

Announcements

- * Reminder: Homework #1 due in two weeks
- Reading assignment (optional for those of you who are new to SQL): "A Critique of the SQL Database Language," by Date in *SIGMOD Record*, 14(3), 1983
 Beware that it is for a rather old version of SQL
- * Recitation session this Friday (January 31) on SQL

Aggregates

- Standard SQL aggregate functions: COUNT, SUM, AVG, MIN, MAX
- Example: number of students under 18, and their average GPA
 - SELECT COUNT(*), AVG(GPA) FROM Student WHERE age < 18;
 - COUNT(*) counts the number of rows

Aggregates with **DISTINCT**

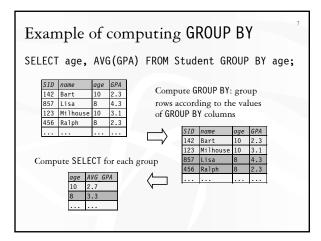
- * Example: How many students are taking classes?
 - SELECT COUNT(DISTINCT SID) FROM Enroll;
 - is equivalent to:
 - SELECT COUNT(*) FROM (SELECT DISTINCT SID, FROM Enroll);

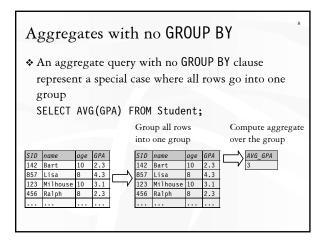
GROUP BY

- ♦ SELECT ... FROM ... WHERE ... GROUP BY list of columns;
- $\boldsymbol{\ast}$ Example: find the average GPA for each age group
 - SELECT age, AVG(GPA) FROM Student GROUP BY age;

Operational semantics of GROUP BY

- SELECT ... FROM ... WHERE ... GROUP BY ...;
- \bullet Compute FROM (×)
- * Compute WHERE (σ)
- Compute GROUP BY: group rows according to the values of GROUP BY columns
- Compute SELECT for each group (π)
 ^ΦOne output row per group in the final output





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

Examples of invalid queries

♦ SELECT → age FROM Student GROUP BY age;

- Recall there is one output row per group
- There can be multiple SID values per group
- ♦ SELECT > MAX(GPA) FROM Student;
 - Recall there is only one group for an aggregate query with no GROUP BY clause
 - There can be multiple SID values
 - Wishful thinking (that the output SID value is the one associated with the highest GPA) does NOT work

HAVING

 Used to filter groups based on the group properties (e.g., aggregate values, GROUP BY column values) 11

- ♦ SELECT ... FROM ... WHERE ... GROUP BY ... HAVING condition;
 - Compute FROM (×)
 - Compute WHERE (σ)
 - Compute GROUP BY: group rows according to the values of GROUP
 BY columns
 - Compute HAVING (another σ over the groups)
 - Compute SELECT (π) for each group that passes HAVING
 - ORDER BY and (SELECT) DISTINCT, if any, are applied last

HAVING examples Find the average GPA for each age group over 10 SELECT age, AVG(GPA) FROM Student GROUP BY age HAVING age > 10; Can be written using WHERE without table expressions List the average GPA for each age group with more than a hundred students SELECT age, AVG(GPA) FROM Student GROUP BY age HAVING COUNT(*) > 100; Can be written using WHERE and table expressions

Summary of SQL features covered so far

- ♦ SELECT-FROM-WHERE statements
- * Set and bag operations
- Table expressions, subqueries
- * Ordering
- * Aggregation and grouping
 - More expressive power than relational algebra
- ☞ Next: NULL's

Incomplete information

- * Example: Student (SID, name, age, GPA)
- ✤ Value unknown
 - We do not know Nelson's age
- * Value not applicable
 - Nelson has not taken any classes yet; what is his GPA?

Solution 1

- * A dedicated special value for each domain (type)
 - GPA cannot be -1, so use -1 as a special value to indicate a missing or invalid GPA
 - Leads to incorrect answers if not careful
 SELECT AVG(GPA) FROM Student;
 - Complicates applications
 SELECT AVG(GPA) FROM Student WHERE GPA <> -1;
 - Remember the pre-Y2K bug?
 09/09/99 was used as a missing or invalid date value

Solution 2

- * A valid-bit for every column
 - Student (<u>SID</u>, name, name_is_valid, age, age_is_valid, GPA, GPA_is_valid)
 - Still complicates applications
 SELECT AVG(GPA) FROM Student WHERE GPA_is_valid;

SQL's solution

A special value NULL

- Same for every domain
- Special rules for dealing with NULL's

* Example: Student (SID, name, age, GPA)

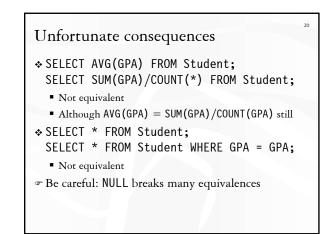
• \langle 789, "Nelson", NULL, NULL \rangle

Rules for NULL's

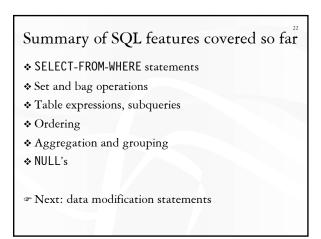
- ♦ When we operate on a NULL and another value (including another NULL) using +, -, etc., the result is NULL
- Aggregate functions ignore NULL, except COUNT(*) (since it counts rows)
- \clubsuit A scalar subquery that return no answer is treated as returning NULL

Three-valued logic

- When we compare a NULL with another value (including another NULL) using =, >, etc., the result is UNKNOWN
- \bullet TRUE = 1, FALSE = 0, UNKNOWN = 0.5
- $* x \text{ AND } y = \min(x, y)$
- $\mathbf{*} x \ \mathbf{0R} \ y = \max(x, y)$
- \bullet NOT x = 1 x
- WHERE and HAVING clauses only select rows for output if the condition evaluates to TRUE
 - UNKNOWN is insufficient



Another problem Example: Who has NULL GPA values? SELECT * FROM Student WHERE GPA = NULL; Does not work; never returns anything (SELECT * FROM Student) EXCEPT ALL (SELECT * FROM Student WHERE GPA = GPA) Works, but ugly Introduced built-in predicates IS NULL and IS NOT NULL SELECT * FROM Student WHERE GPA IS NULL;



INSERT

Insert one row

- INSERT INTO Enroll VALUES (456, 'CPS216');
 Student 456 takes CPS216
- * Insert the result of a query
 - INSERT INTO Enroll (SELECT SID, 'CPS216' FROM Student WHERE SID NOT IN (SELECT SID FROM Enroll WHERE CID = 'CPS216'));
 - Force everybody to take CPS216

DELETE

23

- Delete everything
 - DELETE FROM Enroll;
- Delete according to a WHERE condition
 - Example: Student 456 drops CPS216
 - DELETE FROM Enroll
 WHERE SID = 456 AND CID = 'CPS216';
 - Example: Drop students with GPA lower than 1.0 from all CPS classes

24

 DELETE FROM Enroll WHERE SID IN (SELECT SID FROM Student WHERE GPA < 1.0)
 AND CID LIKE 'CPS%';

UPDATE

- Example: Student 142 changes name to "Barney" and GPA to 3.0
 - UPDATE Student SET name = 'Barney', GPA = 3.0 WHERE SID = 142;
- * Example: Let's be "fair"?
 - UPDATE Student SET GPA = (SELECT AVG(GPA) FROM Student);
 - But update of every row causes average GPA to change!
 - Average GPA is computed over the old Student table

Summary of SQL features covered so far

* Query

25

- SELECT-FROM-WHERE statements
- Set and bag operations
- Table expressions, subqueries
- Ordering
- Aggregation and grouping
- * Modification
 - INSERT/DELETE/UPDATE
- * Next: constraints, triggers, views, indexes, ...