CompSci 516
Database Systems

Lecture 2
SQL

Instructor: Sudeepa Roy
Announcements

• If you are enrolled to the class, but have not received the email from Piazza, please send me an email

• If you are on the waitlist and want to enroll, please send me an email

• HW1 will be released soon (~tomorrow)

• TA office hours:
  – Yuchao: LSRC D309, Mondays 1:30-2:30 pm
  – Tianpeng: LSRC D344, Wednesdays 1:30-2:30 pm
Recap: Lecture 1

• Why use a DBMS
• Structured data model: Relational data model
  – table, schema, instance, tuples, attributes
  – bag and set semantic
• Logical and physical data independence
Today’s topic

• Overview of XML

• SQL in a nutshell
  – Reading material: [RG] Chapters 3 and 5
  – Additional reading for practice: [GUW] Chapter 6

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
XML: an overview
Semi-structured Data and XML

• XML: Extensible Markup Language

• Will not be covered in detail in class, but many datasets available to download are in this form
  – You will download the DBLP dataset 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
XML: Example

<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>
Attribute vs. Elements

- Elements can be repeated and nested
- Attributes are unique and atomic
XML vs. Relational Databases

+ 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 “nulls” if done in relational model, very easy in XML
• NULL = A keyword to denote missing or unknown values

+ 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)?
XML to Relational Model

• Problem 1: Repeated attributes

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

What is a good relational schema?
• Problem 1: Repeated attributes

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

<table>
<thead>
<tr>
<th>Title</th>
<th>Publisher</th>
<th>Author1</th>
<th>Author2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What if the paper has a single author?
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>
  <publisher>Prentice Hall</publisher>
</book>

Does not work
## XML to Relational Model

### Book

<table>
<thead>
<tr>
<th>BookId</th>
<th>Title</th>
<th>Publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>Database Management Systems</td>
<td>McGraw Hill</td>
</tr>
<tr>
<td>b2</td>
<td>Database Systems – The Complete Book</td>
<td>Prentice Hall</td>
</tr>
</tbody>
</table>

### BookAuthoredBy

<table>
<thead>
<tr>
<th>BookId</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>Ramakrishan</td>
</tr>
<tr>
<td>b1</td>
<td>Gehrke</td>
</tr>
<tr>
<td>b2</td>
<td>Garcia-Molina</td>
</tr>
<tr>
<td>b2</td>
<td>Ullman</td>
</tr>
<tr>
<td>b2</td>
<td>Widom</td>
</tr>
</tbody>
</table>
XML to Relational Model

• Problem 2: Missing attributes

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

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

<table>
<thead>
<tr>
<th>BookId</th>
<th>Title</th>
<th>Publisher</th>
<th>Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>Database Management Systems</td>
<td>McGraw Hill</td>
<td>Third</td>
</tr>
<tr>
<td>b2</td>
<td>Database Systems – The Complete Book</td>
<td>Prentice Hall</td>
<td>null</td>
</tr>
</tbody>
</table>

Duke CS, Fall 2018
Summary: Data Models

• Relational data model is the most standard for database managements
  – and is the main focus of this course

• 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
SQL
(Structured Query Language)
Relational Query Languages

• A major strength of the relational model: supports simple, powerful querying of data.

• Queries can be written intuitively, and the DBMS is responsible for an efficient evaluation
  – The key: precise semantics for relational queries
  – Based on a sound theory!
  – Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.
The SQL Query Language

• Developed by IBM (systemR) in the 1970s based on Ted Codd’s relational model
  – First called “SEQUEL” (Structured English Query Language)
• First commercialized by Oracle (then Relational Software) in 1979
• Standards by ANSI and ISO since it is used by many vendors
  – SQL-86, -89 (minor revision), -92 (major revision), -96, -99 (major extensions), -03, -06, -08, -11, -16
Purposes of SQL

• Data Manipulation Language (DML)
  – Querying: SELECT-FROM-WHERE
  – Modifying: INSERT/DELETE/UPDATE

• Data Definition Language (DDL)
  – CREATE/ALTER/DROP
The SQL Query Language

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

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

   ![Table]

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

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

   ```sql
   SELECT S.name, S.login
   ```
Querying Multiple Relations

- What does the following query compute?

```sql
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:

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

we get: ??
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:

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

we get:

```
<table>
<thead>
<tr>
<th>S.name</th>
<th>E.cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Topology112</td>
</tr>
</tbody>
</table>
```
Creating Relations in SQL

• Creates the “Students” relation
  – the type (domain) of each field is specified
  – enforced by the DBMS whenever tuples are added or modified

• As another example, the “Enrolled” table holds information about courses that students take

```
CREATE TABLE Students
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL)

CREATE TABLE Enrolled
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2))
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>
Destroying and Altering Relations

DROP TABLE Students

• Destroys the relation Students
  – The schema information \textit{and} the tuples are deleted.

ALTER TABLE Students
  ADD COLUMN firstYear: integer

• The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a \textbf{NULL} value in the new field.
Adding and Deleting Tuples

• Can insert a single tuple using:

\[
\text{INSERT INTO Students (sid, name, login, age, gpa) VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)}
\]

• Can delete all tuples satisfying some condition (e.g., name = Smith):

\[
\text{DELETE FROM Students S WHERE S.name = 'Smith'}
\]
Integrity Constraints (ICs)

- **IC**: condition that must be true for any instance of the database
  - e.g., domain constraints
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified

- A **legal** instance of a relation is one that satisfies all specified ICs
  - DBMS will not allow illegal instances

- If the DBMS checks ICs, stored data is more faithful to real-world meaning
  - Avoids data entry errors, too!
Keys in a Database

- Key / Candidate Key
- Primary Key
- Super Key
- Foreign Key

- Primary key attributes are underlined in a schema
  - Person(pid, address, name)
  - Person2(address, name, age, job)
Primary Key Constraints

• A set of fields is a key for a relation if:
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key

• Part 2 false? A superkey

• If there are > 1 keys for a relation, one of the keys is chosen (by DBA = DB admin) to be the primary key
  – E.g., sid is a key for Students
  – The set {sid, gpa} is a superkey.

• Any possible benefit to refer to a tuple using primary key (than any key)?
Primary and Candidate Keys in SQL

• Possibly many candidate keys
  – specified using `UNIQUE`
  – one of which is chosen as the primary key.

• “For a given student and course, there is a single grade.”

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY ???)
Primary and Candidate Keys in SQL

• Possibly many **candidate keys**
  – specified using **UNIQUE**
  – one of which is chosen as the primary key.

• “For a given student and course, there is a single grade.”

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2),
     PRIMARY KEY (sid, cid)
    )
```
Primary and Candidate Keys in SQL

• Possibly many candidate keys
  – specified using \texttt{UNIQUE}
  – one of which is chosen as the primary key.

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid, cid))

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARKEY ???,
UNIQUE ???)

• “For a given student and course, there is a single grade.”

• vs.

• “Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.”
Primary and Candidate Keys in SQL

• Possibly many candidate keys
  – specified using UNIQUE
  – one of which is chosen as the primary key.

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid)
  )

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY sid, UNIQUE (cid, grade)
  )

• “For a given student and course, there is a single grade.”

• vs.

• “Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.”
Primary and Candidate Keys in SQL

• Possibly many candidate keys
  – specified using UNIQUE
  – one of which is chosen as the primary key.

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))

• “For a given student and course, there is a single grade.”

• vs.

• “Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.”

• Used carelessly, an IC can prevent the storage of database instances that arise in practice!

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))
Foreign Keys, Referential Integrity

• **Foreign key**: Set of fields in one relation that is used to ‘refer’ to a tuple in another relation
  – Must correspond to primary key of the second relation  
  – Like a ‘logical pointer’

• **E.g. sid** is a foreign key referring to **Students**:
  – If all foreign key constraints are enforced, referential integrity is achieved
  – i.e., no dangling references
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

```
CREATE TABLE Enrolled
    (sid CHAR(20), cid CHAR(20), grade CHAR(2),
     PRIMARY KEY (sid,cid),
     FOREIGN KEY (sid) REFERENCES Students )
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eeecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
Enforcing Referential Integrity

• Consider Students and Enrolled
  – sid in Enrolled is a foreign key that references Students.

• What should be done if an Enrolled tuple with a non-existent student id is inserted?
  – Reject it!

• What should be done if a Students tuple is deleted?
  – Three semantics allowed by SQL
    1. Also delete all Enrolled tuples that refer to it (cascade delete)
    2. Disallow deletion of a Students tuple that is referred to
    3. Set sid in Enrolled tuples that refer to it to a default sid
    4. (in addition in SQL): Set sid in Enrolled tuples that refer to it to a special value null, denoting `unknown’ or `inapplicable’

• Similar if primary key of Students tuple is updated
Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

CREATE TABLE Enrolled
(sid CHAR(20) DEFAULT '000',
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid) REFERENCES Students
  ON DELETE CASCADE
  ON UPDATE SET DEFAULT )
Where do ICs Come From?

• ICs are based upon the semantics of the real-world enterprise that is being described in the database relations

• Can we infer ICs from an instance?
  – We can check a database instance to see if an IC is violated, but we can **NEVER** infer that an IC is true by looking at an instance.
  – An IC is a statement about all possible instances!
  – From example, we know name is not a key, but the assertion that sid is a key is given to us.

• Key and foreign key ICs are the most common; more general ICs supported too
Example Instances

• What does the key (sid, bid, day) in Reserves mean?

• If the key for the Reserves relation contained only the attributes (sid, bid), how would the semantics differ?

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Next...

• Querying using SQL
  – semantic
  – joins
  – group bys and aggregates
  – nested queries
## Basic SQL Query

```sql
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!
### Conceptual Evaluation Strategy

Select

<table>
<thead>
<tr>
<th>Conceptual Evaluation Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Semantics</strong> of an SQL query defined in terms of the following conceptual evaluation strategy:</td>
</tr>
<tr>
<td>- Compute the cross-product of <code>&lt;relation-list&gt;</code></td>
</tr>
<tr>
<td>- Discard resulting tuples if they fail <code>&lt;qualifications&gt;</code></td>
</tr>
<tr>
<td>- Delete attributes that are not in <code>&lt;target-list&gt;</code></td>
</tr>
<tr>
<td>- If <code>DISTINCT</code> is specified, eliminate duplicate rows</td>
</tr>
</tbody>
</table>

- **This strategy is probably the least efficient way to compute a query!**
  - An optimizer will find more efficient strategies to compute the same answers
Example of Conceptual Evaluation

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

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

**Sailor**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

**Reserves**

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Example of Conceptual Evaluation

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

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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
**Example of Conceptual Evaluation**

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

**Sailor**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

**Reserves**

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Step 3: Select the specified attribute(s)

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
A Note on “Range Variables”

• Really needed only if the same relation appears twice in the FROM clause
  – 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!
Find sailor ids who’ve reserved at least one boat

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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>Sailor</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
<td>rating</td>
<td>age</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserves</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bid</td>
<td>day</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td></td>
</tr>
</tbody>
</table>
Find sailors who’ve reserved at least one boat

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

- **Would adding `DISTINCT` to this query make a difference?**
  - Note that if there are multiple bids for the same sid, you get multiple output tuples for the same sid
  - Without distinct, you get them multiple times

- **What is the effect of replacing `S.sid` by `S.sname` in the `SELECT` clause?**
  - Would adding `DISTINCT` to this variant of the query make a difference even if one sid reserves at most one bid?