Views, Privileges, and Catalogs

PDBM 7.4, 7.6–7.7

Dr. Chris Mayfield

Department of Computer Science
James Madison University

Mar 01, 2022

Section 7.4

SQL Views: logical data independence

Virtual views

Three types of *relations* in SQL

1. `CREATE TABLE` — physical
2. `CREATE INDEX` — btree/hash
3. `CREATE VIEW` — virtual

```
-- "store a query" as a VIEW
CREATE VIEW ParamountMovies AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount';

-- use it in the FROM clause
SELECT DISTINCT starName
FROM ParamountMovies, StarsIn
WHERE title = movieTitle
  AND year = movieYear;
```

VIEW syntax tips

You can rename all view attributes

```
CREATE VIEW ParamountMovies(movieTitle, movieYear) AS
  SELECT title, year
  FROM Movies
  WHERE studioName = 'Paramount';
```

You cannot add/remove columns with `ALTER VIEW`

```
  -- removing views
  DROP VIEW ParamountMovies;
```

Inserting into views

If views are "simple" you can `UPDATE` them

- Queries cannot be `SELECT DISTINCT` from R
- `FROM` clause may only involve R (one time)
- `WHERE` clause must not involve R in subquery
- Many other complex rules (but common sense)

For example

```
INSERT INTO ParamountMovies
VALUES ('Star Trek', 1979);
```

- Underlying schema

  `Movies(title, year, length, genre, studioName, producerC#)`

- Will this new tuple be present in the view?

Updating/deleting from views

```
UPDATE ParamountMovies
SET year = 1979
WHERE title LIKE '%Trek%';

DELETE FROM ParamountMovies
WHERE title LIKE '%Trek%';
```

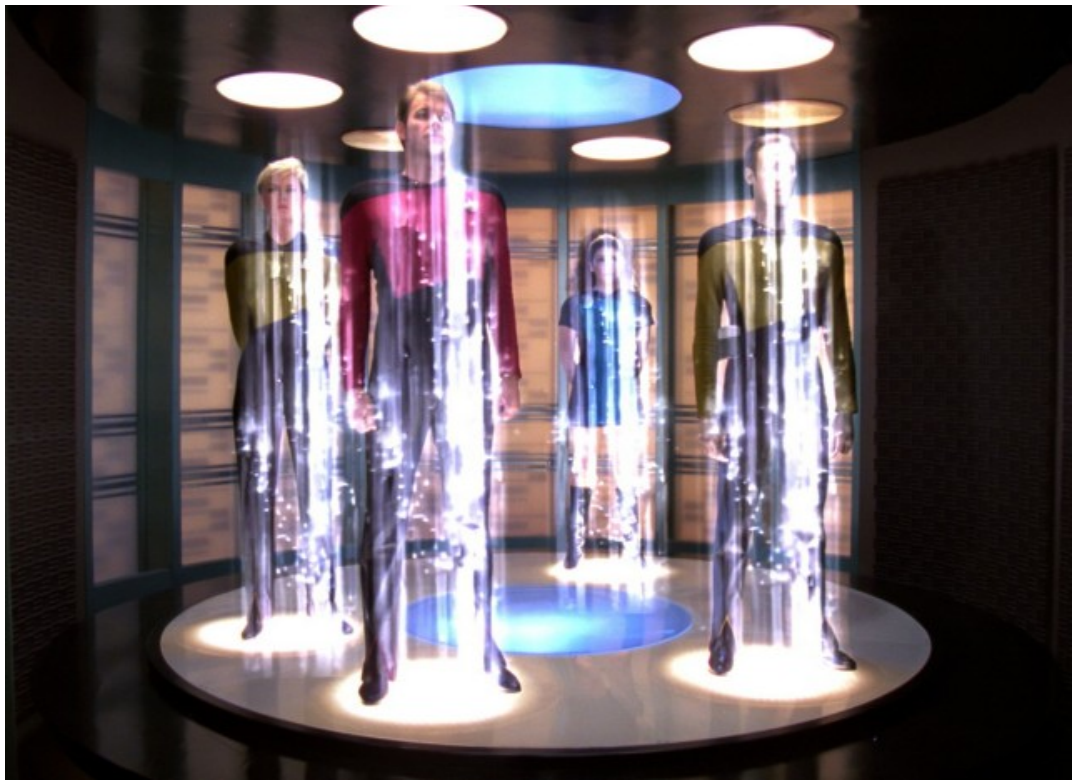
SQL automatically restricts updates to rows in the view

```
UPDATE Movies
SET year = 1979
WHERE title LIKE '%Trek%'
  AND studioName = 'Paramount';

DELETE FROM Movies
WHERE title LIKE '%Trek%'
  AND studioName = 'Paramount';
```

View Performance

What does the word materialize mean?



Materialized views

```
CREATE MATERIALIZED VIEW fallmem_all AS
    SELECT * FROM fall_membership
    -- do not consider sub-groups
    WHERE race = 'ALL'
      AND gender = 'ALL'
      AND disabil = 'ALL'
      AND lep = 'ALL'
      AND disadva = 'ALL';
```

How do we keep the view up to date?

- incremental updates (i.e., eager)
- periodic updates (i.e., lazy)
- manual updates (i.e., snapshot)
  - `REFRESH MATERIALIZED VIEW fallmem_all;`
  - https://www.postgresql.org/docs/11/rules-materializedviews.html

View query rewriting

View	Query
SELECT L_V	SELECT L_Q
FROM R_V	FROM R_Q
WHERE C_V	WHERE C_Q

We can rewrite part of Q using V when:

1. The relations in list R_V all appear in the list R_Q
2. The condition C_Q is equivalent to C_V `AND` C (for some C)
3. If C is needed, then attributes of R_V in C are also in L_V
4. Attributes in L_Q that come from R_V are also in L_V

How to rewrite Q to use V:

- Replace R_Q by V and other relations not in R_Q
- Replace C_Q by C (or remove `WHERE` if C not needed)

Example

```
CREATE MATERIALIZED VIEW MovieProd AS
  SELECT title, year, name    -- LV
  FROM Movies, MovieExec      -- RV
  WHERE producerC# = cert#;   -- CV
```

Original query:

```
  SELECT starName                      -- LQ
  FROM StarsIn, Movies, MovieExec      -- RQ
  WHERE movieTitle = title             -- CQ
    AND movieYear = year
    AND producerC# = cert#
    AND name = 'Max Bialystock';
```

Rewritten query:

```
  SELECT starName
  FROM StarsIn, MovieProd
  WHERE movieTitle = title  -- C atts in LV
    AND movieYear = year
    AND name = 'Max Bialystock';
```

Section 7.6

SQL Privileges: `GRANT` and `REVOKE`

Privileges

POSIX file system:

- {User, Group, Other} may {4=Read, 2=Write, 1=Execute}
- Example: `chmod 755 myfile.txt`

SQL database:

- SELECT, INSERT, UPDATE, DELETE
- TRUNCATE, REFERENCES, TRIGGER
- CREATE, CONNECT, TEMPORARY
- EXECUTE, USAGE, ALL PRIVILEGES

https://www.postgresql.org/docs/11/sql-grant.html

Granting privileges

GRANT <privilege list> ON <database element> TO <user list>

```
GRANT SELECT, INSERT ON Studio
TO kirk, picard WITH GRANT OPTION;
```

PostgreSQL syntax is slightly different from the book

```
GRANT SELECT (title), UPDATE (title)
ON movies TO sisko;
```

Easy way to give everyone read access

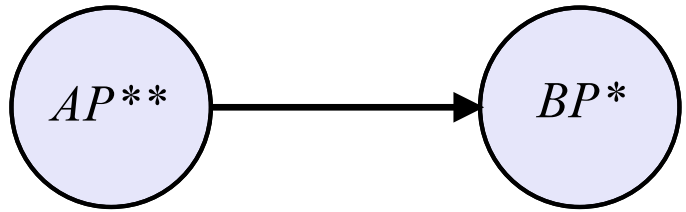
```
GRANT SELECT ON ALL TABLES
IN SCHEMA public TO public;
```

https://www.postgresql.org/docs/11/ddl-priv.html

Grant diagrams

Directed graph

- Nodes = user and privilege
  - ** = owner of element
  - * = with grant option
- Edges = who granted privilege

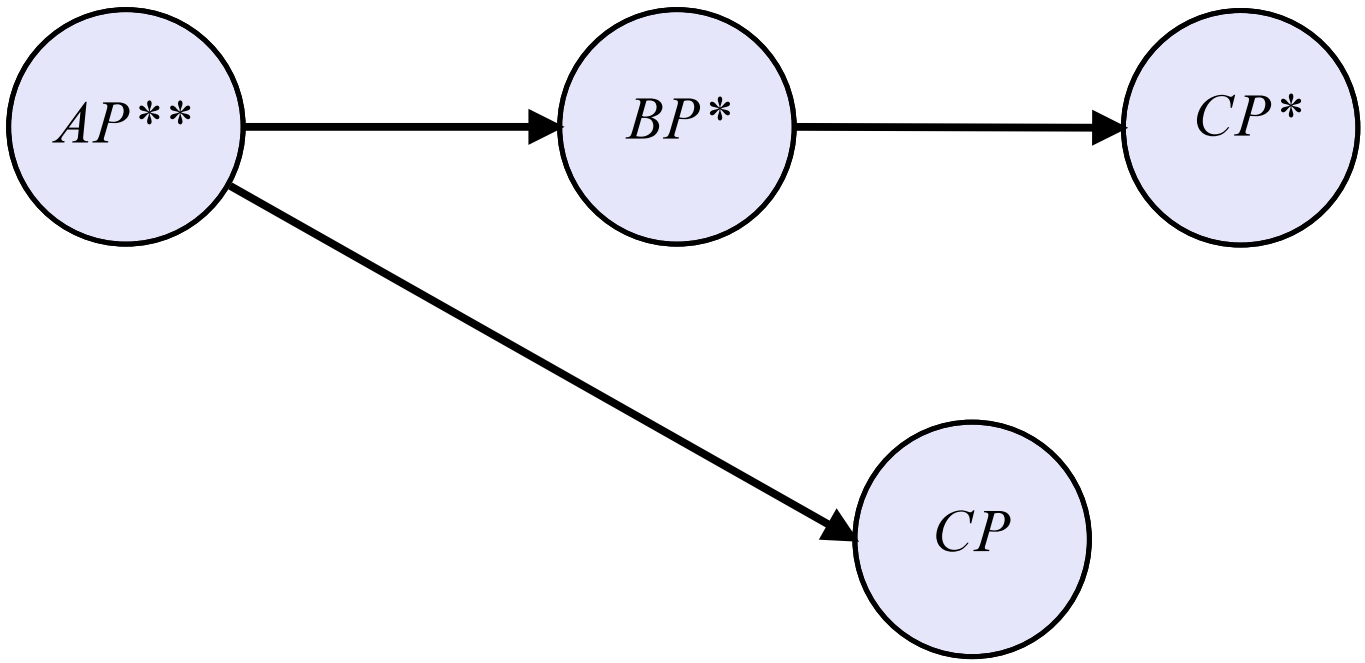


Fundamental rule

- User C has privilege P if and only if:
  - Path from $XQ**$ to $CP**$, $CP*$, or CP
  - X is the owner and $Q \supseteq P$ (superprivilege)
- Remember that P could be Q, and X could be C
- Superusers and object owners have all privileges

Example grant diagram

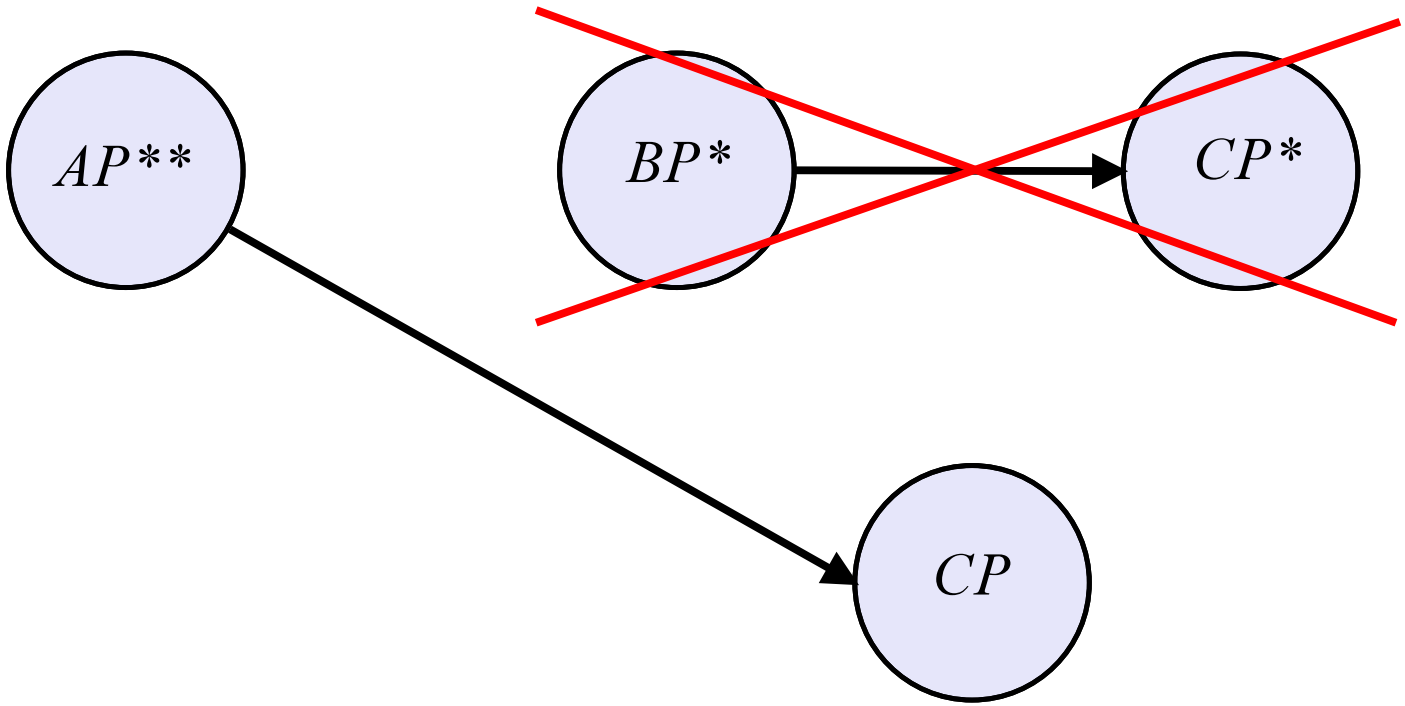
- A owns the object for which P is a privilege
  - User A: `GRANT P TO B WITH GRANT OPTION;`
  - User B: `GRANT P TO C WITH GRANT OPTION;`
  - User A: `GRANT P TO C;`



Example revoke cascade

User A: `REVOKE P FROM B CASCADE;`

- Both B and C lose $P*$
- However, C still has P



Revoking privileges

REVOKE <privilege list> ON <database element> FROM <user list>
[CASCADE | RESTRICT]

Note: `RESTRICT` by default

- Cannot revoke if has any *dependent* privileges

```
REVOKE SELECT, INSERT ON Studio FROM picard CASCADE;

-- PostgreSQL has additional options
REVOKE ALL PRIVILEGES ON Studio FROM picard;
```

Creating initial privileges

How I created your personal schema:

```
CREATE USER mayfiecs PASSWORD '123456789';
CREATE SCHEMA AUTHORIZATION mayfiecs;
REVOKE ALL ON SCHEMA mayfiecs FROM public;
```

How I created your group database:

```
CREATE ROLE teamname USER user1, user2, ...;
CREATE DATABASE teamname OWNER teamname;
REVOKE ALL ON DATABASE teamname FROM public;
```

And made "postgres" read-only:

```
REVOKE CREATE ON DATABASE postgres FROM public;
REVOKE TEMP ON DATABASE postgres FROM public;
REVOKE CREATE ON SCHEMA public FROM public;
```

Privilege-checking process

1. Is the user the owner?
2. Is the object public?
3. Does the user have access?

Group roles:

```
CREATE ROLE absent;
-- NOSUPERUSER NOCREATEDB NOCREATEROLE NOREPLICATION

-- each user has a set of authorization IDs
GRANT absent TO mayfiecs;
```

Super users:

```
CREATE ROLE postgres LOGIN SUPERUSER INHERIT
            CREATEDB CREATEROLE REPLICATION;
```

Section 7.7

What is the difference between data and metadata?

System catalogs

All metadata is stored in tables

- No need to reinvent the wheel
- It's also useful to query it!

Two versions of the metadata:

- `pg_catalog`
  https://www.postgresql.org/docs/11/catalogs.html
- `information_schema`
  https://www.postgresql.org/docs/11/information-schema.html

Most useful data in system views:
https://www.postgresql.org/docs/11/views-overview.html

Example #1: table/attribute names

System catalog views:

```
SELECT * FROM pg_tables;
SELECT * FROM pg_views;
```

System catalog tables:

```
SELECT * FROM pg_namespace; -- schemas
SELECT * FROM pg_class; -- tables
SELECT * FROM pg_attribute; -- columns
```

Putting them together:

```
SELECT t.tablename, a.attnum, a.attname
FROM pg_tables AS t
  JOIN pg_class AS c ON t.tablename = c.relname
  JOIN pg_attribute AS a ON c.oid = a.attrelid
WHERE schemaname = 'public'
ORDER BY t.tablename, a.attnum;
```

Example #2: optimizer statistics

What happens when you `ANALYZE VERBOSE`?

```
SELECT * FROM pg_stats
WHERE schemaname = 'public'
ORDER BY tablename;
```

See https://www.postgresql.org/docs/11/view-pg-stats.html

Idea for pgAdmin feature: GUI for `pg_stats`

Example #3: what can be NOT NULL?

```sql
-- generate queries that count null values
SELECT
  'SELECT count(*) FROM ' || t.tablename
  || ' WHERE ' || A.attname || ' IS NULL;' AS sql
FROM pg_tables AS t
  JOIN pg_class AS c ON t.tablename = c.relname
  JOIN pg_attribute AS a ON c.oid = a.attrelid
WHERE schemaname = 'public'
  AND attnum > 0
ORDER BY tablename, attnum;
```

- You can write queries to write other queries!
- Project tip: write sql to generate JavaScript