

CSE 232: Principles of Database Systems

<http://www.sdsc.edu/~ludaesch/Teaching/CSE232/>

- **Topics**
 - Relational Query Languages: SQL, algebra, calculus
 - Data Storage Issues
 - Index Structures: conventional, multi-level, B-Trees, Hashing
 - Query Execution: rewritings, algorithms for algebra operators
 - Transactions and Concurrency Control
 - Semantic Query Optimization (guest lecturer: Victor Vianu)
 - ...
- **Lectures:** Tuesdays, Thursdays, 3:55-5:15, APM 4218
- **Instructor:** Bertram Ludäscher, SDSC R367, ludaes@sdsc.edu
- **Office Hours:** immediately after classes (@APM) or by appointment (@SDSC)
- **Grading (tentative):** midterm (40%), final (60%)
- **Recommended Texts**
 - *Database System Implementation*, Garcia-Molina, Widom, Ullman, Prentice-Hall
 - *Foundations of Database Systems*, Abiteboul, Hull, Vianu, Addison-Wesley

1

Relational Model

- Structure of relational databases: *Schema + Data*
- *Schema (or db-scheme):*
 - collection of *tables (or relations)*
 - each table has a set of *attributes*
 - unique relation names, unique attributes per table
- *Data (or db-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

2

Query Languages: Overview

- "procedural": describe *how* to retrieve result
 - relational algebra (RA), internal representation within the DBMS
- "declarative": describe *what* to retrieve
 - relational calculus (RC) (tuple-RC and domain-RC):
 - constants: a,b, ..., variables: X,Y,Z..., atomic formulas/predicates: p(a,b,X), formulas: using formulas and $\forall, \exists, \neg, \wedge, \vee$
- commercial query languages:
 - SQL: based on tuple-RC and RA (procedural and declarative elements)
- QLs should be *closed*: the result of a query is always a table (and thus can be queried again)

3

Basic Relational Algebra Operators

- **Selection (σ)**
 - σ_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., $>$, $<$, $=$, \neq , LIKE)
 - Complex conditions using conjunctions (AND) and disjunctions (OR) of atomic predicates

Find tuples where director="Berto"

$\sigma_{Director="Berto"} Movie$

Title	Director	Actor
Sky	Berto	Winger
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	Snyder

Find tuples where director=actor

$\sigma_{Director=Actor} Movie$

Title	Director	Actor
Reds	Beatty	Beatty

Director="Berto" OR Director=Actor Movie

$\sigma_{Director="Berto" \vee Director=Actor} Movie$

Title	Director	Actor
Sky	Berto	Winger
Reds	Beatty	Beatty
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	Snyder

Basic Relational Algebra Operators

- **Projection (π)**
 - $\pi_{attr1, \dots, attrN} R$ 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

Project the title and director of Movie

R	
A	B
0	1
2	4

S	
A	C
a	b
c	d

$R \times S$

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

Basic Relational Algebra Operations

- **Rename (ρ)**
 - $\rho_{A \rightarrow B} R$ 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

$\pi_{People} \rho_{Actor \rightarrow People} Movie$
 $\cup \pi_{People} \rho_{Director \rightarrow People} Movie$

Find all directors who are not actors

$\pi_{Director} Movie$
 $- \rho_{Actor \rightarrow Director} (\pi_{Actor} Movie)$

SQL Queries: The Basic From	
<ul style="list-style-type: none"> Basic form SELECT <i>a1</i>, ..., <i>aN</i> FROM <i>R1</i>, ..., <i>RM</i> WHERE <i>condition</i> Equivalent relational algebra expression $\pi_{a1, \dots, aN} \sigma_{condition}(R1 \times \dots \times RM)$ WHERE clause is optional When more than one relation of the FROM has an attribute named <i>A</i>, we refer to a specific <i>A</i> attribute as <i><RelationName>.A</i> 	<i>Find titles of currently playing movies</i> SELECT Title FROM Schedule
	<i>Find the titles of all movies by "Berto"</i> SELECT Title FROM Schedule WHERE Director="Berto"
	<i>Find the titles and the directors of all currently playing movies</i> SELECT Movie.Title, Director FROM Movie, Schedule WHERE Movie.Title=Schedule.Title

SQL Queries: Aliases
<ul style="list-style-type: none"> 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

SQL Queries: Nesting	
<ul style="list-style-type: none"> The WHERE clause can contain predicates of the form <ul style="list-style-type: none"> <i>attr/value</i> IN <i><SQL query></i> <i>attr/value</i> NOT IN <i><SQL query></i> The predicate is satisfied if the <i>attr</i> or <i>value</i> appears in the result of the nested <i><SQL query></i> 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., <i>find actors playing in every movie by "Berto"</i> : SELECT Actor FROM Movie WHERE Actor NOT IN (SELECT m2.Actor FROM Movie m1 m2, WHERE m1.Director="Berto" AND m2.Actor NOT IN (SELECT Actor FROM Movie WHERE Title=m1.Title)) The shaded query finds actors NOT playing in some movie by "Berto" The top lines complement the shaded part

SQL: Union, Intersection, Difference	
<ul style="list-style-type: none"> • Union <ul style="list-style-type: none"> - <SQL query 1> UNION <SQL query 2> • Intersection <ul style="list-style-type: none"> - <SQL query 1> INTERSECT <SQL query 2> • Difference <ul style="list-style-type: none"> - <SQL query 1> MINUS <SQL query 2> 	<p><i>Find all actors or directors</i></p> <pre>(SELECT Actor FROM Movie) UNION (SELECT Director FROM Movie)</pre>
	<p><i>Find all actors who are not directors</i></p> <pre>(SELECT Actor FROM Movie) MINUS (SELECT Director FROM Movie)</pre> <p style="text-align: right;">10</p>

Nested Queries: Existential and Universal Quantification	
<ul style="list-style-type: none"> • A <i>op</i> ANY <nested query> is satisfied if there is a value X in the result of the <nested query> and the condition A <i>op</i> X is satisfied <ul style="list-style-type: none"> - ANY aka SOME • A <i>op</i> ALL <nested query> is satisfied if for every value X in the result of the <nested query> the condition A <i>op</i> X is satisfied 	<p><i>Find directors of currently playing movies</i></p> <pre>SELECT Director FROM Movie WHERE Title = ANY (SELECT Title FROM Schedule)</pre>
	<p><i>Find the employees with the highest salary</i></p> <pre>SELECT Name FROM Employee WHERE Salary >= ALL (SELECT Salary FROM Employee)</pre> <p style="text-align: right;">11</p>

Nested Queries: Set Comparison	
<ul style="list-style-type: none"> • <nested query 1> CONTAINS <nested query 2> 	<p><i>Find actors playing in every movie by "Berto"</i></p> <pre>SELECT m1.Actor FROM Movie m1 WHERE (SELECT Title FROM Movie m2 WHERE m1.Actor = m2.Actor) CONTAINS (SELECT Title FROM Movie WHERE Director = "Berto")</pre> <p style="text-align: right;">12</p>

SQL Queries: Aggregation and Grouping

- There is no relational algebra equivalent for aggregation and grouping
- Aggregate functions: AVG, COUNT, MIN, MAX, SUM, ... (e.g. user defined functions)
- Group-by

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

Find 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

13

SQL Grouping: Conditions that Apply on Groups

- HAVING** clause

the condition after HAVING defines the desired property of the **group** (not individual tuples)

Find the average salary of each department that has more than 1 employee

```
SELECT Dept, AvgSal= Avg(Salary)
FROM Employee
GROUP-BY Dept
HAVING COUNT(Name)>1
```

14

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"
```

15

...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

16

SQL as a Data Manipulation Language: Insertions

- *Insertion* of 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"
```

17

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
SET Director="Bertoluci"
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
