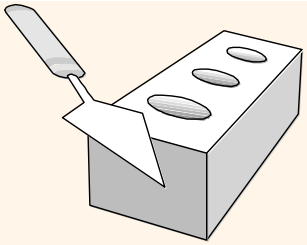
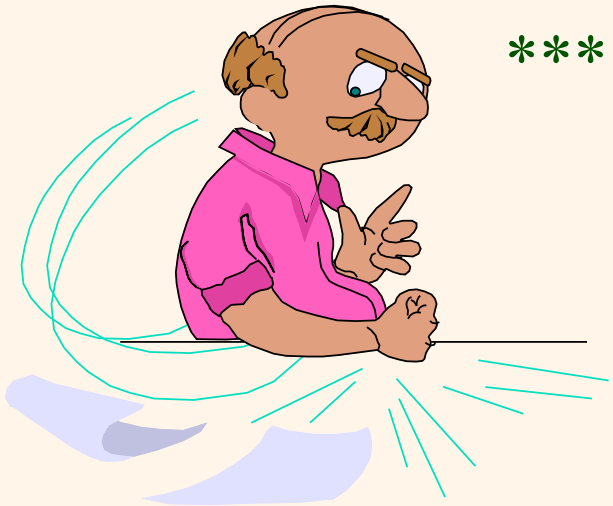


Introduction to Data Management

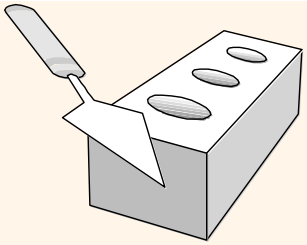
**** The "Flipped" Edition ****



Lecture #17
(Advanced SQL II)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements



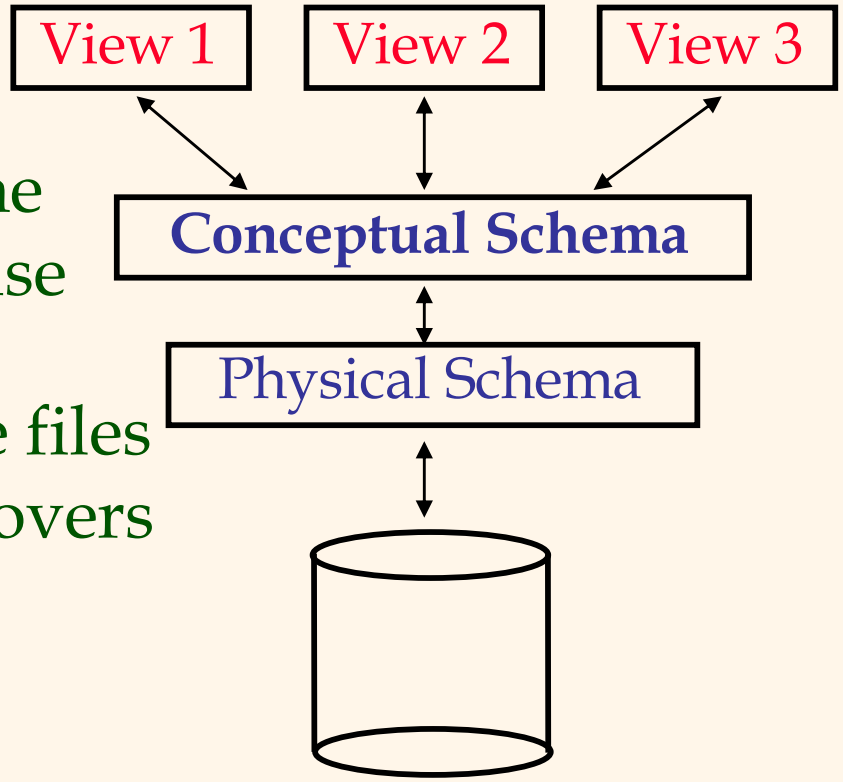
- Again, note that you're over half-way through...!
  - You can ***do*** this....! ☺
- Roadmap reminder:

Relational Algebra	Ch. 2.5-2.7
Relational Calculus	Wikipedia: Tuple relational calculus
SQL Basics (SPJ and Nested Queries)	Ch. 3.3-3.5
SQL Analytics: Aggregation, Nulls, and Outer Joins	Ch. 3.6-3.9, 4.1
Advanced SQL: Constraints, Triggers, Views, and Security	Ch. 4.2, 4.4-4.5, 4.7
Midterm Exam 2	***Mon, Nov 15*** (during lecture time)

- HW#5 is due Friday (we're still in "*Friday 6PM mode*")
  - First of our series of ***SQL-based*** HW adventures

*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



A Simple View Example (PostgreSQL)

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

```
SELECT * FROM YoungSailorsView;
```

```
SELECT yname, yrating, yage
FROM YoungSailorsView
WHERE yrating >= 9;
```

Another View Example (PostgreSQL)

```
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:

```
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...

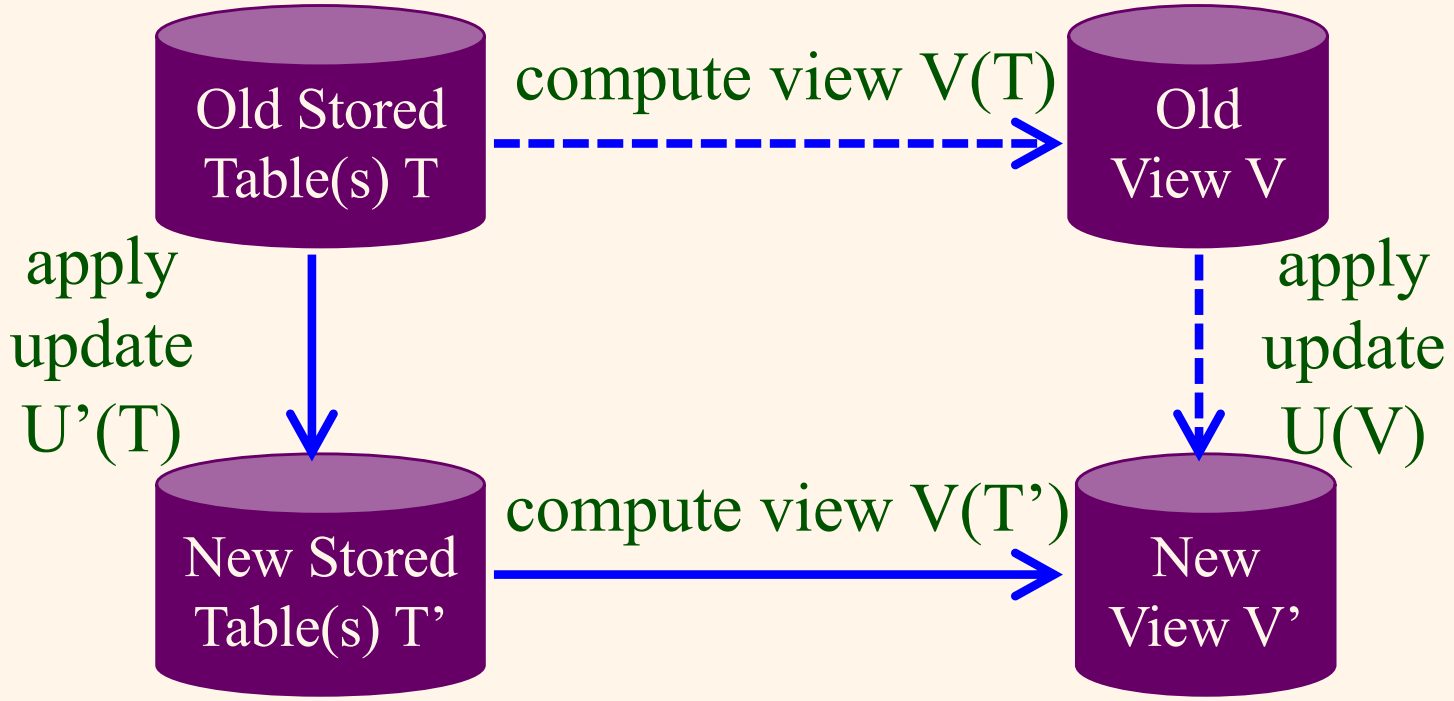
```
UPDATE SailsBoats
SET rating = 12
WHERE sid = 22 AND bid = 101;
```

(?)

*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.)*

- 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, 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)

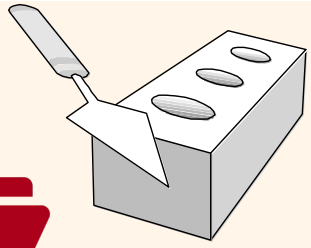
... *Views & Updates (Cont'd.)?*

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

Result Grid Filter Rows: Search

sid	sname	rating	age	bid	bname	color
22	Dustin	~~7~~ 12?	45.0	101	Interlake	blue
64	Horatio	7	35.0	101	Interlake	blue
22	Dustin	7	45.0	102	Interlake	red
31	Lubber	8	55.5	102	Interlake	red
64	Horatio	7	35.0	102	Interlake	red
22	Dustin	7	45.0	103	Clipper	green
31	Lubber	8	55.5	103	Clipper	green
74	Horatio	9	35.0	103	Clipper	green
22	Dustin	7	45.0	104	Marine	red
31	Lubber	8	55.5	104	Marine	red

An Aside: What's a ***Schema****?*



- A ***schema***, as a feature in SQL's DDL, is a namespace used to organize all the objects for a given SQL database-based application
  - It's a home for tables, views, stored procedures, types, ...
  - Kind of like an OS file system folder (but SQL schemas are not hierarchical and its entities are more strongly typed)
- Everything a user does is in the context of *some* schema
  - There's a default schema called 'public' in PostgreSQL
  - You can create a new one: **CREATE SCHEMA** Lecture12;
  - You can also say: **SET** search_path **TO** Lecture12, public;
  - And you can ask where you are: **SELECT** current_schema();
  - Objects can be in the current schema (implicit) or fully qualified:
    - SELECT * FROM Sailors; -- Sailors table in current schema
    - SELECT * FROM Lecture12.Sailors; -- Sailors table in Lecture12 schema

SQL Access Control

- Based on the concept of access rights or privileges for objects (schemas, tables, views, stored procedures, ...) and mechanisms for giving users (or *roles*) privileges (as well as revoking privileges).
- The creator of a database object automatically gets all privileges on it.
  - DBMS keeps track of who gains and loses privileges, and it ensures that only requests from users who have the necessary privileges (when the request is issued) are allowed to execute.
  - Two useful PostgreSQL commands you can play around with:
    - **SET** SESSION AUTHORIZATION 'horatio'; -- a *user* or *role*
    - **SELECT** SESSION_USER, CURRENT_USER; -- a sanity check
  - Check out the blog on PostgreSQL access control on AWS: https://aws.amazon.com/blogs/database/managing-postgresql-users-and-roles/

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 alone means the 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, they can pass the privilege on to other users (with or without passing on the GRANT OPTION).
- Only the owner can execute CREATE, ALTER, or DROP.

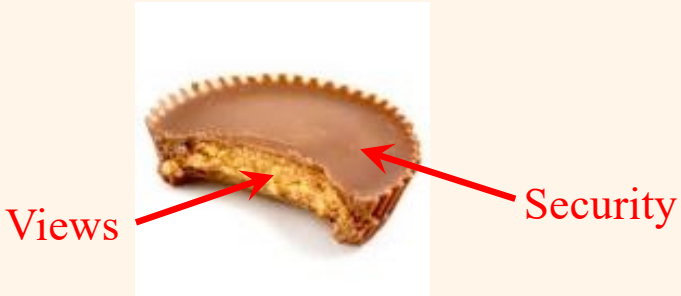
GRANT and REVOKE of Privileges

- 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

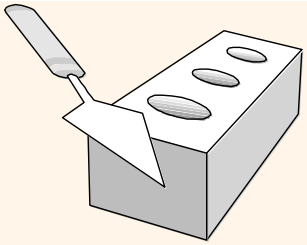
- Great combination to enforce restrictions on *data visibility* for various users/groups
- If a view *creator* loses the SELECT privilege on an underlying table, the view is dropped!
- If view creator loses a privilege held with the grant option on an underlying table, (s)he loses it on the view as 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)



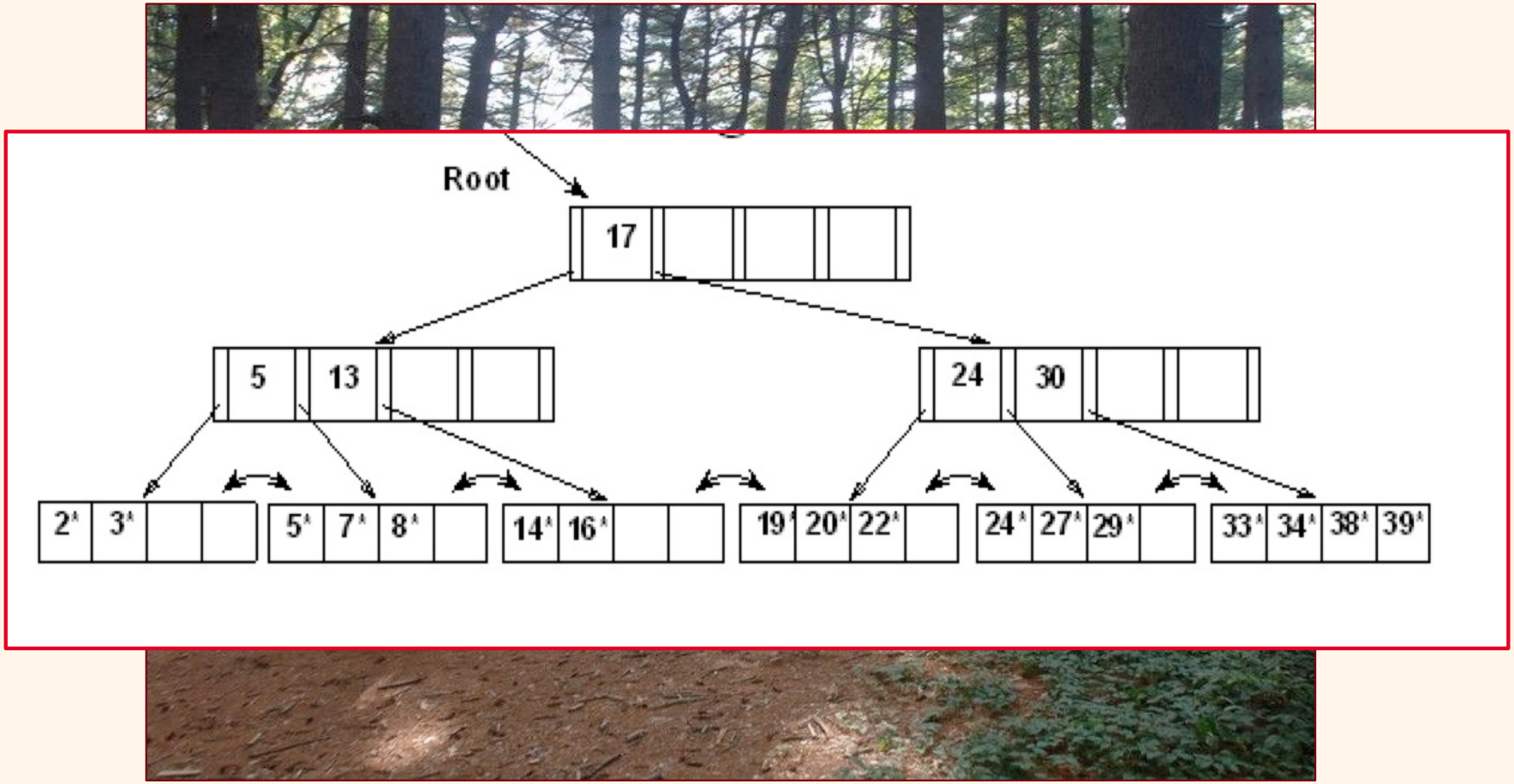
- SQL was a big factor in the early ***acceptance*** of the relational model; users found it more natural than earlier, procedural query languages. *(Thanks, Don!)*
- SQL is ***relationally complete,*** and has significantly ***more*** expressive power than the relational algebra.
- Queries that can be expressed in rel. alg. can often be expressed ***more naturally*** in SQL. (Ex: 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 *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...?



Roadmap Re-check…

Topic Coverage and Exam Schedule

Syllabus

Topic	**Reading (Required!)**
Databases and DB Systems	Ch. 1
Entity-Relationship (E-R) Data Model	Ch. 6.1-6.5, 6.8-6.9
Relational Data Model	Ch. 2.1-2.4, 3.1-3.2
E-R to Relational Translation	Ch. 6.6-6.7
Relational Design Theory	Ch. 7.1-7.4.2
Midterm Exam 1	***Fri, Oct 22*** (during lecture time)
Relational Algebra	Ch. 2.5-2.7
Relational Calculus	Wikipedia: Tuple relational calculus
SQL Basics (SPJ and Nested Queries)	Ch. 3.3-3.5
SQL Analytics: Aggregation, Nulls, and Outer Joins	Ch. 3.6-3.9, 4.1
Advanced SQL: Constraints, Triggers, Views, and Security	Ch. 4.2, 4.4-4.5, 4.7
Midterm Exam 2	***Mon, Nov 15*** (during lecture time)
Storage	Ch. 12.1-12.4, 12.6-12.7
Indexing	Ch. 14.1-14.4, 14.5
Physical DB Design	Ch. 14.6-14.7, 15.1-15.3, 15.5.3
Semistructured Data Management (*a.k.a.* NoSQL)	Ch. 8.1, AsterixDB SQL++ Primer, Couchbase SQL++ Book
Data Science 1: Advanced SQL Analytics	Ch. 5.5, 11.3
Data Science 2: Notebooks, Dataframes, and Python/Pandas	Lecture notes and Jupyter notebook
Basics of Transactions	Ch. 4.3, Ch. 17
Endterm Exam	***Fri, Dec 3*** (during lecture time)