6.170: Software Studio

Relational Databases: SQL
(material adapted from Jennifer Widom's MOOC)

Arvind Satyanarayan +
Daniel Jackson
**Object Model**

Application root references collections of class instances that describe primitive data.

- Quick to prototype.
- Easy to experiment with arbitrary data structures.
- No advanced querying: can only iterate over collections, follow references.
- Refactoring is difficult.

---

**Diagram:**

- **App**
  - showings
  - movies
  - theaters

- **Array**
  - showing
  - movie
  - theater

- **Object**
  - Movie
    - title: "Crazy Rich Asians"
    - rating: 0.93
    - genre: "RomCom"
  - Theater
    - name: "Regal"
    - location: "Fenway"
    - showings

- **Showing**
  - screen: 10
  - time: 7:00pm
NoSQL

"Not Only SQL"

Collections of nested documents (or graph structures).

✓ Quick to prototype (documents stored as JSON).
✓ Easy to experiment with arbitrary data structures.
✓ Pattern matching by document structure.
✓ High scalability (horizontally with primary/replicas) and performance (relaxed robustness tests).

✗ No standardized query language.
✗ (Until recently) no references between collections: complexity of lookups occurs at the application level.
✗ Embedded documents = easier to make poor design decisions.
Relational Model (SQL)

Relations (aka tables) of attributes (aka columns) and tuples (aka rows).

✓ Standardized query language (SQL) regardless of backend engine (MySQL, PostgreSQL, SQLite, ...).

✓ Relational theory encourages better separation of concerns (called "normalization").

✓ Over 40 years of research into performance and robustness (indexing, transactions, integrity, ...).

✘ (Until recently) did not offer JSON types.

✘ (Until recently) difficult to scale horizontally. Vertical scaling (i.e., make machine more powerful) was the easiest option.

### Showings

<table>
<thead>
<tr>
<th>id</th>
<th>theater</th>
<th>screen</th>
<th>movie</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>7:00pm</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Theaters

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>&quot;Regal&quot;</td>
<td>&quot;Fenway&quot;</td>
</tr>
</tbody>
</table>

### Movies

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>“Crazy Rich”</td>
<td>“PG-13”</td>
<td>“RomCom”</td>
</tr>
</tbody>
</table>
Relational Model (SQL)

A database consists of a set of **relations** (or tables).
Each relation has a set of named **attributes** (or columns).
Each **tuple** (or row) has a value for every attribute.
Attributes have a specific type, and that type is enforced for all tuples in the relation. **NULL** is a special value.

**Key**: attribute (or set of attributes) whose value is unique for every tuple. Useful to identify (or find) a tuple or reference tuples in another relation.

This structural description of a database is called the **schema** (Wednesday’s lecture).

### Showings

<table>
<thead>
<tr>
<th>id</th>
<th>theater</th>
<th>screen</th>
<th>movie</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>7:00pm</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Theaters

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>&quot;Regal&quot;</td>
<td>&quot;Fenway&quot;</td>
</tr>
</tbody>
</table>

### Movies

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>“Crazy Rich”</td>
<td>“PG-13”</td>
<td>“RomCom”</td>
</tr>
</tbody>
</table>
```sql
select attr_1, attr_2, ..., attr_n
from rel_1, rel_2, ..., rel_n
where condition
```
```
select attr_1, attr_2, ..., attr_n
from rel_1, rel_2, ..., rel_n
where condition
```

**Subqueries in where**

- `where attr_1 in (select attr_1, ..., from rel_1, ... where condition)`
- `where attr_1 exists (select attr_1, ..., from rel_1, ... where condition)`

**Subqueries in from**

```
from rel_1, ..., (select attr_1, from rel_1, ..., where condition) as R
```
Aggregation functions
min, max, sum, avg, count

select attr₁, attr₂, ..., attrₙ
from rel₁, rel₂, ..., relₙ
where condition
group by attr₁, attr₂, ..., attrₙ
having condition
insert into rel
values(attr_1, attr_2, ..., attr_n)

insert into rel
select-statement
delete from rel
where condition
update rel
set attr_1 = expr_1, ..., attr_n = expr_n
where condition