ICS 321 Spring 2012

The Database Language SQL (i)

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Example Relations

• Sailors(
  sid: integer,
sname: string,
rating: integer,
age: real)

• Boats(
bid: integer,
bname: string,
color: string)

• Reserves(
sid: integer,
bid: string,
day: date)
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 \(<\), \( >\), \( \leq\), \( \geq\), \( =\), \( \neq\)\) 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!
Example Q1

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

Without range variables

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

- Range variables really needed only if the same relation appears twice in the FROM clause.
- Good style to always use range variables
Conceptual Evaluation Strategy

• Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  1. Compute the cross-product of relation-list.
  2. Discard resulting tuples if they fail qualifications.
  3. Delete attributes that are not in target-list.
  4. 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.
Example Q1: conceptual evaluation

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

<table>
<thead>
<tr>
<th>S.sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>R.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>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Conceptual Evaluation Steps:
1. Compute cross-product
2. Discard disqualified tuples
3. Delete unwanted attributes
4. If DISTINCT is specified, eliminate duplicate rows.
Q2: Find sailors who’ve reserved at least one boat

SELECT S1.sid
FROM Sailors S1, Reserves R1
WHERE S1.sid=R1.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? Would adding DISTINCT to this variant of the query make a difference?
Q3: Find the colors of boats reserved by Lubber

```
SELECT B1.color
FROM Sailors S1, Reserves R1, Boats B1
WHERE S1.sid=R1.sid
AND R1.bid=B1.bid
AND S1.sname='Lubber'
```

**S1**

<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>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**R1**

<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>

**B1**

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>Blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>Red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>Red</td>
</tr>
</tbody>
</table>
Expressions

• WHERE-qualification can contain expressions
• SELECT-list can also contain arithmetic or string expressions over the column names
• Example: compute a new "age adjusted" rating for each sailor whose rating satisfies a special formula

```
SELECT S1.sname, 
    S1.rating * S1.age / 100 
AS NewRating 
FROM Sailors S1 
WHERE S1.rating – 5.0 > S1.age / 12.0
```

<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>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
The result of any arithmetic operator +, -, /, \* involving a NULL is always \textbf{NULL}

The result of any comparison operator like =, >, < is always \textbf{UNKNOWN}
The “UNKNOWN” truth-value

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X AND Y</th>
<th>X OR Y</th>
<th>NOT X</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>U</td>
<td>U</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>T</td>
<td>U</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>F</td>
<td>F</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>U</td>
<td>F</td>
<td>U</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

- If TRUE = 1, False = 0, UNKNOWN=0.5
  - AND : min, OR : max, NOT : 1-v
Strings & Pattern Matching

• String comparisons via the comparisons operators ( <, >, =, etc), but take note of collations
  – i.e. determines the ordering. Lexicographic, languages etc
• SQL supports pattern matching via the **LIKE** operator and wildcards
  – ```%```: zero or more arbitrary chars
  – ```_```: any one char

```
SELECT S1.sname, S1.rating
FROM Sailors S1
WHERE S1.sname LIKE `L_%`
```
• Dates and time constants are specified using strings and “cast” into the date/time datatypes using functions.

```
SELECT R*
FROM Reserves R
WHERE R.day = DATE '2010-10-02'
```

```
TIME '15:00:02.5'
TIMESTAMP '2010-10-02 15:00:02'
```
Ordering the Output

- ORDER BY clause sorts the result of the SQL query according to the given column(s).

```
SELECT S1.sname, S1.rating
FROM Sailors S1
ORDER BY S1.rating DESC
```