EECS 647: Introduction to Database Systems

Instructor: Luke Huan

Spring 2009
Administrative

- Homework 4 is assigned today, due April 13th.
  - It involves substantial coding work, starts earlier
- Midterm 2 is scheduled on April 20th. Covering SQL and DBMS Indexing
  - More details later
- The presentation of the final project is due May 4th and the final report is due May 6th
Some Exercises

Sailors (sid: INTEGER, surname: string, rating: INTEGER, age: REAL)
Boats (bid: INTEGER, bname: string, color: string)
Reserves (sid: INTEGER, bid: INTEGER)

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Nancy</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Nina</td>
<td>red</td>
</tr>
<tr>
<td>102</td>
<td>Pinta</td>
<td>blue</td>
</tr>
<tr>
<td>103</td>
<td>Santa</td>
<td>red</td>
</tr>
<tr>
<td></td>
<td>Maria</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
</tr>
</tbody>
</table>
Exercise I

- Find sid’s of sailors who’ve reserved a red AND a green boat
- SELECT R1.sid
  FROM Boats B1, Reserves R1, Boats B2, Reserves R2
  WHERE B1.color='red' AND B2.color='green'
  AND R1.sid=R2.sid

- SELECT sid
  FROM Boats, Reserves
  WHERE B.color = ‘red’
  INTERSECT
  (SELECT sid
   FROM Boats B, Reserves R
   WHERE B.color = ‘green’ and B.sid = R.sid)
How to Write SQL Statement

- Before you start
  - Identify information source
  - Specify target
- Procedures:
  - First write “From”
  - “Where”
  - “Select”
Exercise II

- Find sid’s of sailors who have not reserved a boat

```sql
SELECT sid
FROM Sailors
(EXCEPT
(SELECT R.sid
FROM Reserves R))
```

Non-monotonic operation!
Exercise III (a tough one)

- Find sailors who’ve reserved all boats

```
SELECT  Sid
FROM    Sailor
EXCEPT  
SELECT  Sid
FROM    (SELECT  bid, sid
          FROM    Boat, Sailor
          EXCEPT  Reserves)
```

Non-monotonic operation!

#All sailors

#Those who do not reserve all boats

#All possible combinations between sid and bid

#Existing reservations
Today’s Topic

- NULL value
- Outer join
Incomplete information

- Example: \textit{Student (SID, name, age, GPA)}
- Value \textit{unknown}
  - We do not know Nelson’s age
- Value \textit{not applicable}
  - Nelson has not taken any classes yet; what is his GPA?
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 Y2K bug?
  - “00” was used as a missing or invalid year value
Solution 2

- A valid-bit for every column
  - Student (\textit{SID}, name, name\_is\_valid, age, age\_is\_valid, GPA, GPA\_is\_valid)

- Complicates schema and queries
  - SELECT AVG(GPA) FROM Student WHERE GPA\_is\_valid;
Solution 3?

- Decompose the table; missing row = missing value
  - \textit{StudentName} (\textit{SID}, \textit{name})
  - \textit{StudentAge} (\textit{SID}, \textit{age})
  - \textit{StudentGPA} (\textit{SID}, \textit{GPA})
  - \textit{StudentID} (\textit{SID})

- Conceptually the cleanest solution
- Still complicates schema and queries
  - How to get all information about a student in a table?
  - Would natural join work?
SQL’s solution

- A special value **NULL**
  - For every domain
  - Special rules for dealing with NULL’s

- Example: *Student* (*SID*, name, age, GPA)
  - h 789, “Nelson”, NULL, NULL
Three-valued logic

- TRUE = 1, FALSE = 0, UNKNOWN = 0.5
- $x \text{ AND } y = \min(x, y)$
- $x \text{ OR } y = \max(x, y)$
- $\text{NOT } x = 1 - x$

<table>
<thead>
<tr>
<th>AND</th>
<th>True</th>
<th>False</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>True</th>
<th>False</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WHERE and HAVING clauses only select rows for output if the condition evaluates to TRUE
- UNKNOWN is not enough
Computing with NULL’s

- (Arithmetic operation) 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)
- When we compare a NULL with another value (including another NULL) using =, >, etc., the result is UNKNOWN
Unfortunate consequences

- SELECT AVG(GPA) FROM Student
  - 3.4
- SELECT SUM(GPA)/COUNT(*) FROM Student;
  - 2.72
- SELECT * FROM Student;
- SELECT * FROM Student WHERE GPA = GPA
  - Not equivalent

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>19</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>NULL</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
<td>18</td>
<td>2.9</td>
</tr>
</tbody>
</table>
Another problem

- Example: Who has \texttt{NULL} GPA values?
  - SELECT * FROM Student WHERE GPA = NULL;
    - Does not work; never returns anything
  - (SELECT * FROM Student) EXCEPT ALL (SELECT * FROM Student WHERE GPA = GPA)
    - Works, but ugly
  - Introduced built-in predicates \texttt{IS NULL} and \texttt{IS NOT NULL}
    - SELECT * FROM Student WHERE GPA IS NULL;
Outerjoin motivation

- **Example: a master class list**
  
  ```sql
  SELECT c.CID, s.SID
  FROM Enroll e, Student s, Course c
  WHERE e.SID = s.SID and c.CID = e.CID;
  ```

- What if a student take no classes
  
  - For these students, *CID* column should be **NULL**

- What if a course with no student enrolled yet?
  
  - For these courses, *SID* should be **NULL**
Outerjoin

- **SELECT * FROM R FULL OUTER JOIN S ON p**;

- A **full outer join** between R and S includes all rows in the result of *R joins S*, plus
  - “Dangling” R rows (those that do not join with any S rows) padded with NULL’s for S’s columns
  - “Dangling” S rows (those that do not join with any R rows) padded with NULL’s for R’s columns
Outerjoin (II)

- SELECT * FROM R LEFT OUTER JOIN S ON p;
- SELECT * FROM R RIGHT OUTER JOIN S ON p;
- A **left outer join** includes rows in *R joins S* plus dangling *R* rows padded with **NULL**’s
- A **right outer join** includes rows in *R joins S* plus dangling *S* rows padded with **NULL**’s
Outerjoin examples

**Employee**

<table>
<thead>
<tr>
<th>Eid</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>John Smith</td>
</tr>
<tr>
<td>1234</td>
<td>Mary Carter</td>
</tr>
<tr>
<td>1311</td>
<td>Bob Lee</td>
</tr>
</tbody>
</table>

**Department**

<table>
<thead>
<tr>
<th>Did</th>
<th>Mid</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1123</td>
<td>Research</td>
</tr>
<tr>
<td>5</td>
<td>1234</td>
<td>Finance</td>
</tr>
<tr>
<td>6</td>
<td>1312</td>
<td>HR</td>
</tr>
</tbody>
</table>

```
SELECT * FROM Employee LEFT OUTER JOIN Department ON Eid = Mid
```

```
<table>
<thead>
<tr>
<th>Eid</th>
<th>Name</th>
<th>Did</th>
<th>Mid</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>John Smith</td>
<td>4</td>
<td>1123</td>
<td>Research</td>
</tr>
<tr>
<td>1234</td>
<td>Mary Carter</td>
<td>5</td>
<td>1234</td>
<td>Finance</td>
</tr>
<tr>
<td>1311</td>
<td>Bob Lee</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

```
SELECT * FROM Employee RIGHT OUTER JOIN Department ON Eid = Mid
```

```
<table>
<thead>
<tr>
<th>Eid</th>
<th>Name</th>
<th>Did</th>
<th>Mid</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>John Smith</td>
<td>4</td>
<td>1123</td>
<td>Research</td>
</tr>
<tr>
<td>1234</td>
<td>Mary Carter</td>
<td>5</td>
<td>1234</td>
<td>Finance</td>
</tr>
<tr>
<td>NULL</td>
<td>Bob Lee</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

```
SELECT * FROM Employee FULL OUTER JOIN Department ON Eid = Mid
```

```
<table>
<thead>
<tr>
<th>Eid</th>
<th>Name</th>
<th>Did</th>
<th>Mid</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>6</td>
<td>1312</td>
<td>HR</td>
</tr>
</tbody>
</table>
```

3/31/2009 Luke Huan Univ. of Kansas 21
Summary

- Query
  - SELECT-FROM-WHERE statements
  - Ordering
  - Set and bag operations
  - Aggregation and grouping
  - Table expressions, subqueries
  - NULL
  - Outerjoins
Goal: Get the Name of one’s supervisor

EMPLOYEE (Eid, Mid, Name)

SQL Statement:

SELECT EID, Name FROM EMPLOYEE WHERE Mid = Eid;

SELECT E1.EID, E2.Name FROM EMPLOYEE E1, EMPLOYEE E2 WHERE E1.MId = E2.EId;

<table>
<thead>
<tr>
<th>Eid</th>
<th>Mid</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>1234</td>
<td>John Smith</td>
</tr>
<tr>
<td>1234</td>
<td>1311</td>
<td>Mary Carter</td>
</tr>
<tr>
<td>1311</td>
<td>1611</td>
<td>Bob Lee</td>
</tr>
<tr>
<td>1455</td>
<td>1611</td>
<td>Lisa Wang</td>
</tr>
<tr>
<td>1611</td>
<td>1611</td>
<td>Jack Snow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Eid</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>Mary Carter</td>
</tr>
<tr>
<td>1234</td>
<td>Bob Lee</td>
</tr>
<tr>
<td>1311</td>
<td>Jack Snow</td>
</tr>
<tr>
<td>1455</td>
<td>Jack Snow</td>
</tr>
<tr>
<td>1611</td>
<td>Jack Snow</td>
</tr>
</tbody>
</table>
Exercise

- Goal: group employees according to their department, for each department, list EIDs of its employees and list the head count

  EMPLOYEE (Eid, Name), DEPARTMENT(Eid, Did)

- SQL Statement:

  ```sql
  SELECT Did, Eid, COUNT(Eid)
  FROM EMPLOYEE e, DEPARTMENT d
  WHERE e.Eid = d.Eid
  GROUP BY Did
  ```

- There is something wrong!
  - Wishful thinking (list a group of Eid after grouping) won’t work
  - Solution: produce (1) a summary table listing Did and head count, and (2) sort Department according to Did.
Summary of SQL features covered so far

- **Query**
  - SELECT-FROM-WHERE statements
  - Set and bag operations
  - Table expressions, subqueries
  - Aggregation and grouping
  - Ordering
  - Outerjoins

- **Constraints**

- **Modification**
  - INSERT/DELETE/UPDATE

☞ Next: triggers, views, indexes