



# Chapter 6: Formal Relational Query Languages

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan  
See [www.db-book.com](http://www.db-book.com) for conditions on re-use



# Chapter 6: Formal Relational Query Languages

- Relational Algebra
- Tuple Relational Calculus



# Relational Algebra

- Procedural language
- Six basic operators
  - select:  $\sigma$  (Sigma)
  - project:  $\pi$  (Pi)
  - union:  $\cup$
  - set difference:  $-$
  - Cartesian product:  $\times$
  - rename:  $\rho$  (Rho)



# Relational Algebra

- The operators take one or two relations as inputs and produce a new relation as a result
- **Fundamental and Unary operators**
  - select:  $\sigma$  (Sigma)
  - project:  $\pi$  (Pi)
  - rename:  $\rho$  (Rho)
- **Fundamental and Binary operators**
  - union:  $\cup$
  - set difference:  $-$
  - Cartesian product:  $\times$



# Select Operation – Example

- Relation r

A	B	C	D
$\alpha$	$\alpha$	1	7
$\alpha$	$\beta$	5	7
$\beta$	$\beta$	12	3
$\beta$	$\beta$	23	10

Select the rows where

A and B are same

And

Also  $D > 5$



# Select Operation – Example

- Relation r

A	B	C	D
$\alpha$	$\alpha$	1	7
$\alpha$	$\beta$	5	7
$\beta$	$\beta$	12	3
$\beta$	$\beta$	23	10

■  $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
$\alpha$	$\alpha$	1	7
$\beta$	$\beta$	23	10



# Select Operation

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

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

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

Each **term** is one of:

$\langle \text{attribute} \rangle op \quad \langle \text{attribute} \rangle$  or  $\langle \text{constant} \rangle$

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

- Example of selection:

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



# Project Operation – Example

- Relation  $r$ :

$A$	$B$	$C$
$\alpha$	10	1
$\alpha$	20	1
$\beta$	30	1
$\beta$	40	2

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





# Project Operation – Example

- Relation  $r$ :

$A$	$B$	$C$
$\alpha$	10	1
$\alpha$	20	1
$\beta$	30	1
$\beta$	40	2

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

$A$	$C$
$\alpha$	1
$\alpha$	1
$\beta$	1
$\beta$	2

 $=$ 

$A$	$C$
$\alpha$	1
$\beta$	1
$\beta$	2



# Project Operation

- $\Pi$  Unary Operator
- Notation:

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

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 are removed from result, since relations are sets**
- Example: To eliminate the *dept\_name* attribute of *instructor*

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



# Union Operation – Example

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

- $r \cup s$ :



# Union Operation – Example

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

- $r \cup s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1
$\beta$	3



# Union Operation

- $\cup$  Binary Operator
- 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: 2<sup>nd</sup> column of  $r$  deals with the same type of values as does the 2<sup>nd</sup> column of  $s$ )
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2009}(section)) \cup$

$\Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2010}(section))$



# Set difference of two relations

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

- $r - s$ :



# Set difference of two relations

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

- $r - s$ :

$A$	$B$
$\alpha$	1
$\beta$	1



# Set Difference Operation

- ‘-’ Binary Operator
- 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

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





# Cartesian-Product Operation – Example

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\beta$	2

$r$

$C$	$D$	$E$
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

$s$

- $r \times s$ :



# Cartesian-Product Operation – Example

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\beta$	2

$r$

$C$	$D$	$E$
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

$s$

- $r \times s$ :

$A$	$B$	$C$	$D$	$E$
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b



# Cartesian-Product Operation

- 'X' Binary Operator
- 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 (i.e.,  $R \cap S = \emptyset$ )**
- If attributes of  $r(R)$  and  $s(S)$  are not disjoint, then renaming must be used



# Composition of Operations

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

- $r \times s$

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

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

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b



# Rename Operation

- ‘ $\rho$ ’ Unary Operator
- 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

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

returns the result of expression  $E$  under the name  $X$ , and with the

attributes renamed to  $A_1, A_2, \dots, A_n$



# Example Query

- 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*

$$4 \quad \Pi_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_d(instructor)))$$

- Step 2: Find the largest salary

$$4 \quad \Pi_{salary}(instructor) - \Pi_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_d(instructor)))$$



# 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

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



# 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  $E_1$
  - $\Pi_s(E_1)$ ,  $S$  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

- Additional operations that do not add any power to the relational algebra, but that simplify common queries
- Whenever it is used, utilizes basic operations
  - Set intersection
  - Assignment
  - Join



# Set-Intersection Operation

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



# Set-Intersection Operation – Example

- Relation  $r, s$ :

$A$	$B$
$\alpha$	1
$\alpha$	2
$\beta$	1

$r$

$A$	$B$
$\alpha$	2
$\beta$	3

$s$

- $r \cap s$

$A$	$B$
$\alpha$	2



# Assignment Operation

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



# Join Operation

- Cartesian Product of two relation
  - Gives all the possible tuples that are paired together
  - **Also called cross join**
  - But it might **NOT** be feasible for
    - 4 Huge relations where number of tuples are in thousands and
    - 4 Attributes of both relations are considerable large
- **Join Operation**
  - **Combination of Cartesian product followed by selection process**
  - Join operation pairs two tuples from different relations if and only if the given join condition is satisfied
  - Types
    - 4 Inner Join (Theta Join, Equi Join, Natural Join)
    - 4 Outer Join (Left outer Join, Right outer Join, Full outer Join)



# Cartesian-Product Operation – Example

- Relations  $r, s$ :

$A$	$B$
$\alpha$	1
$\beta$	2

$r$

$C$	$D$	$E$
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

$s$

- $r \times s$ :

$A$	$B$	$C$	$D$	$E$
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b



# Inner Join

- An inner-join **process includes only tuples with matching attributes, rest are discarded in resulting relation**
  - Theta Join
  - Equi Join
  - Natural Join



# Theta ( $\theta$ ) Join

- $\theta$  in Theta join is the join condition
- Combines tuples from different relations provided they satisfy the theta condition
- The **theta join** operation  $r \bowtie_{\theta} s$  is defined as
  - $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$
- $R1 \bowtie_{\theta} R2$ 
  - R1 and R2 are relations with their attributes (A1, A2, .., An ) and (B1, B2,..,Bn) such that **NO Attribute matches** that is  $R1 \cap R2 = \Phi$
- Theta join can use all kinds of comparison operators





# Theta ( $\theta$ ) Join

**Student**

SID	Name	Std
101	Alex	10
102	Maria	11

**Subjects**

Class	Subject
10	Math
10	English
11	Music
11	Sports

- $\text{STUDENT} \bowtie_{\text{Student.Std} = \text{Subject.Class}} \text{SUBJECT}$

SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports



# Equi Join

- When Theta join uses only **equality** comparison operator it is said to be Equi-Join
- The early example corresponds to equi join



# Natural Join

- Does not use any comparison operator
- Does not concatenate the way Cartesian product does
- Instead, Natural Join can only be performed if there is **at least one common attribute exists between relation**
  - Those attributes must have same name and domain
- Acts on those matching attributes where the values of attributes in both relation is same



# Natural Join Operation

- 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

4  $t$  has the same value as  $t_r$  on  $r$

4  $t$  has the same value as  $t_s$  on  $s$

- Example:

$R = (A, \mathbf{B}, C, \mathbf{D})$

$S = (\mathbf{E}, \mathbf{B}, \mathbf{D})$

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

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



# Natural Join Example

- Relations  $r$ ,  $s$ :

$A$	$B$	$C$	$D$
$\alpha$	1	$\alpha$	a
$\beta$	2	$\gamma$	a
$\gamma$	4	$\beta$	b
$\alpha$	1	$\gamma$	a
$\delta$	2	$\beta$	b

$r$

$B$	$D$	$E$
1	a	$\alpha$
3	a	$\beta$
1	a	$\gamma$
2	b	$\delta$
3	b	$\epsilon$

$s$

- $r \bowtie s$

$A$	$B$	$C$	$D$	$E$
$\alpha$	1	$\alpha$	a	$\alpha$
$\alpha$	1	$\alpha$	a	$\gamma$
$\alpha$	1	$\gamma$	a	$\alpha$
$\alpha$	1	$\gamma$	a	$\gamma$
$\delta$	2	$\beta$	b	$\delta$



# Natural Join – Example

- Relation *instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches*

<i>ID</i>	<i>course_id</i>
10101	CS-101
12121	FIN-201
76766	BIO-101

- Join

*instructor* ⋈ *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201



# Natural Join

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  - $\Pi_{name, title} (\sigma_{dept\_name = \text{“Comp. Sci.”} } (instructor \bowtie teaches \bowtie course))$
- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$  is equivalent to  $instructor \bowtie (teaches \bowtie course)$



# Exercise

**Courses**

<b>CID</b>	<b>Course</b>	<b>Dept</b>
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

**HoD**

<b>Dept</b>	<b>Head</b>
CS	Alex
ME	Maya
EE	Mira

Display all courses with their department Head names.

**Courses ⋈ HoD**

<b>Dept</b>	<b>CID</b>	<b>Course</b>	<b>Head</b>
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira





# If...

**Courses**

CID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE
<b>MU01</b>	<b>MusicNotes</b>	<b>MU</b>

**HoD**

Dept	Head
CS	Alex
ME	Maya
EE	Mira
<b>FIN</b>	<b>XYZ</b>

Display all courses with their department Head names.

**Courses ⋈ HoD**

Dept	CID	Course	Head
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

?



# Outer Join

- An extension of the join operation that avoids loss of information
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the resultant relation 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
- Outer join can be expressed using basic operations
  - e.g.  $r \bowtie s$  can be written as

$$(r \bowtie s) \cup (r - \prod_R(r \bowtie s) \times \{(null, ..., null)\})$$



# Outer Join – Example

- Relation *instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches*

<i>ID</i>	<i>course_id</i>
10101	CS-101
12121	FIN-201
76766	BIO-101



# Outer Join – Example

- Relation *instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches*

<i>ID</i>	<i>course_id</i>
10101	CS-101
12121	FIN-201
76766	BIO-101

- Natural Join

*instructor* ⋈ *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201



# Outer Join – Example

- Relation *instructor1*

<i>ID</i>	<i>name</i>	<i>dept_name</i>
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches1*

<i>ID</i>	<i>course_id</i>
10101	CS-101
12121	FIN-201
76766	BIO-101

- Join

*instructor* ⋈ *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

*instructor* ⋈<sub>L</sub> *teaches*

select \* from instructor left outer join teaches on  
instructor.ID = teaches.ID;

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
<b>15151</b>	<b>Mozart</b>	<b>Music</b>	<b>null</b>



# Outer Join – Example

- Right Outer Join

*instructor* ⋈<sub>⊃</sub> *teaches*

select \* from instructor rightt outer join teaches on  
instructor.ID = teaches.ID;

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
<b>76766</b>	<b>null</b>	<b>null</b>	<b>BIO-101</b>

- Full Outer Join

*instructor* ⋈<sub>⊃</sub> *teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>course_id</i>
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
<b>15151</b>	<b>Mozart</b>	<b>Music</b>	<u><b>null</b></u>
<b>76766</b>	<b>null</b>	<u><b>null</b></u>	<b>BIO-101</b>



# Exercise

**Courses**

A	B
100	Database
101	Mechanics
102	Electronics

**HoD**

A	C
100	Alex
102	Maya
104	Mira

**Courses**  $\bowtie$  **HoD**

A	B	C
100	Database	Alex
101	Mechanics	null
102	Electronics	Maya

**Courses**  $\ltimes$  **HoD**

A	B	C
100	Database	Alex
102	Electronics	Maya
104	null	Mira

**Courses**  $\bowtie$  **HoD**

Courses.A	Courses.B	HoD.A	HoD.C
100	Database	100	Alex
101	Mechanics	null	null
102	Electronics	102	Maya
null	null	104	Mira



# Exercise

**Employee**

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

**Dept**

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

**Employee** ⋈ **Dept**

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

**Employee** ⋈ **Dept**

**Employee** ⋈ **Dept**

**Employee** ⋈ **Dept**





# 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.**

$$\Pi_{F_1, F_2, \dots, F_n}(E)$$

- $E$  is any relational-algebra expression
- Each of  $F_1, F_2, \dots, F_n$  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**

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



# 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

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

$E$  is any relational-algebra expression

- $G_1, G_2, \dots, G_n$  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  $\gamma$  instead of  $\mathcal{G}$  (graphic  $G$ )



# Aggregate Operation – Example

- Relation  $r$ :

$A$	$B$	$C$
$\alpha$	$\alpha$	7
$\alpha$	$\beta$	7
$\beta$	$\beta$	3
$\beta$	$\beta$	10

- $G_{\text{sum}(c)}(r)$

<b>sum(<math>c</math>)</b>
27



# Aggregate Operation – Example

- Find the average salary in each department

$dept\_name \quad G_{g(salary)}(instructor)$

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

<i>dept_name</i>	<i>salary</i>
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*      $\mathcal{G}_{avg(salary)} \text{ as } avg\_sal (instructor)$



# SQL and Relational Algebra

- **select**  $A1, A2, \dots, An$   
**from**  $r1, r2, \dots, rm$   
**where**  $P$

is equivalent to the following relational algebra expression

$$\Pi_{A1, \dots, An} (\sigma_P (r1 \times r2 \times \dots \times rm))$$

- **select**  $A1, A2, \text{sum}(A3)$   
**from**  $r1, r2, \dots, rm$   
**where**  $P$   
**group by**  $A1, A2$

is equivalent to the following relational algebra expression

$$A1, A2 \quad G_{\text{sum}(A3)} (\sigma_P (r1 \times r2 \times \dots \times rm))$$



# SQL and Relational Algebra

- Only selected column to display
- 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  $A1$ , sum( $A3$ )
from  $r1$ ,  $r2$ , ...,  $rm$ 
where  $P$ 
group by  $A1$ ,  $A2$ 
  
```

is equivalent to the following relational algebra expression

$$\Pi_{A1, \text{sum} A3}(\sigma_P(r1 \times r2 \times \dots \times rm))$$





# Exercise

- For these queries design relational model, write relational algebra and sql queries.
- 1. Retrieve the name and address of all employees who work for the ‘Physics’ department.
- 2. For the project number 10, retrieve the controlling department number and department manager’s last name and address.
- 3. Find the names of employees who work on all the projects controlled by the department “Physics”.
- 4. Retrieve the list of project numbers for projects that involve an employee whose last name is “Patel”, either as a worker or as a manager of the department that controls the project.



# 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  $\text{not } (A < 5)$  would not be equivalent to  $A \geq 5$
- Three-valued logic using the truth value *unknown*:
  - OR:  $(\text{unknown} \textbf{ or } \text{true}) = \text{true}$ ,  
 $(\text{unknown} \textbf{ or } \text{false}) = \text{unknown}$   
 $(\text{unknown} \textbf{ or } \text{unknown}) = \text{unknown}$
  - AND:  $(\text{true} \textbf{ and } \text{unknown}) = \text{unknown}$ ,  
 $(\text{false} \textbf{ and } \text{unknown}) = \text{false}$ ,  
 $(\text{unknown} \textbf{ and } \text{unknown}) = \text{unknown}$
  - NOT:  $(\textbf{not unknown}) = \text{unknown}$
  - 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*



# Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations can be expressed using the assignment operator



# 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.

$$account \leftarrow account - \sigma_{branch\_name = "Perryridge"}(account)$$

- 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.

$$r_1 \leftarrow \sigma_{branch\_city = "Needham"}(account \bowtie branch)$$

$$r_2 \leftarrow \Pi_{account\_number, branch\_name, balance}(r_1)$$

$$r_3 \leftarrow \Pi_{customer\_name, account\_number}(r_2 \bowtie depositor)$$

$$account \leftarrow account - r_2$$

$$depositor \leftarrow depositor - r_3$$



# 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.

$$account \leftarrow account \cup \{("A-973", "Perryridge", 1200)\}$$
$$depositor \leftarrow depositor \cup \{("Smith", "A-973")\}$$





# 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

$$r \leftarrow \Pi_{F_1, F_2, \dots, F_I}(r)$$

- Each  $F_i$  is either
  - the  $I^{\text{th}}$  attribute of  $r$ , if the  $I^{\text{th}}$  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.

$account \leftarrow \Pi_{account\_number, branch\_name, balance * 1.05} (account)$

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

$account \leftarrow \Pi_{account\_number, branch\_name, balance * 1.06} (\sigma_{BAL > 10000} (account)) \cup \Pi_{account\_number, branch\_name, balance * 1.05} (\sigma_{BAL \leq 10000} (account))$



# 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  
 $s(course\_id) = \Pi_{course\_id}(\sigma_{dept\_name="Biology"}(course))$   
then  $r \div s$  gives us students who have taken all courses in the Biology department

- Can write  $r \div s$  as

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

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

$$result = temp1 - temp2$$

- 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



# Tuple Relational Calculus



# Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form  $\{t \mid P(t)\}$
- It is the set of all tuples  $t$  such that predicate  $P$  is true for  $t$
- $t$  is a *tuple variable*,  $t[A]$  denotes the value of tuple  $t$  on attribute  $A$
- $t \in r$  denotes that tuple  $t$  is in relation  $r$
- $P$  is a *formula* similar to that of the predicate calculus



# Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g.,  $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ,  $>$ ,  $\geq$ )
3. Set of connectives: and ( $\wedge$ ), or ( $\vee$ ), not ( $\neg$ )
4. Implication ( $\Rightarrow$ ):  $x \Rightarrow y$ , if  $x$  is true, then  $y$  is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:

- ▶  $\exists t \in r (Q(t)) \equiv$  "there exists" a tuple  $t$  in relation  $r$  such that predicate  $Q(t)$  is true
- ▶  $\forall t \in r (Q(t)) \equiv Q$  is true "for all" tuples  $t$  in relation  $r$



# Example Queries

- Find the *ID*, *name*, *dept\_name*, *salary* for instructors whose salary is greater than \$80,000

$$\{t \mid t \in instructor \wedge t[salary] > 80000\}$$

- As in the previous query, but output only the *ID* attribute value

$$\{t \mid \exists s \in instructor (t[ID] = s[ID] \wedge s[salary] > 80000)\}$$

Notice that a relation on schema (*ID*) is implicitly defined by the query



# Example Queries

- Find the names of all instructors whose department is in the Watson building

$$\{t \mid \exists s \in instructor (t[name] = s[name] \wedge \exists u \in department (u[dept\_name] = s[dept\_name] \wedge u[building] = \text{"Watson"}))\}$$

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

$$\{t \mid \exists s \in section (t[course\_id] = s[course\_id] \wedge s[semester] = \text{"Fall"} \wedge s[year] = 2009 \vee \exists u \in section (t[course\_id] = u[course\_id] \wedge u[semester] = \text{"Spring"} \wedge u[year] = 2010))\}$$





# Example Queries

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{t \mid \exists s \in \text{section } (t[\text{course\_id}] = s[\text{course\_id}] \wedge s[\text{semester}] = \text{"Fall"} \wedge s[\text{year}] = 2009 \wedge \exists u \in \text{section } (t[\text{course\_id}] = u[\text{course\_id}] \wedge u[\text{semester}] = \text{"Spring"} \wedge u[\text{year}] = 2010))\}$$

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

$$\{t \mid \exists s \in \text{section } (t[\text{course\_id}] = s[\text{course\_id}] \wedge s[\text{semester}] = \text{"Fall"} \wedge s[\text{year}] = 2009 \wedge \neg \exists u \in \text{section } (t[\text{course\_id}] = u[\text{course\_id}] \wedge u[\text{semester}] = \text{"Spring"} \wedge u[\text{year}] = 2010))\}$$



# Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example,  $\{ t \mid \neg t \in r \}$  results in an infinite relation if the domain of any attribute of relation  $r$  is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression  $\{ t \mid P(t) \}$  in the tuple relational calculus is *safe* if every component of  $t$  appears in one of the relations, tuples, or constants that appear in  $P$ 
  - NOTE: this is more than just a syntax condition.
    - 4 E.g.  $\{ t \mid t[A] = 5 \vee \mathbf{true} \}$  is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in  $P$ .



# Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{t \mid \exists r \in \text{student} (t[ID] = r[ID]) \wedge$   
 $(\forall u \in \text{course} (u[\text{dept\_name}] = \text{“Biology”} \Rightarrow$   
 $\exists s \in \text{takes} (t[ID] = s[ID] \wedge$   
 $s[\text{course\_id}] = u[\text{course\_id}]))\}$
  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.



**End of Chapter 6**

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan  
See [www.db-book.com](http://www.db-book.com) for conditions on re-use



- <http://users.cms.caltech.edu/~donnie/cs121/CS121Lec02.pdf>
- <http://www.inf.unibz.it/~nutt/Teaching/IDBs0910/IDBExercises/4-sol-relAlg.pdf>
- [http://cir.dcs.uni-pannon.hu/cikkek/DB\\_relational\\_algebra\\_v2.pdf](http://cir.dcs.uni-pannon.hu/cikkek/DB_relational_algebra_v2.pdf)
- [https://www.inf.usi.ch/faculty/soule/teaching/2014-spring/04\\_Relational\\_Algebra\\_With\\_SQL\\_Equivalents.pdf](https://www.inf.usi.ch/faculty/soule/teaching/2014-spring/04_Relational_Algebra_With_SQL_Equivalents.pdf)
- <http://users.cms.caltech.edu/~donnie/cs121/CS121Lec03.pdf>
- <https://www.cs.cornell.edu/projects/btr/bioinformaticsschool/slides/gehrke.pdf>
- [http://www.cs.toronto.edu/~faye/343/f07/lectures/wk3/03\\_RAlgebra.pdf](http://www.cs.toronto.edu/~faye/343/f07/lectures/wk3/03_RAlgebra.pdf)