2 – Relational Databases

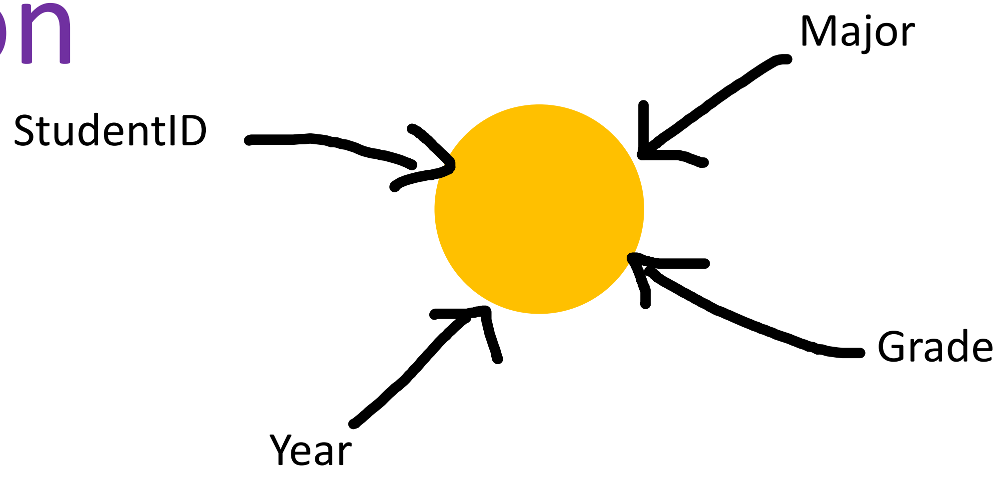
Week 10 Presentation 2

Relational Databases

Because all the attributes are related to a single event, the table is also known as a "relation", hence "relational theory of databases" and "relational database".

Stduent ID	Major	Grade	Year

A Relation



So…

- Table = Class
- Column = Attribute
- Row = Object or instance

… Right?

It is tempting to equate the database elements with the elements used in Object Oriented Programming. However this is misleading.

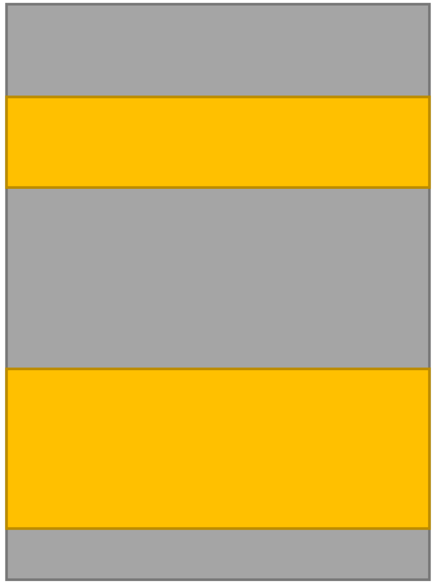
Big Mistake!

- That would be a very big mistake, unfortunately made by many people.
- Because in OO programming the focus is on the objects
- But in a relational database the focus is on the relations

OPERATE on relations

- Codd’s big idea was to define operations on sets to define subsets.
- He insisted that links between pieces of data in different sets should be established by common data values.
- No references – Link through values

Relational Operations



Codd defined several operations, here are the 3 most important ones

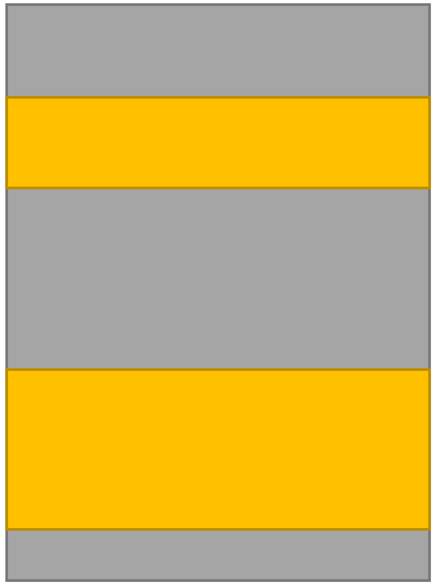
- Select
- Project
- Join

Select

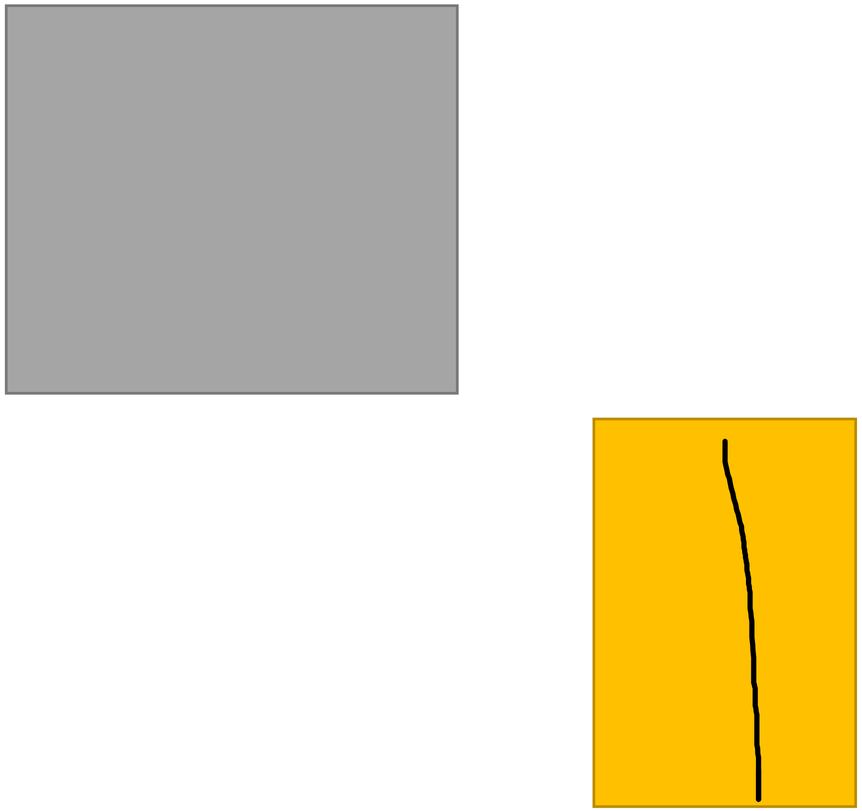
- Select rows with some particular values
- For example:

SELECT students with grade > X

(returns rows with grade more than X)

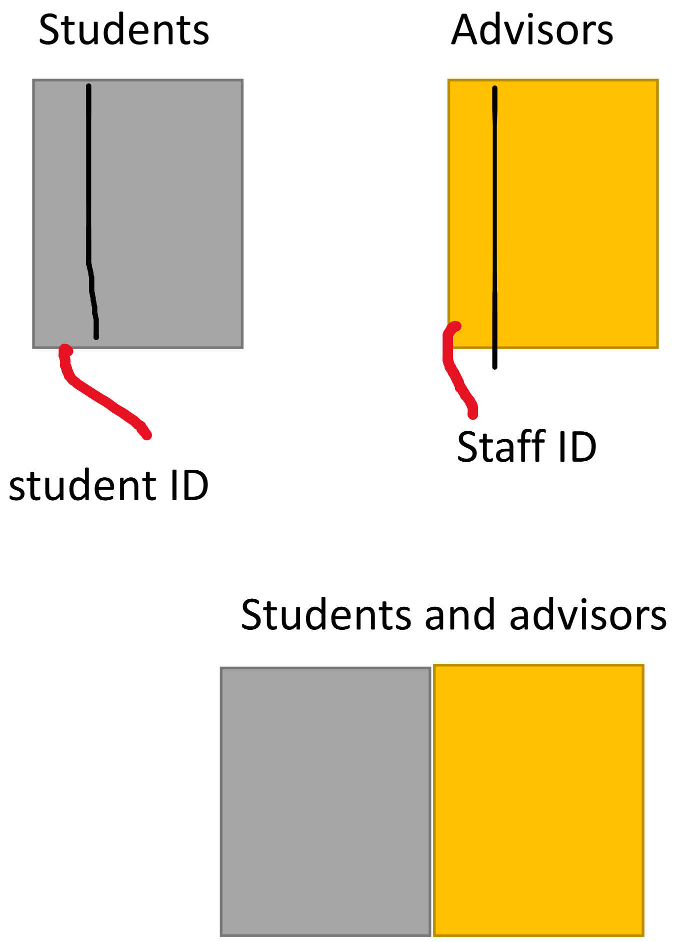


Project



- Get a subset of the columns
- In SQL a projection is done by selecting only certain columns

Join



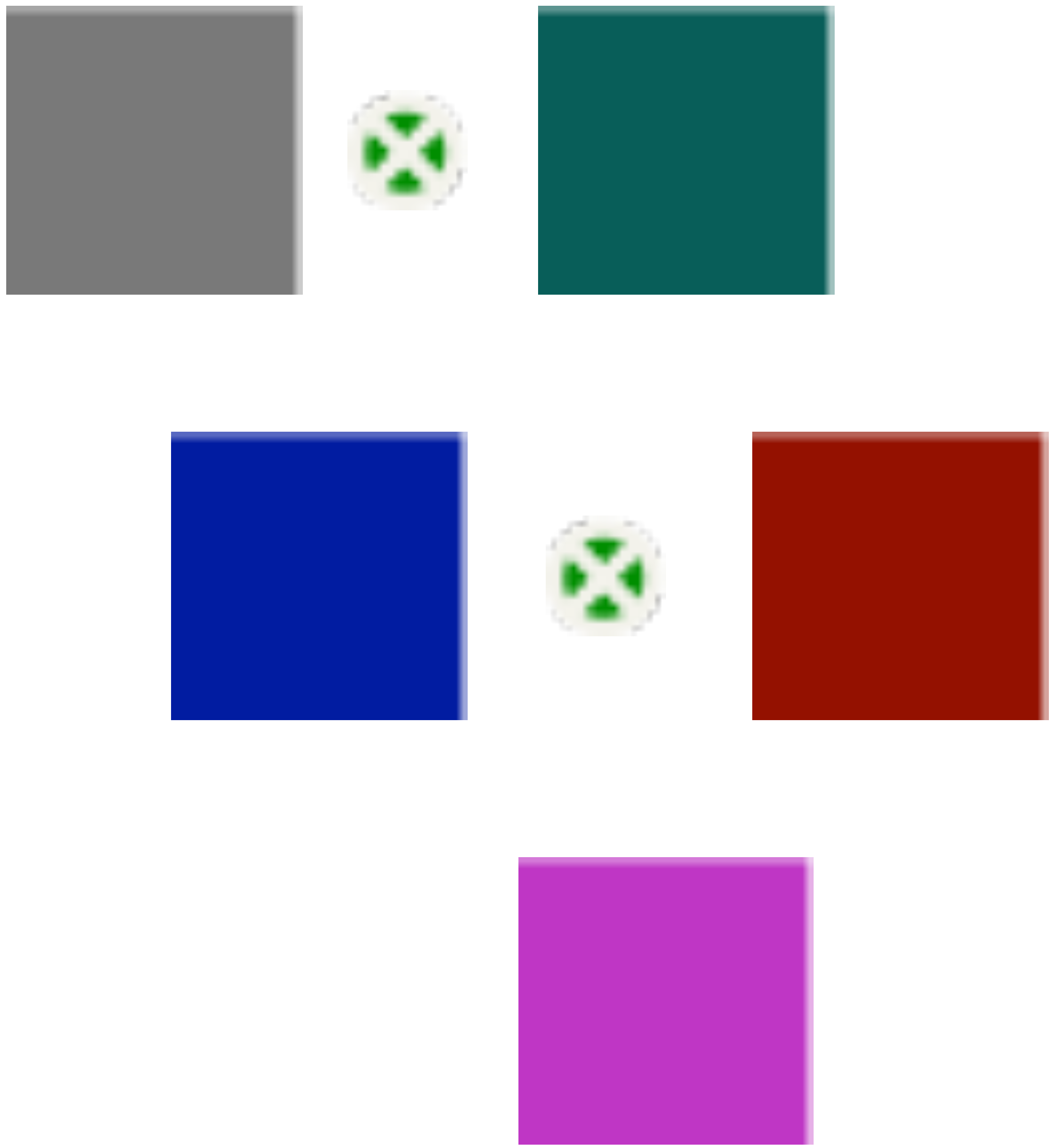
- Join 2 or more relations together
- For example here create a new relation with students and their advisors from two separate relations (students, advisors)
- The join depends on there being a key eg here the students table could have an advisor ID, then the joined relation would have additional information about the advisors for each student (from the advisor table)

Fundamental Concept

- Tables are like variables
- But instead of containing one value they contain one set

KEY POINT: Tables are Variables

And with operations you can combine sets to obtain a result set and then may combine this with further operations until you get exactly the result set that you want…



Good Modelling is Needed

One thing that is very important is that tables should be well designed (they must follow a number of rules, called normal forms). You don't want for instance data to be repeated many times, because it would make changes difficult. You must also know precisely what uniquely identifies a row (it may be all columns, but most often it's one or a few columns)

Entities = “Existing Things”

- Student
- Advisor
- Course

When you design one, you look for “entities”, things that exist independently of the others (a course can be on a catalogue without anyone taking it or teaching it). You must know what identifies each item: a code, a student/employee id, mail address, phone number may be good identifiers. A persons name is not good because several people could have the same name. You then have attributes of the entities (such as name). One entity will be one table.

Relationships

Then you can have relationships between entities – that link entities together. If an entity can be linked to only one other entity (for instance Student → Dormitory) it can be an attribute of an entity.

Student

Advisor

Course

Session

Key = combination of entity keys (StudentID, Advisor ID, Course ID)

Often it will be a table relating identifiers because a student takes many course and there are many students in a course.

Normalization

- Distinguishing between what is an attribute and what should be in a relation is often difficult.
- All this business of organizing tables is rather difficult and often underrated.
- If poorly done it can cause many problems.

What identifies a customer?

- Name
- Email
- Phone number
- Generated customer ID

If you wanted to make a database of customers of an ecommerce site you could potentially use a name (but not unique), an email or a phone, but you are not supposed to modify an identifier. So it may be better to generate an ID.

SQL – Structured Query Language

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

Databases are usually associated with a query language called SQL. That was invented in the early 1970s.

It was originally proposed in the paper "A Structured English Query Langauge" by Don Chamberlin and Ray Boyce in 1974: https://researcher.watson.ibm.com/researcher/files/us-dchamber/sequel-1974.pdf

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consis-

SQL

- SQL was designed to be a very simple language with a syntax similar to English that could be used by people who are not familiar with computers or programming.

  SELECT ..

  FROM …

  WHERE …

- It became at the same time a major success and a major failure: today only computer programmers use SQL – but almost all of them have to use it, and sometimes very often.

Two Main Components

1. SQL provides commands for managing tables (creating, dropping, modifying them)
2. SQL provides commands for managing data (inserting new rows, updating or deleting existing ones – plus of course commands for retrieving data that satisfies some criteria).

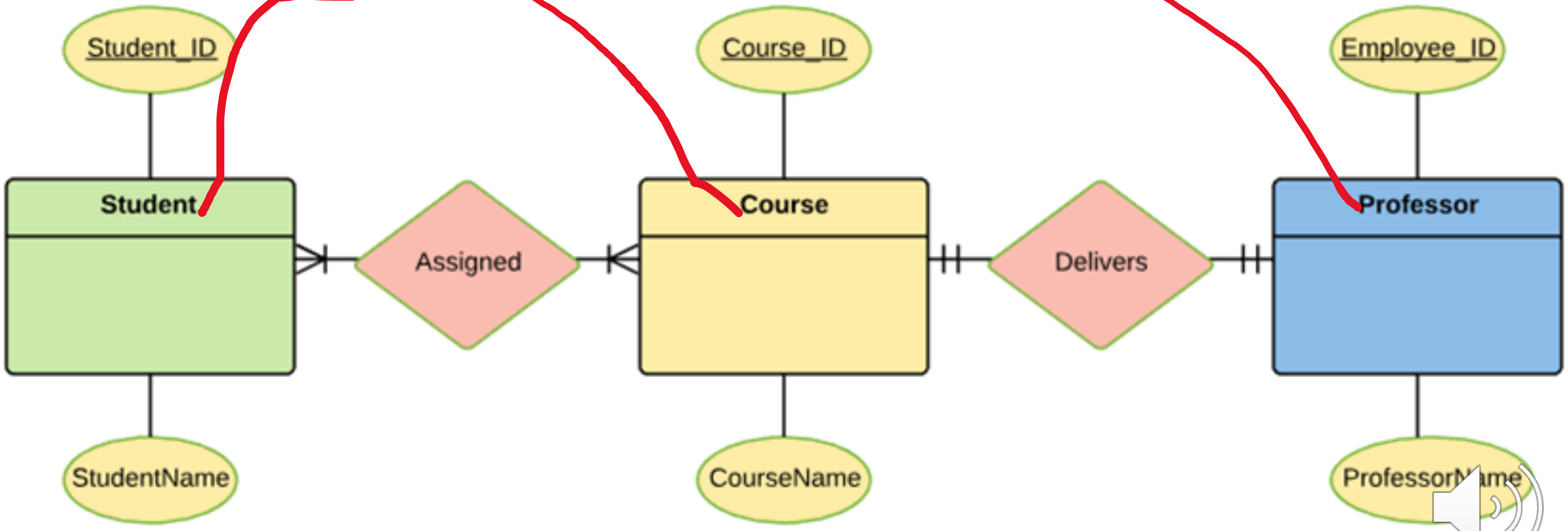
You can learn SQL syntax in about the same time as you can learn how chess pieces move. As with chess, things however quickly become complicated.

Simple Database

This small database represents Students, Courses and Professors.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

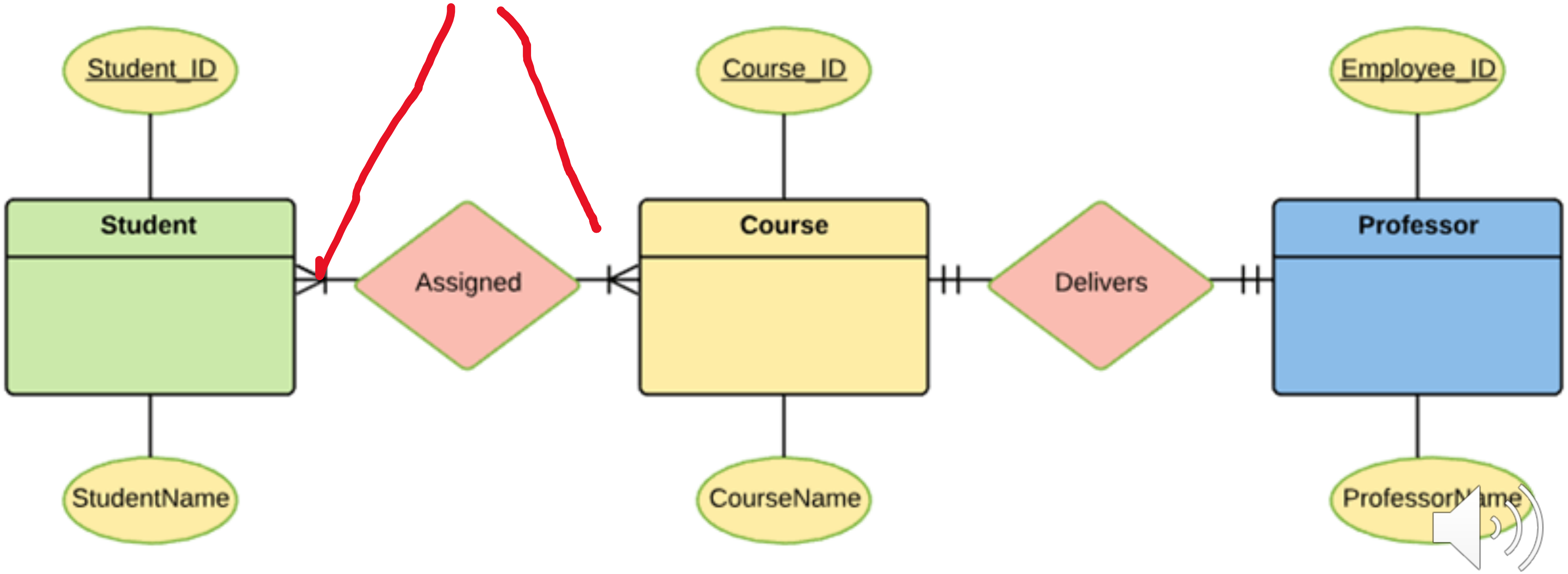
Entities



Simple Database

This small database represents
Students, Courses and Professors.

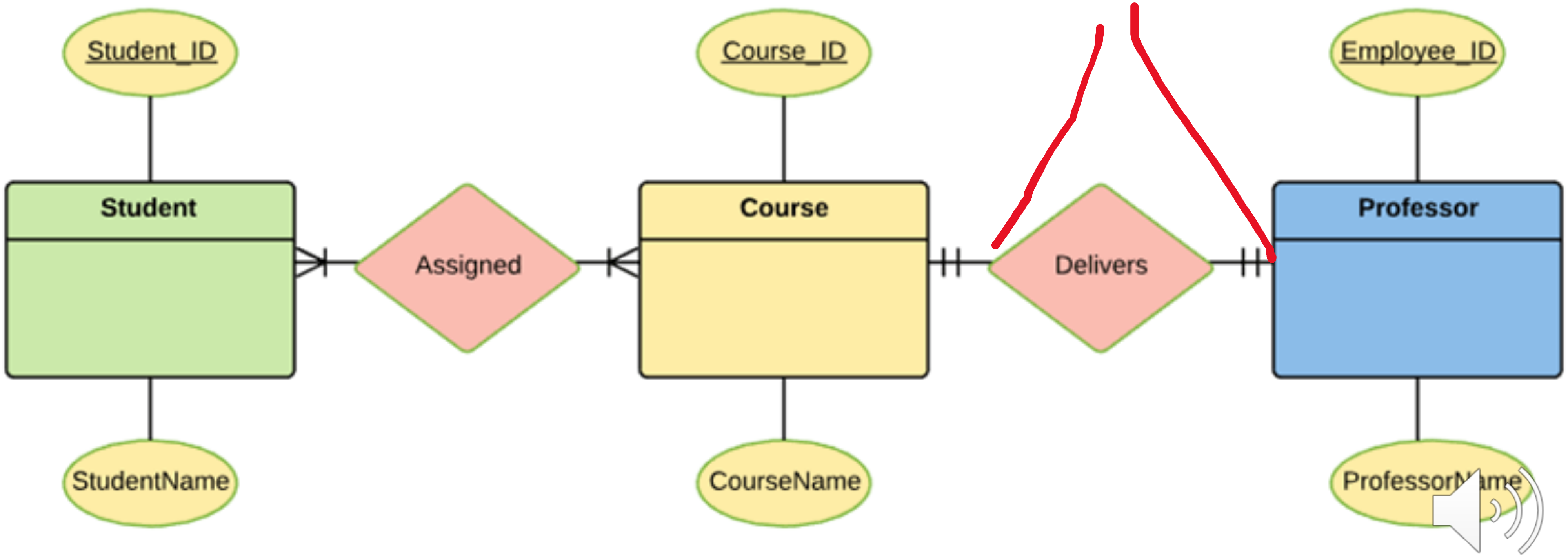
Many to Many



Simple Database

This small database represents
Students, Courses and Professors.

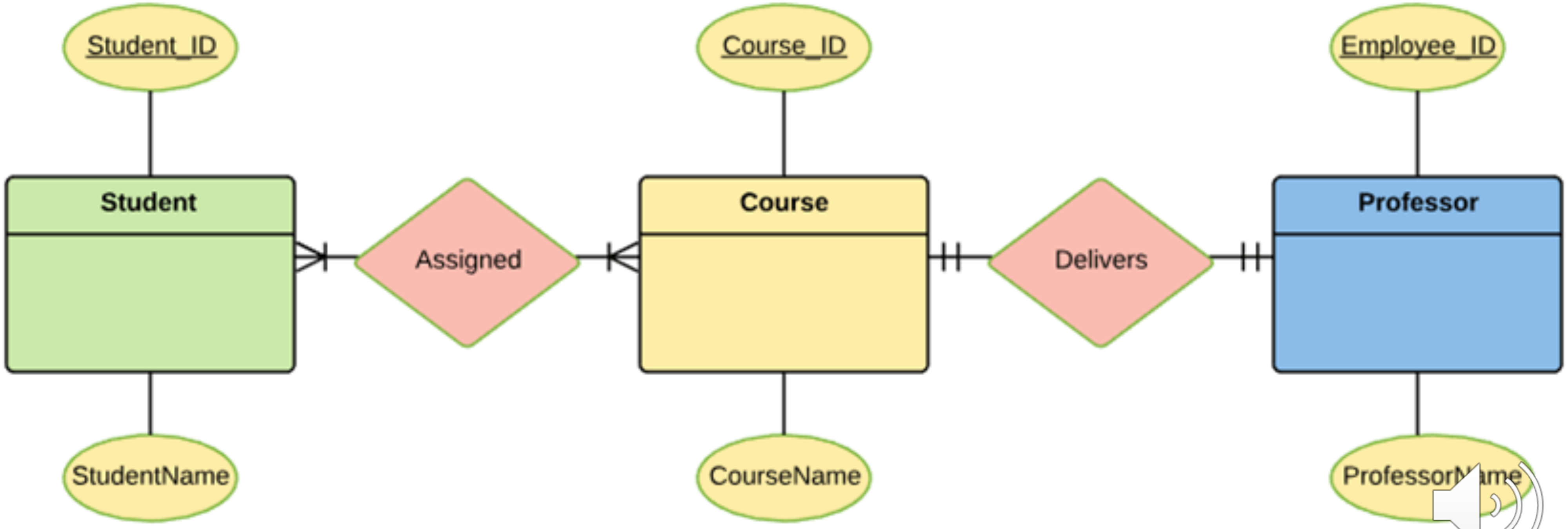
One to one

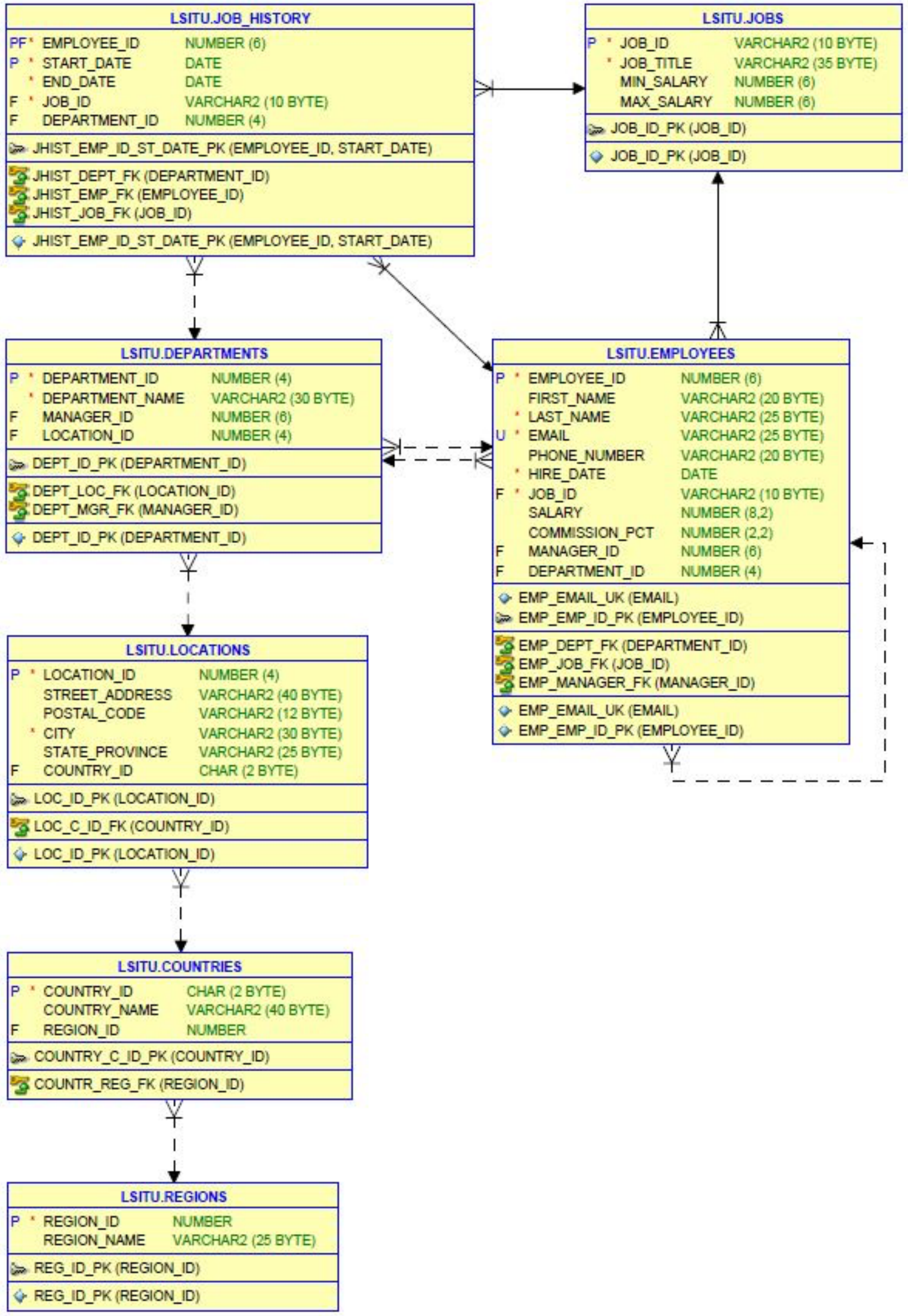


Simple Database

This small database represents Students, Courses and Professors.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



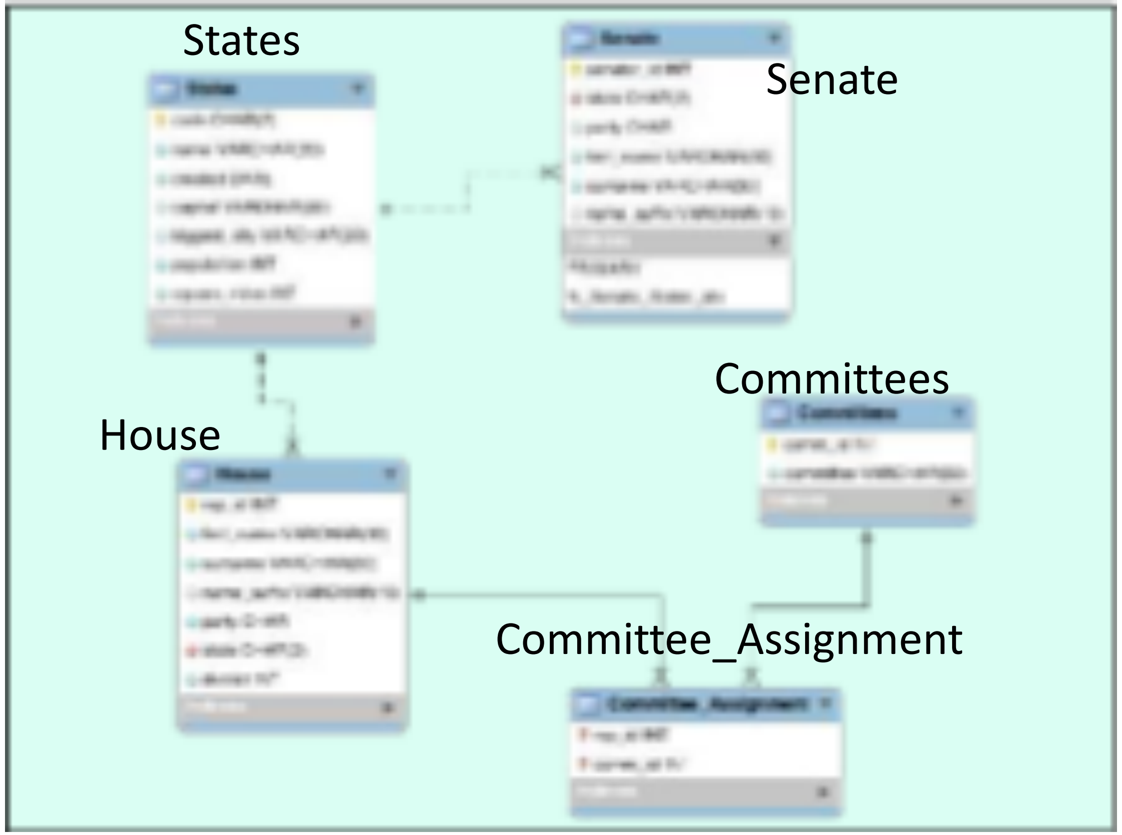


Here is another database which contains information about employees at a company. Not overly complex (7 tables with a few attributes in each).

And here is a query designed to fulfil a task that was "to display all employees and their related info even if some info is missing. Get as much information as you can about the employees".

```sql
SELECT
  e.employee_id AS "Employee #", e.first_name || ' ' || e.last_name AS "Name"
  , e.email AS "Email"  , e.phone_number AS "Phone"
  , TO_CHAR(e.hire_date, 'MM/DD/YYYY') AS "Hire Date"
  , TO_CHAR(e.salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') AS "Salary"
  , e.commission_pct AS "Comission %"
  , 'works as ' || j.job_title || ' in ' || d.department_name || ' department (manager: '
    || dm.first_name || ' ' || dm.last_name || ') and immediate supervisor: ' || m.first_name || ' ' || m.last_name AS "Curr
  , TO_CHAR(j.min_salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') || ' - ' ||
      TO_CHAR(j.max_salary, 'L99G999D99', 'NLS_NUMERIC_CHARACTERS = ''.,'' NLS_CURRENCY = ''$''') AS "Current Salary"
  , l.street_address || ', ' || l.postal_code || ', ' || l.city || ', ' || l.state_province || ', '
    || c.country_name || ' (' || r.region_name || ')' AS "Location"
  , jh.job_id AS "History Job ID"
  , 'worked from ' || TO_CHAR(jh.start_date, 'MM/DD/YYYY') || ' to ' || TO_CHAR(jh.end_date, 'MM/DD/YYYY') ||
    ' as ' || jj.job_title || ' in ' || dd.department_name || ' department' AS "History Job Title"
FROM employees e
-- to get title of current job_id
  JOIN jobs j
    ON e.job_id = j.job_id
-- to get name of current manager_id
  LEFT JOIN employees m
    ON e.manager_id = m.employee_id
-- to get name of current department_id
  LEFT JOIN departments d
    ON d.department_id = e.department_id
-- to get name of manager of current department
-- (not equal to current manager and can be equal to the employee itself)
  LEFT JOIN employees dm
    ON d.manager_id = dm.employee_id
-- to get name of location
  LEFT JOIN locations l
    ON d.location_id = l.location_id
  LEFT JOIN countries c
    ON l.country_id = c.country_id
  LEFT JOIN regions r
    ON c.region_id = r.region_id
-- to get job history of employee
  LEFT JOIN job_history jh
    ON e.employee_id = jh.employee_id
-- to get title of job history job_id
  LEFT JOIN jobs jj
    ON jj.job_id = jh.job_id
-- to get namee of department from job history
  LEFT JOIN departments dd
    ON dd.department_id = jh.department_id
ORDER BY e.employee_id;
```

If you want read more about this query and an explanation here (the purpose of the slide is just to give an example of how a query in SQL can become complex…
https://dev.to/tyzia/example-of-complex-sql-query-to-get-as-much-data-as-possible-from-database-9he



is small database represents for instance part of the US Congress, with the Senate and House of Representatives.

Who participates in the greatest number of House Committees ?

```sql
select h.first_name, h.surname,
            s.name as state, z.num_of_committees
from (select rep_id,
        count(*) as num_of_committees
    from committee_assignment group by rep_id
    having count(*) =
        (select max(num_of_committees) from (select rep_id,
        count(*) as num_of_committees from committee_assignment
        group by rep_id) x)) z
    join house h
        on h.rep_id = z.rep_id
    join states s
        on s.code = h.state
by h.surname, h.first_name, s.name

```

Next

Accessing databases in java