Principles of Database Systems

Formal Relational Query Languages

Formal Relational Query Languages

- **Relational Algebra** (关系代数)
- Tuple Relational Calculus
- Domain Relational Calculus

Relational Algebra

- The relational algebra is **procedural** query language
  - a set of operations, take one or two limited relations as input and produce a new limited relation as output

- Three types of operations/operators on relations
  - **fundamental operations**（基本运算）
  - **additive operations**（附加运算）
  - **extended operations**（扩展运算）

Fundamental Operations

- Six basic operators
  - select: σ
  - project: ∏
  - union: ∪
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Unary operations (select, project)

Binary operations (union, set difference, Cartesian product)

Unary operations (rename)

Select Operation

- **The select operation** σ on ***r*** (R) selects **tuples** that satisfy a given predicate, resulting in a subset of *r*
- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}$ (r)

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- 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** (命题演算) 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: $=, \neq, >, \geq. <. \leq$

Try…

- Find the instructors in Physics with a salary greater than $90,000

$$\sigma_{\,dept\_name="Physics" \wedge salary>90000}(instructor)$$

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

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000

Note: The term select corresponds to what we refer to in SQL as where

Project Operation

- **The project operation**(投影运算) on ***r*** **list some attributes** and their values in relation ***r***
  - the duplicate rows are **removed** from the result, since relations are sets

Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Project Operation

- Notation:

$$\Pi_{A1,\ A2,\ \ldots,\ Ak}\ (\mathbf{r}):$$

  - 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.

Try…

- To eliminate the dept_name attribute of instructor.

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

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

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Note: The term project corresponds to what we refer to in SQL as select

Composition of Relational Operations

- Several relational algebra operations can be composed together into a **relational algebra expression.**

- Explain the meaning of following statement:

  Π_{name} (σ $_{dept\_name=\text{“Physics”}}$ (instructor))

Find the name of all instructors in the Physics department

Union Operation

- **Union operation** on relation **r** and **s** is defined as

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

- Relations r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cup s$:

A	B
α	1
α	2
β	1
β	3

Union Operation

- For r ∪ s to be valid.
  - r, s must have the **same arity** (same number of attributes)
  - The attribute domains must be **compatible**(兼容的) (example: 2nd column of r deals with the same type of values as does the 2nd column of s)

- SQL statement
  - r **Union** s

Try…

- Find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both.

$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{“Fall”}\ \land\ year=2009}\,(section))\ \cup$$
$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{“Spring”}\ \land\ year=2010}\,(section))$$

Set-Difference Operation

- **Set difference operation** on relation **r** and **s** is defined as

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

- Relations r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r – s:

A	B
α	1
β	1

Set-Difference Operation

- 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

- SQL statement

  r **Except** s

Try…

- Find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester.

$$\Pi_{course\_id}\,(\sigma_{\ semester=\text{“Fall”}\ \wedge\ year=2009}\,(section))\ -$$
$$\Pi_{course\_id}\,(\sigma_{\ semester=\text{“Spring”}\ \wedge\ year=2010}\,(section))$$

Cartesian-Product Operation

- **Cartesian-product operation** on relation **r** and **s** is defined as:

$$r \times s = \{t \mid t = t_1\, t_2 \textbf{ and } t_1 \in \mathbf{r} \textbf{ and } t_2 \in \mathbf{s}\}$$

A	*B*
α	1
β	2

r

C	*D*	*E*
α	10	a
β	10	a
β	20	b
γ	10	b

s

A	*B*	*C*	*D*	*E*
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

r × s

Cartesian-Product Operation

- For relation **r(R)** and **s(S)**, **r** ×**s** is a relation whose schema is **R'** , which is the **concatenation**(串联) of R and S

  R **(**A_1, A_2, …, A_m), **S**(B_1, B_2, …, B_n)

  R' (A_1, A_2, …, A_m, B_1, B_2, …, B_n)
- 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.
- SQL Statement:

  Select *

  From R,S

Composition of Operations

- Can build expressions using multiple operations.
- Example: $\sigma_{A=C}(r \times s)$

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

$r \times s$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

$\sigma_{A=C}(r \times s)$

Composition of Operations

- A typical SQL query has the form

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

- Equivalent to the relational algebra expression

$$\Pi_{A1,\ A2,\ \ldots,\ An}(\sigma_{P}(r_1 \times r_2 \times \ldots \times r_m))$$

Try…

- Find the names of all instructors in the Physics department together with the course id of all courses they taught.

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

Try…

- Find the names of all instructors in the Physics department together with the course id of all courses they taught.

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

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

Rename Operation

- 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.

- Given a relational-algebra expression E, the expression

$$\rho_x (E)$$

returns the result of expression E under the name x.

Rename Operation

- 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 , …., A_n .

An Example

- 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

$$\Pi_{salary} (instructor) - \Pi_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_d (instructor)))$$

Positional Notation

- The rename operation is not strictly required, since it is possible to use a **positional notation**(位置标记) for attributes.
- Use \$1, \$2, . . . refer to the first attribute, the second attribute, and so on.
- E.g.

$$\Pi_{\$4}\ (\sigma_{\$4\ <\ \$8}\ (instructor \times instructor\ ))$$

$$instructor(\underline{ID}, name, dept\_name, salary)$$

Formal Definition

- A basic **expression in the relational algebra** consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Additional Operations

- We define additional operations that do not add any power to the relational algebra, but that **simplify common queries**.
  - **Set Intersection**
  - **Natural Join**
  - **Division**(除)
  - **Assignment**(赋值)
  - **Outer Join**

- Note: An additional operation can be replaced/re-represented by basic operations.

Set-Intersection Operation

- For relation **r** and **s**, **set-intersection operation** on **r** and **s** is defined as

$$r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$$

  - **r**, **s** have the same arity
  - attributes of **r** and **s** are compatible

- Note: $r \cap s = r - (r - s)$

- SQL statement
  - **intersect**

Natural-Join Operation

- 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

- Assuming $R \cap S = \{A_1, A_2, \ldots, A_n\}$

$$\mathbf{r} \bowtie \mathbf{s} = \prod_{R \cup S} (\sigma_{r.A1=s.A1 \wedge r.A2=s.A2 \wedge \ldots \wedge r.An=s.An} (r \times s))$$

Natural-Join Operation

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- r⋈s:

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

- first the attributes of both relations, second those unique to the first relation, finally, the second relation.

Natural-Join Operation

- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$
  - $instructor \bowtie (teaches \bowtie course)$
- Natural join is commutative
  - $instructor \bowtie teaches$
  - $teaches \bowtie instructor$

Natural Join and 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 attributes in $R \cup S$

An Example

- 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

    $$\Pi_{salary} (instructor) - \Pi_{instructor.salary} (instructor \bowtie_{instructor.salary < d.salary} \rho_d (instructor))$$

The Division Operation

- Let r (R) and s(S) be relations, and let $S \subseteq R$; that is, every attribute of schema S is also in schema R. Then **r ÷ s** is a relation on **schema R − S** (that is, on the schema containing all attributes of schema R that are not in schema S). A tuple t is in r ÷ s if and only if **both** of two conditions hold:
  - t is in $\Pi_{R-S}(r)$
  - For every tuple t_s in s, there is a tuple t_r in r satisfying both of the following:
    - $t_r [S] = t_s [S]$
    - $t_r [R - S] = t$

The Division Operation

- r(R)和s(S)是两个关系，并且S $\subseteq$ R；也就是说S中的每个属性都在R中，关系r $\div$ s 是模式R – S 上的关系。也就是说，这个模式中包含所有在R中而不在S中的属性。如果以下两个条件同时成立，那么元组t就属于 r $\div$ s
  - t 在$\Pi_{R-S}(r)$中
  - 对s里的每个元组t_s，在r中都有元组t_r同时满足
    - $t_r [S] = t_s [S]$
    - $t_r [R - S] = t$

The Division Operation

- 被除数中独有属性对应的集合是否包含除数
- E.g.

A	B	C	D
a	b	c	d
a	b	e	f
b	c	e	f
e	d	c	d
e	d	e	f
a	b	d	e

R

C	D
c	d
e	f

S

A	B
a	b
e	d

R ÷ S

The Division Operation

- $R \div S = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$
  - $\Pi_{R-S,S}(r)$ 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 $t_s \in s$, $t\,t_s \notin r$.**

- How to use division:
  - fit for queries that include the phrase "**for all**"

Try…

- Find all students who have taken all courses offered in the Biology department.

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

Try…

- Step1. obtain $\mathbf{r_1}$={all courses offered in Biology department}

$$\Pi_{course\_id}\,(\sigma_{department\ =\ \text{“Biology”}}\,(course))$$

- Step2. find all pairs $\mathbf{r_2}$ = (ID,name, course_id), by

$$\Pi_{ID,\ name,\ course\_id}\,(student \bowtie takes)$$

- Step3. find students who appear in $\mathbf{r_2}$ with **every course_id** in $\mathbf{r_1}$, by using division operations.

$$\Pi_{ID,\ name, course\_id}\,(student \bowtie takes) \div \Pi_{course\_id}\,(\sigma_{department\ =\ \text{“Biology”}}\,(course))$$

Assignment Operation

- **Assign**(赋值) (←) relation algebra expression E to a relational variable var_r, i.e. denoting E as var_r.

- The assignment operation (←) provides a convenient way to **express complex queries**.

- E.g. Write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}\ (r)$$

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

$$result = temp1 - temp2$$

Assignment Operation

- For relational-algebra queries, assignment must always be made to a temporary relation variable.
- Assignments to permanent relations constitute a database modification

$$emp \leftarrow emp \cup (12345, \text{'a'}, null, null, null)$$

$$emp \leftarrow emp - \sigma_{sal<1500}\ (emp)$$

$$emp \leftarrow \Pi_{\ ,\ ,\ ,\ sal \times 1.1}(emp)$$

Outer Join

- **Outer join** is 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 **with NULL**.

- Three types:
  - Left Outer Join $⟕$
  - Right Outer Join $⟖$
  - Full Outer Join $⟗$

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

- Join

  instructor $\bowtie$ 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

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

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

  - r ⟖ s can be written as

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

Extended Relational-Algebra-Operations

- We define extended operations that **add power** to the relational algebra
  - **Generalized Projection**(广义投影)
  - **Aggregation**(聚集)

- Note: An extended operation cannot be expressed using the basic relational-algebra operations.

Generalized Projection

- **Generalized project**(广义投影) extends the **projection** operation **by allowing arithmetic functions to be used in the projection list**.

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

where

- 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.

Generalized Projection

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

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

Aggregation

- **Aggregation functions**(聚集函数)
  - mapping of a collection of attribute values in a relation r into **a single value**, i.e. attribute domains → domain

    avg: average value

    min: minimum value

    max: maximum value

    sum: sum of values

    count: number of values

- In SQL Select statement, aggregation functions are embedded in select and having subclause.

Aggregation

- **Aggregate operation**(聚集运算) 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 …, G_n is a list of attributes on which to group (can be empty)
  - if G_1, G_2,…,G_n do not appears, aggregations are conducted on all tuples in E, that is, E are not classified into groups
- Each F_i is an aggregate function
- Each A_i is an attribute name

Try…

- Find the average salary in each department

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

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

Aggregation

- 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

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

Multiset Relational Algebra

- Pure relational algebra removes all duplicates
  - e.g. after projection
- **Multiset**(多重集) relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention(保留) was initially for efficiency, but is now a feature

Multiset Relational Algebra

- Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are m copies of $t1$ in r, and n copies of $t2$ in s, there are $m \times n$ copies of $t1.t2$ in $r \times s$
  - Other operators similarly defined
    - E.g. union: $m + n$ copies, intersection: $\min(m, n)$ copies, difference: $\max(0, m - n)$ copies

Multiset Relational Algebra – Example

- Find out the sum of salaries of all instructors.

$$\mathcal{G}_{\textbf{sum}(\text{salary})}(\text{instructor})$$

- Find the total number of instructors who teach a course in the Spring 2010 semester.

$$\mathcal{G}_{\textbf{count\_distinct}(\text{ID})}(\sigma_{\text{semester}=\text{“Spring”} \wedge \text{year}=2010}(\text{teaches}))$$

SQL and Relational Algebra

- **select** $A_1, A_2, .. A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  is equivalent to the following expression in multiset relational algebra:

$$\Pi_{A1,\ ..,\ An} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

SQL and Relational Algebra

- **select** A_1, A_2, **sum**(A_3)
  from $r_1, r_2, \ldots, r_m$
  where P
  group by A_1, A_2
  is equivalent to the following expression in multiset relational algebra

$$_{A1,\ A2}\,\mathcal{G}_{\ \mathbf{sum}(A3)}\ (\sigma_{\ P}\ (r_1 \times r_2 \times \ldots \times r_m)))$$

SQL and Relational Algebra

- More generally, the non-aggregated attributes in the **select** clause may be a subset of the **group by** attributes, in which case the equivalence is as follows:

  select A_1, **sum**(A_3)
  from $r_1, r_2, \ldots, r_m$
  where P
  group by A_1, A_2

  is equivalent to the following expression in multiset relational algebra

$$\Pi_{A1,sumA3}(\ {}_{A1,A2}\mathcal{G}_{\ \mathbf{sum}(A3)\ \mathbf{as}\ sumA3}(\sigma_P\ (r_1 \times r_2 \times \ldots \times r_m)))$$

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Thanks