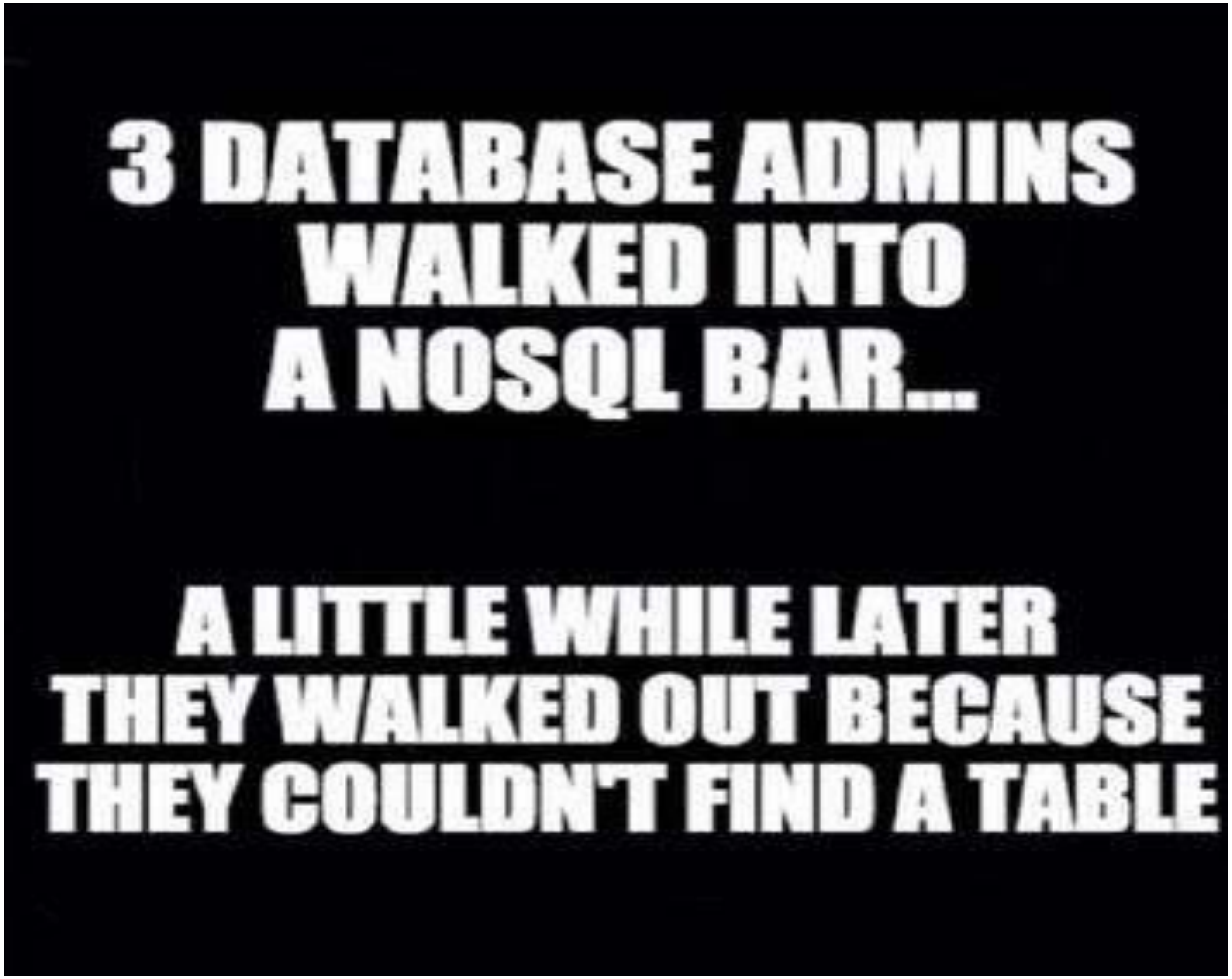
Formal Relational Query Languages

Instructor : Nitesh Kumar Jha

niteshjha@soa.ac.in

ITER,S'O'A(DEEMED TO BE UNIVERSITY)


3 DATABASE ADMINS WALKED INTO A NOSQL BAR...

A LITTLE WHILE LATER THEY WALKED OUT BECAUSE THEY COULDN'T FIND A TABLE

Formal Relational query language

- Relational Algebra
- Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus

Relational Algebra

- Procedural formal query language
- It forms a basis of mathematical foundation on which SQL is developed
- The operators take one or two relations as inputs and produce a new relation as a result.

Relational Algebra

- Six basic operators
  - select: σ --------Unary Operator
  - project: $\prod$ ---------Unary Operator
  - union: $\cup$
  - set difference: –
  - Cartesian product: x

  } Binary Operator

  - rename: ρ ----------Unary operator
- Other operators include
  - Natural Join : ⋈
  - Left Outer Join : ⟕
  - Right Outer Join : ⟖
  - Full Outer Join : ⟗

  Which I have discussed in my Lab lecture on Subquery and joins.

Two Databases

Consider the following tables for Bank Database:

- Customer(C_id, name, ph_no, address)
- Depositor(C_id, A_no)
- Account(A_no, Balance, B_name)
- Borrower(C_id, l_no)
- Loan(l_no, l_amt, B_name)
- Branch(B_name, B_city, B_assets)

Consider the following table for University Database:

- Instructor(ID, Name, Dept_name, Salary)
- Course(Course_id, Title, Dept_name, credits)
- Prereq(Course_id, Prereq_id)
- Department(Dept_name, Building, Budget)
- Teaches(ID, Course_id, sec_id, sem, year)
- Student(id, Name, dept_name, tot_credit)
- Takes(id, course_id, sec_id, semester, year, grade)
- Section(course_id, sec_id, semester, year, building, room_no, timeslot_id)

Select Operation

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

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

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: $=, \neq, >, \geq. <. \leq$

- Example of selection:

$$\sigma_{dept\_name=\text{“Physics”}}(instructor)$$

Project Operation

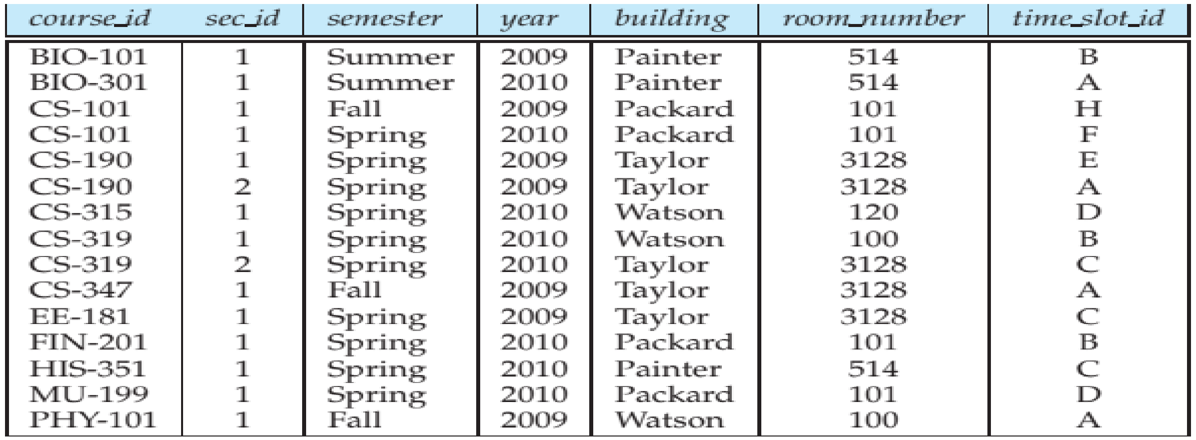
- Notation: $$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  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
- Example: To eliminate the *dept_name* attribute of *instructor*

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

Union Operation

- Notation: $r \cup s$



course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 6.4 The *section* relation.

- 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

$$\Pi_{course\_id}(\sigma_{semester=“Fall” \ \wedge \ year=2009}(Section)) \ \cup$$
$$\Pi_{course\_id}(\sigma_{semester=“Spring” \ \wedge \ year=2010}(Section))$$

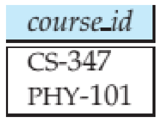
Set Difference Operation

- 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

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



course_id
CS-347
PHY-101

Figure 6.6 Courses offered in the Fall 2009 semester but not in Spring 2010 semester.

Set-Intersection Operation

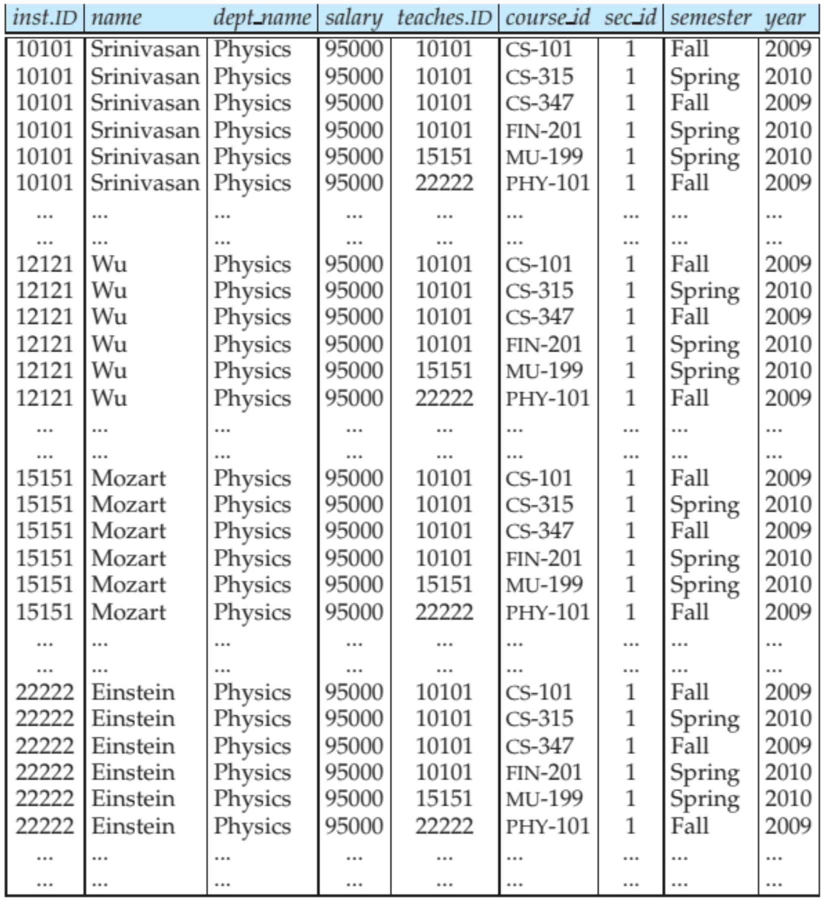
- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Note: $r \cap s = r - (r - s)$

Cartesian-Product Operation

- Notation *r* x *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, $R \cap S = \varnothing$).
- If attributes of *r(R)* and *s(S)* are not disjoint
- then renaming must be used.



inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Physics	95000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Physics	95000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Physics	95000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Physics	95000	10101	FIN-201	1	Spring	2010
10101	Srinivasan	Physics	95000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Physics	95000	10101	CS-101	1	Fall	2009
12121	Wu	Physics	95000	10101	CS-315	1	Spring	2010
12121	Wu	Physics	95000	10101	CS-347	1	Fall	2009
12121	Wu	Physics	95000	10101	FIN-201	1	Spring	2010
12121	Wu	Physics	95000	15151	MU-199	1	Spring	2010
12121	Wu	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Physics	95000	10101	CS-101	1	Fall	2009
15151	Mozart	Physics	95000	10101	CS-315	1	Spring	2010
15151	Mozart	Physics	95000	10101	CS-347	1	Fall	2009
15151	Mozart	Physics	95000	10101	FIN-201	1	Spring	2010
15151	Mozart	Physics	95000	15151	MU-199	1	Spring	2010
15151	Mozart	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	10101	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Figure 6.8 Result of *instructor* × *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.
- 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 $A_1, A_2, \ldots., A_n$.

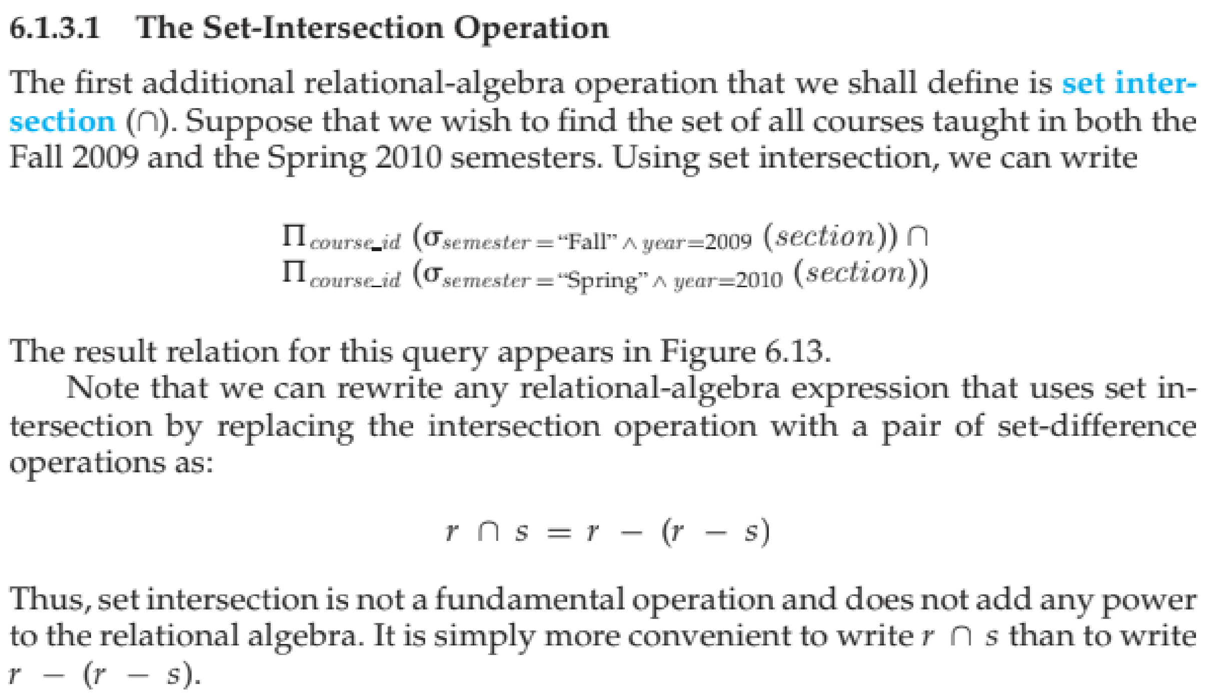
The query to find the largest salary in the university can be written as:

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

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 Relational algebra operations



6.1.3.1 The Set-Intersection Operation

The first additional relational-algebra operation that we shall define is **set intersection** ($\cap$). Suppose that we wish to find the set of all courses taught in both the Fall 2009 and the Spring 2010 semesters. Using set intersection, we can write

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

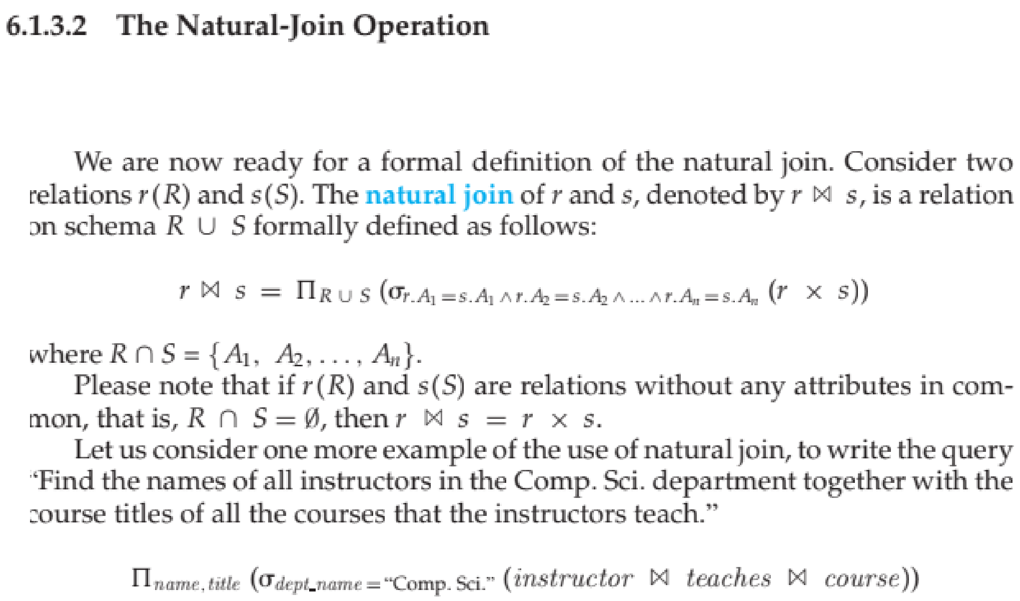
The result relation for this query appears in Figure 6.13.

Note that we can rewrite any relational-algebra expression that uses set intersection by replacing the intersection operation with a pair of set-difference operations as:

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

Thus, set intersection is not a fundamental operation and does not add any power to the relational algebra. It is simply more convenient to write $r \cap s$ than to write $r - (r - s)$.

Additional Relational algebra operations



6.1.3.2 The Natural-Join Operation

We are now ready for a formal definition of the natural join. Consider two relations $r(R)$ and $s(S)$. The **natural join** of r and s, denoted by $r \bowtie s$, is a relation on schema $R \cup S$ formally defined as follows:

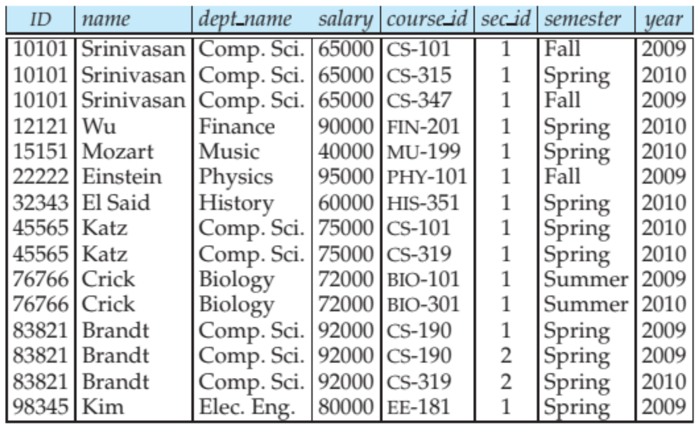
$$r \bowtie s = \Pi_{R \cup S} (\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (r \times s))$$

where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

Please note that if $r(R)$ and $s(S)$ are relations without any attributes in common, that is, $R \cap S = \emptyset$, then $r \bowtie s = r \times s$.

Let us consider one more example of the use of natural join, to write the query "Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach."

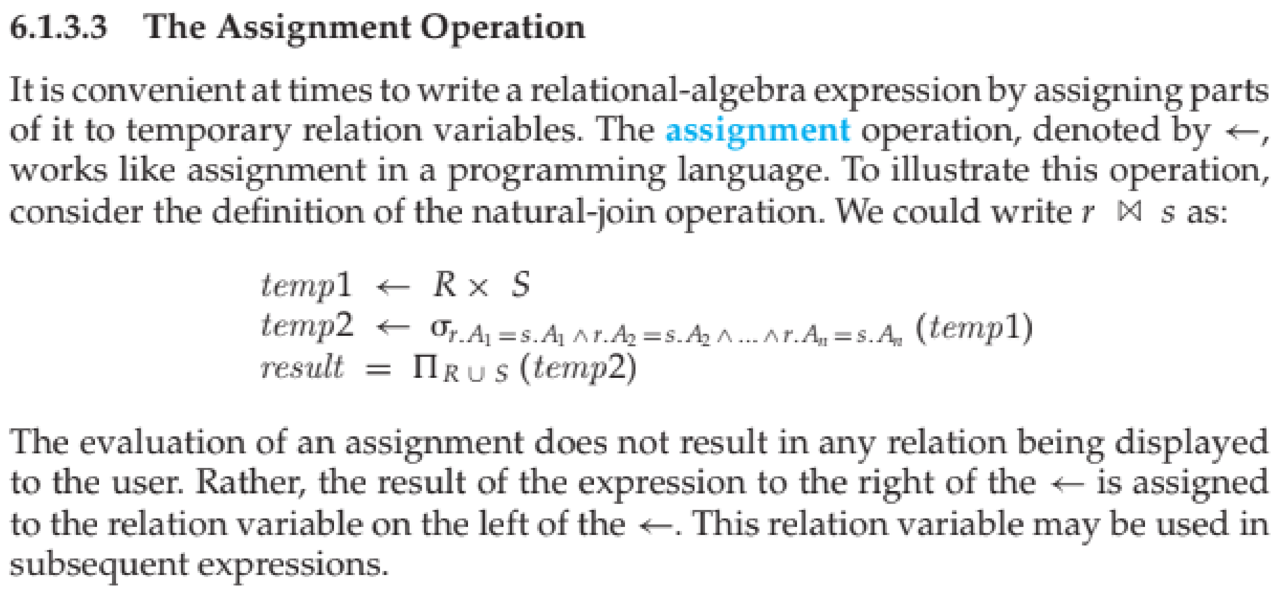
$$\Pi_{name, title} (\sigma_{dept\_name = \text{"Comp. Sci."}} (instructor \bowtie teaches \bowtie course))$$



ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 6.14 The natural join of the *instructor* relation with the *teaches* relation.

Additional Relational algebra operations



6.1.3.3 The Assignment Operation

It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables. The **assignment** operation, denoted by $\leftarrow$, works like assignment in a programming language. To illustrate this operation, consider the definition of the natural-join operation. We could write $r \bowtie s$ as:

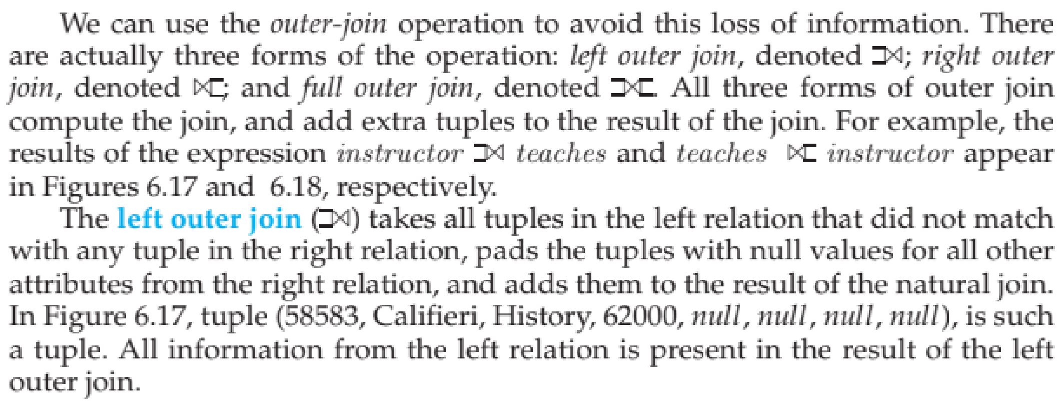
$$\begin{aligned} temp1 &\leftarrow R \times S \\ temp2 &\leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (temp1) \\ result &= \Pi_{R \cup S} (temp2) \end{aligned}$$

The evaluation of an assignment does not result in any relation being displayed to the user. Rather, the result of the expression to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$. This relation variable may be used in subsequent expressions.

Additional Relational algebra operations

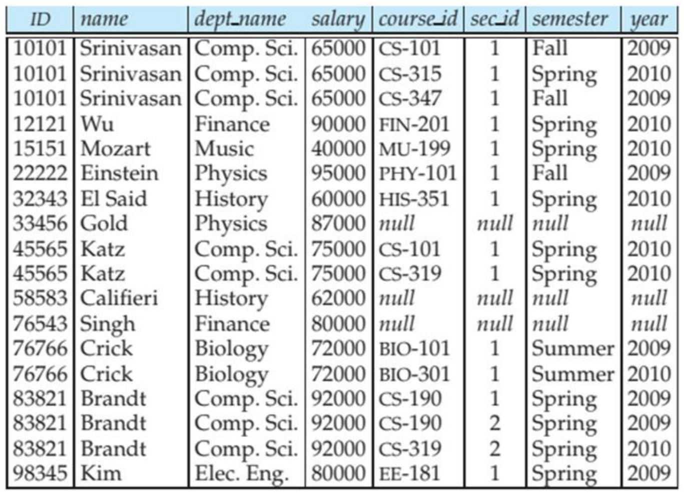
6.1.3.4 Outer join Operations

The outer-join operation is an extension of the join operation to deal with missing information.



We can use the *outer-join* operation to avoid this loss of information. There are actually three forms of the operation: *left outer join*, denoted ⟕; *right outer join*, denoted ⟖; and *full outer join*, denoted ⟗. All three forms of outer join compute the join, and add extra tuples to the result of the join. For example, the results of the expression *instructor* ⟕ *teaches* and *teaches* ⟖ *instructor* appear in Figures 6.17 and 6.18, respectively.

The **left outer join** (⟕) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join. In Figure 6.17, tuple (58583, Califieri, History, 62000, *null*, *null*, *null*, *null*), is such a tuple. All information from the left relation is present in the result of the left outer join.

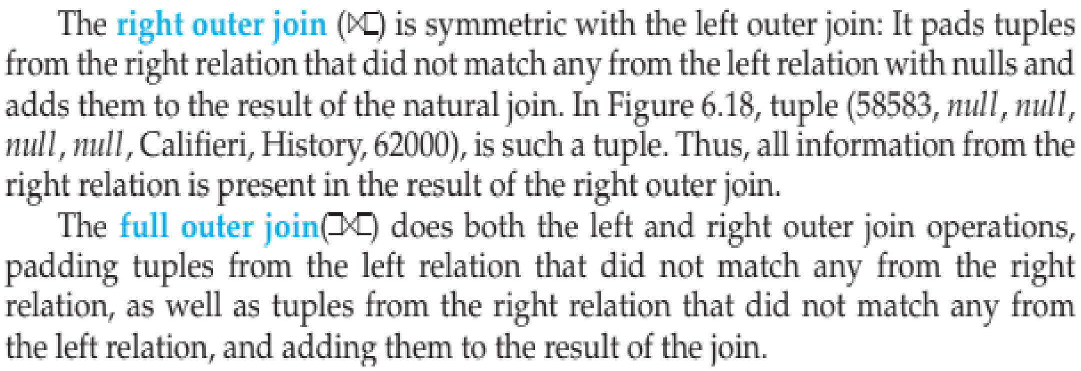


ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
33456	Gold	Physics	87000	*null*	*null*	*null*	*null*
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
58583	Califieri	History	62000	*null*	*null*	*null*	*null*
76543	Singh	Finance	80000	*null*	*null*	*null*	*null*
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 6.17 Result of *instructor* ⟕ *teaches*.

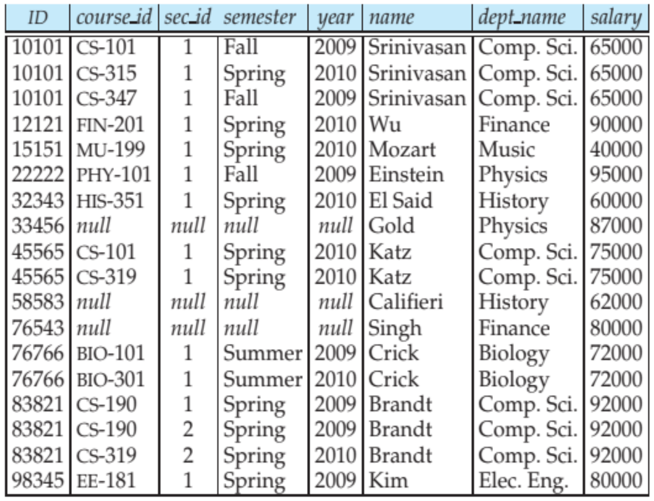
Additional Relational algebra operations

6.1.3.4 Outer join Operations



The **right outer join** (⟖) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join. In Figure 6.18, tuple (58583, *null*, *null*, *null*, *null*, Califieri, History, 62000), is such a tuple. Thus, all information from the right relation is present in the result of the right outer join.

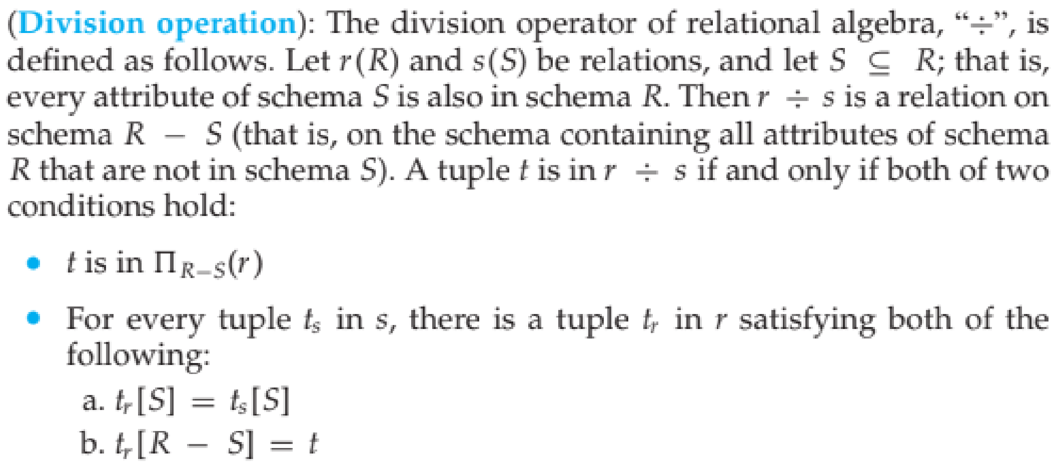
The **full outer join**(⟗) does both the left and right outer join operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.



ID	*course_id*	*sec_id*	*semester*	*year*	*name*	*dept_name*	*salary*
10101	CS-101	1	Fall	2009	Srinivasan	Comp. Sci.	65000
10101	CS-315	1	Spring	2010	Srinivasan	Comp. Sci.	65000
10101	CS-347	1	Fall	2009	Srinivasan	Comp. Sci.	65000
12121	FIN-201	1	Spring	2010	Wu	Finance	90000
15151	MU-199	1	Spring	2010	Mozart	Music	40000
22222	PHY-101	1	Fall	2009	Einstein	Physics	95000
32343	HIS-351	1	Spring	2010	El Said	History	60000
33456	*null*	*null*	*null*	*null*	Gold	Physics	87000
45565	CS-101	1	Spring	2010	Katz	Comp. Sci.	75000
45565	CS-319	1	Spring	2010	Katz	Comp. Sci.	75000
58583	*null*	*null*	*null*	*null*	Califieri	History	62000
76543	*null*	*null*	*null*	*null*	Singh	Finance	80000
76766	BIO-101	1	Summer	2009	Crick	Biology	72000
76766	BIO-301	1	Summer	2010	Crick	Biology	72000
83821	CS-190	1	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-190	2	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-319	2	Spring	2010	Brandt	Comp. Sci.	92000
98345	EE-181	1	Spring	2009	Kim	Elec. Eng.	80000

Figure 6.18 Result of *teaches* ⟖ *instructor*.

Additional Relational algebra operations



(**Division operation**): The division operator of relational algebra, "$\div$", is defined as follows. 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 $r \div s$ 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 $r \div s$ 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:
  a. $t_r[S] = t_s[S]$
  b. $t_r[R - S] = t$

Relational algebra queries (based on previous database)

1. Find out the account details with balance greater than 10000.

2. Find out the loan details where the loan amount is greater than 50000 and loan taken is from khandagiri branch.

3. Find out the phone number and address of all the customers.

4. Find out the phone number and address of customer name sachin

5. Find out the account number and branch name of all those account where the balance is less than 1000.

6. Find out the customer id of those customer who have either an account in the bank or else have borrower loan from the bank.

7. Find out the customer Id of customer who have account in the bank but not taken loan.

8. Find out the name of customer along with their account number.

9. Find out the name, phone number and balance of customer having balance more than 50000.

10. Find out the name of customer who have taken loan from any branch with branch city Bhubaneswar

11. Find out the branch name and branch city of customer having customer name sachin has a account

12. Find out the customer Id along with their loan number and account number

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 (v), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if 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 in *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

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

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

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

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

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

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

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

  $\{t \mid \exists s \in \textit{teaches}\ (t[\textit{course\_id}] = s[\textit{course\_id}] \wedge$
  $\qquad s[\textit{semester}] = \text{“Fall”} \wedge s[\text{year}] = 2009$
  $\quad \vee \exists u \in \textit{teaches}\ (t[\textit{course\_id}] = u[\textit{course\_id}] \wedge$
  $\qquad u[\textit{semester}] = \text{“Spring”} \wedge u[\text{year}] = 2010)\}$

Example Queries

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

$$\{t \mid \exists s \in \textit{teaches}\ (t[\textit{course\_id}] = s[\textit{course\_id}] \land s[\textit{semester}] = \text{“Fall”} \land s[\text{year}] = 2009 \land \exists u \in \textit{teaches}\ (t[\textit{course\_id}] = u[\textit{course\_id}] \land u[\textit{semester}] = \text{“Spring”} \land u[\text{year}] = 2010\ )\}$$

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

$$\{t \mid \exists s \in \textit{teaches}\ (t[\textit{course\_id}] = s[\textit{course\_id}] \land s[\textit{semester}] = \text{“Fall”} \land s[\text{year}] = 2009 \land \neg \exists u \in \textit{teaches}\ (t[\textit{course\_id}] = u[\textit{course\_id}] \land u[\textit{semester}] = \text{“Spring”} \land u[\text{year}] = 2010\ )\}$$

Example Queries

- Q1: Find out the account detail where the balance is greater than 50000.

- Q2: Find out the loan details for branch name Khandagiri, and loan amount greater than 50000.

- Q7: Find out the loan amount and branch name of the loan with loan number=L001.

- Q8: Find out the account number of the customer along with their customer name.

- Q9: Find out the account number of customer name sachin.

Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{t \mid \exists\, r \in student\ (t[ID] = r[ID]) \wedge$
    $(\forall\, u \in course\ (u[dept\_name] = \text{“Biology”} \Rightarrow$
    $\exists\, s \in takes\ (t[ID] = s[ID] \wedge$
    $s[course\_id] = u[course\_id]))\}$

Q10: Find out all customer name who have an account at all branches located in Bhubaneswar.

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.
    - E.g. $\{ t \mid t[A] = 5 \vee \textbf{true} \}$ is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in *P*.

Safety of Expressions (Cont.)

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

Domain Relational Calculus

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula similar to that of the predicate calculus

Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000
  - $\{< i, n, d, s> \mid < i, n, d, s> \in instructor \wedge s > 80000\}$
- As in the previous query, but output only the *ID* attribute value
  - $\{< i> \mid < i, n, d, s> \in instructor \wedge s > 80000\}$
- Find the names of all instructors whose department is in the Watson building

$$\{< n > \mid \exists\, i, d, s\ (< i, n, d, s > \in instructor \wedge \exists\, \text{b, u}\ (< d, b, u> \in department \wedge b = \text{“Watson”}))\}$$

Example Queries

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

  $\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in \mathit{Teaches}\ \wedge$
  $s = \text{“Fall”} \wedge y = 2009\ )$
  $\vee \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in \mathit{Teaches}\ ] \wedge$
  $s = \text{“Spring”} \wedge y = 2010)\}$

  This case can also be written as
  $\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in \mathit{Teaches}\ \wedge$
  $(\,(s = \text{“Fall”} \wedge y = 2009\ )\ \vee (s = \text{“Spring”} \wedge y = 2010))\}$

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

  $\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in \mathit{Teaches}\ \wedge$
  $s = \text{“Fall”} \wedge y = 2009\ )$
  $\wedge \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in \mathit{Teaches}\ ] \wedge$
  $s = \text{“Spring”} \wedge y = 2010)\}$

Safety of Expressions

The expression:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from *dom* (*P*) (that is, the values appear either in *P* or in a tuple of a relation mentioned in *P*).
2. For every "there exists" subformula of the form $\exists\, x\, (P_1(x))$, the subformula is true if and only if there is a value of *x* in *dom* (P_1) such that $P_1(x)$ is true.
3. For every "for all" subformula of the form $\forall_x\, (P_1\, (x))$, the subformula is true if and only if $P_1(x)$ is true for all values *x* from *dom* (P_1).

Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{< i > \mid \exists\ n, d, tc\ (< i, n, d, tc > \in student\ \land$
    $(\forall\ ci, ti, dn, cr\ (< ci, ti, dn, cr > \in course \land dn = \text{“Biology”}$
    $\Rightarrow\ \exists\ si, se, y, g\ (<i, ci, si, se, y, g> \in takes\ ))\}$
  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

- Find out the account detail where the balance is greater than 50000.

- Find out the customer details of customer having phone number 7407983296

- Find out the name of all customers along with their phone numbers.
- Find out the loan number and amount of all loan taken from khandagiri branch.

- Find out the phone number and address of customer named sachin.

- Find out all customer Id where the account are maintained at khandagiri branch.

- Find out the account number of customer name sachin.

- Find out the loan amount of a customer with customer ID C001.

- Find out the branch city where customer with customer Id C001 has his account.

End of Chapter 6