CompSci 516
Database Systems

Lecture 2
Data Models
and
More SQL

Instructor: Sudeepa Roy

1

Announcements - 01/11 (Tues)

- HW1-Part 1 posted on sakai
  - XML -> Relational database
  - Start working on it!
  - Part 2 will have SQL queries and data analysis
  - Both parts due on 01/27/2022
- Office hour times + Zoom link posted on Ed
- Threads for project teams posted on Ed
  - If you are looking for teammates or a team, please post

2

What is a Database?

3

Revisit: Why use a DBMS?

- Recall the book-selling-platform exercise!
- Some nice properties of a DBMS?

4

Revisit: Why use a DBMS?

- A DBMS is a piece of software (i.e., a big program written by someone else) that makes these tasks easier

5

Why use a DBMS?

6

Why use a DBMS?

7

Why use a DBMS?

8

When NOT to use a DBMS?

9

Data Model

- Applications need to model some real-world units
- Entities:
  - Students, Departments, Courses, Faculty, Organization, Employee, …
- Relationships:
  - Course enrollments by students, Product sales by an organization
- A data model is a collection of high-level data description constructs that hide many low-level storage details

10

Data Model

Can Specify:

1. Structure of the data
   - like arrays or structs in a programming language
   - but at a higher level (conceptual model)
2. Operations on the data
   - unlike a programming language, not any operation can be performed
   - allow limited sets of queries and modifications
   - a strength, not a weakness!
3. Constraints on the data
   - what the data can be
   - e.g., a movie has exactly one title

11

Important Data Models

- Structured Data
- Semi-structured Data
- Unstructured Data

What are these?

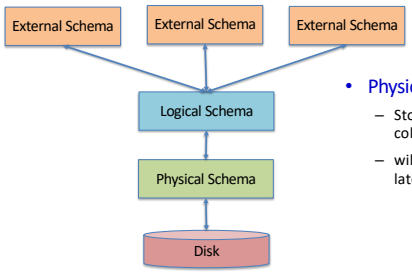
12

Important Data Models

- Structured Data
  - All elements have a fixed format
  - Relational Model (table, Lecture-1)
- Semi-structured Data
  - Some structure but not fixed
  - Hierarchically nested tagged-elements in tree structure
  - XML
- Unstructured Data
  - No structure
  - Text, image, audio, video (still some structure)

13

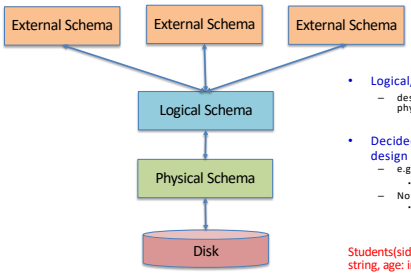
Levels of Abstractions in a DBMS



- Physical schema
  - Storage as files, row vs. column store, indexes
  - will discuss these in later lectures

14

Levels of Abstractions in a DBMS

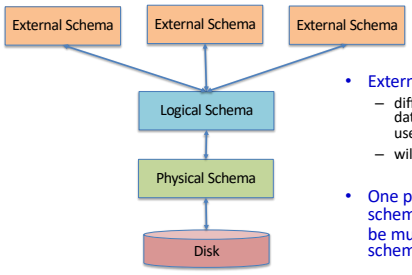


- Logical/Conceptual schema
  - describes the stored data in the physical schema
- Decided by conceptual schema design
  - e.g. ER Diagram
    - not covered in this course
  - Normalization
    - will be covered

Students(sid: string, name: string, login: string, age: integer, gpa: real)

15

Levels of Abstractions in a DBMS



- External schema
  - different "views" of the database to different users
  - will discuss views later
- One physical and logical schema but there can be multiple external schemas

16

Data Independence

- Application programs are insulated from changes in the way the data is structured and stored
- A very important property of a DBMS
- Logical and Physical

17

Logical Data Independence

- Users can be shielded from changes in the logical structure of data
- e.g. Students:
  Students(sid: string, name: string, login: string, age: integer, gpa: real)
- Divide into two relations
  Students_public(sid: string, name: string, login: string)
  Students_private(sid: string, age: integer, gpa: real)
- Still a "view" Students can be obtained using the above new relations
  - by "joining" them with sid
- A user who queries this view Students will get the same answer as before

18

Physical Data Independence

- The logical/conceptual schema insulates users from changes in physical storage details
  - how the data is stored on disk
  - the file structure
  - the choice of indexes
- The application remains unaltered
  - But the performance may be affected by such changes

19

XML: A brief overview

20

Semi-structured Data and XML

- XML: Extensible Markup Language
- Will not be covered in detail in class, but many datasets available to download (DBLP, Yelp) are in this form
  - You will download the DBLP publication dataset (https://dblp.org/, CS Bibliography) in XML format and transform into relational form (in HW1)
- Data does not have a fixed schema
  - "Attributes" are part of the data
  - The data is "self-describing"
  - Tree-structured

21

XML: Example

Attributes

```
<article mdate="2011-01-11" key="journals/acta/Saxena96">
    <author>Sanjeev Saxena</author>
    <title>Parallel Integer Sorting and Simulation Amongst CRCW
        Models.</title>
    <pages>607-619</pages>
    <year>1996</year>
    <volume>33</volume>
    <journal>Acta Inf.</journal>
    <number>7</number>
    <url>db/journals/acta/acta33.html#Saxena96</url>
    <ee>http://dx.doi.org/10.1007/BF03036466</ee>
</article>
```

Elements

22

Attribute vs. Elements

- Elements can be repeated and nested
- Attributes are unique and atomic

23

Why XML?

+ Serves as a model suitable for integration of databases containing similar data with different schemas
  - e.g. try to integrate two student databases: S1(sid, name, gpa) and S2(sid, dept, year)
  - Many "NULL"s (for unknown data) if done in relational model, very easy in XML

+ Flexible – easy to change the schema and data

- Makes query processing more difficult

Which one is easier?

- XML (semi-structured) to relational (structured)

or

- relational (structured) to XML (semi-structured)?

24

XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <title>Database Management Systems</title>
    <pubisher> McGraw Hill
</book>
```

What is a good relational schema?

25

XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <title>Database Management Systems</title>
    <pubisher> McGraw Hill</publisher>
</book>
```

Title	Publisher	Author1	Author2

26

XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Garcia-Molina</author>
    <author>Ullman</author>
    <author>Widom</author>
    <title>Database Systems – The Complete Book</title>
    <pubisher>Prentice Hall</publisher>
</book>
```

Does not work

Title	Publisher	Author1	Author2

Better design?

27

XML to Relational Model

28

XML to Relational Model

- Problem 2: Missing attributes

```
<book>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <title>Database Management Systems</title>
    <pubisher> McGraw Hill
    <edition>Third</edition>
</book>
<book>
    <author>Garcia-Molina</author>
    <author>Ullman</author>
    <author>Widom</author>
    <title>Database Systems – The Complete Book</title>
    <pubisher>Prentice Hall</publisher>
</book>
```

BookId	Title	Publisher	Edition
b1	Database Management Systems	McGraw Hill	Third
b2	Database Systems – The Complete Book	Prentice Hall	null

29

Summary: Data Model

- Relational data model is the most standard for database managements
  - semi-structured model/XML is also used in practice – you will use them in hw assignments
  - unstructured data (text/photo/video) is unavoidable, but won't be covered in this class
- A DBMS provides data independence and insulates the application programmer from many low level details
- We will learn about those low level details as well as high level data management in this course

30

SQL Programming

31

SQL Programming: Working with SQL through an API

- E.g.: Python psycopg2, JDBC, ODBC (C/C++/VB)
  - All based on the SQL/CLI (Call-Level Interface) standard
  - You can use any of these in HW1
- The application program sends SQL commands to the DBMS at runtime
- Responses/results are converted to objects in the application program

32

Optional slide

Example API: Python psycopg2

```
import psycopg2
conn = psycopg2.connect(dbname='beers')
cur = conn.cursor()
# list all drinkers:
cur.execute('SELECT * FROM Drinker')
for drinker, address in cur:
    print(drinker + ' lives at ' + address)
# print menu for bars whose name contains "a":
cur.execute('SELECT * FROM Serves WHERE bar LIKE %s', ('%a%',))
for bar, beer, price in cur:
    print('{} serves {} at ${:,.2f}'.format(bar, beer, price))
cur.close()
conn.close()
```

beers database
Drinker(drinker, address)
Serves(bar, beer, price)

You can iterate over cur one tuple at a time

Placeholder for query parameter

Tuple of parameter values, one for each %s
(note that the trailing "," is needed when the tuple contains only one value)

33

Optional slide

More psycopg2 examples

```
# "commit" each change immediately (later in transactions)—need to set this option just once at the start of the session
conn.set_session(autocommit=True)
# ...
bar = input('Enter the bar to update: ').strip()
beer = input('Enter the beer to update: ').strip()
price = float(input('Enter the new price: '))
try:
    cur.execute("""
UPDATE Serves
SET price   = %s
WHERE bar = %s AND beer = %s""", (price, bar, beer))
    if cur.rowcount != 1:
        print('{} row(s) updated: correct bar/beer?'\
              .format(cur.rowcount))
except Exception as e:
    print(e)
```

# of tuples modified

Exceptions can be thrown
(e.g., if positive-price constraint is violated)

34

Optional slide

Prepared statements: motivation

```
while True:
    # Input bar, beer, price...
    cur.execute("""
UPDATE Serves
SET price = %s
WHERE bar = %s AND beer = %s""", (price, bar, beer))
    # Check result...
```

- Every time we send an SQL string to the DBMS, it must perform parsing, semantic analysis, optimization, compilation, and finally execution
- A typical application issues many queries with a small number of patterns (with different parameter values)
- Can we reduce this overhead?

35

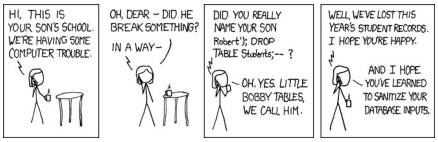
Prepared statements: example

```
cur.execute("""                 # Prepare once (in SQL).
PREPARE update_price AS         # Name the prepared plan,
UPDATE Serves
SET price = $1                  # and note the $1, $2, ... notation for
WHERE bar = $2 AND beer = $3""") # parameter placeholders.
while True:
    # Input bar, beer, price...
    cur.execute('EXECUTE update_price(%s, %s, %s);',\     # Execute many times.
                (price, bar, beer))
                                # Note the switch back to %s for parameter placeholders.
    # Check result...
```

- The DBMS performs parsing, semantic analysis, optimization, and compilation only once, when it "prepares" the statement
- At execution time, the DBMS only needs to check parameter types and validate the compiled plan
- Most other API's have better support for prepared statements than psycopg2
  - E.g., they would provide a cur.prepare() method

36

SQL Injection Attack



http://xkcd.com/327/

- The school probably had something like:

```
cur.execute("SELECT * FROM Students " + \
      "WHERE (name = '" + name + "')")
```

where name is a string input by user

- Suppose name = Robert'; DROP TABLE Students
  - Drop deletes a table
  - -- starts a comment
  - Becomes SELECT * FROM Students WHERE (name = 'Robert'; DROP TABLE Students; -- ')

37

Guarding against SQL injection

- Escape certain characters in a user input string, to ensure that it remains a single string
  - E.g., ', which would terminate a string in SQL, must be replaced by '' (two single quotes in a row) within the input string
- Luckily, most API's provide ways to "sanitize" input automatically (if you use them properly)
  - E.g., pass parameter values in psycopg2 through %s's
- Check out Ashley Madison data breach story or https://medium.com/five-guys-facts/sql-injection-98199af86c9

38

Very important

Understand the Course-Policy

See "what is allowed/not allowed"

39

Back to SQL!

40

The SQL Query Language

- To find all 18 year old students, we can write:

all attributes

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```


41

Querying Multiple Relations

- What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instances of Enrolled and Students:

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enrolled

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get: ??

42

Querying Multiple Relations

- What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instances of Enrolled and Students:

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enrolled

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get:

43

Read yourself, after reading the next few slides first

Basic SQL Query

```
SELECT    [DISTINCT] <target-list>
FROM      <relation-list>
WHERE     <qualification>
```

- relation-list A list of relation names
  - possibly with a "range variable" after each name
- target-list A list of attributes of relations in relation-list
- qualification Comparisons
  - (Attr op const) or (Attr1 op Attr2)
  - where op is one of = , <, >, <=, >= combined using AND, OR and NOT
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates
  - Default is that duplicates are not eliminated!

44

Read yourself, after reading the next few slides first

Conceptual Evaluation Strategy

```
SELECT    [DISTINCT] <target-list>
FROM      <relation-list>
WHERE     <qualification>
```

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *<relation-list>*
  - Discard resulting tuples if they fail *<qualifications>*
  - Delete attributes that are not in *<target-list>*
  - If DISTINCT is specified, eliminate duplicate rows
- This strategy is probably the least efficient way to compute a query!
  - An optimizer will find more efficient strategies to compute the same answers

45

Example of Conceptual Evaluation

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103
```

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

What does this query return?

46

Example of Conceptual Evaluation

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103
```

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Step 1: Form "**cross product**" of Sailor and Reserves

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
22	dustin	7	45	58	103	11/12/96
31	lubber	8	55	22	101	10/10/96
31	lubber	8	55	58	103	11/12/96
58	rusty	10	35	22	101	10/10/96
58	rusty	10	35	58	103	11/12/96

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

47

Example of Conceptual Evaluation

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103
```

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Step 2: Discard tuples that do not satisfy <qualification>

sid	sname	rating	age	sid	bid	day
~~22~~	~~dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10/10/96~~
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
58	rusty	10	35	58	103	11/12/96

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

48

Example of Conceptual Evaluation

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Step 3: Select the specified attribute(s)

sid	sname	rating	age	sid	bid	day
~~22~~	~~dustin~~	~~7~~	~~45~~	~~22~~	~~101~~	~~10/10/96~~
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~rusty~~	~~10~~	~~35~~	~~22~~	~~101~~	~~10/10/96~~
58	rusty	10	35	58	103	11/12/96

49

Recap

```
3  SELECT  S.sname
1  FROM    Sailors S, Reserves R
2  WHERE   S.sid=R.sid AND R.bid=103
```

Always start from "FROM" -- form cross product
Apply "WHERE" -- filter out some tuples (rows)
Apply "SELECT" -- filter out some attributes (columns)

Ques. Does this get evaluated this way in practice in a Database Management System (DBMS)?

No! This is conceptual evaluation for finding what is correct!
We will learn about join and other operator algorithms later

50

A Note on "Range Variables"

- Sometimes used as a short-name
- The previous query can also be written as:

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND bid=103
```

OR

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid
        AND bid=103
```

It is good style, however, to use range variables always!

51

A Note on "Range Variables"

- Really needed only if the same relation appears twice in the FROM clause (called self-joins)
- Find pairs of Sailors of same age

```
SELECT  S1.sname, S2. name
FROM    Sailors S1, Sailors S2
WHERE   S1.age = S2.age AND S1.sid < S2.sid
```

Why do we need the 2nd condition?

52

Find sailor ids who've reserved at least one boat

```
SELECT  ????
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid
```

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

53

Find sailor ids who've reserved at least one boat

```
SELECT  S.sid
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid
```

- Would adding DISTINCT to this query make a difference?

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

54

Find sailors who've reserved at least one boat

```
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
```

- Would adding DISTINCT to this query make a difference?
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?

Sailor

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

55

Simple Aggregate Operators

Check yourself:
What do these queries compute?

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)
single column

```
SELECT COUNT (*)
FROM Sailors S
```

```
SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating=10
```

```
SELECT S.sname
FROM Sailors S
WHERE S.rating= (SELECT MAX(S2.rating)
                 FROM Sailors S2)
```

```
SELECT COUNT (DISTINCT S.rating)
FROM Sailors S
WHERE S.sname='Bob'
```

```
SELECT AVG ( DISTINCT S.age)
FROM Sailors S
WHERE S.rating=10
```


56

Next: different types of joins

- Theta-join
- Equi-join
- Natural join
- Outer Join

57

Condition/Theta Join

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid=R.sid and age >= 40
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Form cross product, discard rows that do not satisfy the condition

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
58	rusty	10	35	22	101	10/10/96
58	rusty	10	35	58	103	11/12/96

sid	bid	day
22	101	10/10/96
58	103	11/12/96

58

Equi Join

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid=R.sid and age = 45
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

A special case of theta join
Join condition only has equality predicate =

sid	sname	rating	age	sid	bid	day
22	dustin	7	45	22	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~58~~	~~103~~	~~11/12/96~~
58	rusty	10	35	22	101	10/10/96
~~58~~	~~rusty~~	~~10~~	~~35~~	~~58~~	~~103~~	~~11/12/96~~

sid	bid	day
22	101	10/10/96
58	103	11/12/96

59

Natural Join

```
SELECT *
FROM Sailors S NATURAL JOIN Reserves R
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

A special case of equi join
Equality condition on ALL common predicates (sid)
Duplicate columns are eliminated

sid	sname	rating	age	bid	day
22	dustin	7	45	101	10/10/96
~~22~~	~~dustin~~	~~7~~	~~45~~	~~103~~	~~11/12/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~101~~	~~10/10/96~~
~~31~~	~~lubber~~	~~8~~	~~55~~	~~103~~	~~11/12/96~~
58	rusty	10	35	101	10/10/96
58	rusty	10	35	103	11/12/96

sid	bid	day
22	101	10/10/96
58	103	11/12/96

60

Outer Join

```
SELECT S.sid, R. bid
FROM Sailors S LEFT OUTER JOIN Reserves R
ON S.sid=R.sid
```

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

Preserves all tuples from the left table whether or not there is a match
if no match, fill attributes from right with null
Similarly RIGHT/FULL outer join

sid	bid
22	101
31	null
58	103

sid	bid	day
22	101	10/10/96
58	103	11/12/96

61

Expressions and Strings

```
SELECT S.age,  age1=S.age-5,  2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching
- *Find triples (of ages of sailors and two fields defined by expressions) for sailors*
  - *whose names begin and end with B and contain at least three characters*
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters
  - You will need these often

62

Find sid's of sailors who've reserved a red or a green boat

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples
  - can themselves be the result of SQL queries
- If we replace OR by AND in the first version, what do we get?
- Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

63

Find sid's of sailors who've reserved a red and a green boat

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

64

Find sid's of sailors who've reserved a red and a green boat

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- Does not work ->
- INTERSECT: Can be used to compute the intersection of any two union-compatible sets of tuples.
  - Included in the SQL/92 standard, but some systems don't support it

65

Nested Queries

Find names of sailors who've reserved boat #103:

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid=103)
```

Sailors (sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- A very powerful feature of SQL:
  - a WHERE/FROM/HAVING clause can itself contain an SQL query
- To find sailors who've not reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a nested loops evaluation
  - For each Sailors tuple, check the qualification by computing the subquery

66

Nested Queries with Correlation

Find names of sailors who've reserved boat #103:

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
              FROM Reserves R
              WHERE R.bid=103 AND S.sid=R.sid)
```

- EXISTS is another set comparison operator, like IN
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple

67

Nested Queries with Correlation

Find names of sailors who've reserved boat #103 at most once:

```
SELECT S.sname
FROM Sailors S
WHERE UNIQUE (SELECT R.bid
              FROM Reserves R
              WHERE R.bid=103 AND S.sid=R.sid)
```

- If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103
  - UNIQUE checks for duplicate tuples

68

More on Set-Comparison Operators

- We've already seen IN, EXISTS and UNIQUE
- Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL, *op* IN
  - where op : >, <, =, <=, >=
- Find sailors whose rating is greater than that of some sailor called Horatio
  - similarly ALL

```
SELECT *
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.sname='Horatio')
```


69

Summary

- Relational Data
- SQL
  - Semantic
  - Join
  - Simple Aggregates
  - Nested Queries

70