Introduction to Data Management

Lecture #17
(SQL, the Never Ending Story…)

Instructor: Mike Carey
mjcarey@ics.uci.edu

It’s time again for....

Friday Nights with Databases...

Brought to you by…
Announcements

- First SQL query HW is now underway
  - Hopefully everyone has MySQL working
  - Get the latest version of the questions! (Sorry…! 😞)
- Grading is in progress for many things
  - HW #2 is done and back (!)
  - Other HW’s are in progress
  - Midterm #1 will be back by Monday

Example Data in MySQL

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
<td>rating</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>4</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
</tr>
<tr>
<td>101</td>
<td>Joan</td>
<td>3</td>
</tr>
<tr>
<td>107</td>
<td>Johan</td>
<td>3</td>
</tr>
</tbody>
</table>
Trigger Example (MySQL)

DELIMITER $$
-- Necessary to make semicolons great again... 😊
-- (Prevents them from ending the input statement!)

CREATE TRIGGER youngsailorupdate
AFTER INSERT ON Sailors
FOR EACH ROW
BEGIN
  IF NEW.age < 18 THEN
    INSERT INTO YoungSailors (sid, sname, age, rating)
    VALUES (NEW.sid, NEW.sname, NEW.age, NEW.rating);
  END IF;
END;
END;

Note: FOR EACH ROW provides less power than FOR EACH STATEMENT (e.g., can’t compute average new age)

Insert Example (MySQL, cont’d.)

- INSERT INTO Sailors(sid, sname, rating, age)
  VALUES (777, 'Lucky', 7, 77);

- INSERT INTO Sailors(sid, sname, rating, age)
  VALUES (778, 'Lucky Jr', 7, 7);

(Note: Look at YoungSailors table content after each one!)
Another Trigger Example (*MySQL*)

-- Let’s implement a poor man’s CHECK constraint!
DELIMITER $$
CREATE TRIGGER checkSailorAge
AFTER INSERT ON Sailors
FOR EACH ROW
BEGIN
  IF NEW.age < 18 THEN
    SIGNAL SQLSTATE '02000'
    SET MESSAGE_TEXT = 'Warning: Sailors can not be under 18!';
  END IF;
END;
END;

Stored Procedures in SQL

- What is a stored procedure?
  - A program executed via a single SQL statement
  - Executes in the process space of the server
- Advantages:
  - Can encapsulate application logic while staying “close” to the data
  - Supports the reuse (sharing) of application logic by different users
  - Can be used to help secure database applications, as we will see a bit later on
Stored Procedures: More Detail

- A **stored procedure** is a function or procedure written in a general-purpose programming language that executes within the DBMS.
- These can perform computations that cannot be expressed in SQL – i.e., they go beyond the limits of relational completeness.
- Procedure execution is requested through a single SQL statement (**CALL**).
- Executes on the (usually remote!) DBMS server.
- SQL **PSM** (Persistent Stored Modules) extends SQL with concepts from general-purpose PLs.

Stored Procedures: **Functions**

**Ex:** Let’s define a simple function that we might want:

```sql
CREATE PROCEDURE ShowNumReservations(bid INT(11))
BEGIN
  SELECT S.sid, S.sname, COUNT(*)
  FROM Sailors S, Reserves R
  WHERE S.sid = R.sid AND R.bid = bid
  GROUP BY S.sid, S.sname;
END;
```

**Q:** What does this “function” do?

**Then:** CALL ShowNumReservations(102);
**Stored Procedures: Procedures**

**Ex:** Let’s define a procedure that might be useful:

- (Possible modes for parameters: IN, OUT, INOUT)

```
CREATE PROCEDURE IncreaseRating(
    IN sailor_sid INT(11),
    IN increase INT(11))
BEGIN
    UPDATE Sailors
    SET rating = rating + increase
    WHERE sid = sailor_sid;
END;
```

**Q:** How is this “procedure” different?

**Then:** CALL IncreaseRating(95,1);

**Stored Procedures: External Logic**

Stored procedures can also be written outside of the SQL language:

```
CREATE PROCEDURE RecklessSailors() LANGUAGE JAVA
EXTERNAL NAME file:///c:/storedProcs/sailorprocs.jar;
```
Main **SQL/PSM** Constructs *(FYI)*

- Supports **FUNCTIONs** and **PROCEDUREs**
- Local variables (**DECLARE**)
- **RETURN** values for **FUNCTION**
- Assign variables with **SET**
- Branches and loops:
  - IF (condition) THEN statements;
  - ELSEIF (condition) statements;
  - … ELSE statements; END IF;
  - LOOP statements; END LOOP
- Queries can be parts of expressions
- Cursors available to iterate over query results

**Note:** SQL PSM is the SQL standard’s language for S.P.’s; not supported by all vendors (due to late standardization...!)

---

A (random+) **SQL/PSM** Example

```sql
CREATE FUNCTION ResvRateSailor(IN sailorId INT(11))
RETURNS INT(11)
BEGIN
  DECLARE resvRating INT(11)
  DECLARE numResv INT(11)
  SET numResv = (SELECT COUNT(*)
                 FROM Reserves R
                 WHERE R.sid = sailorId)
  IF (numResv > 10) THEN resvRating = 1;
    ELSE resvRating = 0;
  END IF;
  RETURN resvRating;
END;
```

**Note:** See your chosen RDBMS’s docs for info about its procedural extension to SQL...
Layers of Schemas: Brief “Re-View”

- Many views of one conceptual (logical) schema and an underlying physical schema
  - Views describe how different users see the data.
  - Conceptual schema defines the logical structure of the database
  - Physical schema describes the files and indexes used under the covers

Views in SQL

- Uses of views
  - Logical data independence (to some extent)
  - Simplified view of data (for users/groups)
  - Unit of authorization (for access control)
- Views can
  - Rename/permute columns
  - Change units/representations of columns
  - Select/project/join/etc. tables

★ Virtual tables, defined via (SQL) queries
Views in SQL (cont’d.)

Provided View

CREATE VIEW RegionalSales(category, sales, state) AS
SELECT P.category, S.sales, L.state
FROM Products P, Sales S, Locations L
WHERE P.pid=S.pid AND S.locid=L.locid

User’s Query

SELECT R.category, R.state, SUM(R.sales)
FROM RegionalSales AS R GROUP BY R.category, R.state

Modified Query (System)

SELECT R.category, R.state, SUM(R.sales)
FROM (SELECT P.category, S.sales, L.state
FROM Products P, Sales S, Locations L
WHERE P.pid=S.pid AND S.locid=L.locid) AS R
GROUP BY R.category, R.state

A Simple View Example (MySQL)

CREATE VIEW YoungSailorsView (yid, yname, yage, yrating) AS
SELECT sid, sname, age, rating
FROM Sailors
WHERE age < 18;

SELECT * FROM YoungSailorsView;

SELECT yname
FROM YoungSailorsView
WHERE yrating > 5;
Another View Example (MySQL)

CREATE VIEW ActiveSailors (sid, sname, rating) AS
SELECT S.sid, S.sname, S.rating
FROM Sailors S WHERE EXISTS
  (SELECT * FROM Reserves R WHERE R.sid = S.sid);

SELECT * FROM ActiveSailors;

UPDATE ActiveSailors
SET rating = 11
WHERE sid = 22;