Data Structure: Relational Model

- Relational databases: Schema + Data
- Schema (also called scheme):
  - collection of tables (also called relations)
  - each table has a set of attributes
  - no repeating relation names, no repeating attributes in one table
- Data (also called instance):
  - set of tuples
  - tuples have one value for each attribute of the table they belong

<table>
<thead>
<tr>
<th>Movie</th>
<th>Title</th>
<th>Director</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wild</td>
<td>Lynch</td>
<td>Winger</td>
<td></td>
</tr>
<tr>
<td>Sky</td>
<td>Berto</td>
<td>Winger</td>
<td></td>
</tr>
<tr>
<td>Reds</td>
<td>Beatty</td>
<td>Beatty</td>
<td></td>
</tr>
<tr>
<td>Tango</td>
<td>Berto</td>
<td>Brando</td>
<td></td>
</tr>
<tr>
<td>Tango</td>
<td>Berto</td>
<td>Winger</td>
<td></td>
</tr>
<tr>
<td>Tango</td>
<td>Berto</td>
<td>Snyder</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Theater</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Odeon</td>
<td>Wild</td>
<td></td>
</tr>
<tr>
<td>Forum</td>
<td>Reds</td>
<td></td>
</tr>
<tr>
<td>Forum</td>
<td>Sky</td>
<td></td>
</tr>
</tbody>
</table>

Programming Interface: JDBC/ODBC

- How client opens connection with server
- How access & modification commands are issued
- ...

SQL Queries: The Basic From

- Basic form:
  SELECT a1, ..., aN
  FROM R1, ..., RM
  WHERE condition
- WHERE clause is optional
- When more than one relation of the FROM have an attribute named A we refer to a specific A attribute as <RelationName>.A

Find titles of currently playing movies
SELECT Title
FROM Schedule

Find the titles of all movies by “Berto”
SELECT Title
FROM Schedule
WHERE Director=“Berto”

Find the titles and the directors of all currently playing movies
SELECT Movie.Title, Director
FROM Movie, Schedule
WHERE Movie.Title=Schedule.Title
SQL Queries: Aliases

- Use the same relation more than once in the FROM clause
- Tuple variables
- Example: find actors who are also directors
  ```sql
  SELECT t.Actor
  FROM Movie t s
  WHERE t.Actor=s.Director
  ```

SQL Queries: Nesting

- The WHERE clause can contain predicates of the form
  - `attr/value IN <SQL query>`
  - `attr/value NOT IN <SQL query>`
- The predicate is satisfied if the `attr` or `value` appears in the result of the nested `<SQL query>`.
- Queries involving nesting but no negation can always be un-nested, unlike queries with nesting and negation.

```
Typical use: “find objects that always satisfy property X”, e.g., find actors playing in every movie by “Berto”.
```
```
SELECT Actor FROM Movie
WHERE Actor NOT IN
(SELECT t.Actor
FROM Movie t s,
WHERE s.Director="Berto"
AND t.Actor NOT IN
(SELECT Actor
FROM Movie
WHERE Title=s.Title))
```
```
The shaded query finds actors NOT playing in some movie by “Berto”
The top lines complement the shaded part.
```

Homework Problem

Compare with shaded sub-query of previous page. The sample data may help you.

```
<table>
<thead>
<tr>
<th>Actor</th>
<th>Director</th>
<th>Movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>X</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>X</td>
<td>3</td>
</tr>
</tbody>
</table>
```

6
Nested Queries: Existential and Universal Quantification

- A \( \text{op ANY } \langle \text{nested query} \rangle \) is satisfied if there is a value \( X \) in the result of the \( \langle \text{nested query} \rangle \) and the condition \( A \text{ op } X \) is satisfied.
- A \( \text{op ALL } \langle \text{nested query} \rangle \) is satisfied if for every value \( X \) in the result of the \( \langle \text{nested query} \rangle \) the condition \( A \text{ op } X \) is satisfied.

Find directors of currently playing movies

```sql```
SELECT Director
FROM Movie
WHERE Title = ANY
(SELECT Title
FROM Schedule)
```

Find the employees with the highest salary

```sql```
SELECT Employee
WHERE Salary = ALL
(SELECT Salary
FROM Employee)
```

Nested Queries: Set Comparison

- \( \langle \text{nested query 1} \rangle \) CONTAINS \( \langle \text{nested query 2} \rangle \)

Find actors playing in every movie by “Berto”

```sql```
SELECT s.Actor
FROM Movie s
WHERE
(SELECT Title
FROM Movie t
WHERE t.Actor = s.Actor)
CONTAINS
(SELECT Title
FROM Movie
WHERE Director = "Berto")
```

SQL: Union, Intersection, Difference

- Union
  - \( \langle \text{SQL query 1} \rangle \) UNION \( \langle \text{SQL query 2} \rangle \)
- Intersection
  - \( \langle \text{SQL query 1} \rangle \) INTERSECT \( \langle \text{SQL query 2} \rangle \)
- Difference
  - \( \langle \text{SQL query 1} \rangle \) MINUS \( \langle \text{SQL query 2} \rangle \)

Find all actors or directors

```sql```
(SELECT Actor
FROM Movie)
UNION
(SELECT Director
FROM Movie)
```

Find all actors who are not directors

```sql```
(SELECT Actor
FROM Movie)
MINUS
(SELECT Director
FROM Movie)
```
SQL Queries: Aggregation and Grouping

- There is no relational algebra equivalent for aggregation and grouping
- Aggregate functions: AVG, COUNT, MIN, MAX, SUM, and recently user defined functions as well
- Group-by

<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, AvgSal=Avg(Salary)
FROM Employee
GROUP-BY Dept

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

SQL Grouping: Conditions that Apply on Groups

- **HAVING** clause

Find the average salary of for each department that has more than 1 employee
SELECT Dept, AvgSal=Avg(Salary)
FROM Employee
GROUP-BY Dept
HAVING COUNT(Name)>1

SQL: More Bells and Whistles ...

- Select all attributes using *
- Pattern matching conditions: `<attr>` LIKE `<pattern>`

Retrieve all movie attributes of currently playing movies
SELECT Movie.*
FROM Movie, Schedule
WHERE Movie.Title=Schedule.Title

Retrieve all movies where the title starts with "Ta"
SELECT *
FROM Movie
WHERE Title LIKE "Ta"
...and a Few “Dirty” Points

- **Duplicate elimination** must be explicitly requested
  - `SELECT DISTINCT ... FROM ... WHERE ...`
- **Null values**
  - all comparisons involving NULL are false by definition
  - all aggregation operations, except `count`, ignore NULL values

<table>
<thead>
<tr>
<th>Title</th>
<th>Director</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wild</td>
<td>Lynch</td>
<td>Winger</td>
</tr>
<tr>
<td>Sky</td>
<td>Berto</td>
<td>Winger</td>
</tr>
<tr>
<td>Reds</td>
<td>NULL</td>
<td>Beauty</td>
</tr>
<tr>
<td>Tango</td>
<td>Berto</td>
<td>Brando</td>
</tr>
<tr>
<td>Tango</td>
<td>Berto</td>
<td>Winger</td>
</tr>
<tr>
<td>Tango</td>
<td>Berto</td>
<td>NULL</td>
</tr>
</tbody>
</table>

---

### SQL as a Data Manipulation Language: Insertions

- inserting tuples
  - `INSERT INTO R VALUES (v1,...,vk);`
- some values may be left NULL
- use results of queries for insertion
  - `INSERT INTO R SELECT ... FROM ... WHERE ...`

- INSERT INTO Movie
  VALUES (“Brave”, “Gibson”, “Gibson”);
- INSERT INTO Movie(Director)
  VALUES (“Brave”, “Gibson”);
- INSERT INTO EuroMovie
  SELECT * FROM Movie
  WHERE Director = “Berto”

---

### SQL as a Data Manipulation Language: Updates and Deletions

- **Deletion** basic form:
  delete every tuple that satisfies `<cond>`
  - `DELETE FROM R WHERE <cond>`
- **Update** basic form:
  update every tuple that satisfies `<cond>` in the way specified by the SET clause
  - `UPDATE R SET A1=<exp1>, ..., Ak=<expk> WHERE <cond>`

- Delete the movies that are not currently playing
  `DELETE FROM Movie WHERE Title NOT IN SELECT Title FROM Schedule`
- Change all “Berto” entries to “Bertoluci”
  `UPDATE Movie WHERE Director = “Berto”`
- Increase all salaries in the Toys dept by 10%
  `UPDATE Employee WHERE Dept = “Toys”`
- The “rich get richer” exercise:
  Increase by 10% the salary of the employee with the highest salary