# Lecture 5: SQL (Basic)

CS348 Spring 2025: Introduction to Database Management

> Instructor: **Xiao Hu** Sections: 001, 002, 003

#### Announcements

- Project Milestone o
  - not graded but due on May 22
- Online office hours by IAs for Assignment 1
  - See Piazza for Zoom information
  - Friday May 23 4pm 5pm
  - Wednesday May 28 4:30 pm 5:30pm

#### SQL features covered so far

#### Basics

- Data-definition language (DDL): define/modify schemas, drop relations
- Data-manipulation language (DML): query data
  - SELECT-FROM-WHERE statements
  - Set/Bag (DISTINCT, UNION/EXCEPT/INTERSECT (ALL))

Next: Nested SQL queries

## Table subqueries

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string)

- Query result as a table that can be used in FROM, set/bag operations, etc.
  - Temporarily exist only in the duration of the outer query
- Example: names of users belonging to at least two groups

SELECT name FROM User, (SELECT DISTINCT m1.uid FROM Member m1, Member m2 WHERE m1.uid=m2.uid AND m1.gid != m2.gid) AS temp WHERE User.uid = temp.uid;

## WITH clause

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string)

- Another way of defining a temporary table
  - Available only to the query in which the WITH clause occurs
- Example: names of users belonging to at least two groups

WITH temp AS (SELECT DISTINCT m1.uid FROM Member m1, Member m2 WHERE m1.uid=m2.uid AND m1.gid != m2.gid) SELECT name FROM User, temp WHERE User.uid = temp.uid;

#### Scalar subqueries

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string)

- A query that returns a single row can be used as a value in SELECT, WHERE, etc.
- Example: users at the same age as Bart

```
SELECT *
FROM User
WHERE age = (SELECT age
FROM User
WHERE name = 'Bart');
```

- When can this query go wrong?
  - Return more than 1 row
  - Return no rows or NULL values (in next lecture)

## IN subqueries

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string)

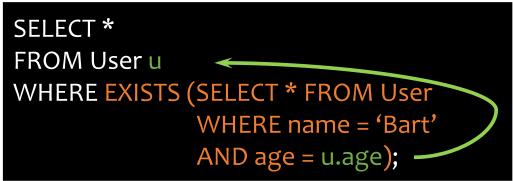
- x IN (subquery) checks if x is in the result of subquery
  - True if *x* equals to some value in the subquery result
- Example: users that have the same age as (some) Bart



#### EXISTS subqueries

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string)

- EXISTS (*subquery*) checks if the result of *subquery* is non-empty
  - True if at least one row is returned by subquery
- Example: users that have the same age as (some) Bart

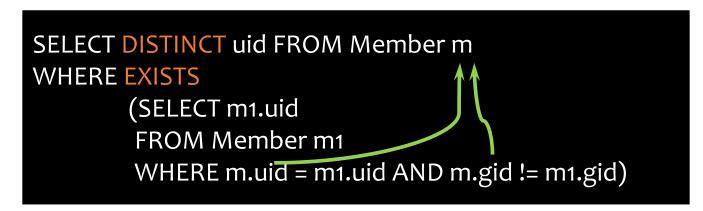


• This happens to be a correlated subquery -- a subquery that references tuple variables in surrounding queries

#### More example

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string)

• IDs of users who join at least two groups



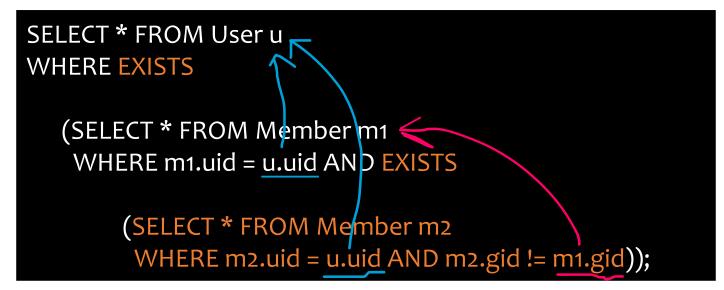
Use *table\_name.column\_name* when appropriate to avoid confusion

- How to find which table a column belongs to?
  - Start with the immediately surrounding query
  - If not found, look in the one surrounding that; repeat if necessary

#### More example

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string)

• All info of users who join at least two groups



Use *table\_name.column\_name* when appropriate to avoid confusion

• Query optimizer can decorrelate correlated subqueries into an equivalent join or aggregation

## Quantified subqueries

- Universal quantification (for all):
  - ... WHERE *x* op ALL(subquery) ...
  - True if for all *t* in the *subquery* result such that *x op t* is true

SELECT \* FROM User WHERE pop >= ALL (SELECT pop FROM User);

- Existential quantification (exists):
  - ... WHERE *x* op ANY(subquery) ...
  - True if there exists some *t* in the *subquery* result such that *x op t* is true

SELECT \* FROM User WHERE NOT (pop < ANY (SELECT pop FROM User));

#### More ways to get the most popular

• Which users are the most popular?



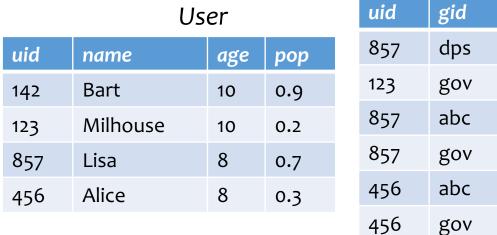
SELECT \* FROM User u WHERE NOT [EXISTS or IN?] (SELECT \* FROM User WHERE pop > u.pop);

SELECT \* FROM User WHERE uid NOT [EXISTS or IN?] (SELECT u1.uid FROM User AS u1, User AS u2 WHERE u1.pop < u2.pop);

## Summary of Subqueries

- We have covered:
  - Table subqueries (FROM)
  - Scalar subqueries (SELECT, WHERE)
  - IN subqueries (WHERE)
  - EXISTS subqueries (WHERE)
  - ALL/ANY subqueries (WHERE)
- Subqueries allow queries to be written in more declarative ways (recall the "most popular" query)
- But in many cases, they don't add expressive power

#### In class exercise



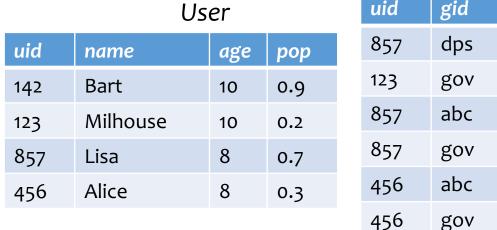
• What is the output of these queries?

SELECT name FROM User WHERE age <= ALL(SELECT age FROM User)

SELECT name FROM User WHERE pop < ANY(SELECT pop FROM User)

Member

#### In class exercise



• What is the output of these queries?

WITH temp AS (SELECT uid FROM User WHERE pop < ANY (SELECT pop FROM User)) SELECT name FROM User WHERE uid NOT IN (SELECT uid FROM temp) How about uid or \*? SELECT uid FROM User u WHERE EXISTS (SELECT gid FROM Member m WHERE m.uid = u.uid)

Member

#### SQL features covered so far

#### • Basics

- Data-definition language (DDL): define/modify schemas, delete relations
- Data-manipulation language (DML): query data
  - SELECT-FROM-WHERE statements
  - Set/Bag (DISTINCT, UNION/EXCEPT/INTERSECT (ALL))
  - Subqueries (table, scalar, IN, EXISTS, ALL, ANY)

Sext: Aggregation and grouping

## Aggregates

- Standard SQL aggregate functions: COUNT, SUM, AVG, MIN, MAX
- Example: number of users under 18, and their average popularity
  - COUNT(\*) counts the number of rows
  - AVG(x) computes the average values in column x

SELECT COUNT(*), AVG(pop)
FROM User
WHERE age <18;

COUNT(*)	AVG(pop)
6	0.625

• Aggregate functions do not appear in WHERE clause

## Aggregates with DISTINCT

• Example: How many users belong to at least one group?

SELECT COUNT(\*) FROM (SELECT DISTINCT uid FROM Member);

#### is equivalent to

SELECT COUNT(DISTINCT uid) FROM Member;

## Grouping

User (<u>uid</u> int, name string, age int, pop float) Group (<u>gid</u> string, name string) Member (<u>uid</u> int, <u>gid</u> string) 19

- SELECT ... FROM ... WHERE ... GROUP BY *list\_of\_columns*;
- Example: compute average popularity for each age group

SELECT age, AVG(pop) FROM User GROUP BY age;

Add much expressive power to SFW statements

## Example of GROUP BY

#### SELECT age, AVG(pop) FROM User GROUP BY age;

uid	name	age	рор
142	Bart	10	0.9
857	Lisa	8	0.7
123	Milhouse	10	0.2
456	Alice	8	0.3

Compute GROUP BY: group rows according to the values of GROUP BY columns

•	uid	name	age	рор
	142	Bart	10	0.9
	123	Milhouse	10	0.2
	857	Lisa	8	0.7
	456	Alice	8	0.3

Compute SELECT for each group

age

10

8

AVG(pop)

0.55

0.50

#### Semantics of GROUP BY

SELECT ... FROM ... WHERE ... GROUP BY ...;

- 1. Compute FROM  $(\times)$
- 2. Compute WHERE ( $\sigma$ )
- 3. Compute GROUP BY: group rows according to the values of GROUP BY columns
- 4. Compute SELECT for each group  $(\pi)$ 
  - For aggregation functions with DISTINCT inputs, first eliminate duplicates within the group

Solution Number of groups = number of rows in the final output

## Aggregates with no GROUP BY

 An aggregate query with no GROUP BY clause = all rows go into one group

SELECT AVG(pop) FROM User;

## Group all rows into one group

Aggregate over the whole group

uid	name	age	рор
142	Bart	10	0.9
857	Lisa	8	0.7
123	Milhouse	10	0.2
456	Alice	8	0.3

uid	name	age	рор	
142	Bart	10	0.9	AVG(pop)
857	Lisa	8	0.7	0.525
123	Milhouse	10	0.2	
456	Alice	8	0.3	

#### **Restriction on SELECT**

- If a query uses aggregation/group by, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column

Why?

This restriction ensures that any SELECT expression produces only one value for each group



#### HAVING

- Used to filter groups based on the group properties (e.g., aggregate values, GROUP BY column values)
- SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...;
  - 1. Compute FROM ( $\times$ )
  - 2. Compute WHERE ( $\sigma$ )
  - 3. Compute GROUP BY: group rows according to the values of GROUP BY columns
  - 4. Compute HAVING (another  $\sigma$  over the groups)
  - 5. Compute SELECT ( $\pi$ ) for each group that passes HAVING

#### Example of HAVING

 List the average popularity for each age group with more than a hundred users

SELECT age, AVG(pop) FROM User GROUP BY age HAVING COUNT(\*) > 100;

• Can be written using WHERE and table subqueries

SELECT Temp.age, Temp.apop
FROM (SELECT age, AVG(pop) AS apop, COUNT(\*) AS gsize
 FROM User GROUP BY age) AS Temp
WHERE Temp.gsize > 100;

#### Example of HAVING

• Find average popularity for each age group over 10

SELECT age, AVG(pop) FROM User GROUP BY age HAVING age > 10;

• Can be written using WHERE without table subqueries

SELECT age, AVG(pop) FROM User WHERE age > 10 GROUP BY age;

#### SQL features covered so far

#### Basics

- Data-definition language (DDL): define/modify schemas, delete relations
- Data-manipulation language (DML): query data
  - SELECT-FROM-WHERE statements
  - Set/Bag (DISTINCT, UNION/EXCEPT/INTERSECT (ALL))
  - Subqueries (table, scalar, IN, EXISTS, ALL, ANY)
  - Aggregation and Grouping
    - More expressive than relational algebra

Next: Ordering output tuples

#### ORDER BY

- SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY output\_column [ASC|DESC], ...;
- ASC = ascending, DESC = descending
- Semantics: After SELECT list has been computed and optional duplicate elimination has been carried out, sort the output according to ORDER BY specification

#### Example of ORDER BY

• List all users, sort them by popularity (descending) and name (ascending)

SELECT uid, name, age, pop FROM User ORDER BY pop DESC, name (ASC);

- ASC can be omitted since it is the default option
- Strictly speaking, only output columns can appear in the ORDER BY clause (although some DBMS support more)
- Can use sequence numbers instead of names to refer to output columns: ORDER BY 4 DESC, 2; Discouraged:

hard to read!

29

#### LIMIT

- The LIMIT clause specifies the number of rows to return
- E.g., Return top 3 users with highest popularities

SELECT uid, name, age, pop FROM User ORDER BY pop DESC LIMIT 3;

#### In class exercise

• What is the output of these queries?

SELECT <mark>COUNT(DISTINCT</mark> gid) FROM Member;

WITH temp AS (SELECT uid, COUNT(\*) AS cnt FROM Member GROUP BY uid )

SELECT name FROM User u, temp t WHERE t.uid = u.uid AND t.cnt = (SELECT MAX(cnt) FROM temp)

User			
uid	name	age	рор
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Alice	7	0.6

11cor

Member

uid	gid
857	dps
123	gov
857	abc
857	gov
456	abc
456	gov

#### In class exercise

• What is the output of these queries?

SELECT AVG(pop) AS apop FROM User GROUP BY age HAVING COUNT(\*) >=2 ORDER BY apop LIMIT 2;

User				
uid	name	age	рор	
142	Bart	10	0.9	
123	Milhouse	10	0.2	
857	Lisa	8	0.7	
456	Alice	7	0.6	

Member

uid	gid
857	dps
123	gov
857	abc
857	gov
456	abc
456	gov

SELECT AVG(pop) AS apop FROM User GROUP BY age HAVING age>5 ORDER BY apop LIMIT 2;

#### Take home exercises

- Using EXISTS, write a query to list the IDs of groups that have at least two users
- Using WITH-AS and (NOT) IN, write a query to list the IDs of groups that Lisa belongs to but Ralph does not
- Write the same query but using EXCEPT (you may or may not use any other keywords)

#### SQL features covered so far

#### Basics

- Data-definition language (DDL): define/modify schemas, delete relations
- Data-manipulation language (DML): query data
  - SELECT-FROM-WHERE statements
  - Set/Bag (DISTINCT, UNION/EXCEPT/INTERSECT (ALL))
  - Subqueries (table, scalar, IN, EXISTS, ALL, ANY)
  - Aggregation and grouping (GROUP BY, HAVING)
  - Ordering (ORDER)

Next: NULL, JOIN, Modification