data model: method to define data structure, method to describe data.

traditional: hierarchical model
network model

# Chapter 2: Intro to Relational Model

very simple, high efficiency
commercially used

**Database System Concepts, 7th Ed.**

# Outline

- Structure of Relational Databases
- Database Schema
- Keys
- Schema Diagrams
- Relational Query Languages
- The Relational Algebra

# Example of a *Instructor* Relation

attributes of the instructor

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)

2 dimension table.

# 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*)
- A relation instance *r* defined over schema *R* is denoted by *r* (*R*).
- The current values a relation are specified by a table
- An element ***t*** of relation ***r*** is called a *tuple* and is represented by a *row* in a table

R

| *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 |

t → 76766 row

R

r

# Attributes

- 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"
- The null value causes complications in the definition of many operations

domain : simple, not allowed to have complex structure

table key words

**create table** *instructor* (

| | |
|---|---|
| *ID* | **char**(5), |
| *name* | **varchar**(20), |
| *dept_name* | **varchar**(20), |
| *salary* | **numeric**(8,2)) |

↑ domain Datatype

# Relations are Unordered

List in Python: have order
Set: no order

- 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 Schema

all the current values types

all the tables managed by database / schema

- Database schema -- is the logical structure of the database.
- Database instance -- is a snapshot of the data in the database at a given instant in time.
  - value always change
- Example:
  - schema: *instructor* (*ID, name, dept_name, salary*)
  - Instance: uniquely identified

| *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 |

ID can be a key.

* name can be a key.

guarantee (no two students have same names)

$R = R(A_1, A_2, \cdots, A_n)$
$K \subseteq R$

# Keys

e.g. $A_1 \subseteq R$
$\{A_1, A_2\} \subseteq R$
$\{A_1, A_2, \cdots, A_n\} = R \subseteq R$

- 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**.
  - Which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation
  - Example: *dept_name* in i*nstructor* is a foreign key from *instructor* referencing *department*

# Foreign key

- **Foreign key**: Value in one relation is the key in another relation
  - **Referencing** relation
  - **Referenced** relation
  - Example: *dept_name* in i *nstructor* is a foreign key from *instructor* referencing *department*

Value comes from department table

| *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 |

(a) The *instructor* table

**Referencing**

values come from another table.

foreign key, values already exist there

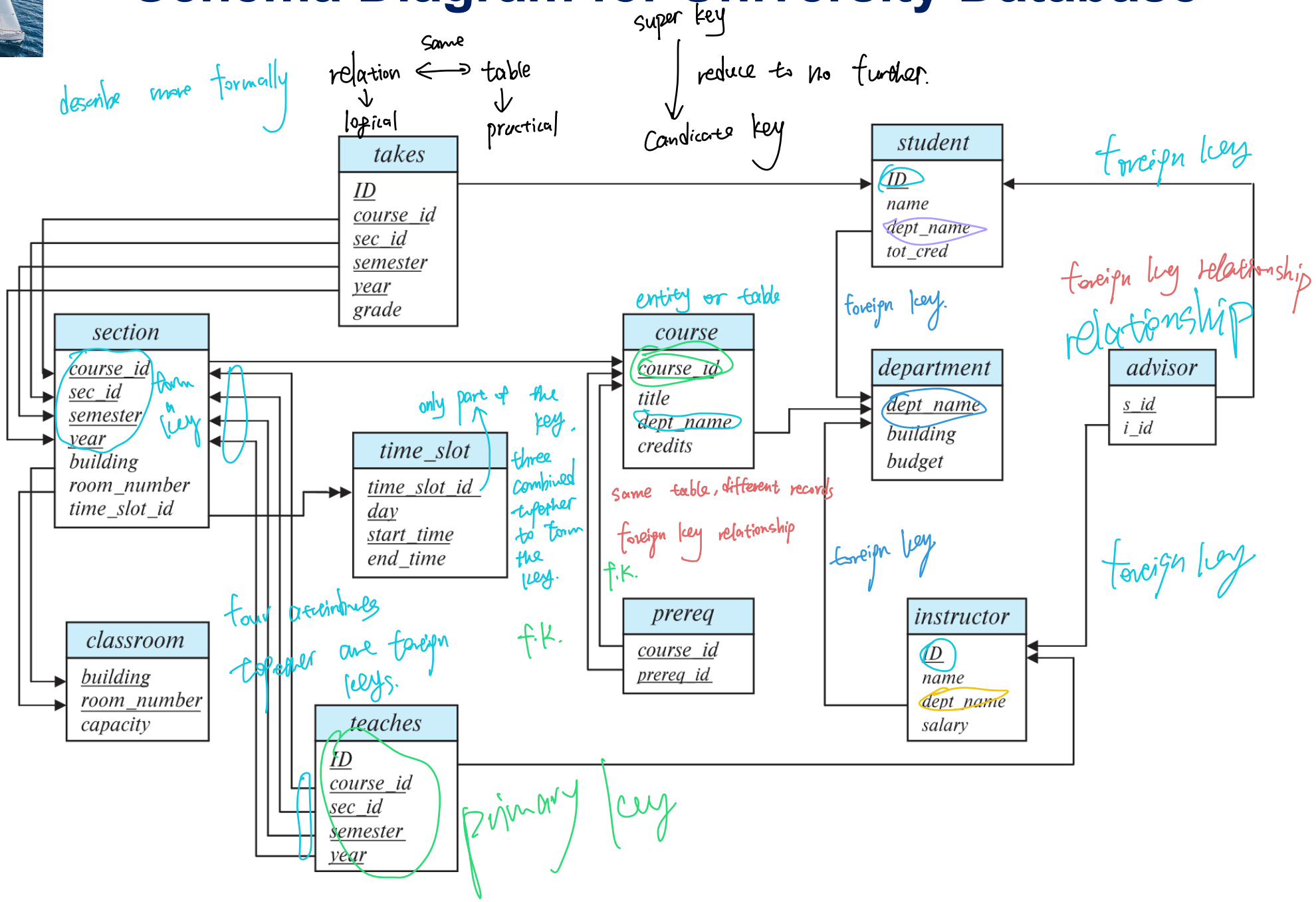
| *dept_name* | *building* | *budget* |
|---|---|---|
| Comp. Sci. | Taylor | 100000 |
| Biology | Watson | 90000 |
| Elec. Eng. | Taylor | 85000 |
| Music | Packard | 80000 |
| Finance | Painter | 120000 |
| History | Painter | 50000 |
| Physics | Watson | 70000 |

primary key in this table.

(b) The *department* table

**Referenced**

# Schema Diagram for University Database



# Relational Query Languages

- Procedural versus non-procedural, or declarative
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this chapter on relational algebra
  - Not Turing-machine equivalent
  - Consists of 6 basic operations

# Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
  - select: $\sigma$ Select individual record now
  - project: $\Pi$ only two attributes for the relation only several attributes columns
  - union: $\cup$ two sets union together
  - set difference: – set 1: all the students records set 2: student records of Macau students
  - Cartesian product: x
  - rename: $\rho$ rename the attributes

Set 1 — Set 2 $S_1=\{a,b,c,d\}$ $S_2=\{a,b\}$ $S_3=S_1-S_2=\{c,d\}$

$S_4=\{a,b,e\}$

$S_5=S_1-S_4=\{c,d\}$ because e not in $S_1$

T

| | $A_1$ | $A_2$ |
|---|---|---|
| $t_1$ | a | b |
| $t_2$ | c | d |

S

| $B_1$ | $B_2$ | |
|---|---|---|
| e | f | $S_1$ |
| x | y | $S_2$ |

$T \times S =$

| $A_1$ | $A_2$ | $B_1$ | $B_2$ |
|---|---|---|---|
| a | b | e | f |
| a | b | x | y |
| c | d | e | f |
| c | d | x | y |

# Select Operation

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

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

dept_name = "Physics"

  - Result

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

AND : both departments Physics & CS : empty because instructor only for one department

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 22222 | Einstein | Physics | 95000 |
| 33456 | Gold | Physics | 87000 |

# Select Operation (Cont.)

no "≠" in SQL, but "<>" or "!="

- We allow comparisons using
  $=, \neq, >, \geq, <, \leq$
  in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:
  - $\wedge$ (**and**)
  - $\vee$ (**or**)
  - $\neg$ (**not**)
- Example: Find the instructors in Physics with a salary greater $90,000, we write:

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

  $\sigma_{\neg(dept\_name='CS')}(instructor)$ = ∅ might occur

- The select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:
  - $\sigma_{dept\_name=building}(department)$

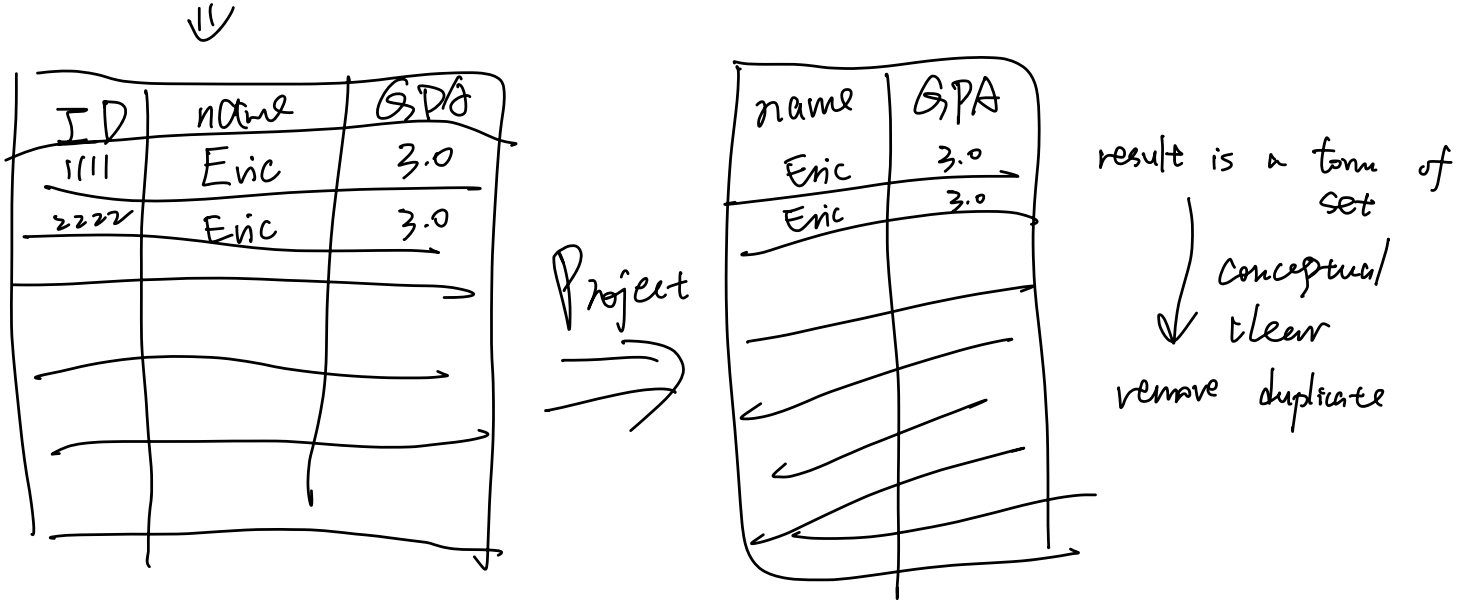
# Project Operation

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

$$\Pi_{A1,A2,A3\ \ldots.Ak}\ (r)$$

where $A_1$, $A_2$, …, $A_k$ 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 Example

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

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

- Result:

| *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 of 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.

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

procedure

cannot change order

if $\Pi$ name first, $\sigma$ dept = physics can't be done

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

# Cartesian-Product Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
- 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* | *teaches.ID* | *course_id* | *sec_id* | *semester* | *year* |
|---|---|---|---|---|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-101 | 1 | Fall | 2017 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-315 | 1 | Spring | 2018 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-347 | 1 | Fall | 2017 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 12121 | FIN-201 | 1 | Spring | 2018 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 15151 | MU-199 | 1 | Spring | 2018 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 22222 | PHY-101 | 1 | Fall | 2017 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 12121 | Wu | Finance | 90000 | 10101 | CS-101 | 1 | Fall | 2017 |
| 12121 | Wu | Finance | 90000 | 10101 | CS-315 | 1 | Spring | 2018 |
| 12121 | Wu | Finance | 90000 | 10101 | CS-347 | 1 | Fall | 2017 |
| 12121 | Wu | Finance | 90000 | 12121 | FIN-201 | 1 | Spring | 2018 |
| 12121 | Wu | Finance | 90000 | 15151 | MU-199 | 1 | Spring | 2018 |
| 12121 | Wu | Finance | 90000 | 22222 | PHY-101 | 1 | Fall | 2017 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 15151 | Mozart | Music | 40000 | 10101 | CS-101 | 1 | Fall | 2017 |
| 15151 | Mozart | Music | 40000 | 10101 | CS-315 | 1 | Spring | 2018 |
| 15151 | Mozart | Music | 40000 | 10101 | CS-347 | 1 | Fall | 2017 |
| 15151 | Mozart | Music | 40000 | 12121 | FIN-201 | 1 | Spring | 2018 |
| 15151 | Mozart | Music | 40000 | 15151 | MU-199 | 1 | Spring | 2018 |
| 15151 | Mozart | Music | 40000 | 22222 | PHY-101 | 1 | Fall | 2017 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 22222 | Einstein | Physics | 95000 | 10101 | CS-101 | 1 | Fall | 2017 |
| 22222 | Einstein | Physics | 95000 | 10101 | CS-315 | 1 | Spring | 2018 |
| 22222 | Einstein | Physics | 95000 | 10101 | CS-347 | 1 | Fall | 2017 |
| 22222 | Einstein | Physics | 95000 | 12121 | FIN-201 | 1 | Spring | 2018 |
| 22222 | Einstein | Physics | 95000 | 15151 | MU-199 | 1 | Spring | 2018 |
| 22222 | Einstein | Physics | 95000 | 22222 | PHY-101 | 1 | Fall | 2017 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

# Join Operation

- The Cartesian-Product

  $$instructor \times teaches$$

  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:

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

  - We get only those tuples of "*instructor* X *teaches*" that pertain to instructors and the courses that they taught.
- The result of this expression, shown in the next slide

# Join Operation (Cont.)

- The table corresponding to:

$$\Theta_{instructor.id = teaches.id}(instructor \times teaches))$$

| *instructor.ID* | *name* | *dept_name* | *salary* | *teaches.ID* | *course_id* | *sec_id* | *semester* | *year* |
|---|---|---|---|---|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-101 | 1 | Fall | 2017 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-315 | 1 | Spring | 2018 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-347 | 1 | Fall | 2017 |
| 12121 | Wu | Finance | 90000 | 12121 | FIN-201 | 1 | Spring | 2018 |
| 15151 | Mozart | Music | 40000 | 15151 | MU-199 | 1 | Spring | 2018 |
| 22222 | Einstein | Physics | 95000 | 22222 | PHY-101 | 1 | Fall | 2017 |
| 32343 | El Said | History | 60000 | 32343 | HIS-351 | 1 | Spring | 2018 |
| 45565 | Katz | Comp. Sci. | 75000 | 45565 | CS-101 | 1 | Spring | 2018 |
| 45565 | Katz | Comp. Sci. | 75000 | 45565 | CS-319 | 1 | Spring | 2018 |
| 76766 | Crick | Biology | 72000 | 76766 | BIO-101 | 1 | Summer | 2017 |
| 76766 | Crick | Biology | 72000 | 76766 | BIO-301 | 1 | Summer | 2018 |
| 83821 | Brandt | Comp. Sci. | 92000 | 83821 | CS-190 | 1 | Spring | 2017 |
| 83821 | Brandt | Comp. Sci. | 92000 | 83821 | CS-190 | 2 | Spring | 2017 |
| 83821 | Brandt | Comp. Sci. | 92000 | 83821 | CS-319 | 2 | Spring | 2018 |
| 98345 | Kim | Elec. Eng. | 80000 | 98345 | EE-181 | 1 | Spring | 2017 |

# 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:

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

- Thus

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

- Can equivalently be written as

$$instructor \bowtie_{Instructor.id = teaches.id} teaches.$$

more generalized

instructor $\bowtie_{Instructor.id \neq teaches.id}$ teachers.

or instructor ... > ... teachers.

# Union Operation

- The union operation allows us to combine two relations
- Notation: $r \cup s$
- 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 2017 semester, or in the Spring 2018 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2017}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2018}(section))$$

# Union Operation (Cont.)

- Result of:

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2017}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2018}(section))$$

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

# 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
- Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2017}(section)) \cap \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2018}(section))$$

- Result

| *course_id* |
| --- |
| CS-101 |

# Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation $r - s$ 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
- Example: to find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2017}(section)) - \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2018}(section))$$

for each element in A, check whether in B and remains if not in B.

| *course_id* |
| --- |
| CS-347 |
| PHY-101 |

CS-101 both in A and B if A−B, CS-101 still eliminated.

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

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

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

  $Physics \cup Music$

- 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 the result of the query.

# The Rename Operation

- The results of relational-algebra expressions do not have a name that we can use to refer to them. The rename operator, $\rho$ is provided for that purpose
- The expression:

  $\rho_x(E)$

  returns the result of expression $E$ under the name $x$
- Another form of the rename operation:

  $\rho_{x(A1,A2,..An)}(E)$

new name

table's new name

attributes' new name.

if $\rho$ name,

instructor is set'n the name of the transtong tables, e.g.,

$R = \text{Instructor}(ID, name, Dept\text{-}name, Salary)$

$\rho_{Ins(ID, iname, Dept\text{-}name, salary)}(R)$

# Equivalent Queries

- There is more than one way to write a query in relational algebra.
- Example: Find information about courses taught by instructors in the Physics department with salary greater than 90,000
- Query 1

  $\sigma_{dept\_name=\text{“Physics”}} (\sigma_{salary > 90{,}000} (instructor))$ better in terms of complexity

- Query 2

  $\sigma_{dept\_name=\text{“Physics”}} (\sigma_{salary > 90.000} (instructor))$ main memory, second memory enter main memory and hard disk twice.

- The two queries are not identical; they are, however, equivalent -- they give the same result on any database.

for 2 DBMS: queries, will chose the first one

# Equivalent Queries

Not User But DBMS will evaluate which query is computationally better.

- There is more than one way to write a query in relational algebra.
- Example: Find information about courses taught by instructors in the Physics department
- Query 1

$$\sigma_{dept\_name=\text{“Physics”}}(instructor \bowtie_{instructor.ID = teaches.ID} teaches)$$

big table as join is very expensive

- Query 2

$$(\sigma_{dept\_name=\text{“Physics”}}(instructor)) \bowtie_{instructor.ID = teaches.ID} teaches$$

better first reduce no. of rows of the huge table.

- The two queries are not identical; they are, however, equivalent -- they give the same result on any database.

# End of Chapter 2