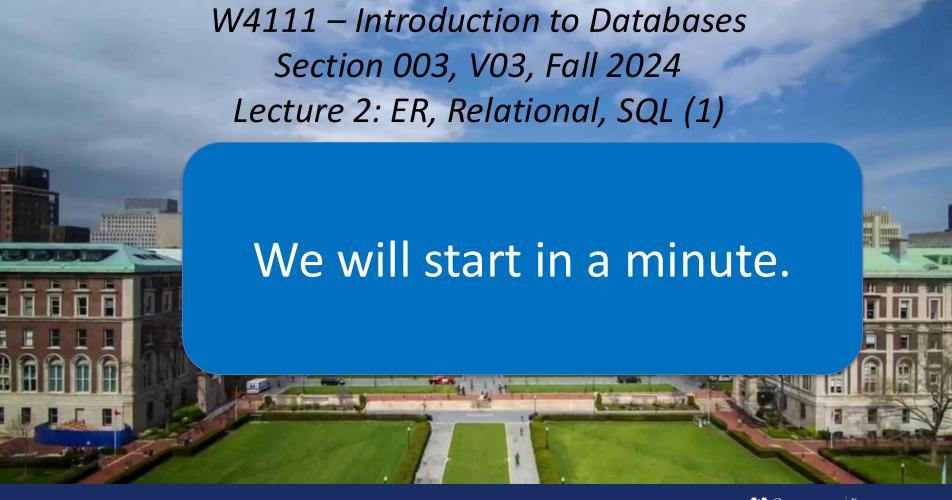
W4111 – Introduction to Databases
Section 003, V03, Fall 2024
Lecture 2: ER, Relational, SQL (1)





# Today's Contents

### Contents

• Introduction and course updates.

• ER Modeling (1).

Relational model and algebra (1).

• SQL (1).

Homework and projects.

Introduction and course updates

## Course Updates

- TAs
  - We should have a complete set of TAs by the weekend.
  - I will be meeting with them to setup GradeScope and submission for HW 0.
- Grading
  - Grading will be on a scale of 0 to 100 points.
  - Exams
    - In class midterm is 25 points (2024-OCT-18). Probably not full lecture time.
    - Final exam is in classroom on scheduled date and is 50 points.
  - Homework
    - There will be 5 homework assignments, each worth 5 points.
    - These will be a combination of written questions and code/practical tasks.
    - After the 5 homework assignments, you will have completed a small project.



### Final Exam Schedule

https://registrar.columbia.edu/content/student-exams

#### **Student Exams**

Final examinations are given at the end of each term. The Projected University Examination Schedule provides a tentative guide to final examinations and is available **online**.

Projected and Final Exam dates may include Religious Holidays for some students. NYS Education Law § 224-A mandates that faculty make available an opportunity to make up any examination missed because of religious beliefs. To facilitate the preparation of makeup exams, students intending to be absent to observe any of these holidays must notify the instructor by October 15 in the fall semester and February 15 in the spring semester.

The definitive schedule of final examinations is usually available in early to mid November for the fall term and early to mid April for the spring term. Exams are scheduled according to a University-wide <a href="#">Final Exam Schedule</a> available shortly after midterms. Prior to its availability, students and faculty should consult the <a href="#">Projected Exam Schedule</a>. Students may access their individual exam schedule in Vergil by clicking on "Schedule," then "My Exam Schedule."

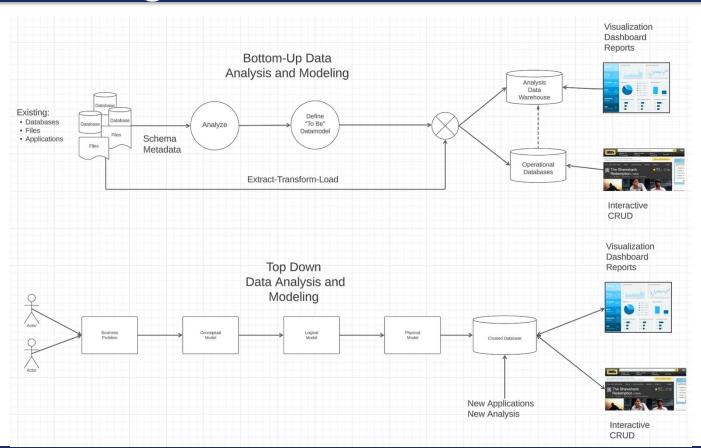
Students who have trouble locating a class on either list should contact the instructor to make sure the class has an exam.



# **ER Modeling**

# Concepts

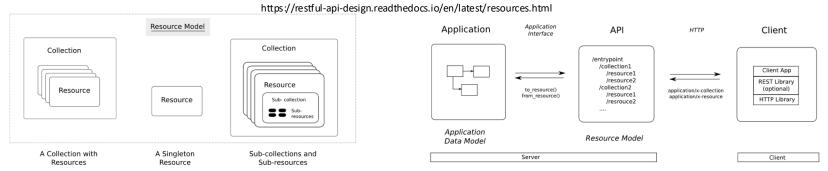
# Modeling



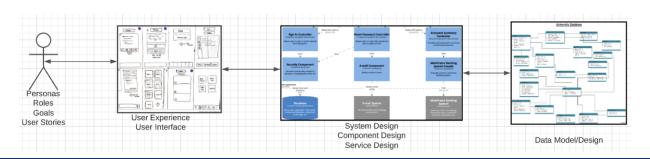
Most of the time there is a mix → Meet-in-the-Middle

## Web Application Problem Statement

- We must build a system that supports create, retrieve, update and delete for IMDB and Game of Thrones Datasets.
- This requires implementing *create, retrieve, update and delete (CRUD)* for resources.



- We will design, develop, test and deploy the system iteratively and continuously.
- There are four core domains.

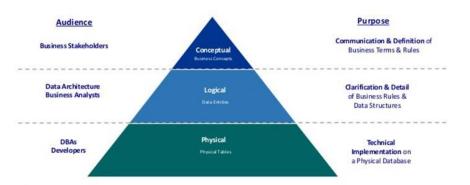


- In this course,
- We focus on the data dimension.
- We will get some insight into the other dimensions.

### A Common and my Approach: Conceptual → Logical → Physical

https://ehikioya.com/conceptual-logical-physical-database-modeling/

#### **Levels of Data Modeling**



- It is easy to get carried away with modeling.
   You can spend all your time modeling and not actually build the schema.
- We will use the approaches in class.
- Mostly to understand concepts and patterns.

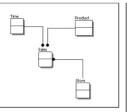
https://www.1keydata.com/datawarehousing/data-modeling-levels.html

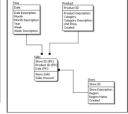
Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

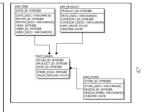
Conceptual Model Design

Logical Model Design

Physical Model Design



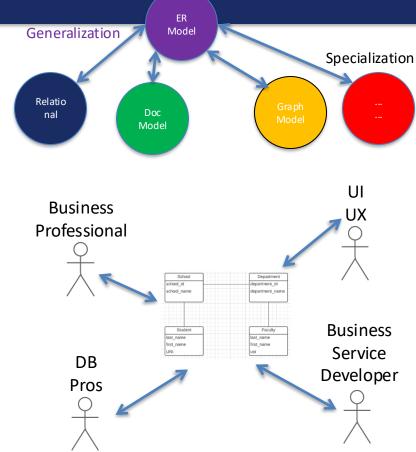




https://www.1keydata.com/datawarehousing/data-modeling-levels.html

# ER Model and ER Modeling

- ER Model: Agility, Separation of Concerns
  - ER model is a generalization that most DB models implement in some form.
  - Using the ER model enables:
    - Thinking about and collaborating on design with getting bogged down in details.
    - Enable flexible choices about how to realize/Implement data.
- ER Diagrams: Communication, Quality, Precision
  - With a little experience, everyone can understand and ER diagram.
  - Easier to discuss and collaborate on application's data than showing SQL table definitions, JSON, ... ...
  - People think visually. That is why we have whiteboards. ER diagrams are precise and unambiguous.
  - Guides you to think about relationships, keys, ... And prevents "redos" later in the process. It is easier to fix a diagram than a database schema.



# ER Modeling – Reasonably Good Summary

#### **Advantages of ER Model**

**Conceptually it is very simple:** ER model is very simple because if we know relationship between entities and attributes, then we can easily draw an ER diagram.

**Better visual representation:** ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.

**Effective communication tool:** It is an effective communication tool for database designer.

**Highly integrated with relational model:** ER model can be easily converted into relational model by simply converting ER model into tables.

**Easy conversion to any data model:** ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

#### Disadvantages of ER Model

Limited constraints and specification

Loss of information content: Some information be lost or hidden in ER model

**Limited relationship representation:** ER model represents limited relationship as compared to another data models like relational model etc.

**No representation of data manipulation:** It is difficult to show data manipulation in ER model.

**Popular for high level design**: ER model is very popular for designing high level design

No industry standard for notation

https://pctechnicalpro.blogspot.com/2017/04/advantages-disadvantages-er-model-dbms.html

#### Note:

- If you get to use Google to help with take home exams, HW, etc.
- I get to use Google to help with slides.



# Entity-Relational Model Reminder



### **Entity Sets**

- An entity is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- An entity set is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays
- An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.
  - Example:

```
instructor = (ID, name, salary )
course= (course_id, title, credits)
```

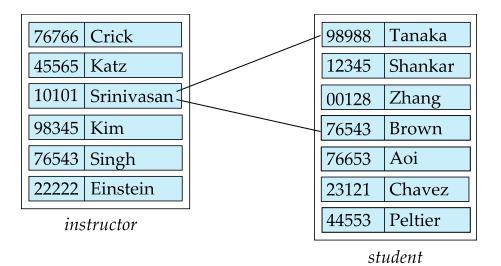
 A subset of the attributes form a primary key of the entity set; i.e., uniquely identifying each member of the set.

#### **DFF Comments:**

- Some of these statements apply primarily to OO systems and the relational/SQL models.
- A motivation for "No SQL" is to relax the constraints.



### **Entity Sets -- instructor and student**



(10101, 98988) (10101,76543)



### **Relationship Sets**

A relationship is an association among several entities

Example:

44553 (Peltier) <u>advisor</u> 22222 (<u>Einstein</u>) student entity relationship set instructor entity

• A **relationship set** is a mathematical relation among  $n \ge 2$  entities, each taken from entity sets

$$\{(e_1, e_2, \dots e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

where  $(e_1, e_2, ..., e_n)$  is a relationship

Example:

$$(44553,22222) \in advisor$$

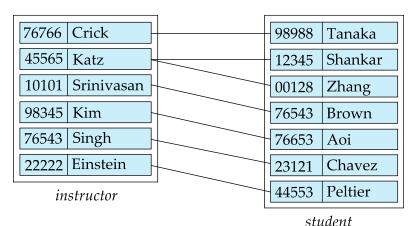
#### **DFF Comments:**

- Nobody thinks about relationships this way.
- There is no idea so simple that a DB professor cannot make it confusing, usually by using math.



### **Relationship Sets (Cont.)**

- Example: we define the relationship set advisor to denote the associations between students and the instructors who act as their advisors.
- Pictorially, we draw a line between related entities.



#### **DFF Comment:**

- In this diagram, the lines are the relationship set.
- Many DB models use a 3<sup>rd</sup> entity set to represent complex relationships.

#### **DFF Comments:**

- Nobody draws the diagrams this way, but ...
- Sometimes thinking this way helps understand other ways to depict the concept.

# **Visual Notation**



### Representing Entity sets in ER Diagram

- Entity sets can be represented graphically as follows:
  - Rectangles represent entity sets.
  - Attributes listed inside entity rectangle
  - Underline indicates primary key attributes

instructor

<u>ID</u>
name
salary

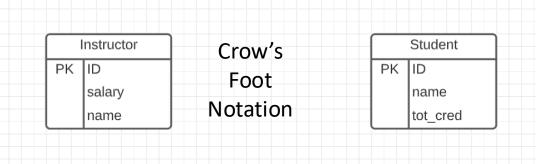
student

ID

name

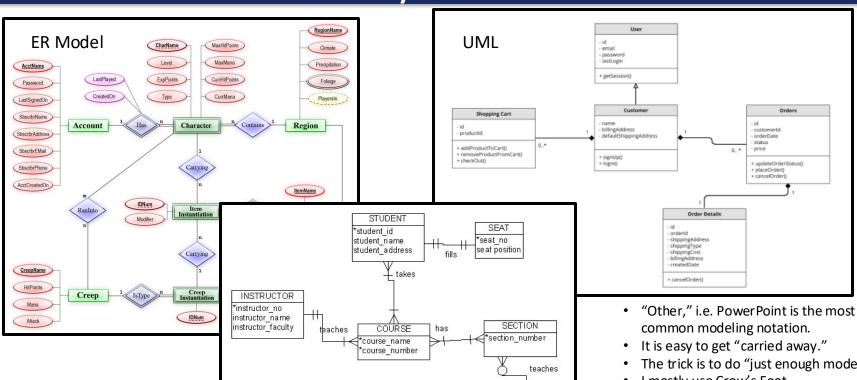
tot\_cred

"Visual modeling is the use of semantically rich, graphical and textual design notations to capture software designs. A notation, such as UML, allows the level of abstraction to be raised, while maintaining rigorous syntax and semantics. In this way, it improves communication in the design team, as the design is formed and reviewed, allowing the reader to reason about the design, and it provides an unambiguous basis for implementation."



# Visual Notation – Many Notations

Crow's Foot



PROFESSOR

professor faculty

professor id professor name

- The trick is to do "just enough modeling."
- I mostly use Crow's Foot
  - It is "just enough"
  - But lacks some capabilities.
- The book uses ER notation.

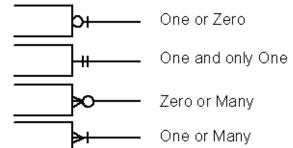
# Notation has Precise Meaning (Crow's Foot)

- Attribute annotations:
  - PK = Primary Key
  - FK = Foreign Key

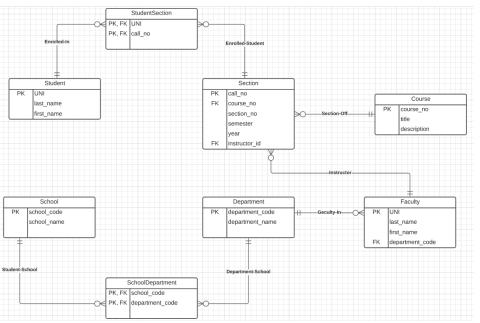
- We will spend a lot of time discussing keys.
- We will start in a couple of slides.

Line annotations:

#### Summary of Crow's Foot Notation



We will learn over time and there are good tutorials (https://www.lucidchart.com/pages/er-diagrams) to help study and refresh.



## What Does this Mean? Let's Get Started

Primary Key means that the value occurs at most once.

School Code	School Name	
СС	Columbia College	
SEAS	Fu Foundation School of Engineering and Applied Science	
GSAS	Graduate School of Arts and Sciences	
GS	General Studies	

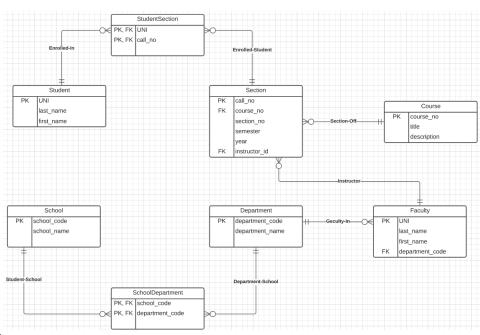
Foreign Key means that if a value occurs in school\_id for any row, there must be a row in School with that key.

UNI	Last name	First name	school_id
dff9	Ferguson	Donald	CC
js11	Smith	John	GS
jp9	Public	James	СС
bb101	Baggins	Bilbo	СС

#### The line notations mean:

- A student is related to EXACTLY ONE school.
- A School may be related to 0, 1 or many students.

Student.school\_id references school.school\_code



# Worked Example

We will do a worked example in LucidChart

# Relational Model



#### **Relational Model**

Rows

- All the data is stored in various tables.
- Example of tabular data in the relational model

		Columns
		//
	<u> </u>	

ID	name	dept_name	salary	
22222	Einstein	Physics	95000	١.
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



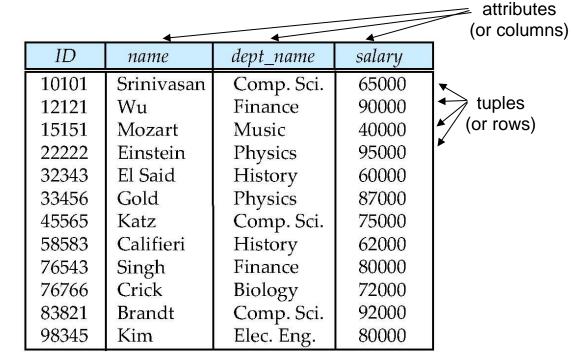
**Ted Codd**Turing Award 1981

- The "relation" is the "table."
  - In my big space of pieces of data.
     ID, name, dept\_name, salary are somehow related.
  - This causes confusion, because the ER and other models use "relation" to mean something else.
- Core concepts:
  - Relation
  - Tuple (Row)
  - Column (Attribute)



### Example of a *Instructor* Relation

00123 123



Integer -213 (integers, >0)



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

Ferguson, Donald

Ferguson Donald

**COMSW4111** 

#### **DFF Comments:**

- I will explain the importance of atomic attributes and null in examples.
- Atomic and use of Null is important!



### **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 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.
- Example:
  - schema: instructor (ID, name, dept\_name, salary)
  - Instance:

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

(<u>ID</u>, name, dept\_name, salary)



### **Keys**

- Let K ⊂ 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 instructor is a foreign key from instructor referencing department

### Notation

#### Classroom relation

building	room_number	capacity
Packard	101	500
Painter	100	125
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

#### classroom schema

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept\_name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined.



- The primary key is a *composite key*. Neither column is a key (unique) by itself.
- Keys are statements about all possible, valid tuples and not just the ones in the relation.
  - Capacity is unique in this specific data, but clearly not unique for all possible data.
  - In this domain, there cannot be two classrooms with the same building and room number.
- Relation schema:
  - Underline indicates a primary key column. There is no standard way to indicate other types of key.
  - We will use **bold** to indicate foreign keys.
  - You will sometimes see things like classroom(<u>building:string</u>, <u>room\_number:number</u>, capacity:number)

### Observations

#### Keys:

- Will be baffling. It takes time and experience to understand/appreciate.
- There are many, many types of keys with formal definitions.
- I explain the formality but focus on the concepts and applications.
- No one uses the formal, relational model. So, why do we study it?
  - Is very helpful when understanding concepts that we cover later in the course, especially query optimization and processing.
  - There are many realizations of the model and algebra, and understanding the foundation helps with understanding language/engine capabilities.
  - The model has helped with innovating new approaches, and you may innovate in data and query models in your future.



### **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 turning-machine equivalent
  - Consists of 6 basic operations

#### **DFF Comments:**

- You will sometimes see other operator, e.g. ← Assignment.
- Relational algebra focuses on retrieve. You can sort of do Create, Update, Delete.
- The SQL Language, which we will see, extends relational algebra.



### **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: σ
  - project: ∏
  - union: ∪
  - set difference: –
  - Cartesian product: x
  - rename:  $\rho$



## **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

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

Result

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:

$$\wedge$$
 (and),  $\vee$  (or),  $\neg$  (not)

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

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

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

 $\prod_{ID, name, salary}$  (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.

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

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



## **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 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

 $R \times R \rightarrow R$   $f(x,y) \rightarrow z$ 



### The instructor x teaches table

- This only sort of makes sense. The result is:
  - Every possible combination of the form
  - (instructor, teaches)
  - Even if the instructor is NOT the instructor in the teaches row.
- Examining in MySQL makes a little clearer.
  - Let's look in the lecture examples notebook.
  - Confusingly, in SQL
     Cartesian Product is JOIN

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
***								

# Simpler Example



Τ

s
5 rows

S



 $S \times T$ 



 $S \times T$ 

T.b	T.d
'a'	100
'd'	200
'f'	400
'g'	120

ገ -	S.b	S.d
5		
1	'a'	100
í	'b'	300
r	'c'	400
	'd'	200
	'e'	150

	S.b	S.d	T.b	T.d
	'a'	100	'a'	100
com	'a'	100	'd'	200
on	'a'	100	'f'	400
ssil	'a'	100	'g'	120
ern	'b'	300	'a'	100
CIII	'b'	300	'd'	200
	'b'	300	'f'	400
	'b'	300	'g'	120
	'c'	400	'a'	100
	'c'	400	'd'	200

	S.b	S.d	T.b	T.d
	'c'	400	'f'	400
pΙ	'c'	400	'g'	120
	'd'	200	'a'	100
	'd'	200	'd'	200
	'd'	200	'f'	400
	'd'	200	'g'	120
	'e'	150	'a'	100
	'e'	150	'd'	200
	'e'	150	'f'	400
	'e'	150	'g'	120



## **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 x 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

#### A fundamental definition:

- $\sigma_{\text{instructor,ID=teaches,ID}}$  (instructor × teaches) = instructor  $\bowtie$  teaches
- ⋈ is the JOIN operations.

## **JOIN Definition**



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



instructor ⋈ teaches

instructor.ID	instructor.name	instructor.dept_name	instructor.salary	teaches.ID	teaches.course_id	teaches.sec_id	teaches.semester	teaches.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
12121	'Wu'	'Finance'	90000	12121	'FIN-201'	1	'Spring'	2010
15151	'Mozart'	'Music'	40000	15151	'MU-199'	1	'Spring'	2010
22222	'Einstein'	'Physics'	95000	22222	'PHY-101'	1	'Fall'	2009
32343	'El Said'	'History'	60000	32343	'HIS-351'	1	'Spring'	2010
45565	'Katz'	'Comp. Sci.'	75000	45565	'CS-101'	1	'Spring'	2010
45565	'Katz'	'Comp. Sci.'	75000	45565	'CS-319'	1	'Spring'	2010
76766	'Crick'	'Biology'	72000	76766	'BIO-101'	1	'Summer'	2009

instructor.ID	instructor.name	instructor.dept_name	instructor.salary	teaches.course_id	teaches.sec_id	teaches.semester	teaches.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

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

instructor ⋈ teaches

instructor ⋈ instructor.ID>teaches.ID teaches





## 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  $r \bowtie_{\theta} s$  is defined as follows:

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

Thus

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

Can equivalently be written as

*instructor* ⋈ <sub>Instructor.id</sub> = teaches.id</sub> teaches.

## The Dreaded Relax Calculator

- Let's look at an online tool that you will use.
- RelaX (<a href="https://dbis-uibk.github.io/relax/calc/local/uibk/local/0">https://dbis-uibk.github.io/relax/calc/local/uibk/local/0</a>)
- The calculator:
  - Has an older version of the data from the recommended textbook.
     (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0)
  - You can also upload new data.
- Some queries:
  - σ dept\_name='Comp. Sci.' V dept\_name='History' (department)
  - π name, dept\_name (instructor)
  - π ID, name (σ dept\_name='Comp. Sci.' (instructor)

# SQL





## **History**

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.



### **SQL Parts**

- DML -- provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- integrity the DDL includes commands for specifying integrity constraints.
- View definition -- The DDL includes commands for defining views.
- Transaction control –includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and dynamic SQL -- define how SQL statements can be embedded within general-purpose programming languages.
- Authorization includes commands for specifying access rights to relations and views.

### DDL DML

## SQL Language Statements

### The core SQL language statements are:

- SELECT: Implements both σ, π
- INSERT
- UPDATE
- DELETE
- CREATE TABLE
- ALTER TABLE
- JOIN, which is an operator within SELECT.
- Many, if not most, SQL statements:
  - Implement multiple relational algebra expressions.
  - Cannot easily (or at all) be represented in relational algebra.

```
π ID, name (
σ dept_name='Comp. Sci.' (instructor)
)

=

SELECT ID, name FROM instructor
WHERE
dept_name='Comp. Sci.'
```



## **Basic Query Structure**

A typical SQL query has the form:

```
select A_1, A_2, ..., A_n from r_1, r_2, ..., r_m where P
```

- A<sub>i</sub> represents an attribute
- R<sub>i</sub> represents a relation
- P is a predicate.
- The result of an SQL query is a relation.

#### Note:

- The SELECT ... FROM ... WHERE ... Combines two relational operators,  $\sigma$  and  $\Pi$ .
- Actually, it also combines other operators, e.g. x



### The select Clause

- The select clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

select name from instructor

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., Name = NAME = name
  - Some people use upper case wherever we use bold font.



## The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword distinct after select.
- Find the department names of all instructors, and remove duplicates

**select distinct** *dept\_name* **from** *instructor* 

The keyword all specifies that duplicates should not be removed.

**select all** dept\_name **from** instructor



## The select Clause (Cont.)

An asterisk in the select clause denotes "all attributes"

select \*
from instructor

An attribute can be a literal with no from clause

select '437'

- Results is a table with one column and a single row with value "437"
- Can give the column a name using:

select '437' as FOO

An attribute can be a literal with from clause

select 'A'
from instructor

 Result is a table with one column and N rows (number of tuples in the instructors table), each row with value "A"



## The select Clause (Cont.)

- The select clause can contain arithmetic expressions involving the operation, +, -, \*, and /, and operating on constants or attributes of tuples.
  - The query:

**select** *ID, name, salary/12* **from** *instructor* 

would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

Can rename "salary/12" using the as clause:

select ID, name, salary/12 as monthly\_salary



#### The where Clause

- The where clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

select name
from instructor
where dept\_name = 'Comp. Sci.'

- SQL allows the use of the logical connectives and, or, and not
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <>.
- Comparisons can be applied to results of arithmetic expressions
- To find all instructors in Comp. Sci. dept with salary > 80000

select name
from instructor
where dept\_name = 'Comp. Sci.' and salary > 80000



#### The from Clause

- The from clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product instructor X teaches

select \*
from instructor, teaches

- generates every possible instructor teaches pair, with all attributes from both relations.
- For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).



## **Examples**

- Find the names of all instructors who have taught some course and the course\_id
  - select name, course\_id
     from instructor, teaches
     where instructor.ID = teaches.ID
- Find the names of all instructors in the Art department who have taught some course and the course\_id
  - select name, course\_id
     from instructor, teaches
     where instructor.ID = teaches.ID and instructor. dept\_name = 'Art'



## **Data Definition Language**

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.



#### **Create Table Construct**

An SQL relation is defined using the create table command:

#### create table r

```
(A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint<sub>1</sub>),
...,
(integrity-constraint<sub>k</sub>))
```

- r is the name of the relation
- each A<sub>i</sub> is an attribute name in the schema of relation r
- D<sub>i</sub> is the data type of values in the domain of attribute A<sub>i</sub>
- Example:

```
create table instructor (

ID char(5),

name varchar(20),

dept_name varchar(20),

salary numeric(8,2))
```



### The Rename Operation

• The SQL allows renaming relations and attributes using the as clause:

old-name as new-name

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
  - select distinct T.name
     from instructor as T, instructor as S
     where T.salary > S.salary and S.dept\_name = 'Comp. Sci.'
- Keyword as is optional and may be omitted instructor as T ≡ instructor T



#### **Null Values**

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
- null signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving null is null
  - Example: 5 + null returns null
- The predicate is null can be used to check for null values.
  - Example: Find all instructors whose salary is null.

select name from instructor where salary is null

 The predicate is not null succeeds if the value on which it is applied is not null.

#### Note:

- NULL is an extremely important concept.
- You will find it hard to understand for a while.



## **Null Values (Cont.)**

- SQL treats as unknown the result of any comparison involving a null value (other than predicates is null and is not null).
  - Example: 5 < null or null <> null or null = null
- The predicate in a where clause can involve Boolean operations (and, or, not); thus the definitions of the Boolean operations need to be extended to deal with the value unknown.
  - and : (true and unknown) = unknown, (false and unknown) = false, (unknown and unknown) = unknown
  - or: (unknown or true) = true,
     (unknown or false) = unknown
     (unknown or unknown) = unknown
- Result of where clause predicate is treated as false if it evaluates to unknown



### **Modification of the Database**

- Deletion of tuples from a given relation.
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation



#### **Deletion**

Delete all instructors

delete from instructor

- Delete all instructors from the Finance department delete from instructor where dept\_name= 'Finance';
- Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building.



## **Deletion (Cont.)**

 Delete all instructors whose salary is less than the average salary of instructors

- Problem: as we delete tuples from instructor, the average salary changes
- Solution used in SQL:
  - 1. First, compute **avg** (salary) and find all tuples to delete
  - 2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)



#### Insertion

Add a new tuple to course

```
insert into course
   values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

or equivalently

```
insert into course (course_id, title, dept_name, credits)
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

Add a new tuple to student with tot\_creds set to null

```
insert into student
  values ('3003', 'Green', 'Finance', null);
```



## **Insertion (Cont.)**

 Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of \$18,000.

```
insert into instructor
    select ID, name, dept_name, 18000
from student
    where dept_name = Music' and total_cred > 144;
```

 The select from where statement is evaluated fully before any of its results are inserted into the relation.

Otherwise queries like

insert into table1 select \* from table1

would cause problem



## **Updates**

Give a 5% salary raise to all instructors

```
update instructor
set salary = salary * 1.05
```

Give a 5% salary raise to those instructors who earn less than 70000 update instructor
 set salary = salary \* 1.05
 where salary < 70000;</li>

Give a 5% salary raise to instructors whose salary is less than average



## **Updates (Cont.)**

- Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others by a 5%
  - Write two update statements:

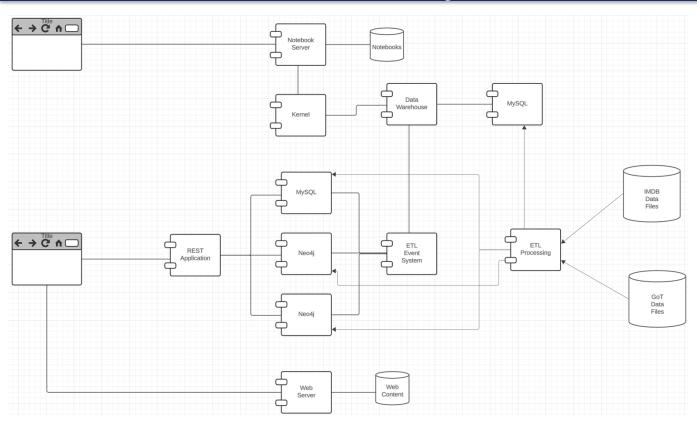
```
update instructor
  set salary = salary * 1.03
  where salary > 100000;
update instructor
  set salary = salary * 1.05
  where salary <= 100000;</pre>
```

- The order is important
- Can be done better using the case statement (next slide)

# Time for Some Hands On?

# Homework -> Incremental Project

# HWs → Incremental Project



Programming track will build a simple, interactive web application.

Non-programming track will build a simple data engineering and insight system.

The TAs and I will provide much of the infrastructure and environment starter/templates.

You will define small individual projects/scenarios and pick the data sets.

We will do worked examples.

# Wrap Up

