L03: Relational Data Model and Relational Algebra

DSAN-6300/PPOL-6810: Relational Databases and SQL Programming

Irina Vayndiner

September 11 & 14, 2023

GEORGETOWN UNIVERSITY

Agenda for today's class

- Logistics
  - Q1 is published, due Tue, 9/19
  - HW1 will be published this Thu 9/14, due Tue 9/26
  - Mon class moved from 9/25 to 9/28
    - Class on 9/28 at 9:30am on zoom will be recorded
    - Also can join at 12:30pm same day on zoom
  - Do not forget to submit your Labs before deadline on Fri (for Mon), Tue (for Thu)
- Reminders:
  - Use Discussion Boards
  - Attend OHs
- Today:
  - Lecture: Relational Data Model and Relational Algebra
  - Lab

MySQL Workbench/AWS Setup – by Fri 9/22

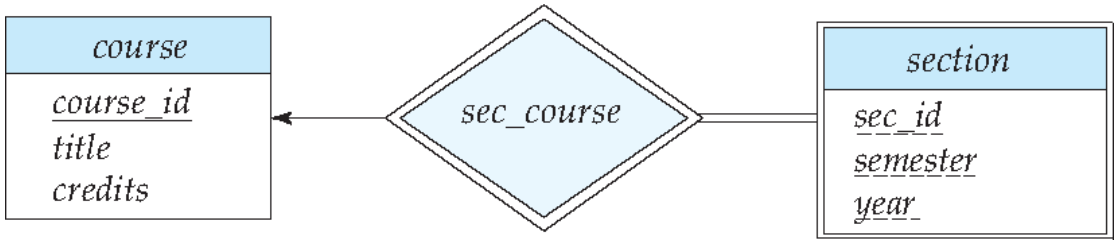
- If you want help with the setup, attend OHs week of 9/18
  - Detailed instructions will be published by 9/17
  - OH for more difficult setup issues: to be setup on the week of 9/18
- We will be using the setup on 9/28 in the Lab

Outline for today's Lecture

- Last topic of E-R Diagrams
  - Weak entity sets
- Relational Data Model and Relational Algebra
  - Structure of Relational Databases
  - Database Schema
  - Keys
  - Schema Diagrams
  - Relational Query Languages
  - The Relational Algebra
  - Translating E-R Diagrams to Relational Schemas

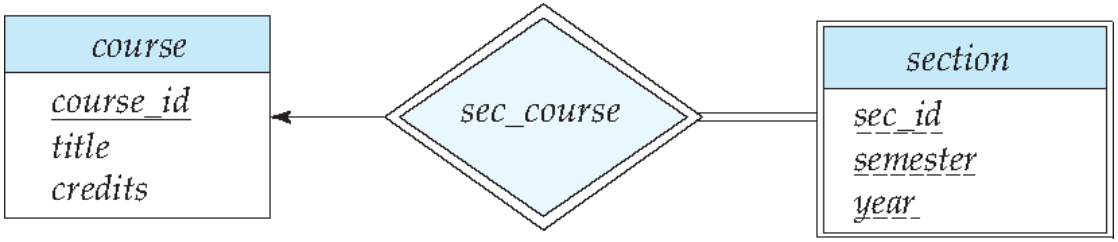
Weak Entity Sets

- Using University Database
  - Consider a situation when a course has several sections each semester. The *section* entity has attributes *sec_id* (with values 01,02, etc.), *year*, and *semester.*
  - In this case the entity set *section* then does not have enough attributes to identify a particular *section* entity uniquely (Why?)
  - To deal with this problem, we treat the relationship *sec_course* as a special relationship that provides extra information, in this case, the *course_id*, is required to identify *section* entities uniquely.



Weak Entity Sets (continued)

- A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity. Q: So what is discriminator here?
- An entity set that is not a weak entity set is called a **strong entity set**.



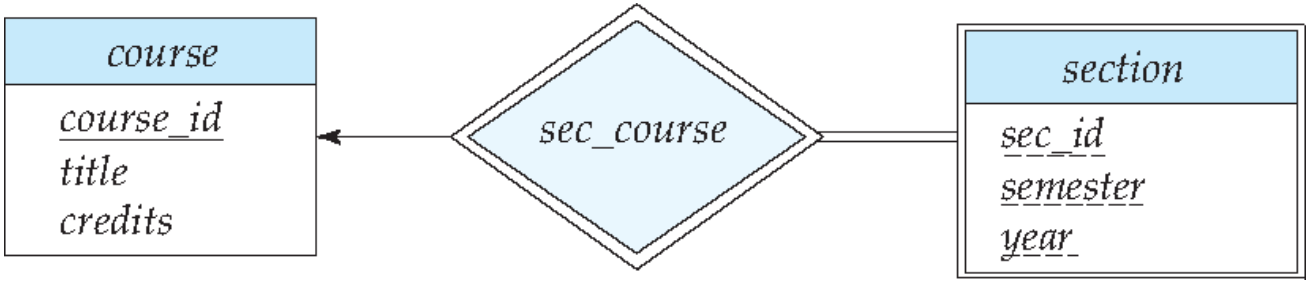
identifying entity

weak entity set

sec_id, semester, year

Expressing Weak Entity Sets in E-R Diagrams

- In E-R diagrams, a weak entity set is depicted via a *double rectangle*.
- We underline the discriminator of a weak entity set with a *dashed line*.
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a *double diamond*.



Today's Lecture: Relational Model

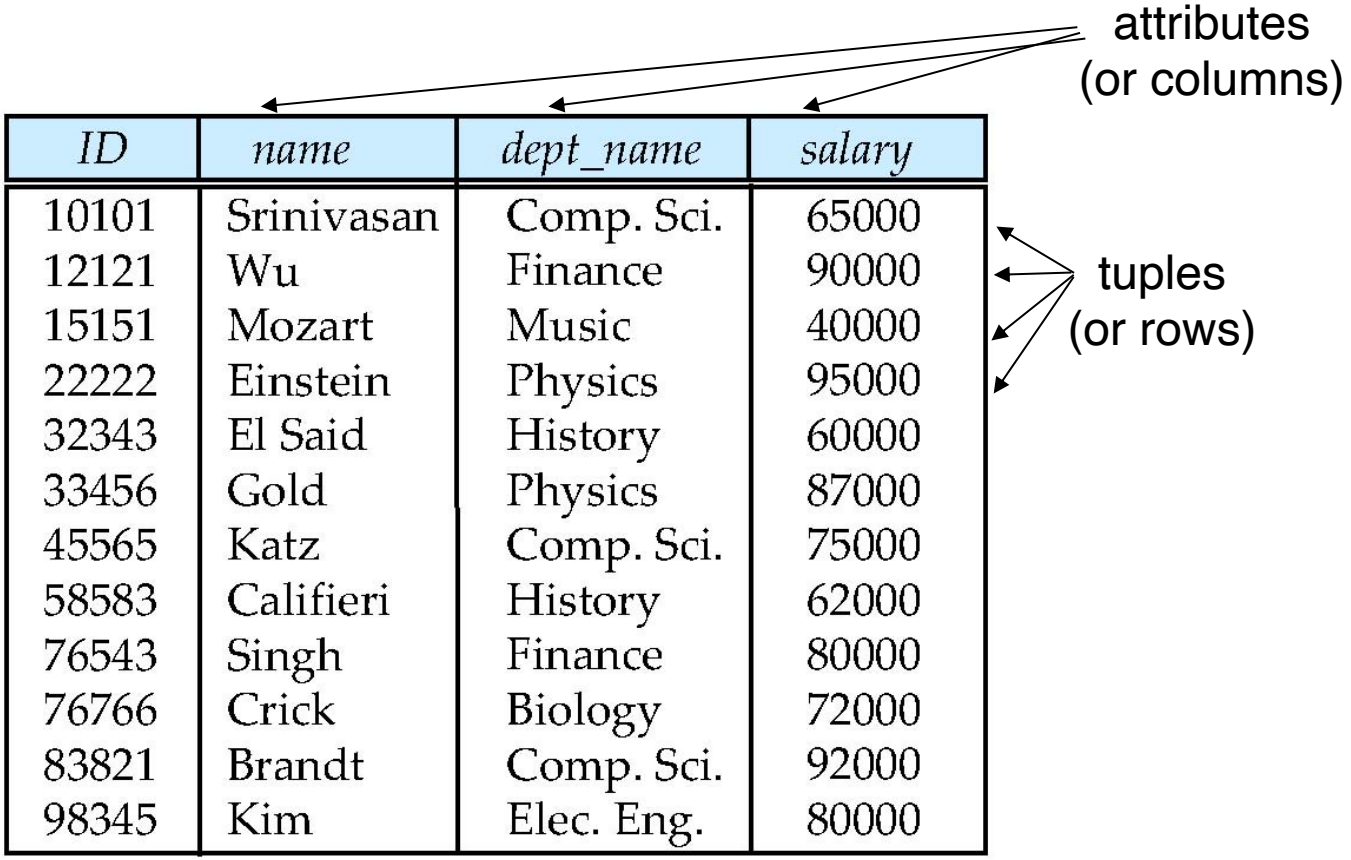
- The Relational Model is a data model that is based on rigorous set theory, e.g. "entity" and "relationships" sets

- The Relational Model is the most widely implemented data model for databases today
  - Traditional Vendors: IBM DB2, Oracle, ...
  - Open source: MySQL, PostgreSQL, Greenplum Database,…
  - Parallel: AWS Redshift, Teradata, Vertica, …

Relational Model (continued)

- **Data Structures**: Relations (or Tables)
- **Integrity Constraints**: Key uniqueness, referential integrity, etc.
- **Operations**: Relational Algebra, SQL.

Data Structure - Relations



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

- # of attributes = **degree, or an arity**

  degree is fixed by the schema
- #of rows = **cardinality**

  the cardinality changes as rows are inserted or deleted.

Data Structures - Relations (continued)

- Relation has two parts: **Schema** and **Instance**
- **Schema** – is the logical structure of the Relation (attributes)
- **Instance** -- is a snapshot of the data at a given moment of time
- Example:
  - Schema: *instructor* (*ID, name, dept_name, salary*)
  - Instance:

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

Example: University Database Schema

---

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

Instance - Tuple Notation

An instance is a set of **tuples** of the relation

A tuple: $< a_1:d_1 , \ldots a_n:d_k >$, where

- a_j is an attribute name,
- d_j is the value of the attribute a_j,
- d_j belongs to Domain(a_j)

for a relation having degree K, its instance consists of “K-tuples”

Example: *instructor* tuples

<ID:633909767,name:Richard Boon, dept_name:Physics, salary:75689>

<ID:674627883,name:Adolfo Laurenti, dept_name:Math, salary:67890>

<ID:193838904,name:Will Smith, dept_name:Math, salary:50000>

Domains and Null value

- The set of allowed values for each attribute is called the **domain** of the attribute
- The special value **NULL** can be a member of any domain
  - Indicates that the value is “unknown” or does not exit
  - The null value causes complications in the definition of many operations

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

Relational Database

- Relational database: a set of relations
- Relational database schema: a collection of schemas for all relations in the database
- Relational database instance: a collection of relation instances
  - One for each relation in the schema
  - Provides the current state of the database

The Relational Model

- Data Structures: Relations (or Tables)
- **Integrity Constraints: key uniqueness, referential integrity, etc.**
- Operations: Relational Algebra, SQL.

Integrity Constraints

- **Integrity Constraint (IC):** a condition that must be true for ***any*** instance of the database
  - ICs are specified when schema is defined, and restrict possible values in database
  - ICs are checked and enforced in real time when relations are modified
  - There are several types of ICs
  - Help to avoids data entry errors, too!
- A legal instance of a relation satisfies all specified ICs
- **Domain** constraints
  - In a legal instance, each tuple has values for its attributes ***drawn from the domains*** of those attributes
  - Specified by a logical expression (ex. “age >0”)

Primary Key Constraints

- Similar to entity sets' definition:
  - A set of attributes is a **candidate key** or a **key** of a relation if
    1. **Unique**: No two distinct tuples can have same values for all key attributes, and
    2. **Minimal**: uniqueness does not hold for any *subset* of that key
    - Ex. instructor(ID, name, dept_name, email, …)
      - {ID}, {email}, possibly {name, dept_name}
  - A set of attributes is a **superkey** of a relation if no two distinct tuples can have same values for all key attributes, but the set is not necessarily minimal
    - Ex. instructor(ID, name, dept_name, email, …) - {ID, email}
  - Any key is a superkey, but not the other way around
  - There may be many candidate keys, one of them is selected to be the **primary key**, it is underlined on schemas.
- **Primary key constraint:** in a legal instance, each tuple in a relation has unique values of the primary key

Foreign Key

- **Foreign key**
  - Set of fields in one relation that is used to 'refer' to a tuple in another relation. Must correspond to primary key of the second relation
  - Like a 'logical pointer'
- Example: Relation *enrolled* relates *students* to their course
  - *students(sid, name, login, age, gpa)*
  - *enrolled(sid, cid, grade,* **foreign key** *sid* **references** *students*)

enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Foreign Key

students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Primary Key

Referential Integrity

- **Referential Integrity** for foreign key constraint:
  - The domain of a foreign key is restricted to currently occurring values of the referenced primary key
  - Informally: no dangling references
- In the example, foreign key constraint ensures that
  - Only students listed in the *students* relation should be allowed to enroll for courses

enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Foreign Key

students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Primary Key

- Q: Can you think of a data model w/o referential integrity?

Foreign Keys and Referential Integrity

enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Foreign Key

students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Primary Key

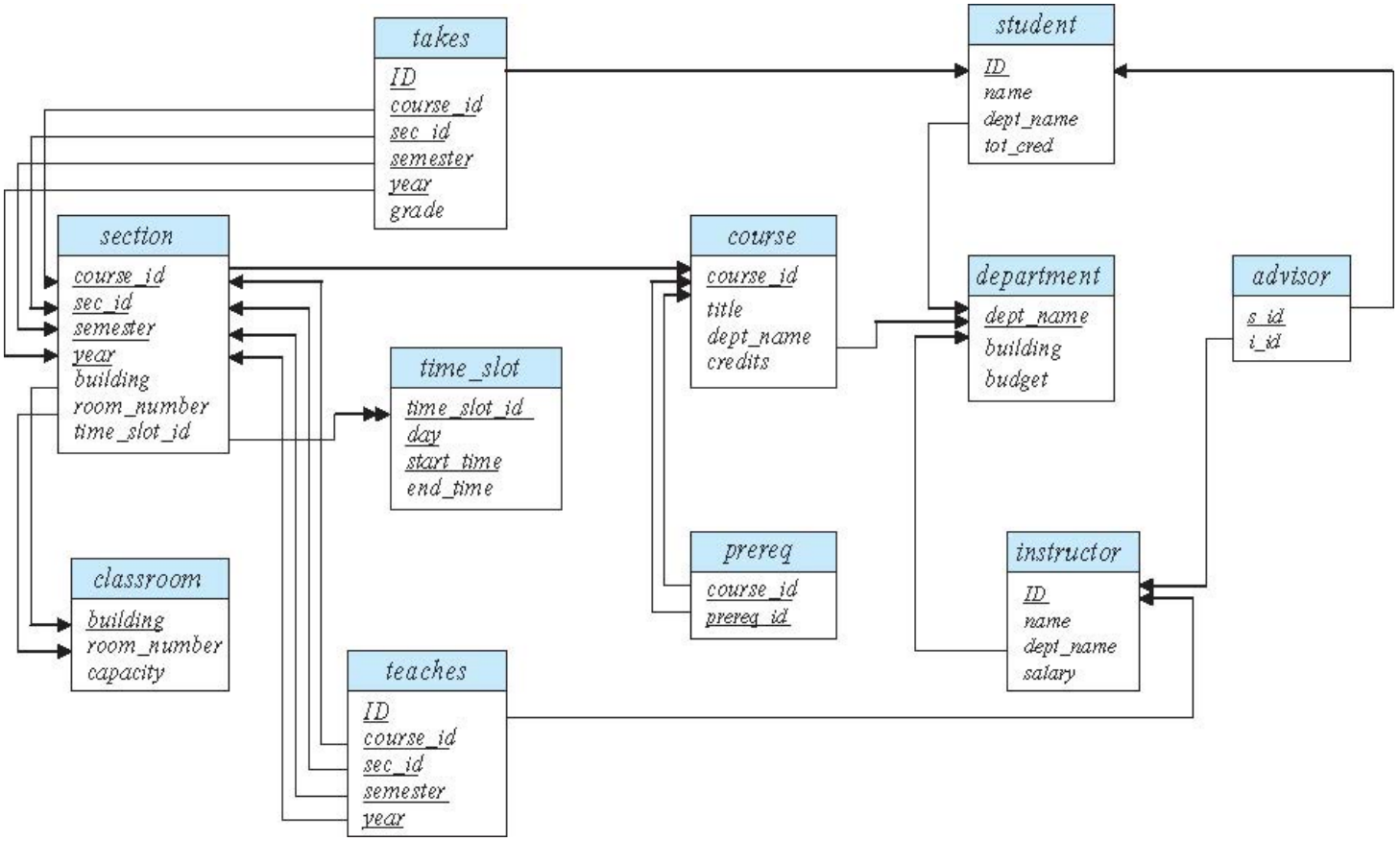
- Foreign Key Constraint
  - A Foreign Key must correspond to the Primary Key of the referenced relation
  - A Foreign Key constraint states a referential IC between two relations
    - A tuple in one relation that refers to another must refer to an *existing* tuple in that relation.
- The more **general** case, a **referential-integrity constraint**, relaxes the requirement that the referenced attributes is a primary key
- Foreign key and referential integrity constraints are used to maintain the consistency among tuples of relations

Enforcing Referential Integrity

- DBMS must dynamically enforce referential integrity by monitoring (and acting on) two types of operations:
  - Insertion/updating of a foreign key value
    - In our example: an *enrolled* tuple with a non-existent sid cannot be inserted
  - Deletion/updating of a primary key row
    - In our example: what should be done if a *students* tuple is deleted? A few options
      - (1) Also delete all *enrolled* tuples that refer to it.
      - (2) Disallow deletion of a *students* tuple that is referred to.
      - (3) Set sid in *enrolled* tuples that refer to it to a *default sid*.

Schema Diagram for University Database

- Schema diagram - a database schema, along with primary key and foreign-key constraints
- Primary-key attributes are underlined
- Foreign-key constraints appear as arrows from the foreign-key attributes of the referencing relation to the primary key of the referenced relation



The Relational Model

- **Data Structures: Relations (or Tables)**
- **Integrity Constraints: key uniqueness, referential integrity**
- **Operations: Relational Algebra, SQL**

Relational Query Languages

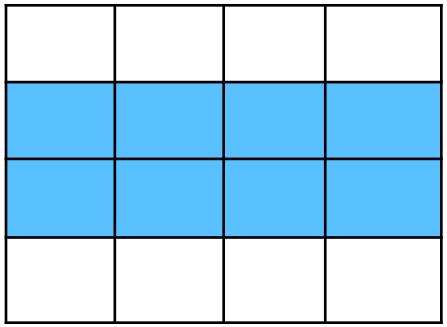
- **Query language** is a programming language in which a user requests information from the database.
- Programming languages can be categorized as
  - **Imperative language** - the user instructs the system to perform a specific sequence of operations
  - **Declarative language** - the user describes the desired information without giving a specific sequence of steps or function
- We will concentrate on **relational algebra language**
  - "Pure" (formal) query language
  - Declarative language
  - Not Turing-machine equivalent
  - Forms the theoretical basis of the SQL query language
    - Which is a Declarative language

Relational Algebra

- A functional language consisting of a set of operations that take one or two relations as input, and produce a new relation as their result.
- It forms a theoretical basis for SQL
- Six basic operations
  - select: σ
  - project: Π
  - Cartesian product: x
  - union: $\cup$
  - Set-difference: –
  - rename: ρ
- Operators can be composed, that is applied to the results of other operators.
  - That makes it a functional language

Select Operation

- The **select** operation selects tuples that satisfy a given selection condition.



- Notation: $\sigma_p(r)$
  - r - relation name
  - p - selection predicate (selection condition)
- Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.
  - Query

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

- Schema of the result is identical to schema of input relation

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

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

Select Operation (Continued)

- 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:

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

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

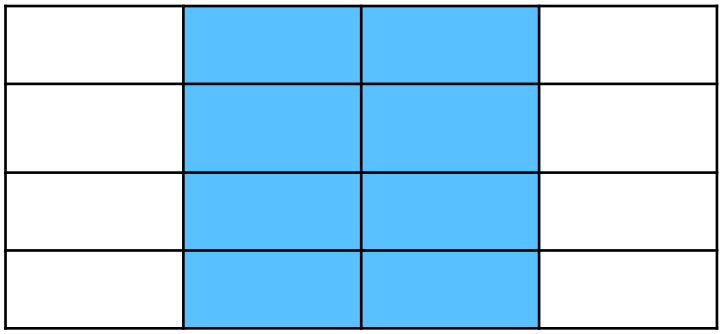
Project Operation

- An operation that returns its argument, with certain attributes left out.
- Notation:

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

where $A_1, \ldots, A_k$ are attribute names and r is a relation name.

- The result is defined as the relation of k attributes obtained by erasing the attributes 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*	*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 a 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.

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

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

name
Einstein
Gold

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* **degree** (or arity) – the same number of attributes
  2. The attribute domains must be **compatible**
     - **E**xample: 2nd column of *r* is compatible if it deals with the same type of values as does the 2nd column of *s*
- Example: to find ids of all courses taught in either the Fall 2017 semester, or in the Spring 2018 semester, or in both

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

$$\cup$$

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

Union Operation (Continued)

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

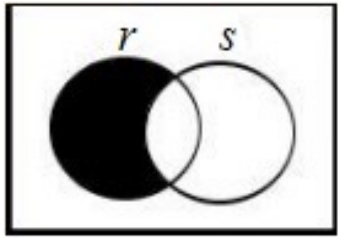
- Result of:

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

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

Set-Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation $r - s$



- 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

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

course_id
CS-347
PHY-101

Set-Intersection Operation

- The set-intersection operation allows us to find tuples that are in both the input relations
  - Not a "basic" operator
  - Can be expressed via union and set-difference
- Notation: $r \cap s$
- Assumes:
  - r, s have the *same* **degree** (or arity)
  - domains of r and s are compatible (same as for union)
- 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

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 *teaches* is written as:

  instructor X *teaches*

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

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

instructor

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

teaches

The *instructor* x *teaches* table

The relational algebra concatenates t_1 and t_2 into a single tuple, as shown.

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

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

Join Operation

- Not a “basic” operator
  - Can be expressed via **X** and σ
- 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:

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

  - By adding select operation, we get only those tuples of “*instructor* X *teaches”* that pertain to instructors and the courses that they taught.

Join Operation (Continued)

- The join operation allows us to combine a selection and a Cartesian product into a single operation.
- Consider relations r and s, and let θ be a predicate (a logical expression). The join operation "⋈" is defined as follows:

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

- Thus,

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

can equivalently be written as

$$\text{instructor} \bowtie_{\text{instructor.ID = teaches.iD}} \text{teaches}$$

- The result of this expression is shown on the next slide

Join Operation (Continued)

- The result of

  instructor $\bowtie_{\text{instructor.ID} = \text{teaches.iD}}$ 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 – Special Cases

- **Equi-Join**: A special case of condition join where the condition contains only *equalities*

$$\text{instructor} \bowtie_{\text{instructor.ID = teaches.iD}} \text{teaches}$$

Notation can be simplified

$$\text{instructor} \bowtie_{\text{ID}} \text{teaches}$$

- **Natural Join**: Equijoin on *all* common fields.

$$\text{instructor} \bowtie \text{teaches}$$

The Assignment Operation

- It is *convenient* at times to write a relational-algebra expression by assigning parts of it to temporary (intermediate) relation variables.
- The assignment operation is denoted by ← and works like assignment in a programming language.
- Example: Find all instructors in the “Physics” and “Music” departments.

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

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, ρ , is provided for that purpose
  - As a synonym or an alias
- The expression:

  $\rho_x (E)$ or $\rho(x, E)$

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

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

  that gives a name to the result of expression E and simultaneously rename the attributes
- Example:

  $\rho_{professor(ID, last\_name, department , salary)}$ *instructor* (*ID, name, dept_name, salary*)

  Defines a new relation

  professor(ID, last_name, department , salary)

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

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

- Query 2

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

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

Lab

To submit:
Word doc with query or queries for #4 only