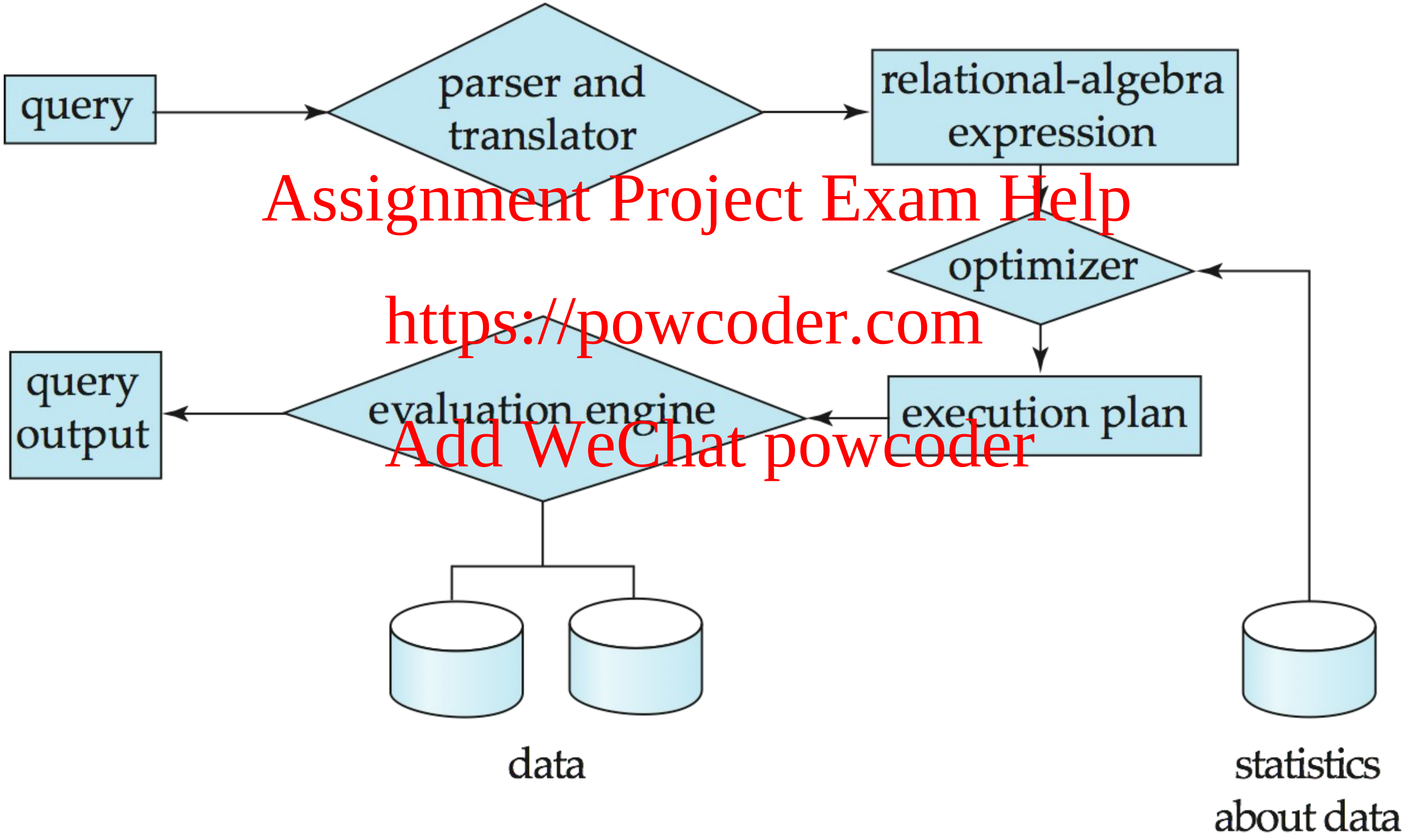
A Big Picture: Basic Steps in Query Processing



A Big Picture: Basic Steps in Query Processing (Cont.)

- A ***SQL query*** can be translated into ***several relational algebra expressions*** (internal form).
  - E.g., **select** *salary* **from** *instructor* **where** *salary* < 75000; →
  - $\sigma_{salary<75000}(\Pi_{salary}(instructor))$; or
  - $\Pi_{salary}(\sigma_{salary<75000}(instructor))$
- The relational algebra representation of a query specifies (partially) how to evaluate a query.

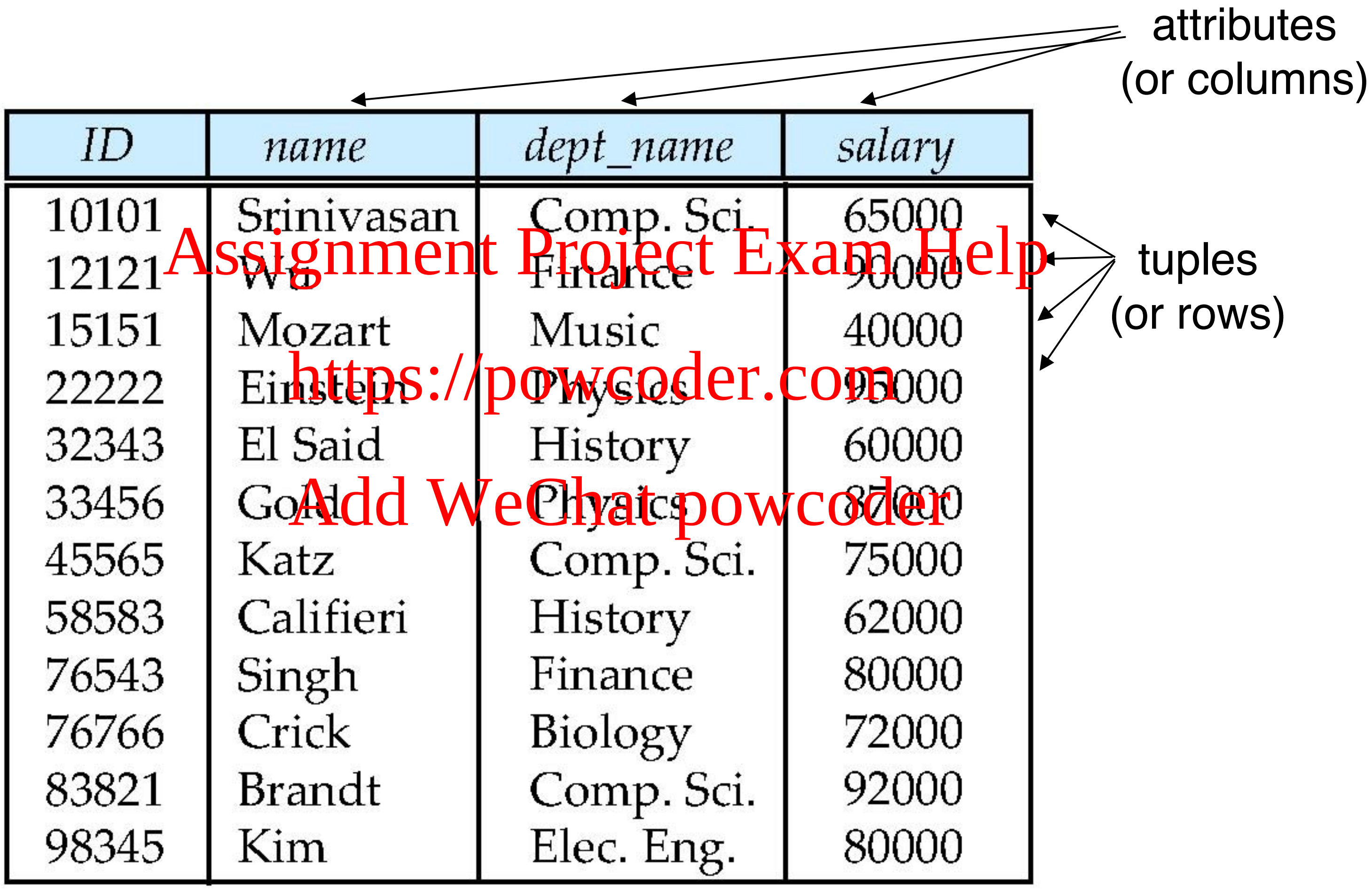
Relational Algebra

Database System Concepts

Relational Algebra

- **A quick review of relational database**
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- Additional operations that simplify common queries
  - set intersection
  - join
  - assignment
  - outer join
- Extended relational algebra operations
  - generalized projection
  - aggregate functions

Example of a *Instructor* Relation



attributes
(or columns)

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

tuples
(or rows)

Attribute

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain, indicated that the value is “unknown”

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are *attributes*
- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

Example:

instructor = (*ID*, *name*, *dept_name*, *salary*)

- Formally, given sets $D_1, D_2, \ldots, D_n$ a **relation** ***r*** is a subset of

$$D_1 \text{ x } D_2 \text{ x } \ldots \text{ x } D_n$$

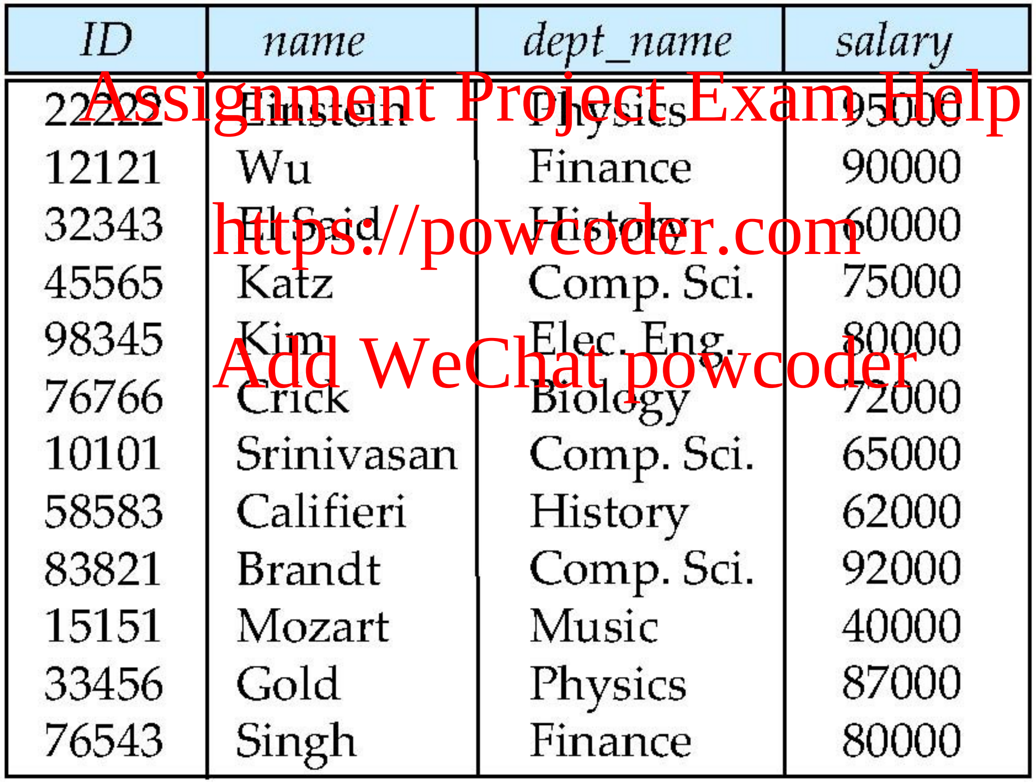
where D_i is the domain of attribute A_i. Thus, a relation is a set of *n*-tuples

$(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

- The current values (**relation instance**) of a relation are specified by a table
- An element ***t*** of ***r*** is a **tuple**, represented by a *row* in a table

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples



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

Database

- A database consists of multiple relations
- Information about a university is broken up into parts

  instructor
  student
  advisor

- Bad design:

  univ (*instructor-ID, name, dept_name, salary, student_ID*, ..)

  results in

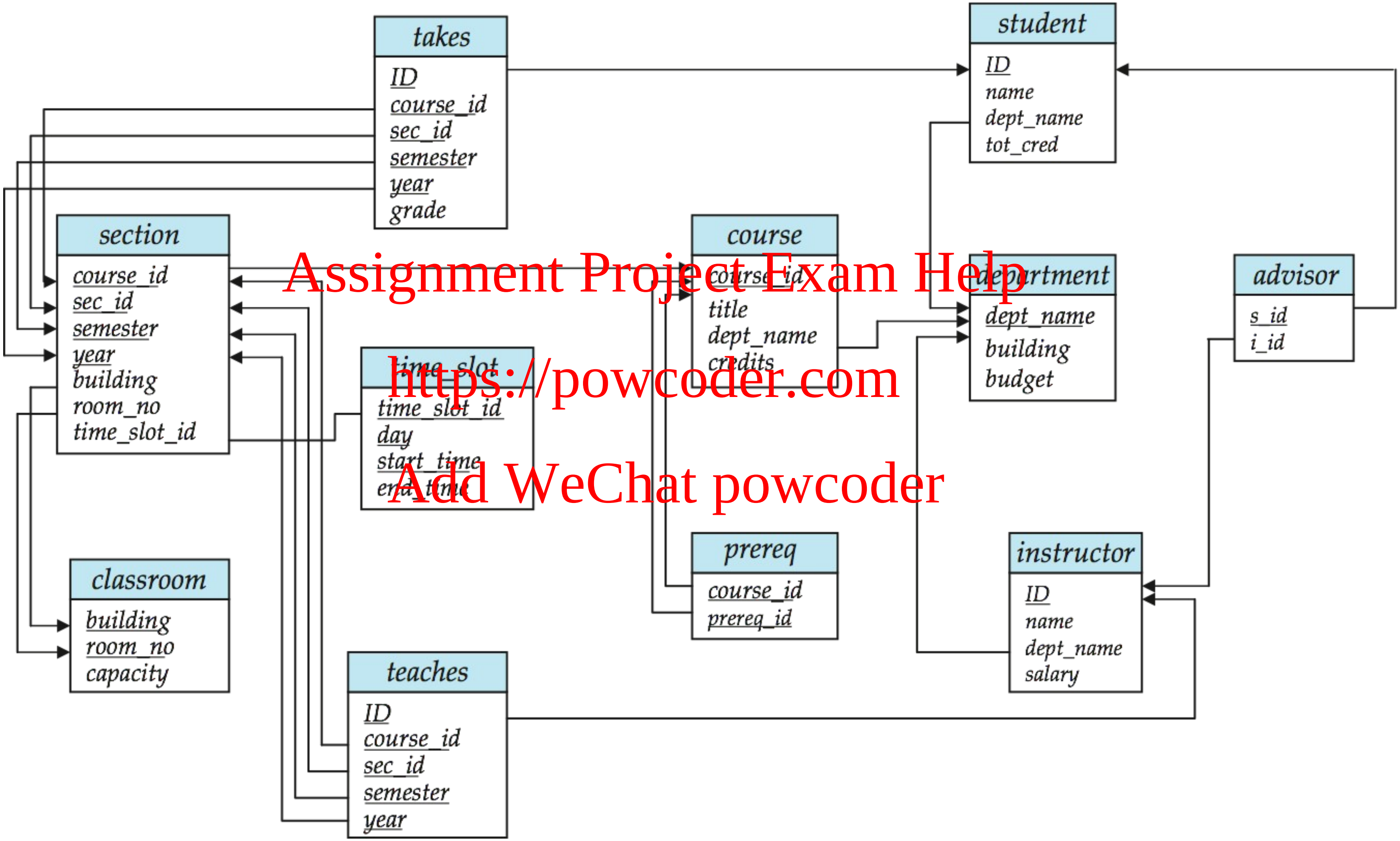
  - repetition of information (e.g., two students have the same instructor)
  - the need for null values (e.g., represent an student with no advisor)

Keys

- Let $K \subseteq R$
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal

  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation
  - Example – *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

Schema Diagram for University Database



Relational Algebra

- A quick review of relational database
- **Six basic operators**
  - **select: σ**
  - **project: ∏**
  - **union: ∪**
  - **set difference: –**
  - **Cartesian product: x**
  - **rename: *ρ***
- Additional operations that simplify common queries
  - set intersection
  - join
  - assignment
  - outer join
- Extended relational algebra operations
  - generalized projection
  - aggregate functions

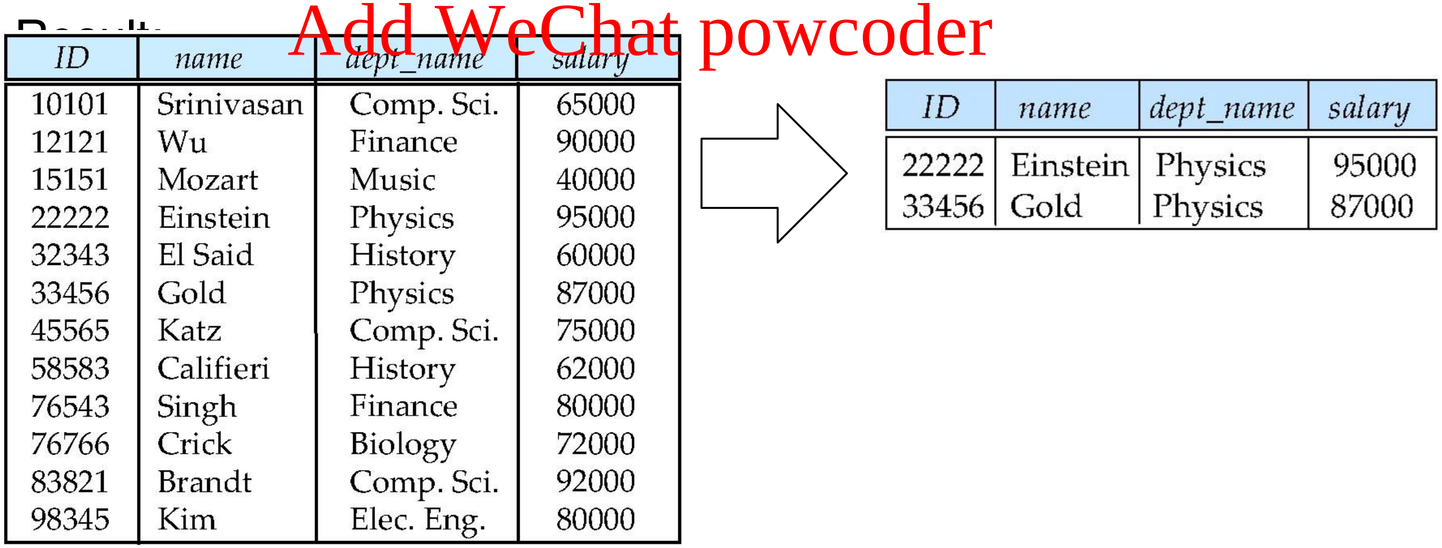
Select Operation

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- *p* is called the **selection predicate**
- Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.

- Query:

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

- Result:



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
33456	Gold	Physics	87000

Select Operation (Cont.)

- We allow comparisons using

  =, ≠, >, , <, ≤

  in the selection predicate.

- We can combine several predicates into a larger predicate by using the connectives:

  ∧ (**and**), ∨ (**or**), ¬ (**not**)

- Example: Find the instructors in Physics with a salary greater $90,000, we write:

  $$\sigma_{\textit{dept\_name}=\textit{“Physics”} \land \textit{salary} > 90{,}000}(\textit{instructor})$$

- Then select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:

    $$\sigma_{\textit{dept\_name}=\textit{building}} \text{ (department)}$$

Project Operation

- A unary operation that returns its argument relation, with certain attributes left out.
- Notation:

$$\prod_{A_1, A_2, A_3, \ldots, A_k} (r)$$

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

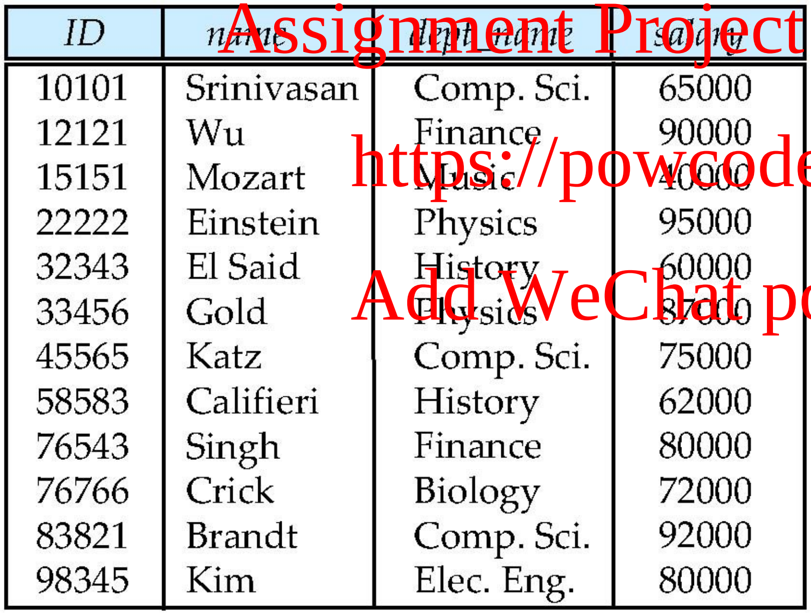
- Duplicate rows removed from result, since relations are sets

Project Operation (Cont.)

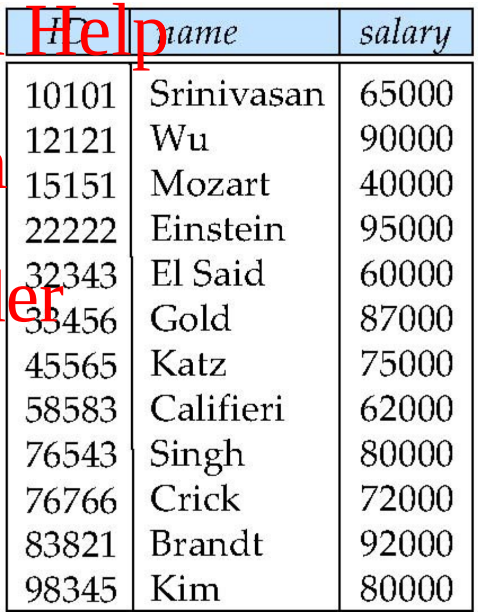
- Example: eliminate the *dept_name* attribute of *instructor*
- Query*:*

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

- Result:



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

Composition of Relational Operations

- The result of a relational-algebra operation is relation and therefore relational-algebra operations can be composed together into a **relational-algebra expression**.

- Consider the query -- Find the names of all instructors in the Physics department.

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

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

Union Operation

- The union operation allows us to combine two relations
- Notation: $r \cup s$
- Output the union of tuples from the two input relations
- 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*)

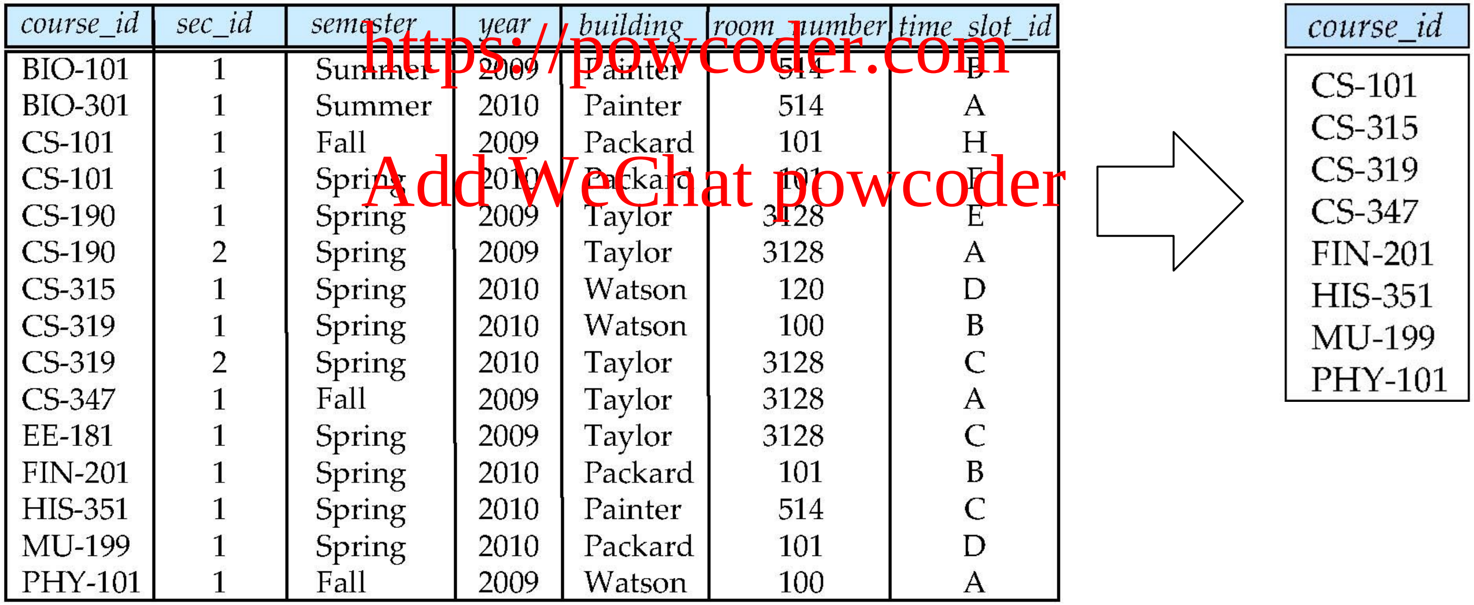
Union Operation – Example

- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both
- Query:

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

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

- Result:



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

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set Difference Operation

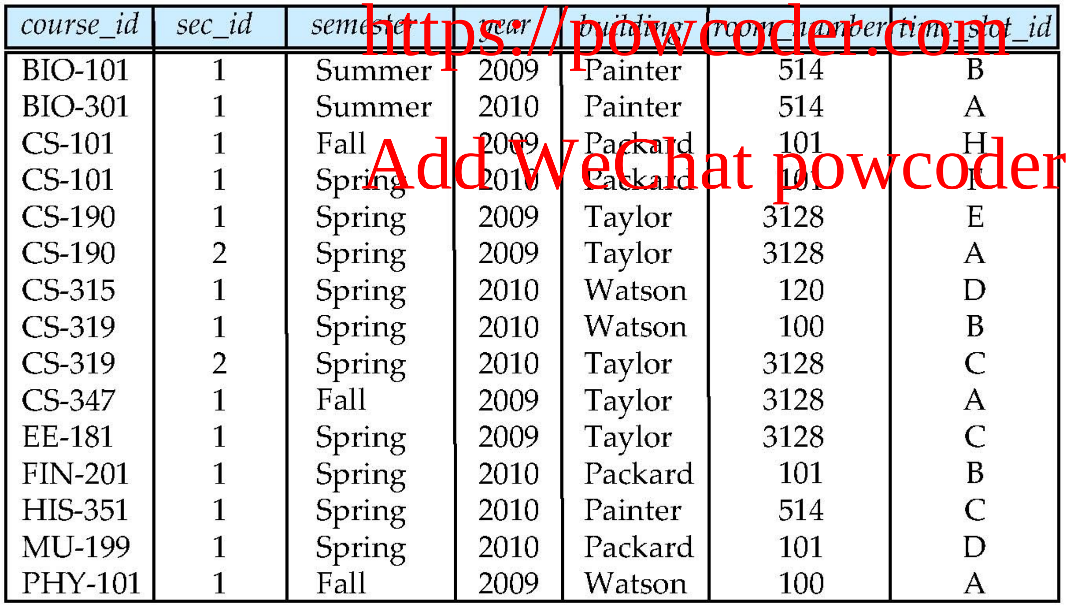
- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation: *r* – *s*
- Produce a relation containing those tuples in *r* but not in *s*
- Set differences must be taken between **compatible** relations.
  - *r* and *s* must have the same arity
  - attribute domains of *r* and *s* must be compatible

Set Difference Operation – Example

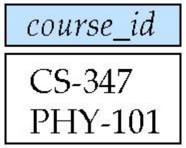
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester
- Query:

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

- Result:



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



course_id
CS-347
PHY-101

Cartesian-Product Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.

- Notation: *r* x *s*

- Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)

- 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.

Cartesian-Product Operation – Example

- Example: the Cartesian product of the relations *instructor* and t*eaches* is written as:

  instructor X *teaches*

- We construct a tuple of the result out of each possible pair of tuples: one from the *instructor* relation and one from the *teaches* relation (see next slide)

- Since the instructor *ID* appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
  - *instructor.ID*
  - *teaches.ID*

The *instructor* x *teaches* table

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

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

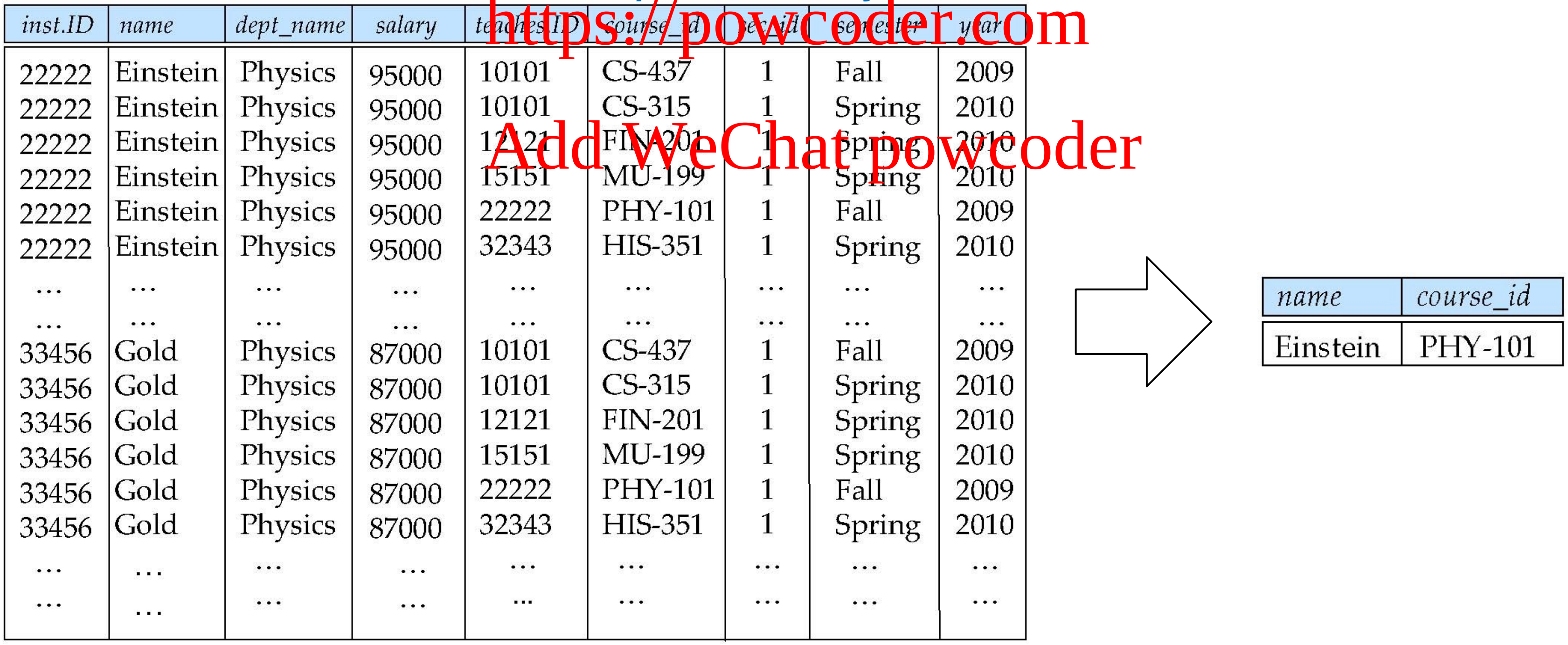
Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2009
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2010
15151	Mozart	Music	40000	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	12121	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
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

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

$$\Pi_{name, course\_id}(\sigma_{instructor.ID=teaches.ID}($$

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



inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
22222	Einstein	Physics	95000	10101	CS-437	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	12121	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
22222	Einstein	Physics	95000	32343	HIS-351	1	Spring	2010
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
33456	Gold	Physics	87000	10101	CS-437	1	Fall	2009
33456	Gold	Physics	87000	10101	CS-315	1	Spring	2010
33456	Gold	Physics	87000	12121	FIN-201	1	Spring	2010
33456	Gold	Physics	87000	15151	MU-199	1	Spring	2010
33456	Gold	Physics	87000	22222	PHY-101	1	Fall	2009
33456	Gold	Physics	87000	32343	HIS-351	1	Spring	2010
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

name	*course_id*
Einstein	PHY-101

Example Queries

- There is often more than one way to write a query in relational algebra
- The following two queries are equivalent, i.e., they give the same result
  - Query 1

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

  - Query 2

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

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.
- The expression

$$\rho_x\,(E)$$

returns the result of expression E under the name X

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

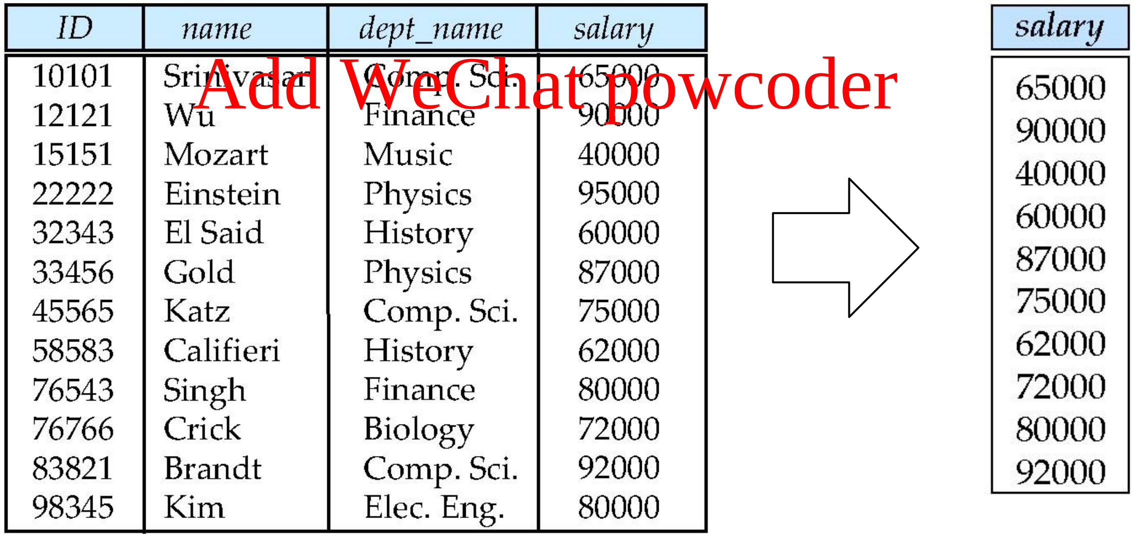
$$\rho_{x(A_1, A_2, \ldots A_n)}\,(E)$$

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

attributes renamed to $A_1, A_2, \ldots., A_n$.

Example Query

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



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

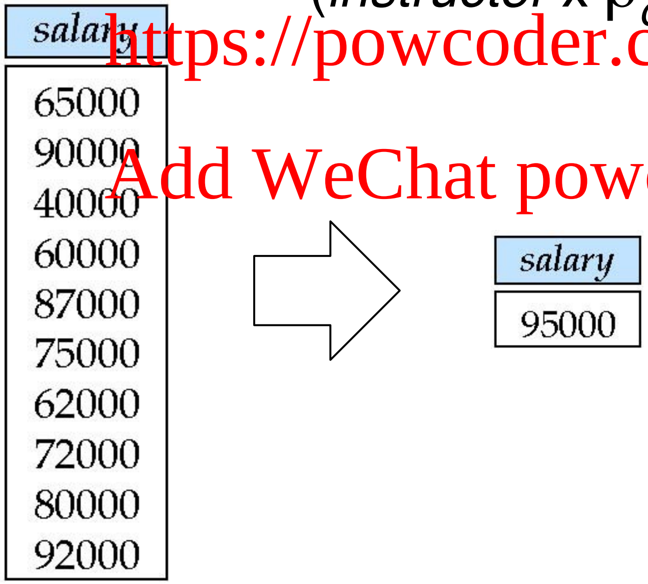
salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

Example Query

- Find the highest salary in the university
  - Step 2: Find the highest salary
    - $\Pi_{salary}(instructor) -$

      $\Pi_{instructor.salary}(\sigma_{instructor.salary < d.salary}$

      $(instructor \times \rho_d (instructor)))$



salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

salary
95000

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

Relational Algebra

- A quick review of relational database
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- **Additional operations that simplify common queries**
  - **set intersection**
  - **Join**
  - **assignment**
  - **outer join**
- Extended relational algebra operations
  - generalized projection
  - aggregate functions

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Join
- Assignment
- Outer join

Set-Intersection Operation

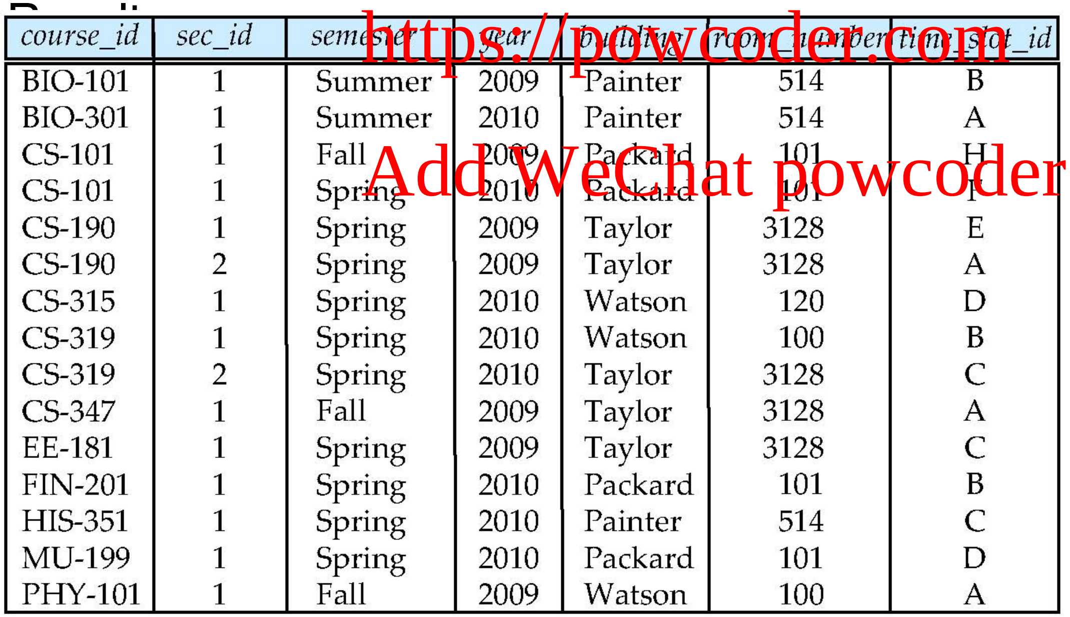
- The set-intersection operation allows us to find tuples that are in both the input relations.
- Notation: $r \cap s$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Set-Intersection Operation – Example

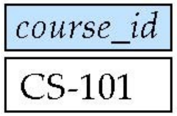
- Example: to find all courses taught in both the Fall 2009 and the Spring 2010 semesters
- Query:

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

- Result:



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



course_id
CS-101

Join Operation

- The Cartesian-Product

  instructor X *teaches*

  associates every tuple of *instructor* with every tuple of *teaches*.

  - Most of the resulting rows have information about instructors who did NOT teach a particular course.

- To get only those tuples of "*instructor* X *teaches* " that pertain to instructors and the courses that they taught, we write:

  $\sigma_{\ instructor.id\ =\ teaches.id}\ (instructor\ \text{x}\ teaches\ )$

  - We get only those tuples of "*instructor* X *teaches"* that pertain to instructors and the courses that they taught.

Join Operation (Cont.)

- The **join** operation allows us to combine a select operation and a Cartesian-Product operation into a single operation.
- Consider relations *r*(*R*) and *s*(*S*)
- Let “theta” be a predicate on attributes in the schema R “union” S. The join operation r s is defined as follows:

- Thus

$$\sigma_{instructor.id = teaches.id}(instructor \ x \ teaches)$$

- Can equivalently be written as

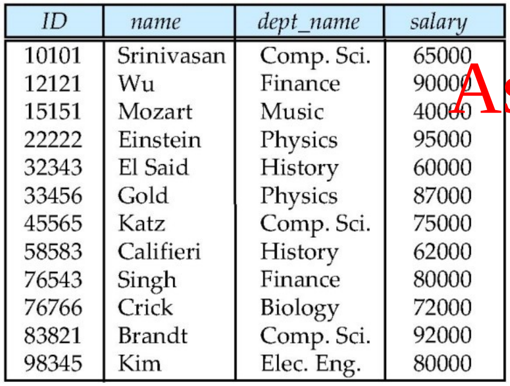
$$instructor_{\ Instructor.id = teaches.id}\ teaches.$$

Join Operation (Cont.)

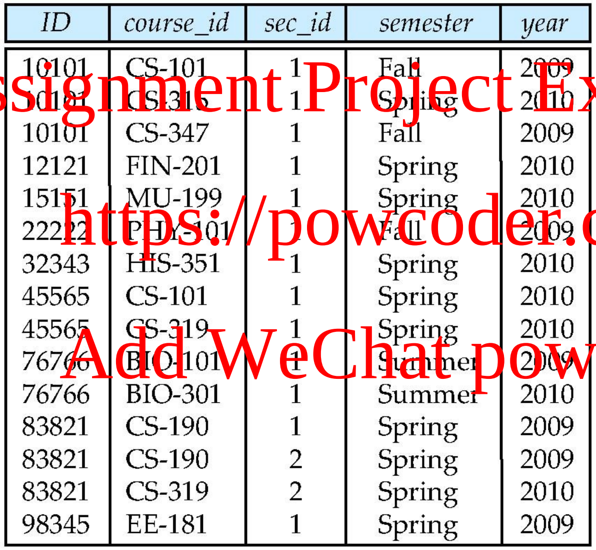
- The join operation without predicate is called **natural join**.
- Notation: *r* *s*
- Output pairs of rows from the two input relations that have the **same value** on all attributes that have the **same name**
- Let *r* and *s* be relations on schemas *R* and *S* respectively.
  Then, *r* *s* is 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*

Join Operation – Example

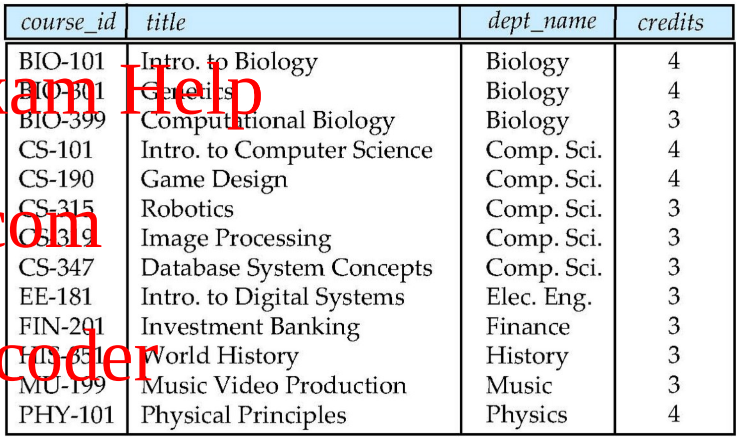
- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  - $\Pi_{name,\ title}\ (\sigma_{dept\_name=\text{“Comp. Sci.”}}\ (instructor \bowtie teaches \bowtie course))$



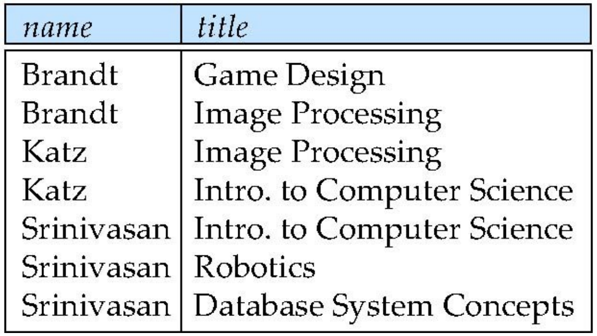
ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821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



ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009



course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4



name	title
Brandt	Game Design
Brandt	Image Processing
Katz	Image Processing
Katz	Intro. to Computer Science
Srinivasan	Intro. to Computer Science
Srinivasan	Robotics
Srinivasan	Database System Concepts

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 is denoted by ← and works like assignment in a programming language.
- Example: Find all instructor in the "Physics" and Music department.

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

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

$$Physics \cup Music$$

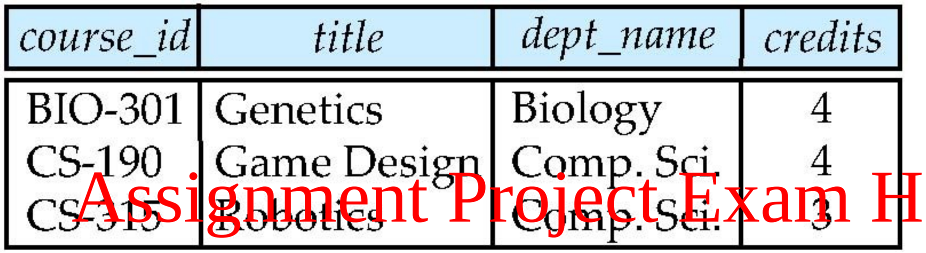
- With the assignment operation, a query can be written as a sequential program consisting of
  - a series of assignments
  - followed by an expression whose value is displayed as a result of the query.

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 result of the join.
- Uses *null* value to signify that the value is unknown or does not exist
- Three forms of outer join:
  - left outer join
  - right outer join
  - full outer join

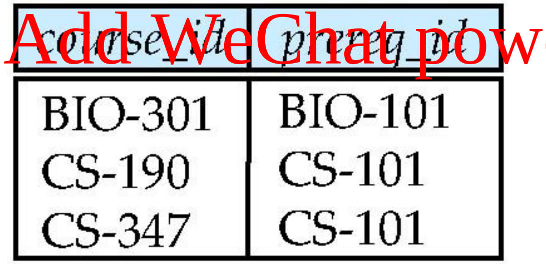
Outer Join – Example

- Relation *course*



course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*



course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

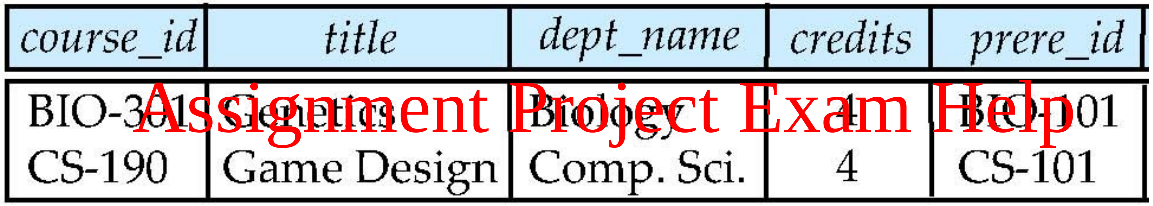
  CS-315 is missing in *prereq* and

  CS-347 is missing in *course*

Outer Join – Example

- Join

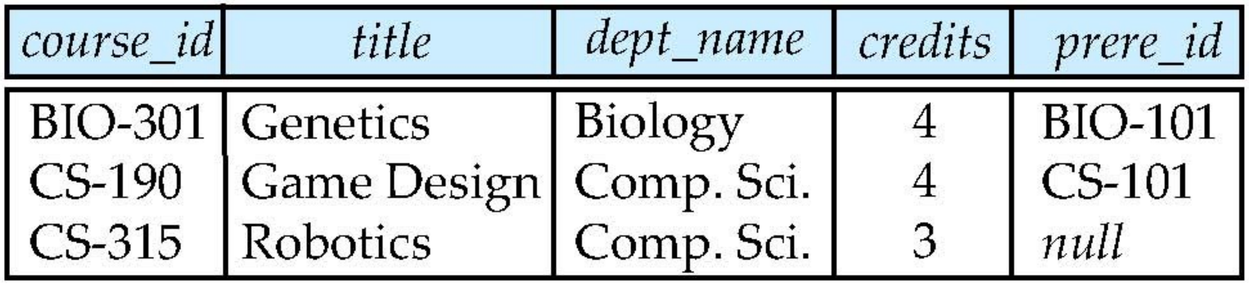
  course prereq



course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101

- Left Outer Join

  course ⟕ prereq

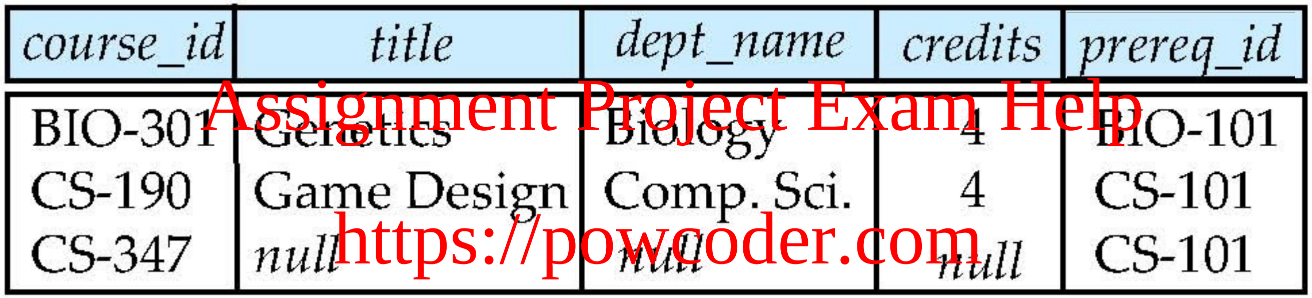


course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

Outer Join – Example

- Right Outer Join

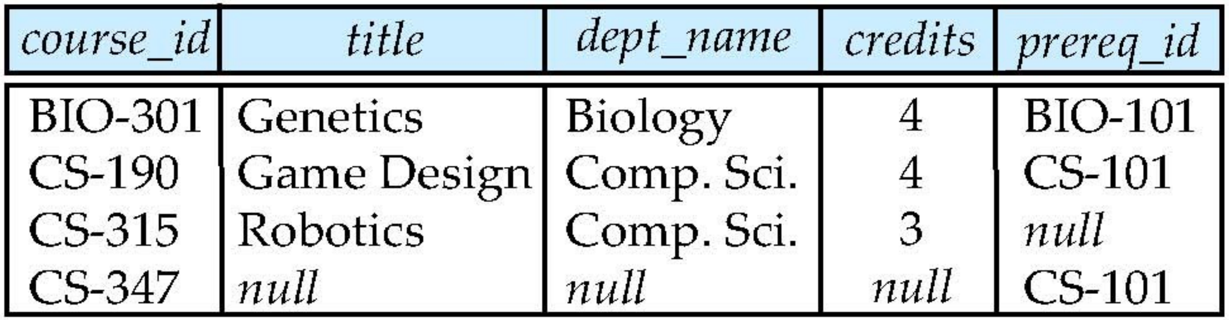
  course ⟖ *prereq*



course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

- Full Outer Join

  course ⟗ *prereq*



course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Outer Join using Joins

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

$$(r \quad s) \cup (r - \Pi_R(r \quad s)) \text{ x } \{(null, \ldots, null)\}$$

where the constant relation $\{(null, \ldots, null)\}$ is on the schema $S - R$

Relational Algebra

- A quick review of relational database
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- Additional operations that simplify common queries
  - set intersection
  - join
  - assignment
  - outer join
- **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.

- *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

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

Aggregate Functions and Operations

- **Aggregate 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

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

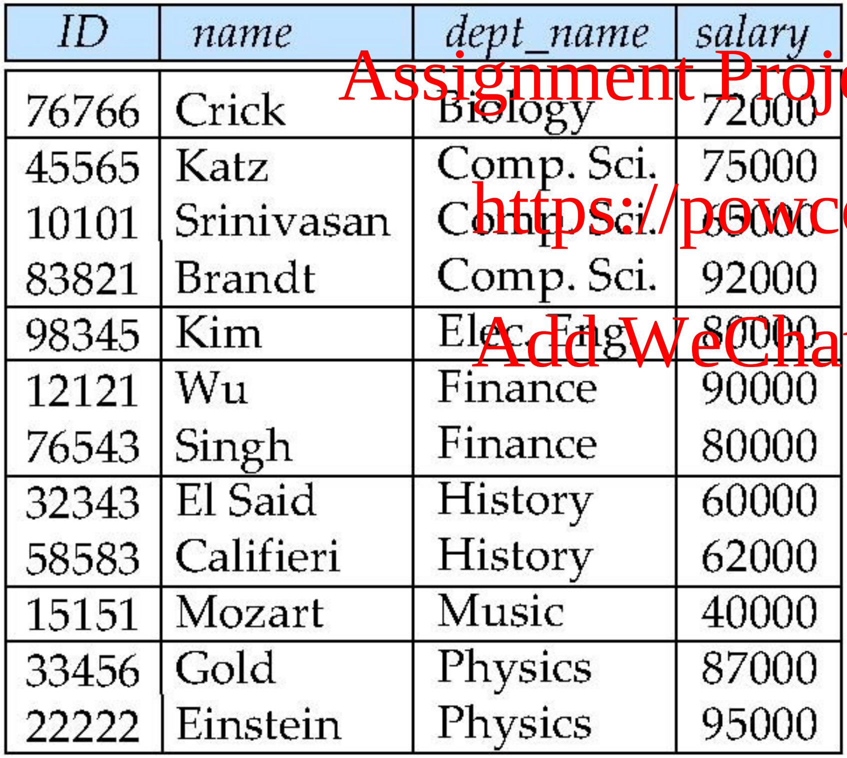
E is any relational-algebra expression

- G_1, G_2 …, G_n is a list of attributes on which to group (can be empty)
- Each F_i is an aggregate function
- Each A_i is an attribute name

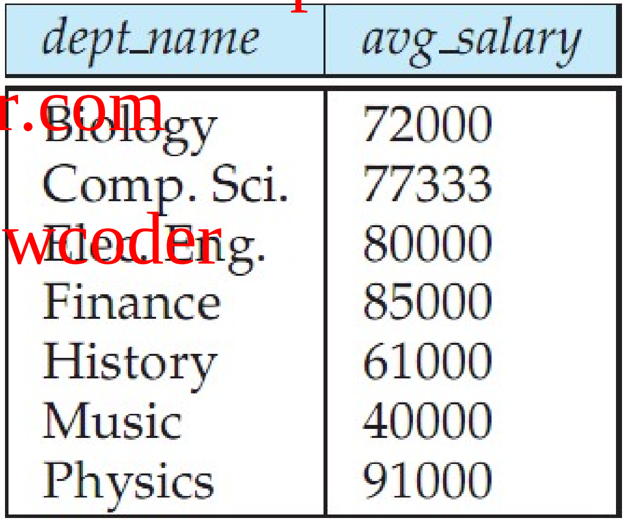
Aggregate Operation – Example

- Find the average salary in each department

- For convenience, we permit renaming as part of aggregate operation



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

Multiset Relational Algebra

- Pure relational algebra removes all duplicates
  - e.g. after projection
- Multiset relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature
- Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
    - σ_θ (r_1): If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selection $\sigma_{\theta,}$, then there are c_1 copies of t_1 in σ_θ (r_1).
  - projection: one tuple per input tuple, even if it is a duplicate
    - $\Pi_A(r_1)$: For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1.

Multiset Relational Algebra (Cont.)

- Multiset relational algebra defined as follows
  - Cartesian product:
    - r_1 x r_2 : If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are c_1 x c_2 copies of the tuple $t_1. t_2$ in r_1 x r_2
  - Other operators similarly defined
    - union: *m + n copies*
    - intersection: min(*m, n*) copies
    - difference: min(0, *m – n*) copies
- Example: Suppose multiset relations r_1 (*A, B*) and r_2 (*C*) are as follows:

  $r_1 = \{(1, a)\ (2,a)\}$ $r_2 = \{(2), (3), (3)\}$

- Then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1)$ x r_2 would be

  $\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$

SQL and Multiset Relational Algebra

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

is equivalent to the following expression in multiset relational algebra

$$\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

- The **select** clause corresponds to the **projection** operation
- The **from** clause corresponds to the **Cartesian product** operation
- The **where** clause corresponds to the **selection** operation

SQL and Multiset Relational Algebra- Examples

- Find courses that ran in Fall 2009 **or** in Spring 2010

(**select** *course_id* **from** *section* **where** *semester* = ‘Fall’ **and** *year* = 2009)
union
(**select** *course_id* **from** *section* **where** *semester* = ‘Spring’ **and** *year* = 2010)

is equivalent to

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

- Find courses that ran in Fall 2009 **and** in Spring 2010

(**select** *course_id* **from** *section* **where** *semester* = ‘Fall’ **and** *year* = 2009)
intersect
(**select** *course_id* **from** *section* **where** *semester* = ‘Spring’ **and** *year* = 2010)

is equivalent to

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

SQL and Multiset Relational Algebra - Examples

- Find courses that ran in Fall 2009 but **not in** Spring 2010

(**select** *course_id* **from** *section* **where** *semester* = ‘Fall’ **and** *year* = 2009)
except
(**select** *course_id* **from** *section* **where** *semester* = ‘Spring’ **and** *year* = 2010)

is equivalent to

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

SQL and Multiset Relational Algebra (Cont.)

- Set operations **union, intersect,** and **except** automatically eliminate duplicates
- To retain all duplicates, use the corresponding multiset versions **union all, intersect all** and **except all.**

  Suppose a tuple occurs *m* times in *r* and *n* times in *s*, then, it occurs:

  - *m + n* times in *r* **union all** *s*
  - min(*m,n)* times in *r* **intersect all** *s*
  - max(0, *m – n)* times in *r* **except all** *s*

SQL and Multiset Relational Algebra (Cont.)

- **select** A_1, A_2, **sum**(A_3)
  from r_1, r_2, ..., r_m
  where P
  group by A_1, A_2 **having count** (A_4) > 2

is equivalent to the following expression in multiset relational algebra

- Find the average salary of instructors in each department

  select *dept_name*, **avg** (*salary*) **as** *avg_salary*
  from *instructor*
  group by *dept_name*;

  is equivalent to