

Announcements (January 20)

- Reading assignment for this week (Ailamaki et al., VLDB 2001) has been posted
 - Due Wednesday night
 - Hunt for related/follow-up work too!
- * Course project will be assigned this Thursday
- Student presentation sign-up sheet will be circulated this Thursday
 - Allows you to drop your lowest homework grade
- ✤ Homework #1 due in two weeks

SQL

- * SQL: Structured Query Language
 - Pronounced "S-Q-L" or "sequel"
 - The standard query language support by most commercial DBMS
- A brief history
 - IBM System R
 - ANSI SQL89
 - ANSI SQL92 (SQL2)
 - SQL3 (still under construction after years!)

Basic queries: SFW statement

- * Also called an SPJ (select-project-join) query
- ♦ Equivalent (not really!) to relational algebra query $\pi_{A_1, A_2, ..., A_n}$ ($\sigma_{condition}$ ($R_1 \times R_2 \times ... \times R_m$))

Example: reading a table

♦ SELECT * FROM Student;

- Single-table query, so no cross product here
- WHERE clause is optional
- * is a short hand for "all columns"

Example: selection and projection

- Name of students under 18
 - SELECT name FROM Student WHERE age < 18;
- * When was Lisa born?
 - SELECT 2004 age FROM Student WHERE name = 'Lisa';
 - SELECT list can contain expressions
 Can also use built-in functions such as SUBSTR, ABS, etc.
 - String literals (case sensitive) are enclosed in single quotes

Example: join

- SID's and name's of students taking courses with the word "Database" in their titles
 - SELECT Student.SID, Student.name FROM Student, Enroll, Course WHERE Student.SID = Enroll.SID AND Enroll.CID = Course.CID AND title LIKE '%Database%';
 - LIKE matches a string against a pattern
 % matches any sequence of 0 or more characters
 - Okay to omit *table_name* in *table_name.column_name* if *column_name* is unique

Example: rename

- * SID's of students who take at least two courses
 - Relational algebra query: $\pi_{e1.SID}$
 - $((\rho_{\epsilon 1} \textit{Enroll}) \bowtie_{\epsilon 1.SID = \epsilon 2.SID \land \epsilon 1.CID \neq \epsilon 2.CID} (\rho_{\epsilon 2} \textit{Enroll}))$ SQL:
 - SELECT e1.SID AS SID FROM Enroll AS e1, Enroll AS e2 WHERE e1.SID = e2.SID AND e1.CID <> e2.CID;
 - AS keyword is completely optional

A more complicated example

 Titles of all courses that Bart and Lisa are taking together

Tip: Write the FROM clause first, then WHERE, and then SELECT

Why SFW statements?

Out of many possible ways of structuring SQL statements, why did the designers choose SELECT-FROM-WHERE?

11

- A large number of queries can be written using only selection, projection, and cross product (or join)
- Any query that uses only these operators can be written in a canonical form: π_L (σ_p (R₁ × ... × R_m))
 - Example: $\pi_{R.A, S.B} (R \bowtie_{p1} S) \bowtie_{p2} (\pi_{T.C} \sigma_{p3} T) = \pi_{R.A, S.B, T.C} \sigma_{p1 \land p2 \land p3} (R \times S \times T)$
- SELECT-FROM-WHERE captures this canonical form

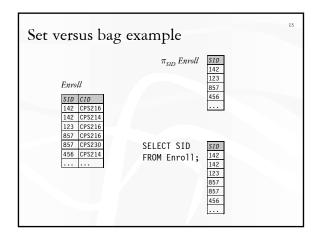
Set versus bag semantics

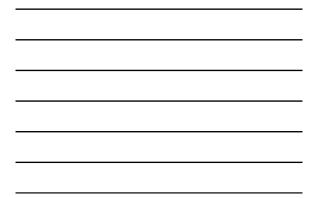
♦ Set

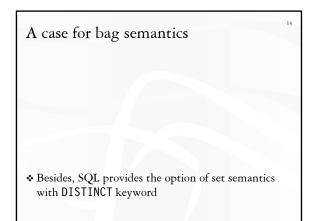
- No duplicates
- Relational model and algebra use set semantics

✤ Bag

- Duplicates allowed
- Number of duplicates is significant
- SQL uses bag semantics by default

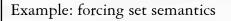






Operational semantics of SFW

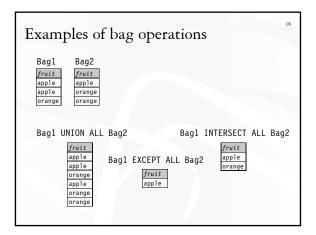
- * SELECT [DISTINCT] E_1 , E_2 , ..., E_n FROM R_1 , R_2 , ..., R_m WHERE condition;
- ★ For each t_1 in R_1 : For each t_2 in R_2 : For each t_m in R_m : If condition is true over $t_1, t_2, ..., t_m$: Compute and output $E_1, E_2, ..., E_n$ If DISTINCT is present Eliminate duplicate rows in output
- t_1, t_2, \dots, t_m are often called tuple variables



- \clubsuit SID's of students who take at least two courses
 - SELECT e1.SID AS SID FROM Enroll AS e1, Enroll AS e2 WHERE e1.SID = e2.SID AND e1.CID <> e2.CID;
 - SELECT DISTINCT e1.SID AS SID ...
 - Duplicate SID values are removed from the output

SQL set and bag operations

- ✤ UNION, EXCEPT, INTERSECT
 - Set semantics
 - Exactly like set \cup , -, and \cap in relational algebra
- ✤ UNION ALL, EXCEPT ALL, INTERSECT ALL
 - Bag semantics
 - Think of each row as having an implicit count (the number of times it appears in the table)
 - Bag union: sum up the counts from two tables
 - Bag difference: proper-subtract the two counts
 - Bag intersection: take the minimum of the two counts





Examples of set versus bag operations

- Enroll(SID, CID), Club Member(club, SID)
 (SELECT SID FROM ClubMember) EXCEPT (SELECT SID FROM Enroll);
 - (SELECT SID FROM ClubMember) EXCEPT ALL (SELECT SID FROM Enroll);

Table expression

- * Use query result as a table
 - In set and bag operations, FROM clauses, etc.
 - A way to "nest" queries

Example: names of students who are in more clubs than classes

20

SELECT DISTINCT name FROM Student, ((SELECT SID FROM ClubMember) EXCEPT ALL (SELECT SID FROM Enroll)) AS S WHERE Student.SID = S.SID;

Summary of SQL features covered so far

- ✤ Basic CREATE/DROP TABLE
- SELECT-FROM-WHERE statements (select-project-join queries)
- * Set and bag operations
- Nesting queries using table expressions
- The So far, not much more than relational algebra
- Pext: aggregation

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

GROUP BY

23

22

♦ SELECT ... FROM ... WHERE ... GROUP BY list of columns;

* Example: find the average GPA for each age group

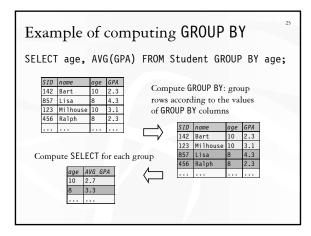
 SELECT age, AVG(GPA) FROM Student GROUP BY age;

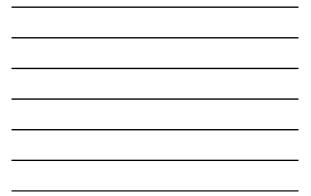
Operational semantics of GROUP BY

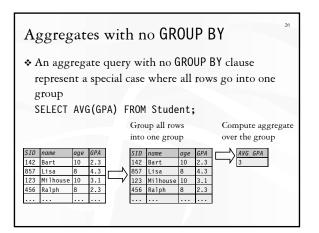
SELECT ... FROM ... WHERE ... GROUP BY ...;

- ✤ Compute FROM (×)
- Compute WHERE (σ)
- Compute GROUP BY: group rows according to the values of GROUP BY columns
- **\diamond** Compute SELECT for each group (π)

To 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

27

- Aggregated, or
- A GROUP BY column
- This restriction ensure that

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

29

30

28

- Used to filter groups based on the group properties (e.g., aggregate values, GROUP BY column values)
- SELECT ... FROM ... WHERE ... GROUP BY ... HAVING condition;
 - Compute FROM (X)
 - 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

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
- 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

Summary of SQL features covered so far

- ✤ Basic CREATE/DROP TABLE
- ✤ SELECT-FROM-WHERE statements
- Set and bag operations
- Table expressions
- 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?

32

33

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)

34

35

 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)

Three-valued logic

When we compare a NULL with another value (including another NULL) using =, >, etc., the result is UNKNOWN 37

39

- \star TRUE = 1, FALSE = 0, UNKNOWN = 0.5
- $\bigstar x \text{ AND } y = \min(x, y)$
- $\mathbf{*} x \, \mathbf{0R} \, y = \max(x, y)$
- $\mathbf{*} \operatorname{NOT} x = 1 x$
- WHERE and HAVING clauses only select rows for output if the condition evaluates to TRUE
 - UNKNOWN is insufficient

Unfortunate consequences

- \$ SELECT AVG(GPA) FROM Student; SELECT SUM(GPA)/COUNT(*) FROM Student;
- \$ SELECT * FROM Student; SELECT * FROM Student WHERE GPA = GPA;
- The careful: NULL breaks many equivalences

Another problem

- Example: Who has NULL GPA values?
 SELECT * FROM Student WHERE GPA = NULL;
 - .
- Works, but ugly
- Introduced built-in predicates IS NULL and IS NOT NULL
 SELECT * FROM Student WHERE GPA IS NULL;

Summary of SQL features covered so far *0

- ✤ Basic CREATE/DROP TABLE
- ♦ SELECT-FROM-WHERE statements
- $\boldsymbol{\diamond}$ Set and bag operations
- * Table expressions
- * Aggregation and grouping
- ✤ NULL's
- P Next: subqueries, modifications, constraints, and views