCT *

FROM R_2

`

$R_1 — R_2$

SELECT DISTINCT *

FROM R_1

EXCEPT

SELECT DISTINCT *

FROM R_2

Not all operations are essential. Intersection of two relations could be expressed using union and set-difference:

$$R_1 \cap R_2 = ((R_1 \cup R_2) — (R_1 — R_2)) — (R_2 — R_1)$$

Compute all tuples belonging to R_1 or R_2

Remove the ones that belong only to R_1

Remove the ones that belong only to R_2

Cartesian Product

Combining two relations

$R_1(a_1, \ldots, a_n)$ and $R_2(b_1, \ldots, b_m)$

$R_1 \text{ X } R_2 = \{ t \mid t_1 \in R_1 \wedge t_2 \in R_2$

$\wedge \quad t.a_1 = t_1.a_1 \ \ldots \wedge \quad t.a_n = t_1.a_n$

$\wedge \quad t.b_1 = t_2.b_1 \ldots \wedge \quad t.b_m = t_2.b_m\}$

Equivalent SQL query: **SELECT DISTINCT * FROM R_1, R_2**

θ-Join

Combining two relations R_1 and R_2 on a condition *c*

$$R_1 \otimes_c R_2 \ = \sigma_c (R_1 \text{ X } R_2)$$

Students $\otimes_{Students.sid=Enrolled.sid}$ Enrolled

Equivalent SQL queries:

SELECT DISTINCT * FROM Students,Enrolled WHERE Students.sid = Enrolled.sid

SELECT DISTINCT * FROM Students INNER JOIN Enrolled ON Students.sid=Enrolled.sid

The Equi-Join

Combines two relations on a condition composed only of equalities of attributes of the first and second relation and projects only one of the redundant attributes (since they are equal)

$$R_1 \otimes_{E(c)} R_2$$

The Natural Join

Combines two relations on the equality of the attributes with the same names and projects only one of the redundant attributes

$$R_1 \otimes R_2$$

Division

Division is not supported as a primitive operator, but it is useful

Let R_1 have 2 fields, x and y; R_2 have only field y:

$$R_1 / R_2 = \{ \langle x \rangle \mid \exists \langle x,y \rangle \in R_1 \quad \forall \langle y \rangle \in R_2 \}$$

i.e., ***R₁* / *R₂* contains all *x* tuples such that for *<u>every</u>* *y* tuple in *R₂*, there is an *xy* tuple in *R₁*.**

Or: If the set of y values associated with an x value in R_1 contains all y values in R_2, the x value is in R_1/R_2. In general, x and y can be any lists of fields; y is the list of fields in R_2, and $x \cup y$ is the list of fields of R_1.

Division is not an essential operation, but just a useful shorthand.

Idea: For R_1 / R_2, compute all x values that are not `disqualified' by some y value in R_2.

x value is *disqualified* if by attaching y value from R_2, we obtain an xy tuple that is not in R_1.

Disqualified x values: $\pi_x ((\pi_x(R_1) \text{ X } R_2) - R_1)$

$$R_1 / R_2 = \pi_x(R_1) - \text{all disqualified values}$$

Renaming

If attributes or relations have the same name (for instance when joining a relation with itself) it may be convenient to rename one

$$\rho(R' (N_1 \rightarrow N'_1, N_2 \rightarrow N'_2), R)$$

alternative notation: $\rho_{R' (N'1, N'2)}(R)$, The new relation R' has the same instance has R, its schema has attribute N'_i instead of attribute N_i

Example:

$$\rho(\text{Courses2 (cid} \rightarrow \text{code, cname} \rightarrow \text{description), Courses)}$$

Courses

cid	cname	credits
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

→

Courses2

code	description	credits
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

```
SELECT cid as code,
       cname as description,
       credits
FROM Courses Courses2
```

Assignment Operation

The assignment operation $\leftarrow$ provides a convenient way to express complex queries.

Assignment must always be made to a temporary relation variable

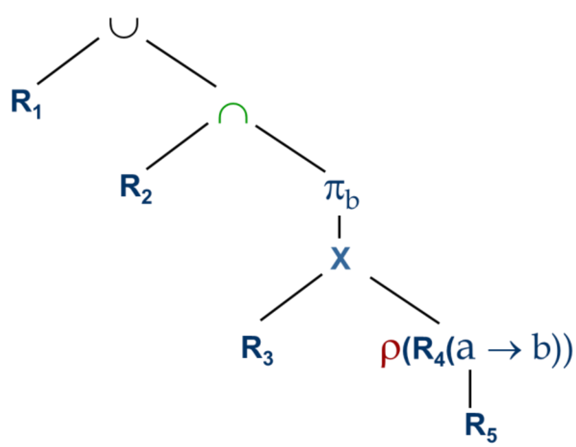
$$\text{Temp} \leftarrow \pi_x(R_1 \text{ X } R_2)$$

The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.

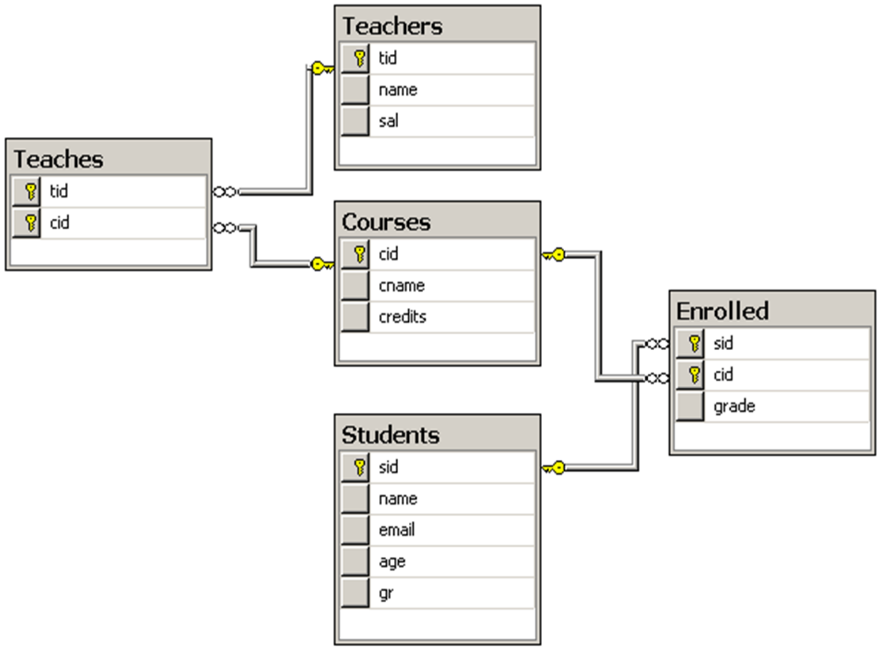
May use variable in subsequent expressions:

$$\text{result} \leftarrow \text{Temp} - R_3$$

The *execution plan* of a complex expression: $R_1 \cup (R_2 \cap \pi_b (R_3 \text{ X } \rho(R_4(a \rightarrow b), R_5)))$



Examples



1. Find names of students enrolled at 'BD1'

Solution 1: π_{name} (($\sigma_{cid=\text{'BD1'}}$(Enrolled)) $\otimes$ Students)

Solution 2: ρ ($Temp_1$, $\sigma_{cid=\text{'BD1'}}$(Enrolled))

ρ ($Temp_2$, $Temp_1 \otimes$ Students)

π_{name} ($Temp_2$)

Solution 3: π_{name} ($\sigma_{cid=\text{'BD1'}}$(Enrolled $\otimes$ Students))

2. Find names of students enrolled at a 5 credits course

Information about course credits only available in Courses; so need an extra join:

$$\pi_{name} ((\sigma_{credits=5}(\text{Courses})) \otimes \text{Enrolled} \otimes \text{Students})$$

A more efficient solution:

$$\pi_{name} (\pi_{sid}(\pi_{cid}(\sigma_{credits=5}(\text{Courses})) \otimes \text{Enrolled}) \otimes \text{Students})$$

A query optimizer can find this, given the first solution!

3. Find students enrolled at a 4 or 5 credits course

Can identify all 4 or 5 credits courses, then find students who're enrolled in one of these courses:

ρ (*TempCourses*, ($\sigma_{\text{credits=4} \vee \text{credits=5}}$(Courses)))

π_{name} (*TempCourses* $\otimes$ Enrolled $\otimes$ Students)

Can also define *TempCourses* using union!

What happens if $\vee$ is replaced by $\wedge$ in this query?

4. Find students enrolled at a 5 <u>and</u> 4 credits course

Previous approach won't work! Must identify students who're enrolled at 4 credits courses, students who're enrolled at 5 credits courses, then find the intersection (note that *sid* is a key for Students):

ρ (*Temp4*, π_{sid}($\sigma_{\text{credits=4}}$ (Courses) $\otimes$ Enrolled))

ρ (*Temp5*, π_{sid}($\sigma_{\text{credits=5}}$ (Courses) $\otimes$ Enrolled))

π_{name} ((*Temp4* $\cap$ *Temp5*) $\otimes$ Students)

5. Find names of students enrolled at all courses

Uses division; schemas of the input relations must be carefully chosen:

ρ (*TempSIDs*, $\pi_{\text{sid, cid}}$(Enrolled) / π_{cid} (Courses))

π_{name}(*TempSIDs* $\otimes$ Students)

Extended Relational Algebra Operations

Generalized projection

Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\pi_{F1,\, F2, \ldots,\, Fn}\ (R)$$

R is any relational-algebra expression

Each of $F_1, F_2, \ldots, F_n$ are are arithmetic expressions involving constants and attributes in the schema of *R*.

Aggregate Functions and Operations

Aggregation function takes a collection of values and returns a single value as a result.

avg: average value
min: minimum value

max: maximum value
sum: sum of values
count: number of values

Aggregate operation in relational algebra

$$_{G_1, G_2, \ldots, G_n}\, \mathcal{G}_{F_1(A_1), F_2(A_2), \ldots, F_n(A_n)}\, (R)$$

R is any relational-algebra expression

G_1, G_2 …, G_n is a list of attributes on which to group (can be empty)

Each F_i is an aggregate function

Each A_i is an attribute name

Example

Relation R:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$g_{\text{sum(C)}}$ (R) =

sum(C)
27

Result of aggregation does not have a name

- Can use rename operation to give it a name
- For convenience, we permit renaming as part of aggregate operation

Outer Join

An extension of the join operation that avoids loss of information.

Left Outer Join ⟕

Right Outer Join ⟖

Full Outer Join ⟗

Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.

Uses null values:

- null signifies that the value is unknown or does not exist
- all comparisons involving null are (roughly speaking) **false** by definition.

Modification of the Database

The content of the database may be modified using the following operations:

Deletion $R \leftarrow R - E$

Insertion $R \leftarrow R \cup E$

Updating $R \leftarrow \pi_{F1, F2, \ldots, Fn}(R)$

All these operations are expressed using the assignment operator.