Relational Algebra

2015-1학기

권동섭

Goal

- Relational Algebra
  - Basic Operation
  - Additional Operation

RELATIONAL ALGEBRA

- Relational Model에서의 Query란?
- Relational Algebra란?

Database의 이용 예

1. Design & Create Database (using DDL)
2. Load Data (using DML)
3. Execute **Query** / Modify Data (using DML)
4. 3을 반복

Query?

- 예
  - “DB930”교과목에서 “A+”학점을 받은 학생의 이름과 학번은?
  - 컴퓨터공학과 교과목 별 수강 인원과 평점 평균은?
  - “운영체제” 교과목을 수강하지 않은 4학년 학생은?
- Relational Query Languages
  - 새로운 Relation을 만들어내는 연산
  - Relation = Query (Relations…)
  - 종류
    - **Relational Algebra**
    - Tuple Relational Calculus
    - Domain Relational Calculus
    - **SQL**

BASIC OPERATION

select: σ
project: Π
union: $\cup$
set difference: $-$
cartesian product: **x**
rename: ρ

Relational Algebra

- Procedural language
- Six basic/primitive operators
  - select: **σ**
  - project: Π
  - union: $\cup$
  - set difference: **–**
  - cartesian product: **x**
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.

Select

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms**
connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq, <, \leq$

- Getting a "**Horizontal Subset**"

- Example of selection:

$$\sigma_{\text{dept\_name} = \text{"Physics"}}(\textit{instructor})$$

$$\sigma_{\text{branch-name} = \text{"Perryridge"}}(\textit{account})$$

MYONGJI UNIVERSITY

Project

- Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets!
- Getting a "**Vertical Subset**"
- May be used for reordering the columns!
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

- To eliminate the *branch-name* attribute of *account*

$$\Pi_{account\text{-}number,\ balance}\ (account)$$

MYONGJI UNIVERSITY

Union

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. r, s must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of r deals with the same type of values as does the 2nd column of s)
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both (section relation)

$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{“Fall”}\ \wedge\ year=2009}\,(section))\ \cup$$

$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{“Spring”}\ \wedge\ year=2010}\,(section))$$

- to find all customers with either an account or a loan (Banking example)

$$\Pi_{customer\text{-}name}\,(depositor)\ \cup\ \Pi_{customer\text{-}name}\,(borrower)$$

Set-Difference

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{"Fall"} \ \land \ year=2009}(section)) \ -$$
$$\Pi_{course\_id}(\sigma_{semester=\text{"Spring"} \ \land \ year=2010}(section))$$

Cartesian-Product

- Notation $r \times s$

- Defined as:

$$r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint.
  (That is, $R \cap S = \emptyset$).

- If attributes of *r(R)* and *s(S*) are not disjoint, then renaming must be used.

Rename

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_{X}(E)$$

returns the expression E under the name X

- If a relational-algebra expression E has arity n, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

e.g., $\rho_d(account)$

Example

학생

학번	이름	주민등록번호	주소	지도교수
60072345	한승연	8816XXXXX	서울	11215
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

교수

교번	이름	학과
11215	권동섭	컴퓨터공학과
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...

학과

학과명	학과장
컴퓨터공학과	11277
...	...
...	...
...	...
...	...
...	...
...	...

- "컴퓨터공학과"에 소속된 교수의 교번과 이름은?
- 각 학과의 "학과명"과 학과장의 "이름"은?
- 모든 학생과 교수의 "번호", "이름"은?

BASIC OPERATOR EXERCISE

Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-city)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Example Queries

- Find all loans of over $1200

$$\sigma_{amount > 1200} (loan)$$

- Find the loan number for each loan of an amount greater than $1200

$$\Pi_{loan\text{-}number} (\sigma_{amount > 1200} (loan))$$

Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer\text{-}name}\ (borrower) \cup \Pi_{customer\text{-}name}\ (depositor)$$

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\text{-}name}\ (borrower) \cap \Pi_{customer\text{-}name}\ (depositor)$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Perryridge”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan))) - \Pi_{customer\text{-}name}(depositor)$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$$

  - Query 2

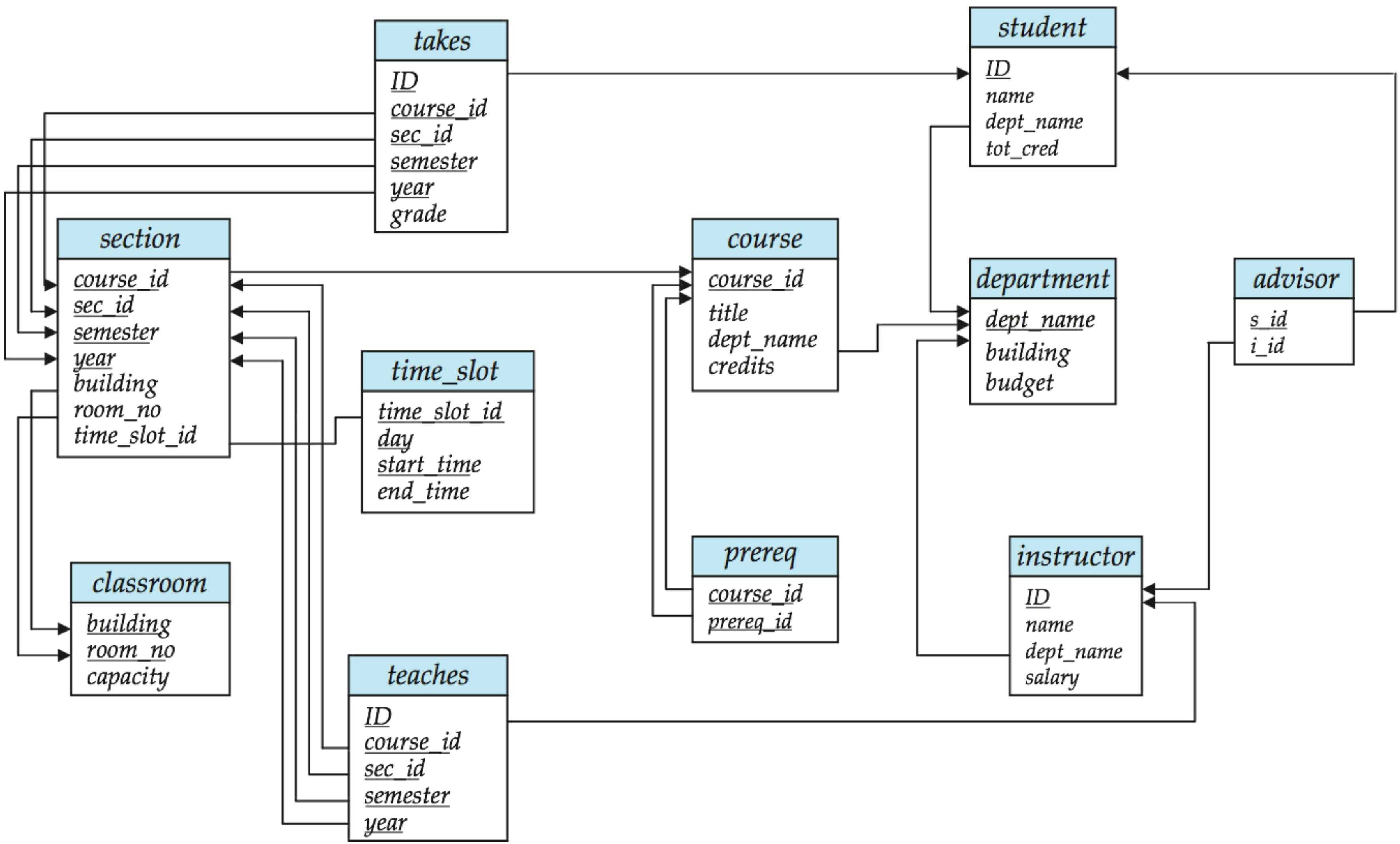
$$\Pi_{customer\text{-}name}(\sigma_{loan.loan\text{-}number\ =\ borrower.loan\text{-}number}((\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}(loan)) \times borrower))$$

Example Queries

- Find the largest account balance
  - Rename *account* relation as *d*
  - Self-join

$$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance\ <\ d.balance}(account \times \rho_d(account)))$$

Schema Diagram for University Database



MYONGJI UNIVERSITY

Example Queries

- Find the names of all instructors in the Physics department, along with the *course_id* of all courses they have taught
  - Query 1

    $$\Pi_{instructor.ID,course\_id}(\sigma_{dept\_name=\text{"Physics"}}(\sigma_{instructor.ID=teaches.ID}(instructor \times teaches)))$$

  - Query 2

    $$\Pi_{instructor.ID,course\_id}(\sigma_{instructor.ID=teaches.ID}(\sigma_{dept\_name=\text{"Physics"}}(instructor) \times teaches))$$

Example Queries

- Find the largest salary in the university
  - Step 1: find instructor salaries that are less than some other instructor salary (i.e. not maximum)
    - using a copy of *instructor* under a new name *d*
    - $\Pi_{instructor.salary}(\sigma_{instructor.salary < d.salary}(instructor \times \rho_d(instructor)))$
  - Step 2: Find the largest salary
    - $\Pi_{salary}(instructor) - \Pi_{instructor.salary}(\sigma_{instructor.salary < d.salary}(instructor \times \rho_d(instructor)))$

MYONGJI UNIVERSITY

ADDITIONAL OPERATION

Set-intersection

Natural join, Theta join

Assignment

Outer join

Set-Intersection

- Notation: $r \cap s$
- Defined as:
  - $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Note: $\boldsymbol{r \cap s = r - (r - s)}$

Natural-Join

- Notation: r $\bowtie$ s
- Let r and s be relations on schemas R and S respectively. Then, r $\bowtie$ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
    - t has the same value as t_r on r
    - t has the same value as t_s on s
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B\ =\ s.B\ \wedge\ r.D\ =\ s.D}\ (r\ \times\ s))$$

Theta Join

- The **theta join** operation $r \bowtie_{\theta} s$ is defined as
  - $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$

  where θ is a predicate on $R \times S$
- The **equi join** operation is a special case of theta join where θ is a predicate on equality (=)

MYONGJI UNIVERSITY

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.

Outer Join

- An extension of the join operation that avoids loss of information.
- 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.
  - Left Outer Join
  - Right Outer 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.
    - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *instructor*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches*

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

Outer Join – Example

- (Natural) Join

instructor ⋈ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

Instructor ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example

- Right Outer Join

instructor ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join

instructor ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Outer Join using Joins

- Outer join can be expressed using basic operations
  - e.g. r ⟕ s can be written as

$$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$$

MYONGJI UNIVERSITY

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
    (*false* **and** *unknown*) = *false,*
    (*unknown* **and** *unknown*) = *unknown*
  - NOT*:* (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown.*
- Result of select predicate is treated as *false* if it evaluates to *unknown.*

APPENDIX: DIVISION

Division Operator

- Given relations r(R) and s(S), such that $S \subset R$, $r \div s$ is the largest relation t(R-S) such that

  $$t \times s \subseteq r$$

  E.g. let $r(ID, course\_id) = \Pi_{ID,\ course\_id}(takes)$ and

  $s(course\_id) = \Pi_{course\_id}(\sigma_{dept\_name=\text{"Biology"}}(course))$

  then $r \div s$ gives us students who have taken all courses in the Biology department

- Can write $r \div s$ as

  $$temp1 \leftarrow \Pi_{R-S}(r)$$
  $$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$
  $$result \leftarrow temp1 - temp2$$

  - 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.

Division Operator (cont.)

- Can be rewritten as
  $r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$
- To see why
  - $\Pi_{R-S,S}(r)$ simply reorders attributes of r
  - $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in

    $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

Division Operator (cont.)

r* ÷ *s

- Suited to queries that include the phrase "for all".

- Let *r* and *s* be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of r ÷ s is a relation on schema
  $R - S = (A_1, \ldots, A_m)$

$$r \div s = \{\ t \mid t \in \prod_{R-S}(r) \wedge \forall_{u \in s} (\ tu \in r\ )\ \}$$

MYONGJI UNIVERSITY

Division Operation – Example

Relations *r, s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

$r \div s$:

A
α
β

MYONGJI UNIVERSITY

Another Division Example

Relations *r, s*:

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

r

D	E
a	1
b	1

s

$r \div s$:

A	B	C
α	a	γ
γ	a	γ

APPENDIX:
RELATIONAL ALGEBRA EXCERCISE

Example Queries

- Find all customers who have an account from at least the "Downtown" and the "Uptown" branches.

  – Query 1

  $$\Pi_{CN}(\sigma_{BN=\text{"Downtown"}}(depositor \bowtie account)) \cap$$
  $$\Pi_{CN}(\sigma_{BN=\text{"Uptown"}}(depositor \bowtie account))$$

  where *CN* denotes customer-name and *BN* denotes *branch-name*.

  – Query 2

  $$\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account)$$
  $$\div\ \rho_{temp(branch\text{-}name)}(\{(\text{"Downtown"}),\ (\text{"Uptown"})\})$$

Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{\textit{customer-name, branch-name}}(\textit{depositor} \bowtie \textit{account})$$
$$\div \Pi_{\textit{branch-name}}(\sigma_{\textit{branch-city} = \text{“Brooklyn”}}(\textit{branch}))$$

APPENDIX:
EXTENDED RELATIONAL-ALGEBRA-OPERATIONS

Generalized Projection

Aggregate Functions

Generalized Projection

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

$$\prod_{F_1, F_2, \ldots, F_n}(E)$$

- *E* is any relational-algebra expression

- Each of F_1, F_2, …, F_n are arithmetic expressions involving constants and attributes in the schema of *E*.

- Given relation *instructor(ID, name, dept_name,* salary) where salary is annual salary, get the same information but with monthly salary

$$\prod_{ID,\ name,\ dept\_name,\ salary/12} (instructor)$$

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

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

  E is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ 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

- Note: Some books/articles use γ instead of $\mathcal{G}$ (Calligraphic G)

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\textbf{sum(c)}}(\mathrm{r})$

sum(*c*)
27

Aggregate Operation – Example

- Find the average salary in each department

$$_{dept\_name}\mathcal{G}_{\textbf{avg}(salary)}(instructor)$$

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregate Functions (Cont.)

- 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

$$_{dept\_name}\mathcal{G}\ _{\textbf{avg}(salary)\ \textbf{as}\ avg\_sal}\ (instructor)$$

APPENDIX: MODIFICATION OF THE DATABASE

Deletion
Insertion
Updating

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.

- Can delete only whole tuples; cannot delete values on only particular attributes

- A deletion is expressed in relational algebra by:

  $$r \leftarrow r - E$$

  where r is a relation and E is a relational algebra query.

Deletion Examples

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{\, branch\text{-}name\ =\ \text{“Perryridge”}}\ (account)$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{\, amount \geq 0 \text{ and } amount \leq 50}\ (loan)$$

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{\, branch\text{-}city\ =\ \text{“Needham”}}\ (account \bowtie branch)$$
$$r_2 \leftarrow \Pi_{account\text{-}number,\ branch\text{-}name,\ balance}\ (r_1)$$
$$r_3 \leftarrow \Pi_{\, customer\text{-}name,\ account\text{-}number}\ (r_2 \bowtie \text{depositor})$$
$$account \leftarrow \text{account} - r_2$$
$$depositor \leftarrow \text{depositor} - r_3$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted

- In relational algebra, an insertion is expressed by:

  $$r \leftarrow r \cup E$$

  where *r* is a relation and *E* is a relational algebra expression.

- The insertion of a single tuple is expressed by letting *E* be a constant relation containing one tuple.

Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

  $account \leftarrow account \cup \{(\text{A-973}, \text{“Perryridge”}, 1200)\}$

  $\text{depositor} \leftarrow depositor \cup \{(\text{“Smith”}, \text{A-973})\}$

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

  $r_1 \leftarrow (\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}(borrower \bowtie \text{loan}))$

  $account \leftarrow account \cup \Pi_{account\text{-}number,\ branch\text{-}name,\ 200}\ (r_1)$

  $\text{depositor} \leftarrow depositor \cup \Pi_{customer\text{-}name,\ loan\text{-}number}(r_1)$

Updating

- A mechanism to change a value in a tuple without changing *all* values in the tuple

- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F1,\ F2,\ \ldots,\ Fn,} (r)$$

- Each F_i is either the i^{th} attribute of r, if the i^{th} attribute is not updated, or, if the attribute is to be updated, F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

Update Examples

- Make interest payments by increasing all balances by 5 percent.

  $$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.05}(account)$$

  where *AN*, *BN* and *BAL* stand for *account-number*, *branch-name* and *balance*, respectively.

- Pay all accounts with balances over \$10,000 6 percent interest and pay all others 5 percent

  $$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.06}(\sigma_{BAL\ >\ 10000}(account)) \cup \Pi_{AN,\ BN,\ BAL\ *\ 1.05}(\sigma_{BAL\ \leq\ 10000}(account))$$

MYONGJI UNIVERSITY