Database Management Systems
L6

Umass Boston
Summer 2023
Cristina Maier

Some slides are based on “Database Management System, 3rd ed., by Ramakrishnan and Gehrke

Topics

- Introduction to DBMS
- Relational Data Model
- *Relational Algebra*
- Conceptual Design: the Entity-Relationship Model
- Structured Query Language (SQL)
- Application Development (Java, Python)
- Schema Refinement and Normal Forms
- Database Security and Authorization
- Some NoSQL topics (If time permitted)

Connecting queries and sub-queries (recap.)

- The result of one query can be:
  - A relation
    - E.g.: SELECT s.sname, s.rating FROM Sailors s;
  - A scalar value (1×1 table)
    - In this case, this result can be used in locations where a constant can be placed
    - E.g.: SELECT AVG(s.rating) from Sailors s;
- Where do subqueries appear?
  - Most often in the WHERE clause of a parent query
  - In FROM clause follow by range variable
  - In HAVING clause

Nested Queries Summary (recap.)

- Nested queries returning a constant
  - Typically the constant is compared with other values in the WHERE clause
  - … WHERE field= (SELECT x FROM…)…
- Nested queries returning a relation
  - In WHERE Clause
    - …WHERE (EXISTS/UNIQUE) (SELECT…)…
    - … WHERE field IN (SELECT col FROM ….)…
    - …WHERE field op ANY|ALL (SELECT col FROM…)….
  - In FROM clause, followed by range variable
    - E.g: ..FROM Sailors, (SELECT bid FROM Boats..) BIDS ….
  - In HAVING clause
    - … HAVING field/const op (SELECT ….) ….

Exercise 1

- sailors(sid:int, sname: string, rating:int, age:, salary:real)
- boats(bid:int, name:string, color:string, manufacturer:string, prod_year:date)
- reserves (sid:int, bid:int, day:date)
- Fine the id and name of the oldest boats

```
SELECT b.bid, b.name
FROM boats b
WHERE b.pub_year = ( SELECT MIN(b2.prod_year)
                     FROM boats b2);
```

Exercise 2

- sailors(sid:int, sname: string, rating:int, age:, salary:real)
- boats(bid:int, name:string, color:string, manufacturer:string, prod_year:date)
- reserves (sid:int, bid:int, day:date)
- Find the sailors who DO NOT have the highest rating. Extract the sailors id and name.

Exercise 2 - Solution #1

- sailors(sid:int, sname: string, rating:int, age:, salary:real)
- boats(bid:int, name:string, color:string, manufacturer:string, prod_year:date)
- reserves (sid:int, bid:int, day:date)
- Find the sailors who DO NOT have the highest rating. Extract the sailors id and name.

```
SELECT s.sid, s.sname, s.rating, s.age, s.salary
FROM sailors s
WHERE s.rating != ( SELECT MAX(s2.rating)
                    FROM sailors s2);
```

Exercise 2 - solution #2 (using ANY)

- sailors(sid:int, sname: string, rating:int, age:, salary:real)
- boats(bid:int, name:string, color:string, manufacturer:string, prod_year:date)
- reserves (sid:int, bid:int, day:date)
- Find the sailors who DO NOT have the highest rating. Extract the sailors id and name.

```
SELECT s.sid, s.sname, s.rating, s.age, s.salary
FROM sailors s
WHERE s.rating < ANY ( SELECT s2.rating
                       FROM sailors s2);
```

Exercise 2 - solution #3

```
SELECT s.sid, s.sname, s.rating, s.age, s.salary
FROM sailors s
WHERE EXISTS ( SELECT *
                    FROM sailors s2
                    WHERE s2.pub_year >
s.pub_year);
```

SQL Division

- Remember the division from Relational Algebra
- SQL does not implement the division (/) operator

Division (recap)

$$R/S$$

- Also noted as $R \div S$
- Useful for expressing certain kinds of queries, such as “Find the names of all sailors who have reserved all boats.”
- It does not have the same importance as the other operators because it is not needed as often, but it is extremely important for certain queries
  - SQL does not have a specific construct for it. When needed, it is done using a combination of other constructs

Division (recap.)

- A(x:type1, y:type2)
- B(y:type2)
- A/B is the set of all values x (in the form of unary tuples) such that for every y value in B, there is a < x, y> tuple in A
- For each x value in the first column of A, consider the set of values that appear in the y column of A associated with that value of x. If this set contains all values y in B, then x is in the result for A/B
- A value x from A is disqualified from the result if by attaching a value y from B, we have a tuple <x, y> that is not present in A
- $A/B = \pi_x A - \pi_x((\pi_x(A) \times B) - A)$
- To understand the division operation in full generality, we have to consider the case when both x and y are replaced by a set of attributes

Division Example (recap.)

R

sid	bid
12	101
10	102
12	102
11	101
10	101

B

bid
101
102

Find the ids of sailors who have reserved all boats

R/B

sid
12
10

Division in SQL

- SQL does not implement the division operator
- One of the most subtle query
- Solutions follow specific patterns

Division in SQL

- Two ways of writing it
  - 1) using the set-difference operator (EXCEPT, MINUS)
    - 2 levels of nested queries
  - 2) without set-difference (EXCEPT, MINUS)
    - 3 levels of nested queries

Schema

- sailors(sid:int, sname: string, rating:int, age:, salary:real)
- boats(bid:int, name:string, color:string, manufacturer:string, prod_year:date)
- reserves (sid:int, bid:int, day:date)

Solution with set-difference (EXCEPT or MINUS)

- Find the sailors who reserved all boats
- This is equivalent to *'find all sailors for whom there is no boat they did not reserved'*

```
SELECT s.sid, s.sname
FROM sailors s
WHERE NOT EXISTS (
        ( SELECT b.bid FROM boats b)
         MINUS
         (SELECT r.bid FROM reserves r
         WHERE r.sid=s.sid)
);
```

Solution without set-difference (EXCEPT or MINUS)

- Find the sailors who reserved all boats
- SELECT *a sailor for whom*
- *there is no boat*
- *for which there is no reservation made by that sailor*

```
SELECT s.sid, s.sname
FROM sailors s
WHERE NOT EXISTS (
        SELECT b.bid FROM boats b
        WHERE NOT EXISTS (
                SELECT * FROM reserves r
                 WHERE r.sid=s.sid AND r.bid=b.bid
         )
);
```

Example 2

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age:int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the id and name of actors who played in all movies ?

Example 2 - solution #1

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age:int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the id and name of actors who played in all movies

```
SELECT a.actor_id, a.name
FROM Actors a
WHERE NOT EXISTS(
        (SELECT m.movie_id FROM Movies m)
         MINUS
        (SELECT p.movie_id FROM PlaysIn p
         WHERE p.actor_id=a.actor_id)
);
```

Example 2 - solution #2

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age:int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the id and name of actors who played in all movies

```
SELECT a.actor_id, a.name
FROM Actors a
WHERE NOT EXISTS (
        SELECT m.movie_id FROM Movies m
         WHERE NOT EXISTS (
             SELECT * from PlaysIn p
             WHERE p.actor_id=a.actor_id AND p.movie_id=m.movie_id
           )
);
```

Example 3

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age:int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the id and name of actors who played in all movies produced by 'Universal' studio.

?

Example 3 - solution #1

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age:int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the id and name of actors who played in all movies produced by ‘Universal' studio.

```
SELECT a.actor_id, a.name
FROM Actors a
WHERE NOT EXISTS(
        (SELECT m.movie_id FROM Movies m
         WHERE m.studio='Universal')
         MINUS
        (SELECT p.movie_id FROM PlaysIn p
         WHERE p.actor_id=a.actor_id)
);
```

Example 3 - solution #2

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age :int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the id and name of actors who played in all movies produced by ‘Universal' studio.

```
SELECT a.actor_id, a.name
FROM Actors a
WHERE NOT EXISTS (
        SELECT m.movie_id FROM Movies m
         WHERE m.studio='Universal' AND NOT EXISTS (
             SELECT * from PlaysIn p
             WHERE p.actor_id=a.actor_id AND p.movie_id=m.movie_id
           )
);
```

Example 4

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age: real)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the average age of actors for each movie in which at least 4 actors play.
- ?

Example 4 - solution

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age: real, age:int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the average age of actors for each movie in which at least 4 actors play.

```
SELECT m.movie_id, AVG(a.age)
FROM Movies m, Actors a, PlaysIn p
WHERE m.movie_id=p.movie_id AND p.actor_id=a.actor_id
GROUP BY m.movie_id
HAVING COUNT(*) >=4;
```

Example 4 - solution #2

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age: real)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find the average age of actors for each movie in which at least 4 actors play.

```
SELECT p.movie_id, avg(a.age)
FROM Actors a, PlaysIn p
WHERE a.actor_id=p.actor_id
GROUP BY p.movie_id
HAVING  4 <= ( SELECT COUNT(p.actor_id)
                    FROM PlaysIn p2
                     WHERE p2.movie_id=p.movie_id
                    );
```

Practice queries

- Please practice at home with queries from PracticeSessionSQL4.sql

SQL

- Create, alter, delete tables
- Insert Statement
- Basic Select Statement Structure
- LIKE, AS keywords
- Dates, Text case sensitivity
- Count
- Set Operations
- Aggregates
- Nested Queries
- SQL Division
- NULL Constraints

Example Schema

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age: int)
- PlaysIn(actor_id:int, movie_id: int, character: string)

NULL Constraints

- It has to do with missing values from columns
- Columns that do not have the NOT NULL constraint can contain null values
- PRIMARY KEYS have NOT NULL constraints !!!
- If you try to insert a record and do not provide values for primary key columns or for NOT NULL columns, you will get an error
- We can use conditions with IS NOT NULL to filter out all records that have a null value for that column
- We can use conditions with IS NULL to filter out all records that have a non-null value for that column

Example - Table Definition

- CREATE TABLE Movies(
  movie_id NUMBER(9) PRIMARY KEY,
  title VARCHAR(2),
  year int,
  studio VARCHAR(20)
  );

- Column movie_id cannot contain null values, because it is a primary key. By default primary keys have NOT NULL constrains!
  - When inserting a record, movie_id must be present. If not present, we will get an error
- Columns title, year and studio can have null values. There is no NOT NULL constraint
  - When inserting a record, it is ok if we do not provide values for title, year or/and studio. If not present, they will contain null

Example 2 - Table Definition

- CREATE TABLE Movies2(
  movie_id NUMBER(9) PRIMARY KEY,
  title VARCHAR(2),
  year int,
  studio VARCHAR(20) NOT NULL
  );

- Column movie_id cannot contain null values, because it is a primary key. By default primary keys have NOT NULL constrains!

- Column studio cannot contain null values because it has the NOT NULL constraint

- Columns title and year can have null values. There is no NOT NULL constraint

Example
IS NOT NULL

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age :int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find all movies that have a studio (i.r. column studio has a value)

```
SELECT *
FROM Movies
WHERE studio IS NOT NULL;
```

Example
IS NULL

- Movies(movie_id: int, title: string, year: int, studio: string)
- Actors(actor_id: int, name: string, age :int)
- PlaysIn(actor_id:int, movie_id: int, character: string)
- Find all movies that DO NOT have a studio (i.r. column studio is null)

```
SELECT *
FROM Movies
WHERE studio IS NULL;
```

Problems with NULL Values

- It complicates things
- E.g.:
- **WHERE rating >7**
- What if rating allows NULLs?
- For NULL values, how NULL >7 evaluate?
- Expressions can have three logic values
  - True
  - False
  - Unknown
- For a record that does have rating Null, the Condition WHERE rating >7 will evaluate to Unknown

Problems with NULL Values

- WHERE condition
  - Returns only the records for where condition is evaluated to True !!!
- Unknown AND True evaluates to Unknown
- Unknown OR False evaluates to Unknown
- Unknown OR True evaluates to True
- Unknown = Unknown evaluates to Unknown

Null Values and Aggregates

- COUNT(*) counts all records (regardless on whether some attributes are NULL)
- COUNT(attribute) only counts the records in which this attribute is not null
- All other aggregates skips NULL values if the aggregate is on a field that is NULL (e.g. SUM(attribute) only adds up the values where attribute is not null)
  - If all values of the attribute from the aggregate are null the aggregate will return NULL, except the case when the aggregate is COUNT. If COUNT, it will return 0.

Example

- SELECT COUNT(*) FROM Movies;
- SELECT COUNT(studio) FROM Movies;
- Do not return the same result if studio contains NULL values
- SELECT COUNT(*) FROM Movies m;
- SELECT COUNT(*) FROM Movies m
  WHERE m.year < 2010 or m.year>=2010;
- Do not return the same result is year contains NULL

Null Values And Duplicates

- When comparing two NULL values, the result is unknown
- There is an anomaly when checking for duplicates
  - In this case NULL values are considered equal
  - DISTINCT, UNIQUE keywords
    - If multiple rows have equal values in all non-null columns, then only one row is returned in the result

Group By

- Group by columnY
  - If columnY contains NULL values, then a common group for NULL is created

Oracle Session

- Please practice at home with:
  - PracticeSessionSQL4.sql
  - PracticeSessionSQL5.sql

Questions?