# Discussion 2

## SQL DDL & Relational Algebra

EECS 484

# Logistics

- Homework 1
  - Due Today, 11:45 PM ET
  - Individual, No Groups!
- Project 1
  - Due Sep 24th, 11:45 PM ET
  - Groups of 2. Make sure to add each other as a group before submitting to the Gradescope and Autograder
  - Part 1 is due on Gradescope
  - Parts 2-4 are due on the Autograder
- Homework 2 Released
  - Due Oct 4th, 11:45 PM ET
  - Individual, No Groups!

# SQL

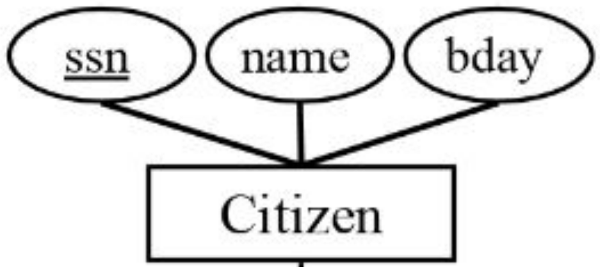
- Structured Query Language
  - Allows you to interact with a DBMS
  - Commands differ slightly from distribution to distribution
    - All compliant languages should be about the same though
  - Other languages exist but are far less popular
  - Vulnerable to things like SQL injection
    - Way of interacting where SQL cannot tell the difference between text and code
  - Convention: All SQL keywords are uppercase
    - SQL doesn’t actually care

# SQL – Data Definition Language

# Creating Tables

- CREATE TABLE Table_Name (field_1 field_1_type, field_2, field_2_type … );
    - Makes an empty table with those columns
    - Can specify constraints too
        - Primary keys, not null, unique, foreign keys, etc.
- Example:
    - CREATE TABLE Citizen (

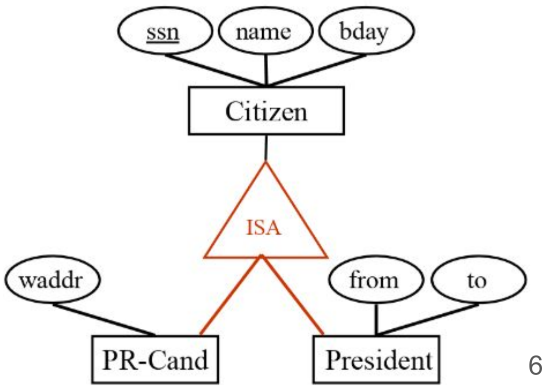
```
        ssn NUMBER PRIMARY KEY,
        name VARCHAR2(100) NOT NULL,
        bday DATETIME
);
```



# Constraints

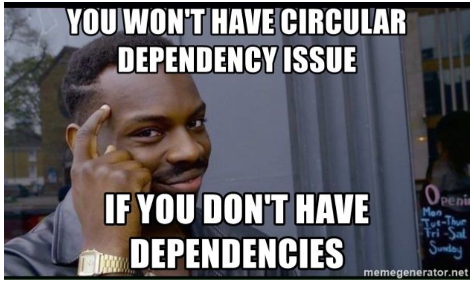
- Constraints allow the database to check the data as it's inserted
  - "Laws" of the schema that are enforced
- Examples
  - PRIMARY KEY - Necessary for every table. A unique and not null value
  - NOT NULL - Field must be populated
  - UNIQUE - Field must be unique across all rows
  - FOREIGN KEY - Links entries across tables (references primary key of another table)
  - CHECK - Makes sure data in a column meets some condition
  - Constraints can be applied across multiple fields in the relation
  - CREATE TABLE President (

```
        ssn NUMBER PRIMARY KEY,
        name VARCHAR2(100) NOT NULL,
        from DATETIME NOT NULL,
        to DATETIME NOT NULL,
        UNIQUE (from, to)   );
```



# Circular Dependency

- Circular dependencies are when two tables depend on each other
  - Foreign key reference each other
  - Programming version of what came first, chicken or the egg




- Example: A Student has be involved in exactly one Club, and a Club must be led by exactly one Student.

# Circular Dependency Example

A Student has be involved in exactly one Club, and a Club must be led by exactly one Student.

```
CREATE TABLE Student (
        sid INTEGER PRIMARY KEY,
        sname VARCHAR(100) NOT NULL,
        club INTEGER NOT NULL,
        FOREIGN KEY (club) REFERENCES Club (cid)
);
```

```
CREATE TABLE Club (
        cid INTEGER PRIMARY KEY,
        cname VARCHAR(100) NOT NULL,
        student_leader INTEGER NOT NULL,
        FOREIGN KEY (student_leader)
REFERENCES Student (sid)
);
```

Won’t work!
Solution: first create both tables without dependencies, then add the dependencies after.

# Circular Dependency Example

A Student has be involved in exactly one Club, and a Club must be led by exactly one Student.

```
CREATE TABLE Student (
	sid INTEGER PRIMARY KEY,
	sname VARCHAR(100) NOT NULL,
	club INTEGER NOT NULL
);
```

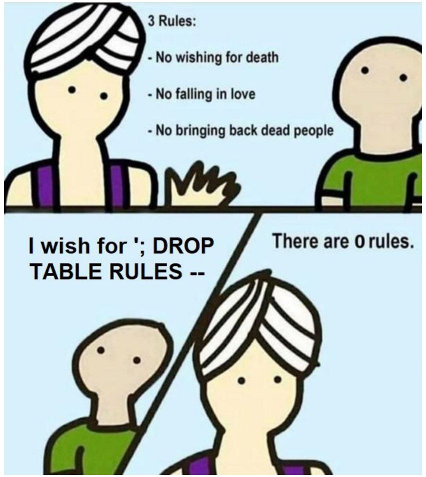
```
CREATE TABLE Club (
	cid INTEGER PRIMARY KEY,
	cname VARCHAR(100) NOT NULL,
	student_leader INTEGER NOT NULL
);
```

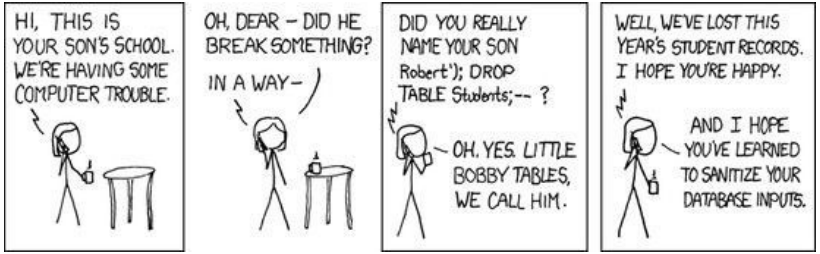
```
ALTER TABLE Student ADD CONSTRAINT club_participation
FOREIGN KEY (club) REFERENCES Club(cid) INITIALLY DEFERRED DEFERRABLE;

ALTER TABLE Club ADD CONSTRAINT leader_is_student
FOREIGN KEY (student_leader) REFERENCES Student(sid) INITIALLY DEFERRED DEFERRABLE;
```

# Drop

- DROP TABLE Table_Name CASCADE CONSTRAINTS;
- Deletes the table and all constraints it contains
    - includes foreign key constraints and triggers
    - does not include sequences




# Enforcing Referential Integrity

# Disallow deletion (default)

- **Disallow deletion** (default): If a tuple you are trying to delete is referred in another table, the record won't be deleted and instead will return an error message.
  - This is default behavior in Oracle, but ON DELETE NO ACTION in MySQL
- Example of this constraint in action: **DELETE FROM Student WHERE sno = 1**
  - If there is a row in Enroll where sno = 1 the statement above will throw an error
- Example of how to add constraint (don't need to do anything special because it is default):

```
CREATE TABLE Enroll (
    sno INT,
    cno INT,
    jdate date,
    PRIMARY KEY(sno,cno),
    FOREIGN KEY(sno) REFERENCES Student(sno),
    FOREIGN KEY(cno) REFERENCES Course(cno)
);
```

# Delete all

- **Delete all**: If a user deletes a row in the parent table, then the affected row is deleted in the child table.
- Example of this constraint in action: **DELETE FROM Student WHERE sno = 1**
    - If there are any rows in Enroll where sno = 1, all of those rows in Enroll will be deleted
- Example of how to add constraint:

```
CREATE TABLE Enroll (
    sno INT,
    cno INT,
    jdate date,
    PRIMARY KEY(sno,cno),
    FOREIGN KEY(sno) REFERENCES Student(sno)
    ON DELETE CASCADE,
    FOREIGN KEY(cno) REFERENCES Course(cno)
    ON DELETE CASCADE
);
```

# Set to null

- **Set to null**: If a user deletes a row in the parent table, then the affected field is now set to null.
- Example of this constraint in action: **DELETE FROM Student WHERE sno = 1**
    - If there are any rows in Enroll where sno = 1, sno will now be set to NULL in those rows
- Example of how to add constraint:

```
CREATE TABLE Person (
    person_id INT,
    car_id INT,
    dob date,
    PRIMARY KEY(person_id),
    FOREIGN KEY(car_id) REFERENCES Cars(car_id)
     ON DELETE SET NULL,
);
```

# `CREATE TABLE` PracticeProblems

# Practice Problem

Consider the following example code:

```
CREATE TABLE Enroll (
    sno INT,
    cno INT,
    jdate date,
    PRIMARY KEY(sno,cno),
    FOREIGN KEY(sno) REFERENCES Student(sno)
     ON DELETE SET NULL,
    FOREIGN KEY(cno) REFERENCES Course(cno)
    ON DELETE SET NULL
);
```

Why is ON DELETE SET NULL improperly used in this instance?

# Practice Problem

1. Take some time to write the SQL commands to create a table with the following schema:

- Table Name: Teas
- Columns: (column_name - type)
    i. Tea_Name - VARCHAR2(100)
    ii. Brew_Time - NUMBER
    iii. Brand - VARCHAR2(100)
- Constraints:
    i. Teas are stored by their primary key, the name of the tea
    ii. Each tea must have a brand

# Practice Problem

1. Take some time to write the SQL commands to create a table with the following schema:

- Answer:

```
CREATE TABLE Teas (
    Tea_Name VARCHAR2(100) PRIMARY KEY,
    Brew_Time NUMBER,
    Brand VARCHAR2(100) NOT NULL
);
```

# Practice Problem

2. Now that we have some tea, let's make a menu table. Write the SQL statements to create the menu table:

- Table Name: Menus
- Columns: (column_name - type)
    1. Tea_Name - VARCHAR2(100)
    2. Menu_Name - VARCHAR2(100)
    3. Cost - NUMBER
- Constraints:
    1. Menu items are stored by Menu_Name as their primary key
    2. Each menu item must have a Tea_Name which corresponds to an item in the Teas table
    3. Each menu item must have a Cost

# Practice Problem

2. Now that we have some tea, let's make a menu table. Write the SQL statements to create the menu table:

- Answer:

```
CREATE TABLE Menus (
    Tea_Name VARCHAR2(100) NOT NULL,
    FOREIGN KEY (Tea_Name) REFERENCES Teas (Tea_Name),
    Menu_Name VARCHAR2(100) PRIMARY KEY,
    Cost NUMBER NOT NULL
);
```

# Practice Problem

3. Finally, let’s drop all of our tables. Write the SQL Commands to drop them

# Practice Problem

3. Finally, let’s drop all of our tables. Write the SQL Commands to drop them

- DROP TABLE Menus;
  DROP TABLE Teas;

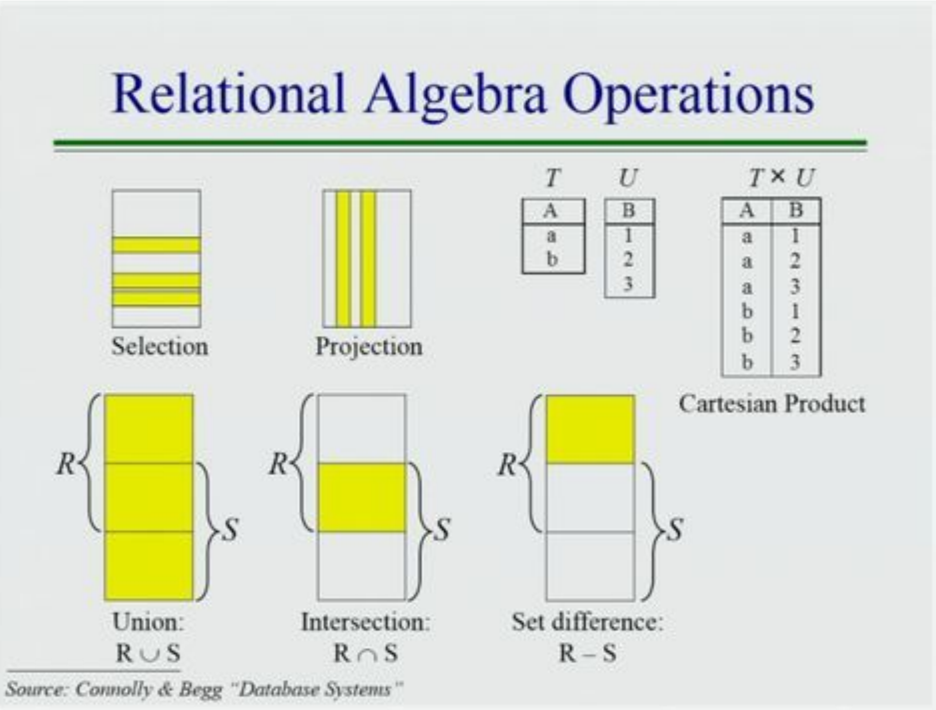
  or

- DROP TABLE Teas CASCADE CONSTRAINTS;
  DROP TABLE Menus;

# Relational Algebra

# Relational Algebra

- Way to represent imperative execution plans for queries
- Helpful to define what you are doing without relying on a specific SQL language



# Selections!

# Selection

- What SQL ‘WHERE …’ is equivalent to
- Retrieve rows that satisfy a logical condition
    - Conditions can be series of boolean statements
    - Compare attributes with other attributes or constants
    - Ahhhh what does this mean, let’s see examples!!

# Selection Example 1

- Format: $\sigma_{condition}(relation)$

- $\sigma_{drink='tea' \wedge major='CS'}(Student)$
  - We should be returning…

Student

| UserID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Undeclared | Tea |
| 3 | Cathy | Aerospace | Coffee |
| 4 | Dylan | CS | Coffee |

# Selection Example 1

- Format: $\sigma_{condition}(relation)$

Notice: what we meant by “series of boolean statements”

- $\sigma_{\boxed{drink='tea' \wedge major='CS'}}(Student)$
  - We should be returning…
    - Row 1

Student

| UserID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Undeclared | Tea |
| 3 | Cathy | Aerospace | Coffee |
| 4 | Dylan | CS | Coffee |

# Selection with Relational Algebra Example 2

- Format: $\sigma_{condition}(relation)$

- $\sigma_{drink='tea' \vee major='Aerospace'}(Student)$
  - What should we return?

Student

| UserID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Undeclared | Tea |
| 3 | Cathy | Aerospace | Coffee |
| 4 | Dylan | CS | Coffee |

# Selection with Relational Algebra Example 2

- Format: $\sigma_{condition}$(relation)

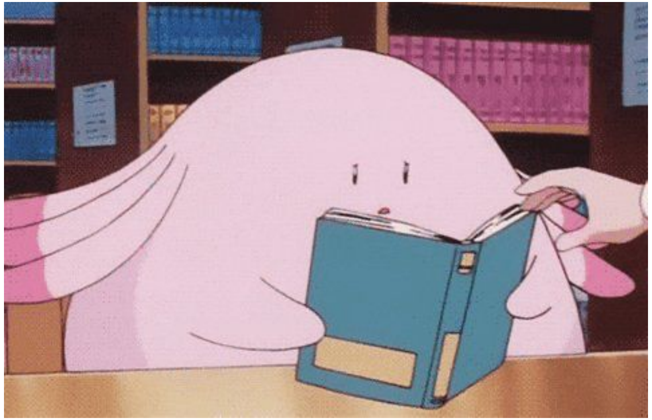
Notice: the ‘v’ for or, comparing attrs with constant values

- $\sigma_{\boxed{\text{drink='tea' v major='Aerospace'}}}$(Student)
  - What should we return?
    - Row 1, 2, 3

Student

| UserID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Undeclared | Tea |
| 3 | Cathy | Aerospace | Coffee |
| 4 | Dylan | CS | Coffee |

Projections!



# Projection

- Equivalent to SQL SELECT DISTINCT
- Choose ***subset*** of columns
    - Deletes attributes ***not*** in projection list
    - Removes ***all duplicates*** rows in the final output
- Written like: $\pi_{\text{projection list}}(\text{relation})$

# Projection Example

- RA: $\pi_{\text{projection list}}(\text{relation})$
- Example: $\pi_{\boxed{\text{major, drink}}}(\text{Student})$

Student

| UserID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Undeclared | Tea |
| 3 | Cathy | Undeclared | Tea |

Notice: what cols are left in the table here

| Major | Drink |
|---|---|
| CS | Tea |
| Undeclared | Tea |

# Set Operators

| | |
|---|---|
| Union | $\cup$ |
| Intersection | $\cap$ |
| Set Difference | $-$ |
| Cross Product | $\times$ |

# Union


- Creates a relation that contains records that appear in Relation1 ***or*** Relation2
- Removes duplicates records
- Example:   RA: Undergrads ∪ Grads

Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major |
|---|---|
| Alice | CS |
| Cathy | Aerospace |

Notice: Alice only appears once

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |
| Cathy | Aerospace |

# Question about Union

Can you do Undergrads ∪ Grads here?

Undergrads

| Name | Major |
| --- | --- |
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major | UMID |
| --- | --- | --- |
| Alice | CS | 123 |
| Cathy | Aerospace | 457 |

# Question about Union

## Can you do Undergrads ∪ Grads here?

No, because how would we compare the UMID column that Undergrads don't have

Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major | UMID |
|---|---|---|
| Alice | CS | 123 |
| Cathy | Aerospace | 457 |

# Question about Union

Can you do Undergrads ∪ Grads here?

No, because how would we compare the UMID column that Undergrads don't have

This concept is called:

**Compatibility**

The relations must be:

- **Same** number of fields
- Corresponding **fields** are of the **same datatype**

Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major | UMID |
|---|---|---|
| Alice | CS | 123 |
| Cathy | Aerospace | 457 |

Needs Compatibility

# Intersect

- Creates a relation that contains records that appear in Relation1 ***and*** Relation2
- ***Removes duplicates*** records
- Example: RA: Undergrads ∩ Grads

Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major |
|---|---|
| Alice | CS |
| Cathy | Aerospace |

| Name | Major |
|---|---|
| Alice | CS |

Needs Compatibility

# Set difference

- Creates a relation that contains records that appear in Relation1 ***and do not appear*** in Relation2
- **Order of relations** matters!
- Example: RA: Undergrads – Grads


Spot the set difference

Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major |
|---|---|
| Alice | CS |
| Cathy | Aerospace |

→

| Name | Major |
|---|---|
| Bob | Aerospace |

***DOES NOT*** Need Compatibility

# Cross product

- RA: RelationA × RelationB
- Combine two relations
- **Every combination** represented
- Example: Student × Course

Student

| ID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Math | Tea |

Course

| Major | CID |
|---|---|
| CS | 575 |
| Chem | 548 |

| ID | Name | (Major) | Drink | (Major) | CID |
|---|---|---|---|---|---|
| 1 | Alice | CS | Tea | CS | 575 |
| 2 | Bob | Math | Tea | CS | 575 |
| 1 | Alice | CS | Tea | Chem | 548 |
| 2 | Bob | Math | Tea | Chem | 548 |

# Cross product

- RA: RelationA × RelationB
- Combine two relations
- **Every combination** represented
- Example: Student × Course

Is this correct?

Student

| ID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Math | Tea |

Course

| Major | CID |
|---|---|
| CS | 575 |
| Chem | 548 |

| ID | Name | (Major) | Drink | (Major) | CID |
|---|---|---|---|---|---|
| 1 | Alice | CS | Tea | CS | 575 |
| 2 | Bob | Math | Tea | CS | 575 |
| 1 | Alice | CS | Tea | Chem | 548 |
| 2 | Bob | Math | Tea | Chem | 548 |

# Cross product

- RA: RelationA × RelationB
- Combine two relations
- **Every combination** represented
- Example: Student × Course

Student

| ID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Math | Tea |

Course

| Major | CID |
|---|---|
| CS | 575 |
| Chem | 548 |

Is this correct?

NO, we don't want ***duplicate*** column names!!!

| ID | Name | (Major) | Drink | (Major) | CID |
|---|---|---|---|---|---|
| 1 | Alice | CS | Tea | CS | 575 |
| 2 | Bob | Math | Tea | CS | 575 |
| 1 | Alice | CS | Tea | Chem | 548 |
| 2 | Bob | Math | Tea | Chem | 548 |

# Renaming

- Allows us to rename **attributes** and/or **relations**
- Represented by ρ(newRelationName(oldAttr1->newAttr1, …), expression)
    - newRelationName is the desired name of the relation
        - Can omit if same as previous relation name
            - Syntax becomes ρ(oldAttr1->newAttr1, …, expression)
    - oldAttr1 is the name of the column before renaming
        - newAttr1 is the name of the same column after renaming
        - Can have as many oldAttr -> newAttr as desired comma separated (...)
        - Don’t need to rename any columns necessarily
            - Syntax becomes ρ(newRelationName, expression)
    - Expression is what is being renamed
        - Can be an RA expression or simply a relation

# Renaming

The almost correct cross product that we just saw

Let's rename the duplicate columns

$$\rho(\text{Major -> Major1, Student}) \times \rho(\text{Major -> Major2, Course})$$

Student

| ID | Name | Major1 | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Math | Tea |

Course

| Major2 | CID |
|---|---|
| CS | 575 |
| Chem | 548 |

| ID | Name | Major1 | Drink | Major2 | CID |
|---|---|---|---|---|---|
| 1 | Alice | CS | Tea | CS | 575 |
| 2 | Bob | Math | Tea | CS | 575 |
| 1 | Alice | CS | Tea | Chem | 548 |
| 2 | Bob | Math | Tea | Chem | 548 |

# Renaming

Example 2:

- RA: ρ(Students(major->Area of study), Undergrads ∪ Grads)


Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major |
|---|---|
| Alice | CS |
| Cathy | Aerospace |

Undergrads ∪ Grads

# Renaming

Example 2:

- RA: ρ(Students(major->Area of study), Undergrads ∪ Grads)


Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major |
|---|---|
| Alice | CS |
| Cathy | Aerospace |

Undergrads ∪ Grads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |
| Cathy | Aerospace |

# Renaming


Example 2:

- RA: ρ(Students(major->Area of study), Undergrads ∪ Grads)

Undergrads

| Name | Major |
|---|---|
| Alice | CS |
| Bob | Aerospace |

Grads

| Name | Major |
|---|---|
| Alice | CS |
| Cathy | Aerospace |

Students

| Name | Area of study |
|---|---|
| Alice | CS |
| Bob | Aerospace |
| Cathy | Aerospace |

# Division

- C = A / B
  - B is a proper subset of A
  - Proper subset means a subset that is not equivalent to original set
- Ahhh what does this mean?
  - Look through the data in A and look at the columns not included in B
    - Which rows would work so that in conjunction with every single row in B we get data in the original relation

- Another way to look at it
  - Look only at the columns in A that are not in B
    - Iterate over the remaining data in A
    - If there is a row in the remaining dataset where if I cross product that row with B, the resulting product has rows that are not in A, discard that row
- One final way: Final set C has property that every row in C×B is in A

# Division

A

| UserID | CourseID |
|---|---|
| 1 | 548 |
| 1 | 575 |
| 2 | 412 |
| 2 | 484 |
| 2 | 575 |
| 2 | 588 |
| 3 | 388 |
| 3 | 412 |
| 3 | 484 |

B

| CourseID |
|---|
| 412 |
| 484 |

C

| CourseID |
|---|
| 412 |
| 575 |

D

| CourseID |
|---|
| 388 |
| 548 |

A / B

| UserID |
|---|
| 2 |
| 3 |

A / C

| UserID |
|---|
| 2 |

A / D

| UserID |
|---|

# Division

- A (x, y), B (y)

$$A/B:\ \pi_x(A) - \underbrace{\pi_x((\pi_x(A) \times B) - A)}_{\text{Disqualified } x \text{ values}}$$

x value is disqualified if by attaching y value from B, we obtain an <x,y> tuple that is not in A

# Division

A/B: $\pi_x(A) - \pi_x(\underbrace{(\pi_x(A) \times B) - A)}_{\text{Disqualified } x \text{ values}}$

A

| uid | cid |
|---|---|
| 1 | 548 |
| 1 | 575 |
| 2 | 412 |
| 2 | 484 |
| 2 | 575 |
| 2 | 588 |
| 3 | 388 |
| 3 | 412 |
| 3 | 484 |

B

| cid |
|---|
| 412 |
| 484 |

$\pi_{uid}(A) \text{ X } B$

| uid | cid |
|---|---|
| 1 | 412 |
| 1 | 484 |
| 2 | 412 |
| 2 | 484 |
| 3 | 412 |
| 3 | 484 |

$\pi_{uid}(A) \text{ X } B - A$

| uid | cid |
|---|---|
| 1 | 412 |
| 1 | 484 |

$\pi_{uid}(\pi_{uid}(A) \text{ X } B - A)$

| uid |
|---|
| 1 |

$\pi_{uid}(A) - \pi_{uid}(\pi_{uid}(A) \text{ X } B - A)$

| uid |
|---|
| 2 |
| 3 |

# Division in SQL

A/B: $\pi_x(A) - \pi_x(\underbrace{(\pi_x(A)\times B) - A}_{\text{Disqualified } x \text{ values}})$

Direct Translation to SQL:

```
SELECT x FROM A
MINUS SELECT x FROM (
    SELECT A1.x, B1.y FROM
        (SELECT x FROM A) A1
        CROSS JOIN
        (SELECT y FROM B) B1
    MINUS
    SELECT x, y FROM A
);
```

Alternative:

```
SELECT DISTINCT x FROM A A1
WHERE NOT EXISTS (
    SELECT y FROM B B1
    MINUS
    SELECT y from A A2
    WHERE A1.x = A2.x
);
```

Disclaimer: not thoroughly tested yet

# Join

- Way to combine information from two tables with correlation
- Conditional join
  - RA: Relation1 $\bowtie_{condition}$ Relation2
    - Equivalent to $\sigma_{condition}$(Relation1 X Relation2)
    - condition can include multiple expressions
  - Equijoin is a conditional join with restrictions on condition
    - Only equalities between fields and $\wedge$ connectors

# Join

- Way to combine information from two tables with correlation
- Natural join (without specifying condition)
    - Relation1 ⋈ Relation2
    - Equijoin but automatic on all columns with the same name (must be same type)
    - Duplicate columns are dropped

# Course ⋈ Student (Example of natural join)

Course

| CID | Major |
|---|---|
| 575 | CS |
| 548 | Aerospace |
| 484 | CS |
| 412 | Math |

Student

| UserID | Name | Major | Drink |
|---|---|---|---|
| 1 | Alice | CS | Tea |
| 2 | Bob | Undeclared | Tea |
| 3 | Cathy | Aerospace | Tea |
| 4 | Dylan | CS | Coffee |

| CID | Major | UserID | Name | Drink |
|---|---|---|---|---|
| 575 | CS | 1 | Alice | Tea |
| 484 | CS | 1 | Alice | Tea |
| 548 | Aero | 3 | Cathy | Tea |
| 575 | CS | 4 | Dylan | Coffee |
| 484 | CS | 4 | Dylan | Coffee |

# Set Operators Summary for Your Reference

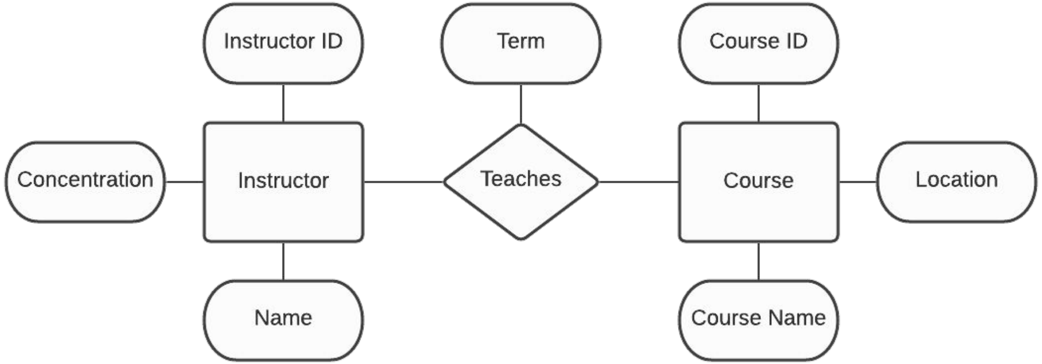
| | |
|---|---|
| Union | Relation1 ∪ Relation2 |
| Intersection | Relation1 ∩ Relation2 |
| Set Difference | Relation1 – Relation2 |
| Cross Product | Relation1 × Relation2 |

- Relations must be “compatible” to perform union, intersection, or set difference (but **not** cross product)
- What is “compatibility”?:
  - Relations have the **same number of fields**
  - Corresponding **fields** are of the **same datatype**

$$\sigma_{topic='RA'}(\text{PracticeProblems})$$

# Example Problem

- Three tables:
  - Courses, Instructors, Teaches




| Course ID | Course Name | Location |
|---|---|---|
| EECS 575 | Crypto | 1690BBB |
| EECS 484 | Databases | 1670BBB |
| EECS 482 | OS | 1690BBB |
| EECS 388 | Security | 404BBB |
| AERO 548 | Astrodynamics | FXB1012 |
| EECS 999 | Redacted | ??? |

| Instructor ID | Name | Concentration |
|---|---|---|
| 1 | Alice | Cryptography |
| 2 | Bob | Boring |
| 3 | SQL | Databases |
| 4 | Eve | Hacking :D |
| 5 | Buzz | Space |
| 6 | Mr. Meow | Meowing |

| Instructor ID | Course ID | Term |
|---|---|---|
| 2 | EECS 575 | F20 |
| 3 | EECS 484 | F20 |
| 1 | EECS 482 | W19 |
| 2 | AERO 548 | W19 |
| 3 | EECS 388 | W19 |
| 4 | EECS 388 | W21 |

# Q1

- Write the RA statement to select the names of all the instructors who teach in 1690 BBB and the courses that they teach
  - Don’t select courses that have no instructor or instructors that don’t teach anything

| Course ID | Course Name | Location |
|---|---|---|
| EECS 575 | Crypto | 1690BBB |
| EECS 484 | Databases | 1670BBB |
| EECS 482 | OS | 1690BBB |
| EECS 388 | Security | 404BBB |
| AERO 548 | Astrodynamics | FXB1012 |
| EECS 999 | Redacted | ??? |

| Instructor ID | Name | Concentration |
|---|---|---|
| 1 | Alice | Cryptography |
| 2 | Bob | Boring |
| 3 | SQL | Databases |
| 4 | Eve | Hacking :D |
| 5 | Buzz | Space |
| 6 | Mr. Meow | Meowing |

| Instructor ID | Course ID | Term |
|---|---|---|
| 2 | EECS 575 | F20 |
| 3 | EECS 484 | F20 |
| 1 | EECS 482 | W19 |
| 2 | AERO 548 | W19 |
| 3 | EECS 388 | W19 |
| 4 | EECS 388 | W21 |

# Q1

- Write the RA statement to select the names of all the instructors who teach in 1690 BBB and the courses that they teach
    - Don’t select courses that have no instructor or instructors that don’t teach anything
- $\pi_{\text{name, Course\_ID}}(\text{Instructor} \bowtie \text{Teaches} \bowtie \sigma_{\text{Location='1690 BBB'}}(\text{Course}))$ OR
- $\pi_{\text{name, Course\_ID}}(\sigma_{\text{Location='1690 BBB'}}(\text{Instructor} \bowtie \text{Teaches} \bowtie \text{Course}))$
- Why do I not need conditions on the joins?

# Q1

- Write the RA statement to select the names of all the instructors who teach in 1690 BBB and the courses that they teach
  - Don't select courses that have no instructor or instructors that don't teach anything
- $\pi_{\text{name, Course\_ID}}(\text{Instructor} \bowtie \text{Teaches} \bowtie \sigma_{\text{Location='1690 BBB'}}(\text{Course}))$ OR
- $\pi_{\text{name, Course\_ID}}(\sigma_{\text{Location='1690 BBB'}}(\text{Instructor} \bowtie \text{Teaches} \bowtie \text{Course}))$
- Why do I not need conditions on the joins?
  - No condition implies natural join

# Q2

- Here are two RA statements that finds all instructor ids (iid) who taught in terms F20 and W19. What is incorrect about each one?

1. Teaches / $\pi_{term}(\sigma_{term='F20' \vee term='W19'}(\text{Teaches}))$
   a.

1. $\pi_{iid}(\ \rho(T1, \text{Teaches}) \bowtie_{T1.iid=T2.iid \wedge T1.term='F20' \wedge T2.term='W19'} \rho(T2, \text{Teaches})$
   $)$
   b.

| iid | cid | term |
|---|---|---|
| 2 | EECS 575 | F20 |
| 3 | EECS 484 | F20 |
| 1 | EECS 482 | W19 |
| 2 | AERO 548 | W19 |
| 3 | EECS 484 | W19 |
| 4 | EECS 388 | W21 |

# Q2

- Here are two RA statements that finds all instructor ids (iid) who taught in terms F20 and W19. What is incorrect about each one?

1. Teaches / $\pi_{term}(\sigma_{term='F20' \vee term='W19'}(Teaches))$
   a. $\pi_{iid,\ term}(Teaches)$ / $\pi_{term}(\sigma_{term='F20' \vee term='W19'}(Teaches))$

1. $\pi_{iid}(\ \rho(T1, Teaches) \bowtie_{T1.iid=T2.iid \wedge T1.term='F20' \wedge T2.term='W19'} \rho(T2, Teaches)$
   )
   b.

| iid | cid | term |
|---|---|---|
| 2 | EECS 575 | F20 |
| 3 | EECS 484 | F20 |
| 1 | EECS 482 | W19 |
| 2 | AERO 548 | W19 |
| 3 | EECS 484 | W19 |
| 4 | EECS 388 | W21 |

# Q2

- Here are two RA statements that finds all instructor ids (iid) who taught in terms F20 and W19. What is incorrect about each one?

1. Teaches / $\pi_{term}(\sigma_{term='F20' \vee term='W19'}(Teaches))$
   a. $\pi_{iid,\ term}(Teaches)$ / $\pi_{term}(\sigma_{term='F20' \vee term='W19'}(Teaches))$

1. $\pi_{iid}(\ \rho(T1, Teaches) \bowtie_{T1.iid=T2.iid \wedge T1.term='F20' \wedge T2.term='W19'} \rho(T2, Teaches)\ )$
   b. $\pi_{T1.iid}(\ \rho(T1, Teaches) \bowtie_{T1.iid=T2.iid \wedge T1.term='F20' \wedge T2.term='W19'} \rho(T2, Teaches)\ )$

| iid | cid | term |
|---|---|---|
| 2 | EECS 575 | F20 |
| 3 | EECS 484 | F20 |
| 1 | EECS 482 | W19 |
| 2 | AERO 548 | W19 |
| 3 | EECS 484 | W19 |
| 4 | EECS 388 | W21 |

# Q2

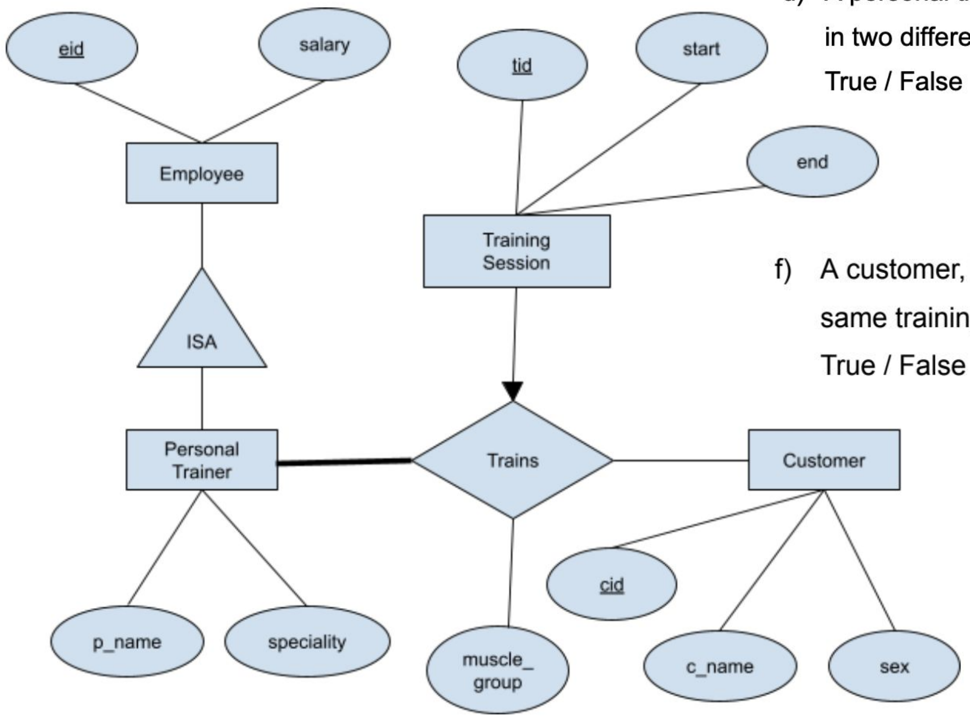
- Another solution using intersection if you want to take a look :)

$$(\pi_{iid}(\sigma_{term='F20'}(Teaches))) \cap (\pi_{iid}(\sigma_{term='W19'}(Teaches)))$$

| iid | cid | term |
|---|---|---|
| 2 | EECS 575 | F20 |
| 3 | EECS 484 | F20 |
| 1 | EECS 482 | W19 |
| 2 | AERO 548 | W19 |
| 3 | EECS 484 | W19 |
| 4 | EECS 388 | W21 |

# HW 1 Problem 3

The following ER diagram shows the relationships between a personal trainer and customers at a gym.




d) A personal trainer, Sam (eid=24), can train two customers Ellie (cid=18) and Riya (cid=7) in two different training sessions (tid=3 and tid=4).
True / False

f) A customer, Max (cid=9), can train two different muscle groups, legs and arms, in the same training session (tid=1) with the same personal trainer Gabe (eid=27).
True / False

# Get started with P1!

We're here if you need any help!!

- Office Hours: Schedule is here, both virtual and in person offered
- Piazza
- Next week's discussion!!!