

Yannis Papakonstantinou, UCSD

## Relational Model (defn)

- Structure of 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

Movie		
Title	Director	Actor
Wild	Lynch	Winger
Sky	Berto	Winger
Reds	Beatty	Beatty
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	Snyder

Schedule	
Theater	Title
Odeon	Wild
Forum	Reds
Forum	Sky

Yannis Papakonstantinou, UCSD

## Query Languages: Overview

- SQL**
  - used by the database user
  - declarative*: we only describe **what** we want to retrieve
  - based on tuple relational calculus
- Relational Algebra**
  - used internally by the database system
  - procedural* (operational): we describe **how** we retrieve
- Relational Calculus**
- The result of a query is always a table (regardless of the query language used)

Yannis Papakonstantinou, UCSD

## Basic Relational Algebra Operators

- Selection ( $\sigma$ )**
  - $\sigma_R$  selects tuples of the argument relation  $R$  that satisfy the condition  $c$ .
  - The condition  $c$  consists of atomic predicates of the form
    - $attr = value$  ( $attr$  is attribute of  $R$ )
    - $attr1 = attr2$
    - other operators possible (e.g.,  $>$ ,  $<$ ,  $!=$ , LIKE)
  - Bigger conditions constructed by conjunctions (AND) and disjunctions (OR) of atomic predicates

$\sigma_{Director="Berto"} Movie$		
Title	Director	Actor
Sky	Berto	Winger
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	Snyder

$\sigma_{Director=Actor} Movie$		
Title	Director	Actor
Reds	Beatty	Beatty

$\sigma_{Director="Berto" \text{ OR } Director=Actor} Movie$		
Title	Director	Actor
Sky	Berto	Winger
Reds	Beatty	Beatty
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	Snyder

Yannis Papakonstantinou, UCSD

## Basic Relational Algebra Operators

- Projection ( $\pi$ )**
  - $\pi_{attr1, \dots, attrN} R$  returns a table that has only the attributes  $attr1, \dots, attrN$  of  $R$
  - no** duplicate tuples in the result (notice the example has only one (Tango,Berto) tuple)
- Cartesian Product ( $\times$ )**
  - the schema of the result has all attributes of both  $R$  and  $S$
  - for every pair of tuples  $r$  from  $R$  and  $s$  from  $S$  there is a result tuple that consists of  $r$  and  $s$
  - if both  $R$  and  $S$  have an attribute  $A$  then rename to  $R.A$  and  $S.A$

$\pi_{Title, Director} Movie$	
Title	Director
Wild	Lynch
Sky	Berto
Reds	Beatty
Tango	Berto

R		S	
A	B	A	C
0	1	a	b
2	4	c	d

R x S			
R.A	R.B	S.A	S.C
0	1	a	b
0	1	c	d
2	4	a	b
2	4	c	d

Yannis Papakonstantinou, UCSD

## Basic Relational Algebra Operations

- Rename ( $\rho$ )**
  - $\rho_A^B R$  renames attribute  $A$  of relation  $R$  into  $B$
  - $\rho_S R$  renames relation  $R$  into  $S$
- Union ( $\cup$ )**
  - applies to two tables  $R$  and  $S$  with same schema
  - $R \cup S$  is the set of tuples that are in  $R$  or  $S$  or both
- Difference ( $-$ )**
  - applies to two tables  $R$  and  $S$  with same schema
  - $R - S$  is the set of tuples in  $R$  but not in  $S$

Find all people, ie, actors and directors of the table Movie		
People	Actor	Movie
$\rho_{People} \rho_{Actor} \rho_{Movie}$		
$\cup \rho_{People} \rho_{Director} \rho_{Movie}$		

Find all directors who are not actors		
Director	Movie	
$\rho_{Director} \rho_{Movie}$		
$-\rho_{Director} \rho_{Actor} \rho_{Movie}$		

Yannis Papakonstantinou, UCSD

## SQL Queries: The Basic From

- Basic form**  
`SELECT  $a_1, \dots, a_N$  FROM  $R_1, \dots, R_M$  WHERE  $condition$`
- Equivalent relational algebra expression  
 $\rho_{a_1, \dots, a_N} \sigma_{condition} (R_1 \times \dots \times R_M)$
- WHERE clause is optional
- When more than one relations of the FROM have an attribute named  $A$  we refer to a specific  $A$  attribute as  $\langle RelationName \rangle.A$

Find titles of currently playing movies		
Title	Schedule	
<code>SELECT Title FROM Schedule</code>		

Find the titles of all movies by "Berto"		
Title	Schedule	
<code>SELECT Title FROM Schedule WHERE Director="Berto"</code>		

Find the titles and the directors of all currently playing movies		
Movie	Title	Schedule
<code>SELECT Movie.Title, Director FROM Movie, Schedule WHERE Movie.Title=Schedule.Title</code>		

Yannis Papakonstantinou, UCSD

## SQL Queries: Aliases

- Use the same relation more than once in the FROM clause
- Example: find actors who are also directors
 

```
SELECT t.Actor
FROM Movie t s
WHERE t.Actor=s.Director
```

Yannis Papakonstantinou, UCSD

## 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

Yannis Papakonstantinou, UCSD

## 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

Employee		
Name	Dept	Salary
Joe	Toys	45
Nick	PCs	50
Jim	Toys	35
Jack	PCs	40

Find the average salary of all employees

```
SELECT AvgSal=Avg(Salary)
FROM Employee
```

AvgSal
42.5

Find the average salary for each department

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

Dept	AvgSal
Toys	40
PCs	45

Yannis Papakonstantinou, UCSD

## SQL: Union, Intersection, Difference

- Union**
  - `<SQL query 1> UNION <SQL query 2>`
- Intersection**
  - `<SQL query 1> INTERSECT <SQL query 2>`
- Difference**
  - `<SQL query 1> MINUS <SQL query 2>`

Find all actors or directors

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

Find all actors who are not directors

```
(SELECT Actor
FROM Movie)
MINUS
(SELECT Director
FROM Movie)
```

Yannis Papakonstantinou, UCSD

## Nested Queries: Existential and Universal Quantification

- A op ANY <nested query>* is satisfied if **there is** a value *X* in the result of the `<nested query>` and the condition *A op X* is satisfied
  - ANY aka SOME
- A op ALL <nested query>* is satisfied if **for every** value *X* in the result of the `<nested query>` the condition *A op X* is satisfied

Find directors of currently playing movies

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

Find the employees with the highest salary

```
SELECT Name
FROM Employee
WHERE Salary >= ALL
  (SELECT Salary
   FROM Employee)
```

Yannis Papakonstantinou, UCSD

## Nested Queries: Set Comparison

- `<nested query 1> CONTAINS <nested query 2>`

Find actors playing in every movie by "Berto"

```
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")
```

Yannis Papakonstantinou, UCSD

## 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

Employee		
Name	Dept	Salary
Joe	Toys	45
Nick	PCs	50
Jim	Toys	35
Jack	PCs	40

Find the average salary of all employees

```
SELECT AvgSal=Avg(Salary)
FROM Employee
```

AvgSal
42.5

Find the average salary for each department

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

Dept	AvgSal
Toys	40
PCs	45

Yannis Papakonstantinou, UCSD

## 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
```

Yannis Papakonstantinou, UCSD

## 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"
```

Yannis Papakonstantinou, UCSD

## ...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

```
SELECT Title
FROM Movie
```

Title
Tango
Tango
Tango

```
SELECT DISTINCT Title
FROM Movie
```

Title
Tango

Title	Director	Actor
Wild	Lynch	Winger
Sky	Berto	Winger
Reds	NULL	Beatty
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	NULL

Yannis Papakonstantinou, UCSD

## 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(Title,Director)
VALUES ("Brave", "Gibson");
```

```
INSERT INTO EuroMovie
SELECT * FROM Movie
WHERE Director = "Berto"
```

Yannis Papakonstantinou, UCSD

## 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 "Bertolucci"

```
UPDATE Movie
SET Director="Bertolucci"
WHERE Director="Berto"
```

Increase all salaries in the Toys dept by 10%

```
UPDATE Employee
SET Salary = 1.1 * Salary
WHERE Dept = "Toys"
```

The "rich get richer" exercise:  
Increase by 10% the salary of the employee with the highest salary