

## SQL: Part I

CPS 116  
Introduction to Database Systems

## Announcements (September 13)

- ❖ Homework #1 due next Tuesday
  - Do we need a help session on Monday?
- ❖ Course project assigned today
  - Choice of “standard” or “open”
  - One- to three-person teams
  - Two milestones + demo/report
  - Milestone #1 due in 4 weeks, right after fall break

## SQL

- ❖ SQL: Structured Query Language
  - Pronounced “S-Q-L” or “sequel”
  - The standard query language supported by most commercial DBMS
- ❖ A brief history
  - IBM System R
  - ANSI SQL89
  - ANSI SQL92 (SQL2)
  - ANSI SQL99 (SQL3)
  - ANSI SQL 2003 (+OLAP, XML, etc.)

## Creating and dropping tables

- ❖ CREATE TABLE *table\_name* (... , *column\_name*, *column\_type*, ... );
- ❖ DROP TABLE *table\_name*;
- ❖ Examples

```
create table Student (SID integer,
                    name varchar(30), email varchar(30),
                    age integer, GPA float);
create table Course (CID char(10), title varchar(100));
create table Enroll (SID integer, CID char(10));
drop table Student;
drop table Course;
drop table Enroll;
-- everything from -- to the end of the line is ignored.
-- SQL is insensitive to white space.
-- SQL is insensitive to case (e.g., ...Course... is equivalent to
-- ...COURSE...)
```

## Basic queries: SFW statement

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

## 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

7

- ❖ Name of students under 18
  - SELECT name FROM Student WHERE age < 18;
- ❖ When was Lisa born?
  - SELECT 2007 - 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

8

- ❖ SID's and names 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

9

- ❖ SID's of all pairs of classmates
  - Relational algebra query:  
 $\pi_{e1.SID, e2.SID} (\rho_{e1} Enroll \bowtie_{e1.CID = e2.CID \wedge e1.SID > e2.SID} \rho_{e2} Enroll)$
  - SQL:  
SELECT e1.SID AS SID1, e2.SID AS SID2  
FROM Enroll AS e1, Enroll AS e2  
WHERE e1.CID = e2.CID  
AND e1.SID > e2.SID;
  - AS keyword is completely optional

## A more complicated example

10

- ❖ Titles of all courses that Bart and Lisa are taking together
  - SELECT c.title  
FROM Student sb, Student sl, Enroll eb, Enroll el, Course c  
WHERE sb.name = 'Bart' AND sl.name = 'Lisa'  
AND eb.SID = sb.SID AND el.SID = sl.SID  
AND eb.CID = c.CID AND el.CID = c.CID;
  - Tip: Write the FROM clause first, then WHERE, and then SELECT

## Why SFW statements?

11

- ❖ Out of many possible ways of structuring SQL statements, why did the designers choose SELECT-FROM-WHERE?
  - 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:  $\pi_L (\sigma_p (R_1 \times \dots \times R_m))$ 
    - Example:  $\pi_{R.A, S.B} (R \bowtie_{p1} S) \bowtie_{p2} (\pi_{T.C} \sigma_{p3} T) = \pi_{R.A, S.B, T.C} \sigma_{p1 \wedge p2 \wedge p3} (R \times S \times T)$
  - SELECT-FROM-WHERE captures this canonical form

## Set versus bag semantics

12

- ❖ Set
  - No duplicates
  - Relational model and algebra use set semantics
- ❖ Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - SQL uses bag semantics by default

## Set versus bag example

13

Enroll		$\pi_{SID} Enroll$
SID	CID	SID
142	CPS116	142
142	CPS114	123
123	CPS116	857
857	CPS116	456
857	CPS130	...
456	CPS114	...
...	...	...

SELECT SID  
FROM Enroll;

SID
142
142
123
857
857
456
...

## A case for bag semantics

14

- ❖ Efficiency
  - Saves time of eliminating duplicates
- ❖ Which one is more useful?
  - $\pi_{GPA} Student$
  - SELECT GPA FROM Student;
  - The first query just returns all possible GPA's
  - The second query returns the actual GPA distribution
- ❖ Besides, SQL provides the option of set semantics with DISTINCT keyword

## Forcing set semantics

15

- ❖ SID's of all pairs of classmates
  - SELECT e1.SID AS SID1, e2.SID AS SID2  
FROM Enroll AS e1, Enroll AS e2  
WHERE e1.CID = e2.CID  
AND e1.SID > e2.SID;
    - Say Bart and Lisa both take CPS116 and CPS114
  - SELECT DISTINCT e1.SID AS SID1, e2.SID AS SID2  
...
  - With DISTINCT, all duplicate (SID1, SID2) pairs are removed from the output

## Operational semantics of SFW

16

- ❖ SELECT {DISTINCT}  $E_1, E_2, \dots, E_n$   
FROM  $R_1, R_2, \dots, R_m$   
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  $t_1, t_2, \dots, t_m$ :  
Compute and output  $E_1, E_2, \dots, E_n$  as a row  
If DISTINCT is present  
Eliminate duplicate rows in output
- ❖  $t_1, t_2, \dots, t_m$  are often called tuple variables

## SQL set and bag operations

17

- ❖ UNION, EXCEPT, INTERSECT
  - Set semantics
    - Duplicates in input tables, if any, are first eliminated
  - 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 bag operations

18

Bag1	Bag2													
<table border="1"><tr><td>fruit</td></tr><tr><td>apple</td></tr><tr><td>apple</td></tr><tr><td>orange</td></tr></table>	fruit	apple	apple	orange	<table border="1"><tr><td>fruit</td></tr><tr><td>apple</td></tr><tr><td>orange</td></tr><tr><td>orange</td></tr></table>	fruit	apple	orange	orange					
fruit														
apple														
apple														
orange														
fruit														
apple														
orange														
orange														
Bag1 UNION ALL Bag2		Bag1 INTERSECT ALL Bag2												
<table border="1"><tr><td>fruit</td></tr><tr><td>apple</td></tr><tr><td>apple</td></tr><tr><td>orange</td></tr><tr><td>apple</td></tr><tr><td>orange</td></tr><tr><td>orange</td></tr></table>	fruit	apple	apple	orange	apple	orange	orange	<table border="1"><tr><td>fruit</td></tr><tr><td>apple</td></tr></table>	fruit	apple	<table border="1"><tr><td>fruit</td></tr><tr><td>apple</td></tr><tr><td>orange</td></tr></table>	fruit	apple	orange
fruit														
apple														
apple														
orange														
apple														
orange														
orange														
fruit														
apple														
fruit														
apple														
orange														
Bag1 EXCEPT ALL Bag2														
<table border="1"><tr><td>fruit</td></tr><tr><td>apple</td></tr><tr><td>orange</td></tr><tr><td>orange</td></tr></table>	fruit	apple	orange	orange	<table border="1"><tr><td>fruit</td></tr><tr><td>apple</td></tr></table>	fruit	apple							
fruit														
apple														
orange														
orange														
fruit														
apple														

## Examples of set versus bag operations <sup>19</sup>

### ❖ *Enroll*(SID, CID), *ClubMember*(club, SID)

- (SELECT SID FROM ClubMember)  
EXCEPT  
(SELECT SID FROM Enroll);
  - SID's of students who are in clubs but not taking any classes
- (SELECT SID FROM ClubMember)  
EXCEPT ALL  
(SELECT SID FROM Enroll);
  - SID's of students who are in more clubs than classes

## Summary of SQL features covered so far <sup>20</sup>

- ❖ SELECT-FROM-WHERE statements (select-project-join queries)
- ❖ Set and bag operations
- ☞ Next: how to nest SQL queries

## Table expression <sup>21</sup>

- ❖ 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

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

## Scalar subqueries <sup>22</sup>

- ❖ A query that returns a single row can be used as a value in WHERE, SELECT, etc.

- ❖ Example: students at the same age as Bart

```
SELECT *           What's Bart's age?
FROM Student
WHERE age = (SELECT age
             FROM Student
             WHERE name = 'Bart');
```

- ❖ Runtime error if subquery returns more than one row
  - Under what condition will this runtime error never occur?
    - *name* is a key of *Student*
- ❖ What if subquery returns no rows?
  - The value returned is a special NULL value, and the comparison fails
- ❖ Can be used in SELECT to compute a value for an output column

## IN subqueries <sup>23</sup>

- ❖  $x$  IN (*subquery*) checks if  $x$  is in the result of *subquery*
- ❖ Example: students at the same age as (some) Bart

```
SELECT *           What's Bart's age?
FROM Student
WHERE age IN (SELECT age
             FROM Student
             WHERE name = 'Bart');
```

## EXISTS subqueries <sup>24</sup>

- ❖ EXISTS (*subquery*) checks if the result of *subquery* is non-empty

- ❖ Example: students at the same age as (some) Bart

```
▪ SELECT *
  FROM Student AS s ←
  WHERE EXISTS (SELECT * FROM Student
               WHERE name = 'Bart'
               AND age = s.age);
```

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

## Operational semantics of subqueries

25

- ❖ 

```
SELECT *
FROM Student AS s
WHERE EXISTS (SELECT * FROM Student
              WHERE name = 'Bart'
              AND age = s.age);
```
- ❖ For each row *s* in Student
  - Evaluate the subquery with the appropriate value of *s.age*
  - If the result of the subquery is not empty, output *s.\**
- ❖ The DBMS query optimizer may choose to process the query in an equivalent, but more efficient way (example?)

## Scoping rule of subqueries

26

- ❖ To find out which table a column belongs to
  - Start with the immediately surrounding query
  - If not found, look in the one surrounding that; repeat if necessary
- ❖ Use *table\_name.column\_name* notation and AS (renaming) to avoid confusion

## Another example

27

```
SELECT * FROM Student s
WHERE EXISTS
  (SELECT * FROM Enroll e
   WHERE [SID] = s.SID
   AND EXISTS
     (SELECT * FROM Enroll
      WHERE [SID] = [s.SID]
      AND CID <> e.CID));
```

Students who are taking at least two courses

## Quantified subqueries

28

- ❖ A quantified subquery can be used as a value in a WHERE condition
- ❖ Universal quantification (for all):  
... WHERE *x op ALL (subquery)* ...
  - True iff for all *t* in the result of *subquery*, *x op t*
- ❖ Existential quantification (exists):  
... WHERE *x op ANY (subquery)* ...
  - True iff there exists some *t* in the result of *subquery* such that *x op t*
- ☞ Beware
  - In common parlance, "any" and "all" seem to be synonyms
  - In SQL, ANY really means "some"

## Examples of quantified subqueries

29

- ❖ Which students have the highest GPA?
  - ```
SELECT *
FROM Student
WHERE GPA >= ALL
  (SELECT GPA FROM Student);
```
  - ```
SELECT *
FROM Student
WHERE NOT
  (GPA < ANY (SELECT GPA FROM Student);
```
- ☞ Use NOT to negate a condition

## More ways of getting the highest GPA

30

- ❖ Which students have the highest GPA?
  - ```
SELECT *
FROM Student AS s
WHERE NOT EXISTS
  (SELECT * FROM Student
   WHERE GPA > s.GPA);
```
  - ```
SELECT * FROM Student
WHERE SID NOT IN
  (SELECT s1.SID
   FROM Student AS s1, Student AS s2
   WHERE s1.GPA < s2.GPA);
```

## Summary of SQL features covered so far <sup>31</sup>

- ❖ SELECT-FROM-WHERE statements
- ❖ Set and bag operations
- ❖ Table expressions, subqueries
  - Subqueries allow queries to be written in more declarative ways (recall the highest GPA query)
  - But they do not add much expressive power
    - Try translating other forms of subqueries into [NOT] EXISTS, which in turn can be translated into join (and difference)

☞ Next: aggregation and grouping

## Aggregates <sup>32</sup>

- ❖ 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 <sup>33</sup>

- ❖ 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 <sup>34</sup>

- ❖ 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 <sup>35</sup>

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

- ❖ Compute FROM ( $\times$ )
  - ❖ Compute WHERE ( $\sigma$ )
  - ❖ Compute GROUP BY: group rows according to the values of GROUP BY columns
  - ❖ Compute SELECT for each group ( $\pi$ )
    - For aggregation functions with DISTINCT inputs, first eliminate duplicates within the group
- ☞ Number of groups = number of rows in the final output

## Example of computing GROUP BY <sup>36</sup>

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

SID	name	age	GPA
142	Bart	10	2.3
857	Lisa	8	4.3
123	Milhouse	10	3.1
456	Ralph	8	2.3
...	...	...	...

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



SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
...	...	...	...

Compute SELECT for each group



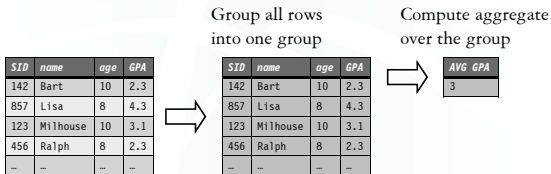
age	AVG GPA
10	2.7
8	3.3
...	...

## Aggregates with no GROUP BY

37

- ❖ An aggregate query with no GROUP BY clause represent a special case where all rows go into one group

```
SELECT AVG(GPA) FROM Student;
```



## Restriction on SELECT

38

- ❖ If a query uses aggregation/group by, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column
- ☞ This restriction ensures that any SELECT expression produces only one value for each group

## Examples of invalid queries

39

- ❖ ~~SELECT~~ ~~SID~~, age FROM Student GROUP BY age;
  - Recall there is one output row per group
  - There can be multiple SID values per group
- ❖ ~~SELECT~~ ~~SID~~, 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
- ☞ Another way of writing the max GPA query?

## HAVING

40

- ❖ 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 ( $\times$ )
  - Compute WHERE ( $\sigma$ )
  - Compute GROUP BY: group rows according to the values of GROUP BY columns
  - Compute HAVING (another  $\sigma$  over the groups)
  - Compute SELECT ( $\pi$ ) for each group that passes HAVING

## HAVING examples

41

- ❖ 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

42

- ❖ SELECT-FROM-WHERE statements
- ❖ Set and bag operations
- ❖ Table expressions, subqueries
- ❖ Aggregation and grouping
  - More expressive power than relational algebra
- ☞ Next: ordering output rows

## ORDER BY <sup>43</sup>

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

## ORDER BY example <sup>44</sup>

- ❖ List all students, sort them by GPA (descending) and name (ascending)
  - SELECT SID, name, age, GPA  
FROM Student  
ORDER BY GPA DESC, name;
  - ASC is the default option
  - Strictly speaking, only output columns can appear in 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;

## Summary of SQL features covered so far <sup>45</sup>

- ❖ SELECT-FROM-WHERE statements
  - ❖ Set and bag operations
  - ❖ Table expressions, subqueries
  - ❖ Aggregation and grouping
  - ❖ Ordering
- ☞ Next: NULL's, outerjoins, data modification, constraints, ...