CS 61:
Database Systems

Joins

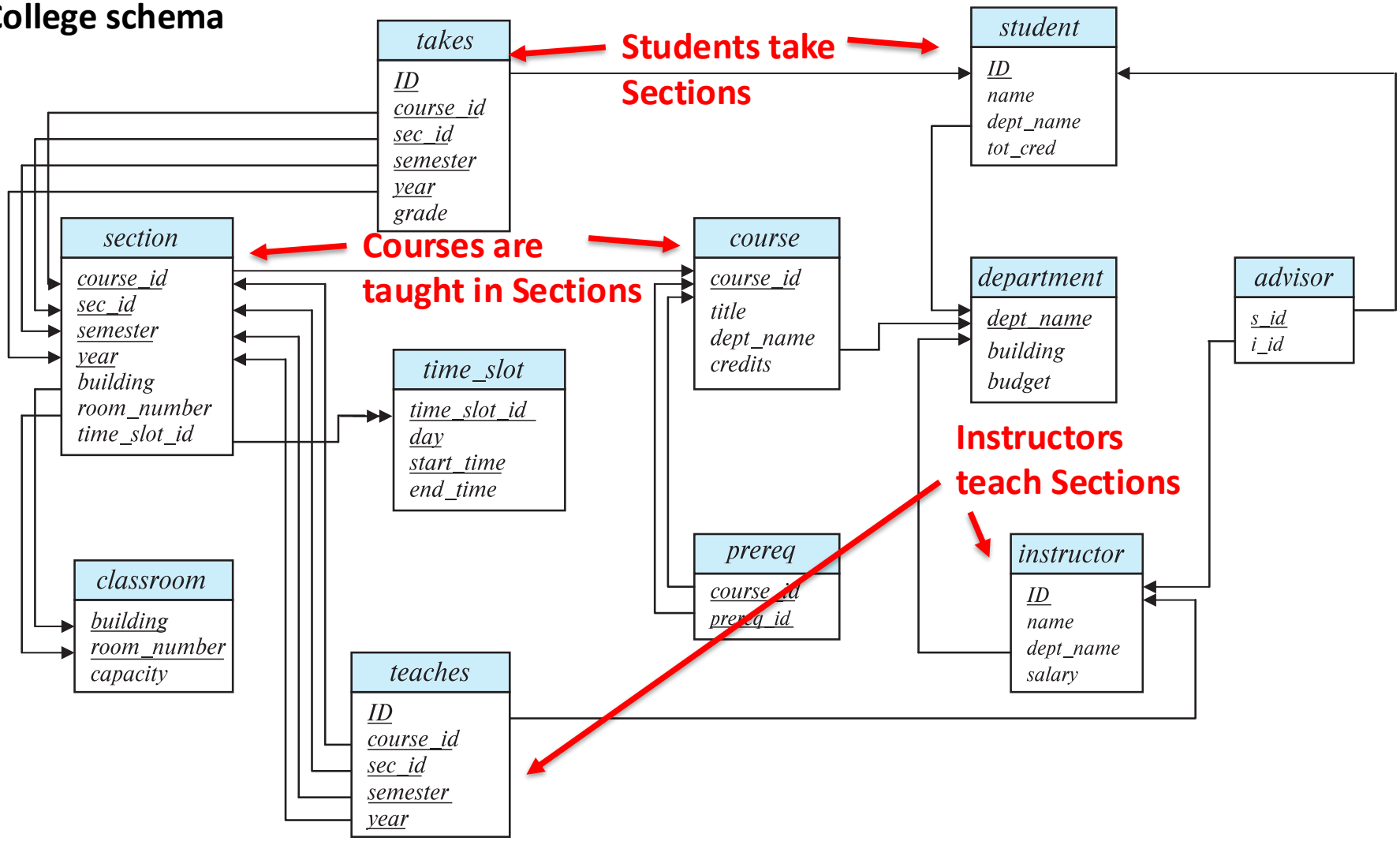
Adapted from Silberschatz, Korth, and Sundarshan unless otherwise noted
Augmented by Claude Opus

Agenda

1. Relational algebra part 2
2. Joins
3. Dates

College schema maintains information about college instructors, students, classes

College schema



I'll use the textbook's instructor and teaches tables

instructor 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

Taught by Srinivasan

teaches table

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-315	1	Spring	2018
45565	CS-101	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 table lists courses and sections that are taught by instructors

ID in *teaches* is a foreign key into *instructor* table (so all IDs in *teaches* must appear in *instructor*)

Cartesian product: combines every pair of tuples from two different relations

Cartesian product: r X s

Cartesian product (or cross join)

r → instructor X teaches ← *s*

Each tuple from *instructor* matched with each tuple from *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
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

Result has attributes from both relations

Note: ID appears in both *instructor* and *teaches* table, some systems prefix with table name

This is probably not what we want!

Most rows about an instructor who did NOT teach a course

Number of rows = |r|*|s|

Rows can get big quickly!

Cartesian product: combines every pair of tuples from two different relations

Cartesian product: r X s

Cartesian product (or cross join)

r → instructor X teaches ← *s*

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

Srinivasan did not teach these sections!

But, cartesian product matches each tuple in r with each tuple in s

Combine Cartesian product with SELECT to produce a JOIN operation

Join operation

Select from cartesian product

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

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

Now we get courses taught by instructors

Attributes from both relations combined into a new relation

This is a JOIN operation

JOIN: returns attributes from r and s where attributes in predicate θ match

Join notation: r $\bowtie_\theta$ s

Given relations *r* (*R*) and *s* (*S*)

Let "theta" be a predicate on attributes R "union" S

The **JOIN** operation r $\bowtie_\theta$ s is defined as $r \bowtie_\theta s = \sigma_\theta (r \times s)$

Select from cartesian product based on θ

θ = instructor.id=teaches.id

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

Same as: *instructor* $\bowtie_{Instructor.id = teaches.id}$ *teaches*

Same procedure, just different notation

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

Agenda

1. Relational algebra part 2
2. Joins
3. Dates

JOIN tables in FROM clause using predicate in WHERE, return attributes in SELECT

Join tables ***Project name, course_id***

$\Pi_{\text{name, course\_id}}$ (*instructor* $\bowtie_{\text{Instructor.id = teaches.id}}$ *teaches)*

On JOIN instructor X teaches where IDs match

SELECT *name, course_id*
FROM *instructor , teaches*
WHERE *instructor.ID = teaches.ID*

Conceptual sequence of events

1. Perform Cartesian product over all relations in **FROM** clause
   - Result is Cartesian product like in slide 5
   - If three tables, number of tuples = $|t_1| * |t_2| * |t_3|$,
     where $|x|$ = number of tuples in table x
   - This result is not particularly useful
   - Real databases do not actually go to this trouble (too time consuming)
2. Apply predicates in **WHERE** clause to result from step 1 (gives rows wanted)
3. Project attributes from **SELECT** clause (gives columns wanted)

Can use aliases for table and attribute names

Joins with alias and 'AND' in WHERE

Find the names of all instructors in the Finance department and the courses they have taught

```
SELECT name, course_id AS `course number`
FROM instructor i, teaches t
WHERE i.ID = t.ID
      AND  i.dept_name = 'Finance'
```

Can alias table names

Could have said 'FROM instructor *AS* i ' but 'AS' is not required here

This is an "old style" join, we will soon look at another method

Can now reference table names by alias

Could have used instructor.ID, but i.ID is shorter

Attribute will be called *'course number'* instead of *'course_id'* thanks to AS keyword (makes an alias)

Use backtick (single quote character near the 1 key on your keyboard) for multiple word names `course number`

With JOIN store data one time in multiple tables; combine to form larger table

instructor 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

teaches table

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-315	1	Spring	2018

Recall *teaches* table lists courses and sections that are taught by *instructors*

Book's schema also has additional table for *courses* (not shown)

With JOIN store data one time in multiple tables; combine to form larger table

instructor 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

teaches table

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-315	1	Spring	2018

SELECT *i.*, t.**
FROM *instructor i, teaches t* *-- cartesian product*
WHERE *i.ID = t.ID;* *-- filter cartesian product*

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

If we kept data in cartesian product table, there are several anomalies that can occur

Problems keeping one large table with many attributes

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

Anomalies if keep one large table instead of multiple tables

- **Insert**
- **Update**
- **Delete**

If we kept data in cartesian product table, there are several anomalies that can occur

Problems keeping one large table with many attributes

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

Insert anomalies

- **If hire a new instructor, they do not show up in database until they teach a course**

If we kept data in cartesian product table, there are several anomalies that can occur

Problems keeping one large table with many attributes

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

Update anomalies

- **If instructor gets a raise, must update salary in <u>all</u> rows for that instructor**
- **Can lead to inconsistent data!**
- **What is the instructor's true salary?**

If we kept data in cartesian product table, there are several anomalies that can occur

Problems keeping one large table with many attributes

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

Delete anomalies

- **If course is only taught one time by an instructor who taught one course, if delete row, loose instructor and course!**
- **If delete PHY-101, loose Einstein and PHY-101!**

We can avoid these anomalies by keeping data in multiple tables

instructor 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

teaches table

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-315	1	Spring	2018

Better to store data in multiple tables

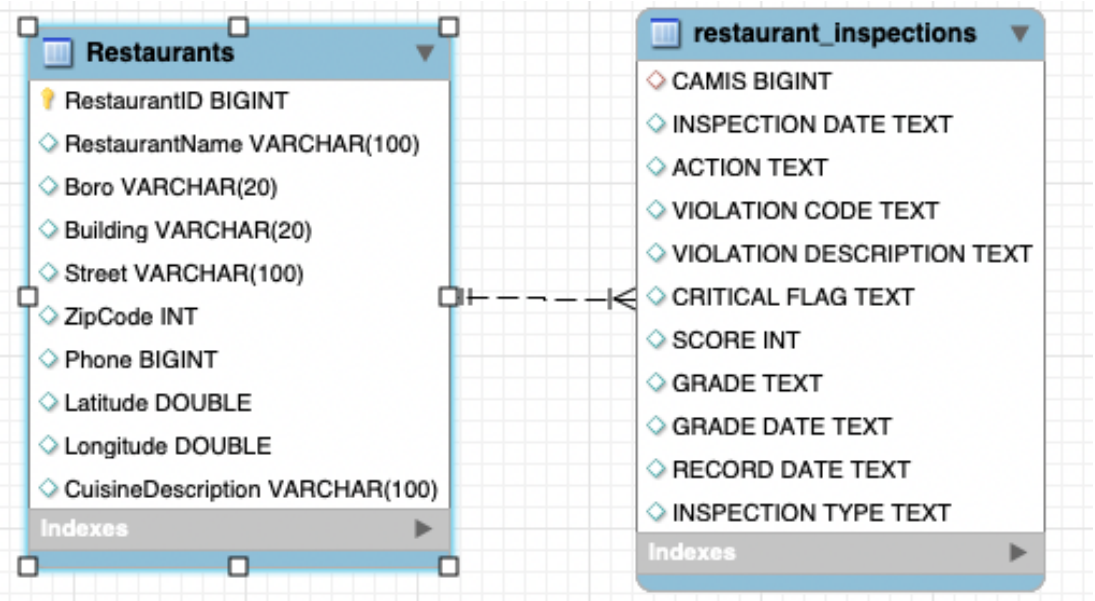
Insert: new instructor can be added to database without teaching a class

Update: instructor gets a raise, only update one row in instructor table

Delete: can delete course; instructor will still exist in instructor table

In our NYC health inspections schema, we left off with two tables

We can tie Restaurants with their inspections using RestaurantID = CAMIS



restaurant_inspections

- **Each row is data about one violation in one inspection of a restaurant**
- **We removed the duplicate information about the restaurant's name, address, cuisine**
- **We just keep the CAMIS (ID) integer for each restaurant in this inspections table**

Restaurants

- **Each row is data about one restaurant**
- **Can change name, cuisine, or move to new address -> just update one row in this table**

Note: CAMIS is a Foreign Key into Restaurants (can't have an inspection for a restaurant that doesn't exist!)

CAMIS is NOT a Primary Key on restaurant_inspections (multiple inspections/violations, so CAMIS appears multiple times, does not uniquely identify a row!)

Practice 1

- Find the unique types of cuisines restaurants have (list each type once)

- You're thinking about opening a new fruit/vegetable restaurant in Manhattan
  - List the fruit or vegetable restaurants in Manhattan (these restaurants are your competition). Recall the LIKE operator
  - Pick one of your competition and find all their health inspections
    - For each inspection, show the RestaurantID, RestaurantName, and all inspections details
    - Your output should look like this

RestaurantID	RestaurantName	CAMIS	INSPECTION DATE	ACTION	VIOLATION CODE	VIOLATION DESCRIPTION	CRITICAL FLAG	SC
50048030	JAMBA JUICE	50048030	02/12/2025	Violations were cited in the following area(s).	06D	Food contact surface not properly washed, rinsed and sanitized af…	Critical	7
50048030	JAMBA JUICE	50048030	11/22/2024	Violations were cited in the following area(s).	02G	Cold TCS food item held above 41 °F; smoked or processed fish h…	Critical	20
50048030	JAMBA JUICE	50048030	05/03/2022	Violations were cited in the following area(s).	10F	Non-food contact surface improperly constructed. Unacceptable m…	Not Critical	8
50048030	JAMBA JUICE	50048030	11/22/2024	Violations were cited in the following area(s).	03A	Food, prohibited, from unapproved or unknown source, home can…	Critical	20
50048030	JAMBA JUICE	50048030	06/16/2023	Violations were cited in the following area(s).	19-10	Failure to display required signage about plastic straw availability	Not Critical	NU
50048030	JAMBA JUICE	50048030	06/16/2023	Violations were cited in the following area(s).	10F	Non-food contact surface or equipment made of unacceptable mat…	Not Critical	12
50048030	JAMBA JUICE	50048030	06/16/2023	Violations were cited in the following area(s).	19-06	Providing single-use, non-compostable plastic straws to customer…	Not Critical	NU
50048030	JAMBA JUICE	50048030	02/12/2025	Violations were cited in the following area(s).	10F	Non-food contact surface or equipment made of unacceptable mat…	Not Critical	7
50048030	JAMBA JUICE	50048030	11/22/2024	Violations were cited in the following area(s).	10B	Anti-siphonage or back-flow prevention device not provided where…	Not Critical	20
50048030	JAMBA JUICE	50048030	06/16/2023	Violations were cited in the following area(s).	02G	Cold TCS food item held above 41 °F; smoked or processed fish h…	Critical	12
50048030	JAMBA JUICE	50048030	05/03/2022	Violations were cited in the following area(s).	06C	Food not protected from potential source of contamination during…	Critical	8

Recommended way to join is not in the WHERE clause, but with JOIN command

JOIN

Thus far we have joined relations by matching in the WHERE clause, but JOIN is preferred

Format:

SELECT $A_1, A_2, \ldots A_n$
FROM r_1 **{type} JOIN** r_2 **ON {p} JOIN ON {p} … JOIN** r_n **ON {p}**
where P ;
{type} = **[NATURAL | INNER | OUTER [LEFT RIGHT FULL]].** If type not specified, INNER

Joins store results in a temporary table in the database

Example: find all violations found for one restaurant

```
SELECT r.RestaurantID, RestaurantName, i.*
FROM Restaurants r, restaurant_inspections i
WHERE r.RestaurantID = i.CAMIS
AND i.CAMIS = 50048030;
```

Our previous way:
Join is done in the WHERE clause
Must specify which table attribute from

Both do the same thing!

Preferred way:

Recommended way:
Join is done in the FROM clause using JOIN; implicitly an INNER join

```
SELECT r.RestaurantID, RestaurantName, i.*
FROM Restaurants r JOIN Inspections i ON r.RestaurantID = i.CAMIS
WHERE i.CAMIS = 50048030;
```

Can still use WHERE to limit results

NATURAL join only returns rows if comparison attribute is in both tables

NATURAL JOIN

Rows returned based attributes common to both tables

I prefer NOT to use NATURAL JOIN because I like to know for sure which attributes were used in the join

What if we have a LastModified column?

```
SELECT *
FROM TableA NATURAL JOIN TableB
```

TableA

ID	A_1
1	m
2	n
4	o

TableB

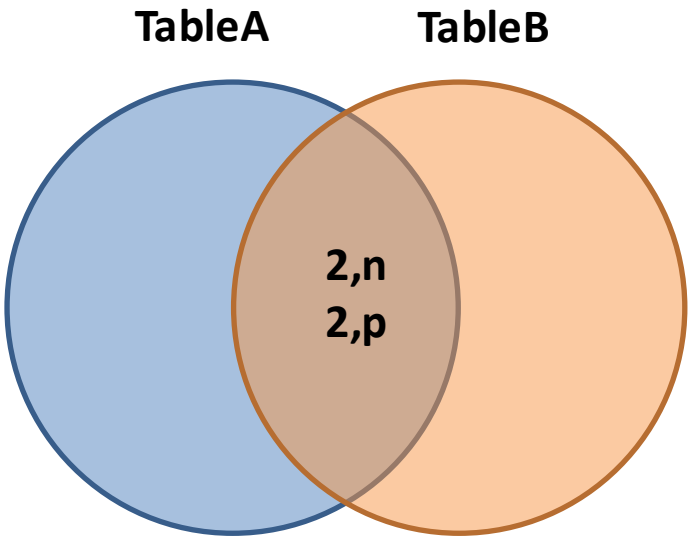
ID	A_2
2	p
3	q
5	r

ID attribute is common to both tables

ID = 2 is in both tables so those rows are returned, other rows are not

Result has attributes from both tables (ID, A_1 and A_2)

Like *SELECT * FROM TableA a, TableB b WHERE a.ID = b.ID*
But SQL chooses which columns to join



Result (temp table)

ID	A_1	A_2
2	n	p

NATURAL JOIN join on attributes with the same name

NATURAL JOIN omits duplicate attributes (could also pick in SELECT)

Adapted from: https://www.w3resource.com/slides/sql-joins-slide-presentation.php

INNER JOIN only returns rows if compared attribute is in both tables

INNER JOIN

TableA

ID	A_1
1	m
2	n
4	o

TableB

ID	A_2
2	p
3	q
5	r

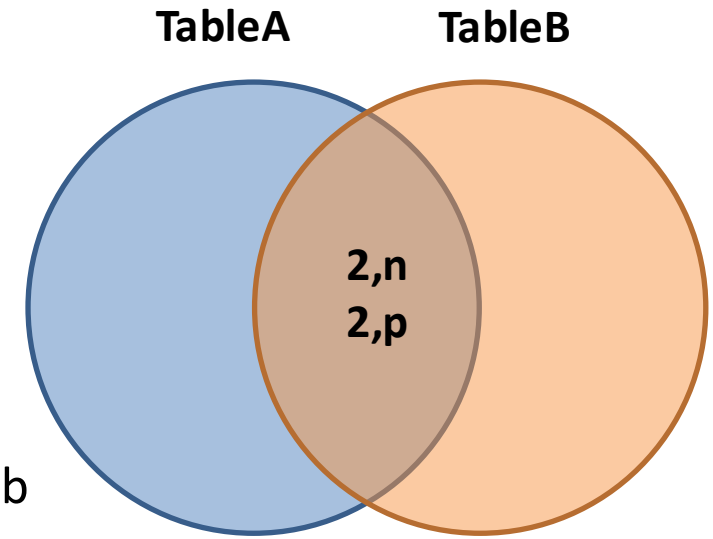
Rows returned with attributes from both tables if match between values in compared columns

Note: 'INNER' is optional

TableA a JOIN TableB b
ON a.ID = b.ID

```
SELECT *
FROM TableA a INNER JOIN TableB b
    ON a.ID=b.ID
```

Also: TableA JOIN TableB USING (ID)



Result (temp table)

ID	A_1	ID	A_2
2	n	2	p

ID of 2 is in both tables so it is returned, others are not

Result has attributes from both tables (A_1 and A_2) and duplicate ID

ID 1 and 4 from TableA and ID 3 and 5 from TableB not returned

NOTE: ID shown twice, once from each table

Same as join in WHERE clause

```
SELECT *
    FROM TableA a, TableB b
    WHERE a.ID = b.ID;
```

Adapted from: https://www.w3resource.com/slides/sql-joins-slide-presentation.php

LEFT OUTER JOIN returns all rows from the left table

LEFT [OUTER] JOIN

TableA

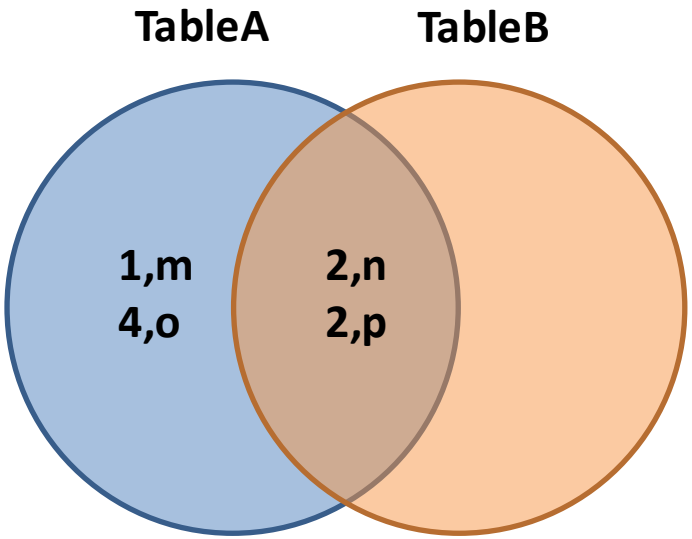
ID	A_1
1	m
2	n
4	o

TableB

ID	A_2
2	p
3	q
5	r

OUTER is optional
'LEFT OUTER JOIN' = 'LEFT JOIN'

```
SELECT *
FROM TableA a LEFT JOIN TableB b
    ON a.ID=b.ID
```



Result (temp table)

ID	A_1	ID	A_2
1	m	NULL	NULL
2	n	2	p
4	o	NULL	NULL

ID of 2 is in both tables so it is returned

All rows from left table (TableA) as written in command are returned

3 and 5 in TableB not returned because those keys not in TableA

All TableA rows
NULLs for TableB attributes if no ID

Adapted from: https://www.w3resource.com/slides/sql-joins-slide-presentation.php

RIGHT OUTER JOIN returns all rows from the right table

RIGHT [OUTER] JOIN

TableA

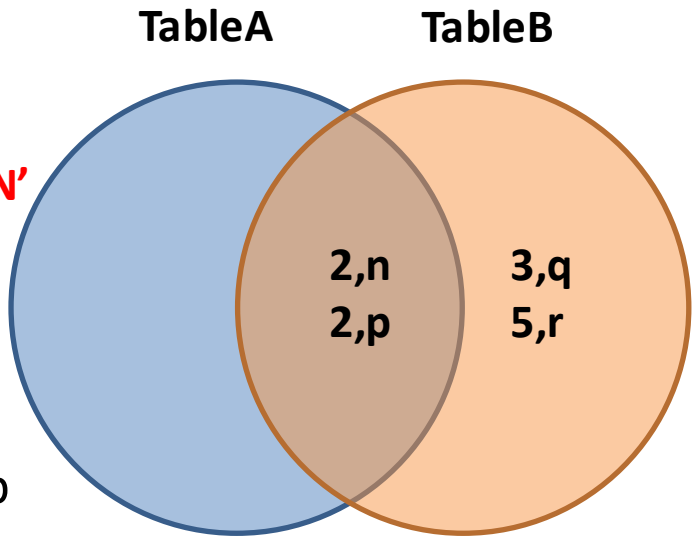
ID	A_1
1	m
2	n
4	o

TableB

ID	A_2
2	p
3	q
5	r

OUTER is optional
'RIGHT OUTER JOIN' = 'RIGHT JOIN'

```
SELECT *
FROM TableA a RIGHT JOIN TableB b
    ON a.ID=b.ID
```



Result (temp table)

ID	A_1	ID	A_2
2	n	2	p
NULL	NULL	3	q
NULL	NULL	5	r

ID of 2 is in both tables so it is returned

All rows from right table (TableB) as written in command are returned

1 and 4 in TableA not returned because those keys not in TableB

All TableB rows
NULLs for TableA attributes if no ID

Adapted from: https://www.w3resource.com/slides/sql-joins-slide-presentation.php

FULL OUTER JOIN returns all rows from both tables

FULL [OUTER] JOIN

All rows from both tables returned

TableA

ID	A_1
1	m
2	n
4	o

TableB

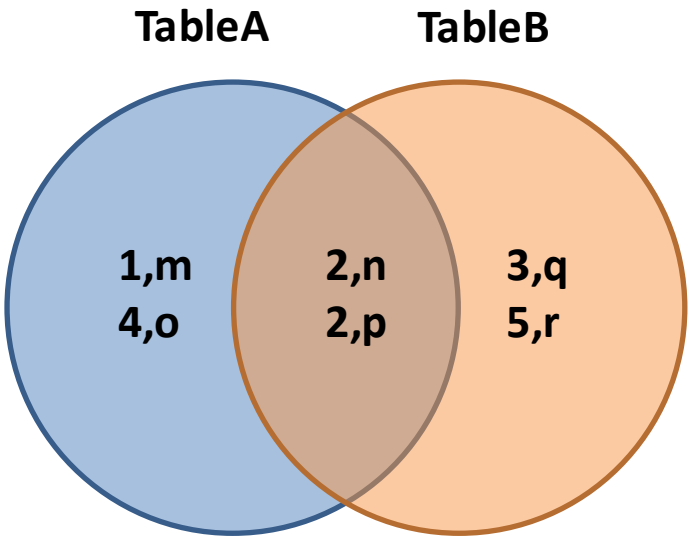
ID	A_2
2	p
3	q
5	r

```
SELECT *
FROM TableA a FULL JOIN TableB b
    ON a.ID=b.ID
```

ID of 2 is in both tables so it is returned

All rows from both tables are returned

NOTE: MySQL does not support FULL OUTER JOIN



Result (temp table)

ID	A_1	ID	A_2
1	m	NULL	NULL
2	n	2	p
4	o	NULL	NULL
NULL	NULL	3	q
NULL	NULL	5	r

Adapted from: https://www.w3resource.com/slides/sql-joins-slide-presentation.php

Practice

Use nyc_inspections

You've opened a new fruits and vegetables restaurant in Manhattan called 'Tim's Tasty Treats' (keep the apostrophe in the name!):

- Insert a new row in your Restaurants table for this restaurant
  - Set the RestaurantID to 1111
  - Set the CuisineDescription to the proper value for a fruits and vegetables restaurant (e.g., do not make a new description, use one from Restaurants)
  - You can set address, phone, lat/long to NULL (or another value)
- See how many other fruit/vegetable restaurants there are (your competition), use your variable from above
- Using a JOIN, count how many violations has been noted for each fruit/vegetable restaurant, include:
  - RestaurantID, RestaurantName, Count of violations
  - Make sure Tim's restaurant is on the list and shows zero inspections! (note: this is tricky! Pay attention to the type of JOIN and what you count)

Agenda

1. Relational algebra part 2
2. Joins
3. Dates

SQL offers a DATE field

```
-- convert a field with strings for dates into DATE field
-- add a DATE field to an existing table
ALTER TABLE TableName ADD COLUMN FieldName DATE;

-- fill new field
UPDATE TableName
    SET FieldName = STR_TO_DATE(DateStringField, '%m/%d/%Y');

-- drop old string date field
ALTER TABLE TableName DROP COLUMN DateStringField;
```

Match the format of the existing date string

SQL has a wide variety of date functions

1. Current Date & Time **Useful for practice on next page**

- **NOW()**: Returns the current date and time (YYYY-MM-DD HH:MM:SS).
- **CURDATE()**: Returns only the current date (YYYY-MM-DD).
- **CURTIME()**: Returns only the current time (HH:MM:SS).

2. Extracting Parts of a Date

If you have a full DATETIME but only want one piece of it:

- **YEAR(date)**, **MONTH(date)**, **DAY(date)**: Returns the year, month (1-12), or day (1-31) as an integer.
- **MONTHNAME(date)**: Returns the full name of the month (e.g., 'October').
- **DAYNAME(date)**: Returns the name of the weekday (e.g., 'Friday').
- **EXTRACT(unit FROM date)**: A versatile function to pull any part (e.g., EXTRACT(WEEK FROM date)).

3. Date Arithmetic ("Date Math")

- **DATE_ADD(date, INTERVAL value unit)**: Adds time to a date.
  - *Example:* DATE_ADD(NOW(), INTERVAL 7 DAY)
- **DATE_SUB(date, INTERVAL value unit)**: Subtracts time from a date.
  - *Example:* DATE_SUB(NOW(), INTERVAL 1 MONTH)
- **DATEDIFF(date1, date2)**: Returns the number of days between two dates.
- **TIMESTAMPDIFF(unit, datetime1, datetime2)**: Returns the difference in specific units like HOUR, MINUTE, or SECOND

Practice

Updates fields in restaurant_inspections that represent dates from a VARCHAR to a DATE data type

- Create a new attribute for each attribute that represents a date
- **UPDATE** restaurant_inspections

  SET <new attribute name> = **STR_TO_DATE**(varchar field, '%m/%d/%Y');
- (Optional) DROP varchar field
- NOTE: because there is no Primary Key on restaurant_inspections, we need to turn off MySQL's safe update protections: **SET SQL_SAFE_UPDATES = 0;**

Select all restaurants and return:

- RestaurantID and name
- Date of last inspection
- Number of days since last inspection (use DATEDIFF, CURDATE)
- Return most recently inspected restaurants first

(remember I pulled this data around December 1, 2025)