# The World of SQL

Part 2: Queries

# Heads up

Relational databases are an older, pre-Web technology. So, the syntax here is unlike anything we've done before.

# Content

- SQL
- Queries

# SQL

# What is SQL?

- Structured Query Language
- The language we use to query relational databases

# Terms

| SQL term | Equivalent Mongo term | Meaning |
|---|---|---|
| Record | Document | A single instance of an entity |
| Table | Collection | A collection of records |
| Field | Field | An attribute of the record |

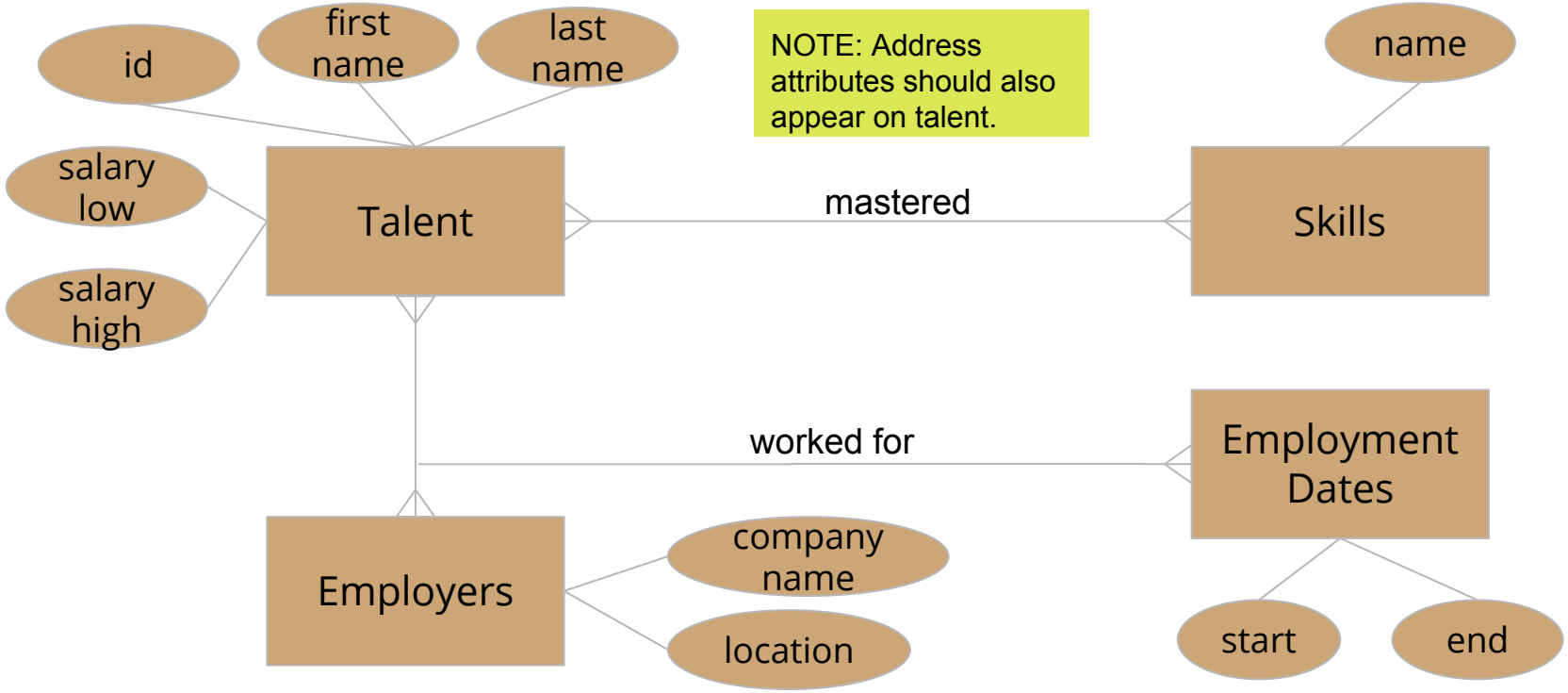
# Queries

# Approach

We will use our talent entity-relationship diagram to create the Rando database.

In the interest of time, we will create the database structure for only two of our entities: talent and skills.

# Remember: Our ER model



# Setup

1. Open a terminal window
2. Type `postgres` to start our Postgres server
3. Open a new terminal window
4. Type `psql` to start the Postgres command line client

# Part A: Creation and insertion queries

# Steps

We will

- Create a database
- Create 2 tables
  - talent for our talent data
  - skills for our skills data
- Insert records into each table

# Create a database

```
CREATE DATABASE rando;
```

# Connect to database

\c rando

# Create talent table

```
CREATE TABLE talent (
    talent_id serial PRIMARY KEY,
    first_name varchar(80),
    last_name varchar(80),
    city varchar(40),
    state varchar(2),
    salary_low int,
    salary_high int
);
```

# Aside: Common data types

| Name | Aliases | Description |
|---|---|---|
| boolean | bool | logical Boolean (true/false) |
| character varying | varchar(n) | variable-length character string |
| date | | calendar date (year, month, day) |
| integer | int, int4 | signed four-byte integer |
| timestamp | | date and time |

More data types for PostgreSQL

# Aside: Keywords

- PRIMARY KEY
  - Uniquely identifies a record in a table
  - Cannot be null
  - Helps database locate a record faster
- serial
  - An int that auto-increments
  - Postgres-specific keyword

# Create skills table

```
CREATE TABLE skills (
    skill_id serial PRIMARY KEY,
    name VARCHAR(80)
);
```

# Insert records into talent

```
INSERT INTO talent(first_name, last_name, city, state, salary_low,
salary_high)
    VALUES('Michael', 'Jordan' 'Chicago', 'IL', 250000, 500000),
    ('Aimee', 'Mann', 'Portland', 'OR', 100000, 200000);
```

# Insert records into skills

```
INSERT INTO skills(name)
    VALUES('basketball'),
    ('singing'),
    ('acting'),
    ('entrepreneurship');
```

# Part B: Find queries

# Details

To become familiar with SQL's syntax for retrieving data, we will demonstrate

- finding all records in a table
- finding only certain records in a table
- sorting records
- limiting records retrieved

# Find all records in talent

```
SELECT *
    FROM talent;
```

# Find all records in talent

```
SELECT * FROM talent
    WHERE first_name = 'Aimee';
```

# Sorting records in skill

```
SELECT * FROM skill
    ORDER BY name ASC;
```

# Limiting records in skill

```
SELECT * FROM skill
    ORDER BY name ASC
    LIMIT 3;
```

# Part C: Implementing many-to-many relationships

# Rule of thumb

The implementation of a many-to-many relationship between 2 entities typically requires 3 tables.

# Create talent_skill table

```
CREATE TABLE talent_skill (
    talent_id int,
    skill_id int,
    PRIMARY KEY(talent_id, skill_id)
);
```

# Aside: Composite keys

Sometimes a record can only be uniquely identified via a combination of fields.

# Part C: Leveraging relationships

# Details

We will use our knowledge of the relationships between entities to make queries that span tables.

We will

- Find all the skills that belong to talent
- Find all the skills that belong to a specific talent

# Find all skills for all talent (first pass)

```
SELECT * FROM talent
    JOIN talent_skill ON talent.talent_id = talent_skill.
talent_id;
-- This gets us the skill ids, but not the skill names...
```

# Find all skills for all talent (second pass)

```
SELECT * FROM talent
    JOIN talent_skill ON talent.id = talent_skill.id
    JOIN skills on talent_skill.id = skills.id;
-- This gets us the skill names in addition to the ids
```

# Find all skills for all talent (third pass)

```
SELECT talent.first_name, talent.last_name, skills.name
    FROM talent
    JOIN talent_skill ON talent.id = talent_skill.id
    JOIN skills on talent_skill.id = skills.id;
-- We specify field names to get only the data we need
```

# Find all skills for a specific talent

```
SELECT talent.first_name, talent.last_name, skills.name
    FROM talent
    JOIN talent_skill ON talent.id = talent_skill.id
    JOIN skills on talent_skill.id = skills.id
    WHERE talent.last_name = 'Jordan';
```

# Resources

# psql setup

If psql won't work for you, try the following:

1. Make sure postgres is running.
2. In a new terminal window, type `createdb` (to create a database with your username).
3. Type `createuser -s postgres` to create the postgres role.

# Useful psql commands

`\l` list databases

`\dt` list tables

# Useful links

- SQL Commands -- one of many places to get info on SQL commands
- psql reference

# If you want more info...

- SQL indexes

# Peer Challenge

# Modes

- Regular
- Hard

# Regular mode

# Task 1 of 2

Create the database structure to support Rando's company data.

Things you will need to do

- Create database
- Create tables
  - company
  - opportunities
- Insert 2-3 records in each table

# Task 2 of 2

Perform the following queries.

- Find all opportunities for all companies.
- Find all opportunities for a specific company.
- Sort all companies by name descending.

# Hard mode

# Task

Implement the remainder of the talent data as based on my E-R diagram.