# Basic SQL DML (no joins)

followup

CS383  Jan 31

# Another look at how select works

- You can think of select as defining, building and refining a relation based on other relations.
- After each stage of select, there is a valid relation
  - Unlikely that this is actually how the query executed internally … Why?   Do I care?

```
SELECT selection_list  # Define what the columns in the relation will be

FROM table_list        # fill in the columns from the listed tables
                       # Does cross product if there are multiple tables

WHERE constraint       # Select the rows in the temp table after FROM completes
                       # such that the rows match the given constraint

GROUP BY columns       # groups the remaining rows by the given columns
    HAVING group constraints     # select the grouped rows by the constraint

ORDER BY sorting_cols  # Order the remaining rows by the given columns

LIMIT count;           # Limit on results
```

# Data types

## in SQL standard

`bigint`,
`bit`,
`bit varying`,
`boolean`,
`char(n)`,
`character varying (n)`,
`character(n)`,
`varchar(n)`,
`date`,
`double precision`,
`integer`,
`interval`,
`numeric(p,d)`,
`decimal`,
`real`,
`smallint`,
`time` (with or without time zone),
`timestamp` (with or without time zone),
`xml`.

## In PostgreSQL (not in standard)

| Name | Aliases | Description |
|---|---|---|
| `bigserial` | `serial8` | autoincrementing eight-byte integer |
| `box` | | rectangular box on a plane |
| `bytea` | | binary data ("byte array") |
| `cidr` | | IPv4 or IPv6 network address |
| `circle` | | circle on a plane |
| `inet` | | IPv4 or IPv6 host address |
| `json` | | textual JSON data |
| `jsonb` | | binary JSON data, decomposed |
| `line` | | infinite line on a plane |
| `lseg` | | line segment on a plane |
| `macaddr` | | MAC (Media Access Control) address |
| `money` | | currency amount |
| `path` | | geometric path on a plane |
| `pg_lsn` | | PostgreSQL Log Sequence Number |
| `point` | | geometric point on a plane |
| `polygon` | | closed geometric path on a plane |
| `smallserial` | `serial2` | autoincrementing two-byte integer |
| `serial` | `serial4` | autoincrementing four-byte integer |
| `text` | | variable-length character string |
| `time [ (`*`p`*`) ] [ without time zone ]` | | time of day (no time zone) |
| `tsquery` | | text search query |

# Null

- Every data type can have a value or null
  - ie no data -- the value is unknown
    - in DDL can specify that null is NOT allowed
    - special operator "is" for handling null
- SQL null is not Java null

or "A is not B"

| A | B | A = B | not A = B | A is B | not A is B |
|---|---|---|---|---|---|
| **1** | **1** | TRUE | FALSE | error | error |
| **0** | **1** | FALSE | TRUE | error | error |
| **1** | **null** | null | null | FALSE | TRUE |
| **0** | **null** | null | null | FALSE | TRUE |
| **null** | **null** | null | null | TRUE | FALSE |
| **null** | **1** | null | null | error | error |

# Using alaised columns

## in the univ database

- Recall aliased columns
  - select salary as s, name from instructor;
  - ~~select salary as s, name from instructor where s>100000~~; /// DOES NOT WORK!!!!
    - cannot use alias within the query, at least in this way.
  - select salary as s, name from instructor where salary>100000;

  - aliases in the output relations are not available, Aliases in subrelations are
    - select salary, name from instructor, (select avg(salary) as av from instructor) as ie where salary>ie.av;
    - select salary, name from instructor where salary> (select avg(salary) as av from instructor);
    - select * from (select salary as s, name from instructor) as inn, (select avg(salary) as av from instructor) as ie where inn.s>ie.av;

# Formatting Numbers

## and column names

- select s as "hello kitty", name,av::numeric(8,2) as "the average" from (select salary as s, name from instructor) as inn, (select avg(salary) as av from instructor) as ie where inn.s>ie.av;
  - for reals, cast into numeric and specify format
  - In column name alias use double quotes to get spaces
    - Postgres: double quotes only on top level column names

- Realistically, I get the data into python ….

## Grouping problems

- get list of all people who earn max salary in their department

```
select max(salary), dept_name, name from instructor group by dept_name;

ERROR:  column "instructor.name" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: select avg(salary), dept_name, name from instructor group by...
```

  - this query does not work
    - WHY?

# Grouping 2

## temporary relations  the WITH clause

- get list of all people who earn max salary in their department
- with aaa as (select max(salary) as maxx, dept_name from instructor group by dept_name) select * from aaa, instructor where aaa.dept_name=instructor.dept_name and salary=maxx;

- With is often unneeded
  - select * from (select max(salary) as maxx, dept_name from instructor group by dept_name)  as aaa, instructor where aaa.dept_name=instructor.dept_name and salary=maxx;
- The more complex the query the more I like with

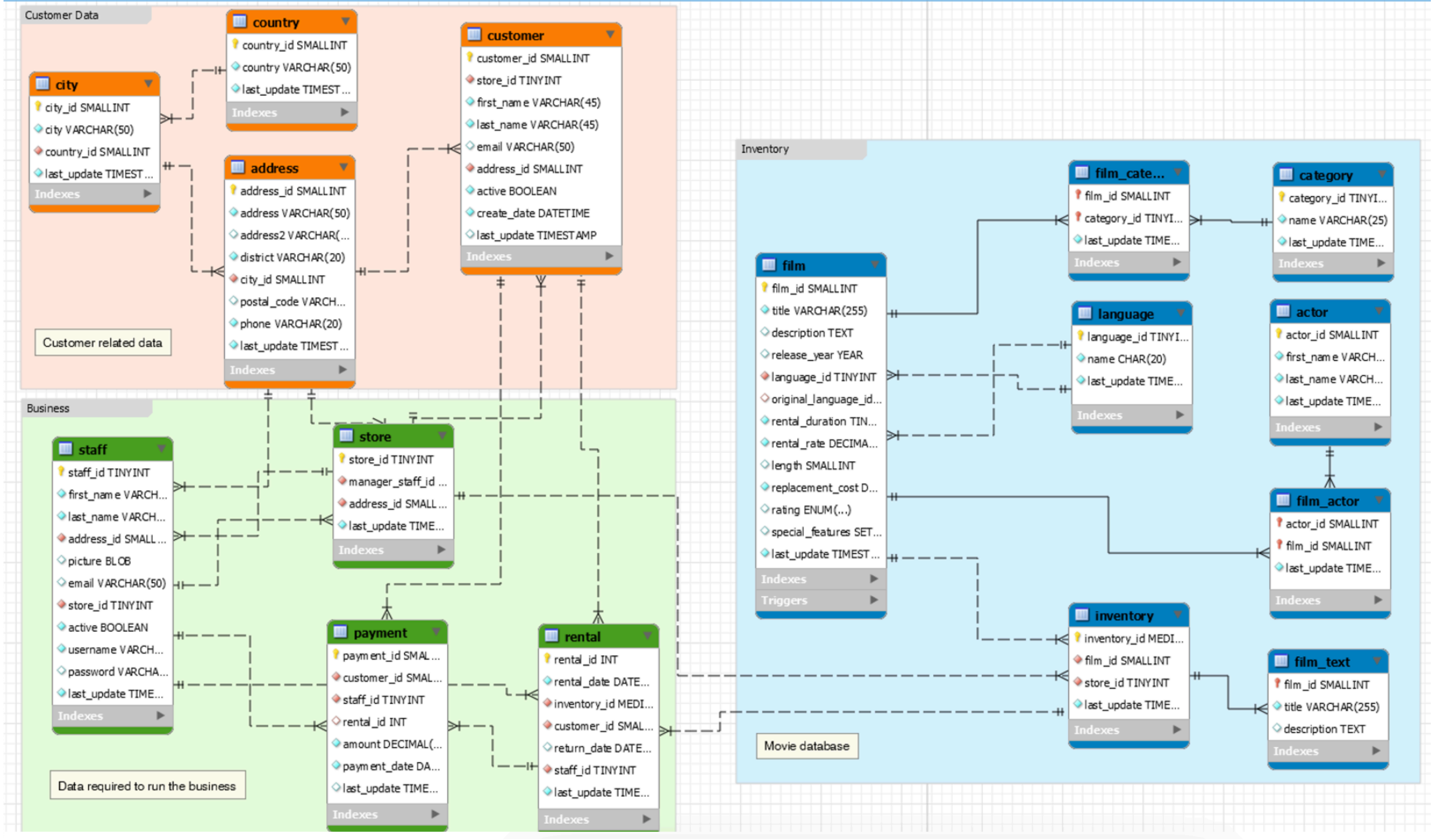
## With and Having

**Or having is just feels weird …**

- Show all departments whose average salary is greater than the university average
- University average: select avg(salary) from instructor;
- select avg(salary), dept_name from instructor group by dept_name having avg(salary) > (select avg(salary) from instructor);

- Use with to avoid having
  - alternately, to precisely explain having.
- with aaa as (select avg(salary) as avg, dept_name from instructor group by dept_name) select * from aaa where avg>(select avg(salary) from instructor);

# Sakila Database



https://codeandwork.github.io/courses/java/sqlJoins.html

## Sakila database

### \c[onnect] sakila

- how many rows are then in the film_actor table
- retrieve the actor id, first name and last name for all actors. Sort by last name, first name
    - use the actor table
  - show only 10 actors
  - show only actors whose last names begin with Z
- retrieve actor id, first name, last name for all actors whose last name equal WILLIAMS or DAVIS
- retrieve all customers whose first name has 2 D.
- Get the customer ID for all customers who rented a film on July 5, 2005 (use the rental table).
  - use date_part function
- from the film table, rank films by first letter, alphabetically within that