CS122A: Introduction to Data Management

Lecture #11: SQL (4) -- Triggers, Views, Access Control

Instructor: Chen Li

Un-intuitive side of SQL on NULL
Nulls w/Aggregates & Grouping

sid	bid	date
22	101	1998-10-10
22	102	1998-10-10
22	103	1998-10-08
22	104	1998-10-07
31	102	1998-11-10
31	103	1998-11-06
31	104	1998-11-12
64	101	1998-09-05
64	102	1998-09-08
74	103	1998-09-08
NULL	103	1998-09-09
1	NULL	2001-01-11
1	NULL	2002-02-02

SELECT bid, COUNT(*)
FROM Reserves
GROUP BY bid

bid	COUNT(*)
NULL	2
101	2
102	3
103	4
104	2

"SQL treats two rows as duplicates if the corresponding columns are either equal or both contain null." **Very inconsistent!**

Un-intuitive side of SQL on NULL
Nulls w/Aggregates & Grouping

sid	bid	date
22	101	1998-10-10
22	102	1998-10-10
22	103	1998-10-08
22	104	1998-10-07
31	102	1998-11-10
31	103	1998-11-06
31	104	1998-11-12
64	101	1998-09-05
64	102	1998-09-08
74	103	1998-09-08
NULL	103	1998-09-09
1	NULL	2001-01-11
1	NULL	2002-02-02

SELECT COUNT(bid)
FROM Reserves (11)

SELECT COUNT(*)
FROM Reserves

(13)

Inner Join and Outer Join Syntax

```
SELECT column_name(s)
FROM table1 INNER JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT column_name(s)
FROM table1 OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

ORDER BY clause

Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors, ordered by rating in an descending order.

```
SELECT  S.rating,  MIN (S.age)
                          AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) >= 2
ORDER BY S.rating DESC.
```

Answer relation:

rating	minage
8	25.5
7	35.0
3	25.5

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

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!

Trigger Example (MySQL)

```
DELIMITER $$
```
←*(Don't ask.... ☺)*

```
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)

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);

Trigger Syntax (MySQL)

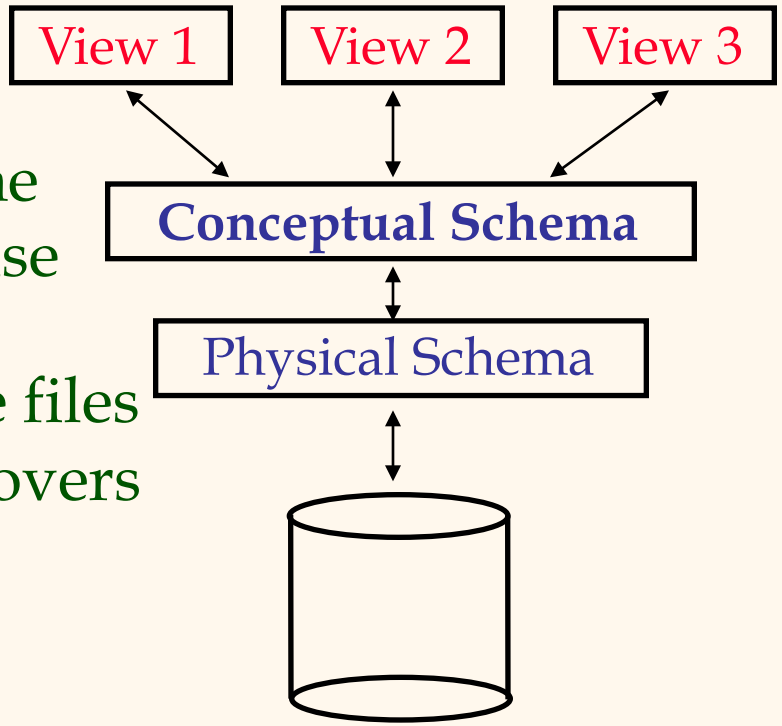
CREATE [DEFINER = { *user* | CURRENT_USER }]
TRIGGER *trigger_name*
trigger_time trigger_event
ON *tbl_name*
FOR EACH ROW
[trigger_order]
trigger_body

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)

3 Schema Levels

- 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*

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;
```

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
```

Rewritten Query

```
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
```

So What About Views & Updates?

```
CREATE VIEW SBR 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 say...

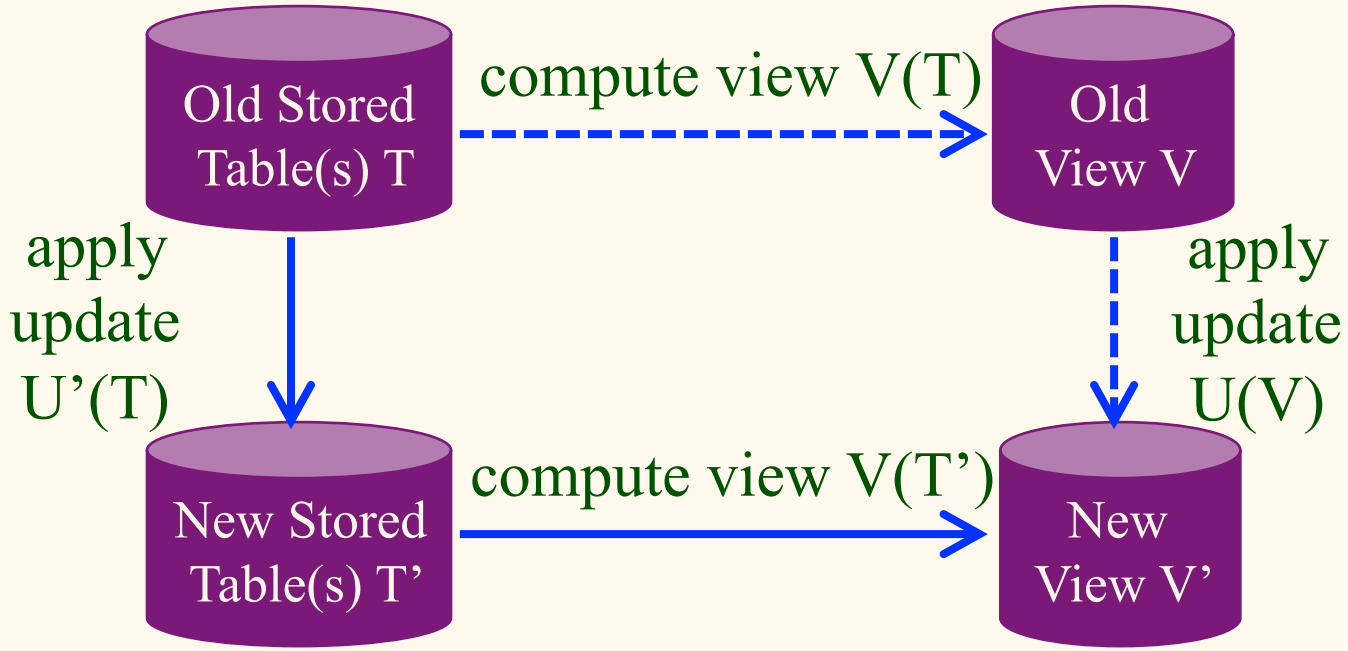
```
UPDATE SBR
SET rating = 11
WHERE sid = 22 AND bid = 33;
```

(?)

This view is **not** updatable since there is no update to the real, stored tables that would have just the asked-for effect.
Hint: If we change rating of sailor (sid = 22) to 11 in the Sailors table, what will happen to the rating of an SBR record with sid = 30 and bid = 555?

... *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.*:



- If this isn't possible, a 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 and views), and mechanisms for giving users privileges (and revoking privileges).
- Creator of a table or a view automatically gets all privileges on it.
  - DBMS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.

GRANT Command

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 owner can execute CREATE, ALTER, and DROP.

GRANT 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 of Privileges

- REVOKE INSERT, SELECT ON Sailors FROMHoratio

- When a privilege is revoked from X, it is also revoked from all users who got it *solely* from X.

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 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!

SQL Summary

- 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 *R.A.* can often be expressed more naturally in SQL. (Ex: 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 (Cont'd.)

- NULL for unknown field values brings many complications (as well as a SQL specification divergence for Oracle *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 (!)