SQL review

• primarily declarative query language
• starting point: relational calculus
  aka first-order predicate logic
• many additions, bells and whistles…
• corresponding procedural language: relational algebra

will discuss relational calculus and algebra later
Running example: Movie database

<table>
<thead>
<tr>
<th>Movie</th>
<th>Title</th>
<th>Director</th>
<th>Actor</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Theater</th>
<th>Title</th>
</tr>
</thead>
</table>
Running example: Movie database

<table>
<thead>
<tr>
<th>Movie</th>
<th>Title</th>
<th>Director</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>Lucas</td>
<td>Lucas</td>
<td>Ford</td>
</tr>
<tr>
<td>Star Wars</td>
<td>Lucas</td>
<td>Lucas</td>
<td>Fischer</td>
</tr>
<tr>
<td>Mad Max</td>
<td>Miller</td>
<td>Miller</td>
<td>Hardy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Theater</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hillcrest</td>
<td>Star Wars</td>
<td></td>
</tr>
<tr>
<td>Hillcrest</td>
<td>Mad Max</td>
<td></td>
</tr>
<tr>
<td>Paloma</td>
<td>Rocky Horror</td>
<td></td>
</tr>
</tbody>
</table>

.................................
Find titles of currently playing movies
SELECT  s.Title
FROM Schedule s

Find the titles of all movies by “Lucas”
SELECT  m.Title
FROM Movie m
WHERE m.Director=“Lucas”

Find the titles and the directors of all currently playing movies
SELECT  m.Title, m.Director
FROM Movie m, Schedule s
WHERE m.Title=s.Title
Basic form

SELECT $t_{i1}.A_{i1}, \ldots, t_{in}.A_{in}$
FROM $R_1 t_1 , \ldots, R_m t_m$
WHERE $condition(t_1 , \ldots, t_m)$

WHERE clause is optional
Informal semantics of basic form

\[ \text{SELECT } t_{i1}.A_{i1}, \ldots, t_{in}.A_{in} \]
\[ \text{FROM } R_1 t_1, \ldots, R_m t_m \]
\[ \text{WHERE } \text{condition}(t_1, \ldots, t_m) \]

\[(a_{i1}, \ldots, a_{in}) \text{ is in the answer iff } \]
\[\text{there exist tuples } t_1 \in R_1, \ldots, t_m \in R_m \]
\[\text{for which } \text{condition}(t_1, t_2, \ldots, t_m) \text{ is satisfied and } \]
\[a_{i1} = t_{i1}(A_{i1}), \ldots, a_{in} = t_{in}(A_{in}) \]
Informal semantics of basic form

**SELECT** \( t_{i1}.A_{i1}, \ldots, t_{in}.A_{in} \)
**FROM** \( R_1 t_1, \ldots, R_m t_m \)
**WHERE** \( \text{condition}(t_1, \ldots, t_m) \)

Simplest case:
\( \text{condition}(t_1, \ldots, t_m) \) is a Boolean combination of comparisons on the attribute values of \( t_1, \ldots, t_m \)

Note: such a query \( Q \) is **monotonic**: \( \text{DB1} \subseteq \text{DB2} \implies Q(\text{DB1}) \subseteq Q(\text{DB2}) \)
SQL Queries: Nesting

- The WHERE clause can contain predicates of the form
  - attr/value IN <SQL query>
  - attr/value NOT IN <SQL query>
- The IN predicate is satisfied if the attr or value appears in the result of the nested <SQL query>

Examples:
find directors of current movies

SELECT m.director FROM Movie m
WHERE m.title IN (SELECT s.title FROM schedule s)

The nested query finds currently playing movies
Simple use of $\text{NOT IN}$

*Find all movies that are not currently playing:*

```
SELECT  m.title FROM Movie m
WHERE m.title NOT IN
       (SELECT s.title FROM Schedule s)
```
Simple use of **NOT IN**

*Find all movies in which Hitchcock does not act:*

```
SELECT  m.title FROM Movie m
Where  m. title NOT IN
    (SELECT t.title FROM Movie t
     WHERE t. actor = 'Hitchcock')
```
More complex use of **NOT IN**

*Find the names of employees with the maximum salary*

```sql
SELECT name FROM Employee
WHERE salary NOT IN
    (SELECT e.salary
     FROM Employee e, Employee f
     WHERE e.salary < f.salary)
```

Intuition: salary is maximum if it is **not** among salaries e.salary lower than some f.salary
More complex use of **NOT IN**

*Find the names of employees with the maximum salary*

```
SELECT  name FROM Employee
WHERE salary NOT IN
    (SELECT  e.salary
     FROM Employee e, Employee f
     WHERE   e.salary < f.salary)
```

**Intuition:** salary of e is maximum if for every employee f e.salary \( \geq \) f.salary there does not exist an employee f such that e.salary < f.salary
More complex use of **NOT IN**

*Find the names of employees with the maximum salary*

```
SELECT name FROM Employee
WHERE salary NOT IN
  (SELECT e.salary
   FROM Employee e, Employee f
   WHERE e.salary < f.salary)
```

**Intuition:** salary of e is maximum if

\[ \forall \text{employee } f \ ( e.\text{salary} \geq f.\text{salary}) \]

\[ \neg \exists \text{employee } f \ ( e.\text{salary} \geq f.\text{salary}) \]

\[ \neg \exists \text{employee } f \ ( e.\text{salary} < f.\text{salary}) \]
More complex use of \texttt{NOT IN}:

\textit{Find actors playing in every movie by \textquote{\textquote{Berto}}}
Aggregate Functions

These functions operate on the multiset of values of a column of a relation, and return a value

- **avg**: average value
- **min**: minimum value
- **max**: maximum value
- **sum**: sum of values
- **count**: number of values
## Aggregate (example)

### Find average salary of all employees

```sql
SELECT Avg(Salary) AS AvgSal
FROM Employee
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Toys</td>
<td>45</td>
</tr>
<tr>
<td>Nick</td>
<td>PCs</td>
<td>50</td>
</tr>
<tr>
<td>Jim</td>
<td>Toys</td>
<td>35</td>
</tr>
<tr>
<td>Jack</td>
<td>PCs</td>
<td>40</td>
</tr>
</tbody>
</table>

AvgSal 42.5
Grouping (example)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Toys</td>
<td>45</td>
</tr>
<tr>
<td>Nick</td>
<td>PCs</td>
<td>50</td>
</tr>
<tr>
<td>Jim</td>
<td>Toys</td>
<td>35</td>
</tr>
<tr>
<td>Jack</td>
<td>PCs</td>
<td>40</td>
</tr>
</tbody>
</table>

Find the average salary for each department

SELECT Dept, Avg(Salary) AS AvgSal
FROM Employee
GROUP BY Dept

<table>
<thead>
<tr>
<th>Dept</th>
<th>AvgSal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toys</td>
<td>40</td>
</tr>
<tr>
<td>PCs</td>
<td>45</td>
</tr>
</tbody>
</table>