Announcements

- HW’s and quizzes:
  - HW #6 (!) underway, deadline next Tuesday (5 pm)
  - Last Friday’s attendees passed a foul-weather quiz (😊)
  - Midterm #2, on All Things Query, is on Friday, 3/3
- This week’s meetings
  - Monday was a holiday! (No class, no discussion(s))
- Today’s lecture plan:
  - STILL more about SQL...! (Advanced topics)
**Triggers in SQL**

- **Trigger**: a procedure that runs automatically if specified changes occur to the DBMS.
- **Three parts:**
  - Event (activates the trigger)
  - Condition (tests if the trigger should run)
  - Action (what happens if the trigger runs)
- **Can be used to do “whatever”!**
  - One SQL statement or sequence/flow of statements; can also cause the current update to bail out.
  - Details vary WIDELY from vendor to vendor (!)
  - Major source of “vendor lock-in”, along with the stored procedure language (= trigger action language)

---

**Trigger Example (SQL:1999)**

```
CREATE TRIGGER youngSailorUpdate
AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
INSERT INTO YoungSailors(sid, sname, age, rating)
SELECT sid, sname, age, rating
FROM NewSailors N
WHERE N.age < 18
```

*Note: NewSailors provides access to the changes here!*
Trigger Syntax (**MySQL**)

```
CREATE [DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
  trigger_time trigger_event
ON tbl_name
FOR EACH ROW
  [trigger_order]
  trigger_body

trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
trigger_order: { FOLLOWS | PRECEDES } other_trigger_name

(http://dev.mysql.com/doc/refman/5.7/en/create-trigger.html)
```

Trigger Example (**MySQL**)

```
DELIMITER $$

(Don’t ask... ☹️; can use MySQL Workbench dialog....!)

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;

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

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
**Trigger 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);

---

**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 the 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.
- They 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 (often remote) DBMS server.
- SQL PSM (Persistent Stored Modules) extends SQL with concepts from general-purpose PLs.

Stored Procedures: Functions

**Ex:** Let’s define a function that we might wish to have:

```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)

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

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

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

**Stored Procedures: External Logic**

Stored procedures can be written outside SQL:

```sql
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

CREATE FUNCTION ResRateSailor(IN sailorId INT(11))
RETURNS INT(11)
BEGIN
    DECLARE resRating INT(11)
    DECLARE numRes INT(11)
    SET numRes = (SELECT COUNT(*)
                   FROM Reserves R
                   WHERE R.sid = sailorId)
    IF (numRes > 10) THEN resRating = 1;
    ELSE resRating = 0;
    END IF;
    RETURN resRating;
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 by (SQL) queries
Views in SQL (cont’d.)

**Provided View**

```sql
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**

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

**Modified Query (System)**

```sql
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)

```sql
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;