Announcements

- HW #6 is due on **Thursday**
  - No late day available for this one
  - Solution will be published right away
- Midterm #2 is **this Friday** in class
  - Relational languages (see syllabus)
  - Come a little early
  - Bring an optional cheat sheet
  - Sample from last year available online
- Let’s finish relational languages!
Trigger Example (*MySQL*)

DELIMITER $$
-- Needed to make semicolons great again... 😊

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)

---

**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);
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 (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:

```
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;
```

Then: CALL ShowNumReservations(102);

**Q:** What does this “function” do?

**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;
```

Then: CALL IncreaseRating(95,1);

**Q:** How is this “procedure” different?
**Stored Procedures: External Logic**

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

```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

View 1       View 2       View 3
             ↓
Conceptual Schema
             ↓
Physical Schema
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.)

<table>
<thead>
<tr>
<th>Provided View</th>
<th>User’s Query</th>
<th>Modified Query (System)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CREATE VIEW RegionalSales(category, sales, state) AS</code></td>
<td><code>SELECT R.category, R.state, SUM(R.sales)</code></td>
<td><code>SELECT R.category, R.state, SUM(R.sales)</code></td>
</tr>
<tr>
<td><code>WHERE P.pid=S.pid AND S.locid=L.locid</code></td>
<td></td>
<td><code>GROUP BY R.category, R.state</code></td>
</tr>
</tbody>
</table>
**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;
So What About Views & Updates?

Ex:

```sql
CREATE VIEW SailsBoats AS
SELECT DISTINCT S.*, B.*
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid and R.bid = B.bid;
```

Q: What if we now try...

```sql
UPDATE SailsBoats
SET rating = 12
WHERE sid = 22 AND bid = 101;
```

This view is not updatable since there is no update to the real (stored) tables that would have (just) the asked-for effect – see next slide...

... Views & Updates (Cont’d.)?

```sql
UPDATE SailsBoats
SET rating = 12
WHERE sid = 22 AND bid = 101;
```
... Views & Updates? (Cont’d.)

- A legal update $U$ to view $V$ must be translatable into an equivalent update $U'$ on the underlying table(s) $T$, i.e.:

  ![Diagram showing the process of updating views]

  - If this isn’t possible, the system will reject the update
  - Systems differ in how well they do this and err on the conservative side (i.e., declining more view updates)

SQL Access Control

- Based on the concept of access rights or privileges for objects (tables, views, stored procedures, ...) and mechanisms for giving users privileges (and revoking privileges).
- Creator of a database object automatically gets all privileges on it.
  - DBMS keeps track of who subsequently gains and loses privileges, and it ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed to execute.
GRANT Command

```sql
GRANT privileges ON object TO users [WITH GRANT OPTION]
```

- The following **privileges** can be specified:
  - **SELECT**: Can read all columns (including those added later via ALTER TABLE command).
  - **INSERT**(col-name): Can insert tuples with non-null or non-default values in this column.
    - INSERT means same right with respect to all columns.
  - **DELETE**: Can delete tuples.
  - **REFERENCES**(col-name): Can define foreign keys (in other tables) that refer to this column.
  - If a user has a privilege with the **GRANT OPTION**, can pass privilege on to other users (with or without passing on the **GRANT OPTION**).
  - Only the owner can execute CREATE, ALTER, or DROP.

---

**GRANT and REVOKE of Privileges**

- **GRANT INSERT, SELECT ON Sailors TO Horatio**
  - Horatio can query Sailors or insert tuples into it.
- **GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION**
  - Yuppy can delete tuples and can authorize others to do so.
- **GRANT UPDATE** *(rating)* **ON Sailors TO Dustin**
  - Dustin can update (only) the *rating* field of Sailors tuples.
- **GRANT SELECT ON ActiveSailors TO Guppy, Yuppy**
  - This does NOT allow the ‘uppies to query Sailors directly!
- **REVOKE**: When a privilege is revoked from X, it is also revoked from all users who got it solely from X.
**GRANT/REVOKE on Views**

- Great combination to enforce restrictions on data visibility for various users/groups.
- If view creator loses the SELECT privilege on an underlying table, the view is dropped!
- If view creator loses a privilege held with the grant option on an underlying table, (s)he loses it on the view as well – and so do users who were granted the privilege on the view!

**Views & Security**

- Views can be used to present just the necessary information (or a summary) while hiding some details of the underlying relation(s):
  - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the bid’s of boats that have been reserved.
- Creator of a view has a privilege on the view if (s)he has the privilege on all underlying tables.
- Used together with GRANT/REVOKE commands, views are a very powerful access control tool.
- Stored procedures can be utilized similarly!
SQL Summary (I)

- SQL was a big factor in the early acceptance of the relational model; users found it more natural than earlier, procedural query languages.
- SQL is relationally complete; it has significantly more expressive power than the relational algebra.
- Queries that can be expressed in rel. alg. can often be expressed more naturally in SQL. (Ex: \textit{\texttt{max}})
- Many alternative ways to write a query; optimizer will look for the most efficient evaluation plan.
  - In practice, expert users are aware of how queries are optimized and evaluated. (Optimizers are imperfect.)

SQL Summary (II)

- NULL for unknown field values brings many complications (as well as a SQL specification divergence for Oracle \textit{w.r.t.} VARCHAR data).
- Allows specification of rich integrity constraints (real RDBMSs implement just some of SQL IC spec).
- Triggers can respond to changes in the database (and make up the difference when the set of available integrity features falls short).
- Stored procedures (and CALL) are also available.
- Views and authorization are both useful features, and can be especially powerful in combination. (!)
That’s it for SQL!

ANY LINGERING QUESTIONS...?