Database views

DATABASE DESIGN

Lis Sulmont
Curriculum Manager

Database views

> In a database, a **view** is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object *(Wikipedia)*

Virtual table that is not part of the physical schema

- Query, not data, is stored in memory
- Data is aggregated from data in tables
- Can be queried like a regular database table
- No need to retype common queries or alter schemas

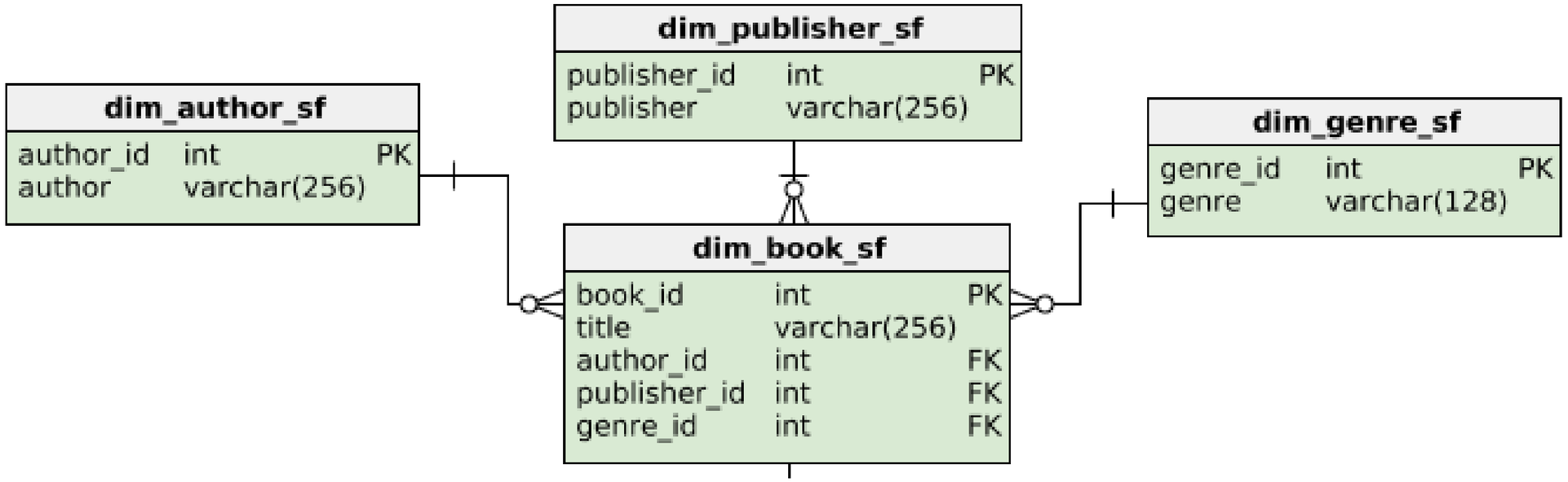
[1] https://en.wikipedia.org/wiki/View_(SQL)

Creating a view (syntax)

```
CREATE VIEW view_name AS
```

```
SELECT col1, col2
FROM table_name
WHERE condition;
```

Creating a view (example)



Goal: Return titles and authors of the `science fiction` genre

Creating a view (example)

```
CREATE VIEW scifi_books AS
```

```
SELECT title,  author, genre
FROM dim_book_sf
JOIN dim_genre_sf ON dim_genre_sf.genre_id = dim_book_sf.genre_id
JOIN dim_author_sf ON dim_author_sf.author_id = dim_book_sf.author_id
WHERE dim_genre_sf.genre = 'science fiction';
```

Querying a view (example)

```
SELECT * FROM scifi_books
```

```
| title                          | author            | genre           |
|--------------------------------|-------------------|-----------------|
| The Naked Sun                  | Isaac Asimov      | science fiction |
| The Robots of Dawn             | Isaac Asimov      | science fiction |
| The Time Machine               | H.G. Wells        | science fiction |
| The Invisible Man              | H.G. Wells        | science fiction |
| The War of the Worlds          | H.G. Wells        | science fiction |
| Wild Seed (Patternmaster, #1)  | Octavia E. Butler | science fiction |
| ...                            | ...               | ...             |
```

Behind the scenes

```
SELECT * FROM scifi_books
```

=

```
SELECT * FROM
(SELECT title,  author, genre
FROM dim_book_sf
JOIN dim_genre_sf ON dim_genre_sf.genre_id = dim_book_sf.genre_id
JOIN dim_author_sf ON dim_author_sf.author_id = dim_book_sf.author_id
WHERE dim_genre_sf.genre = 'science fiction');
```

Viewing views

(in PostgreSQL)

```
SELECT * FROM INFORMATION_SCHEMA.views;
```

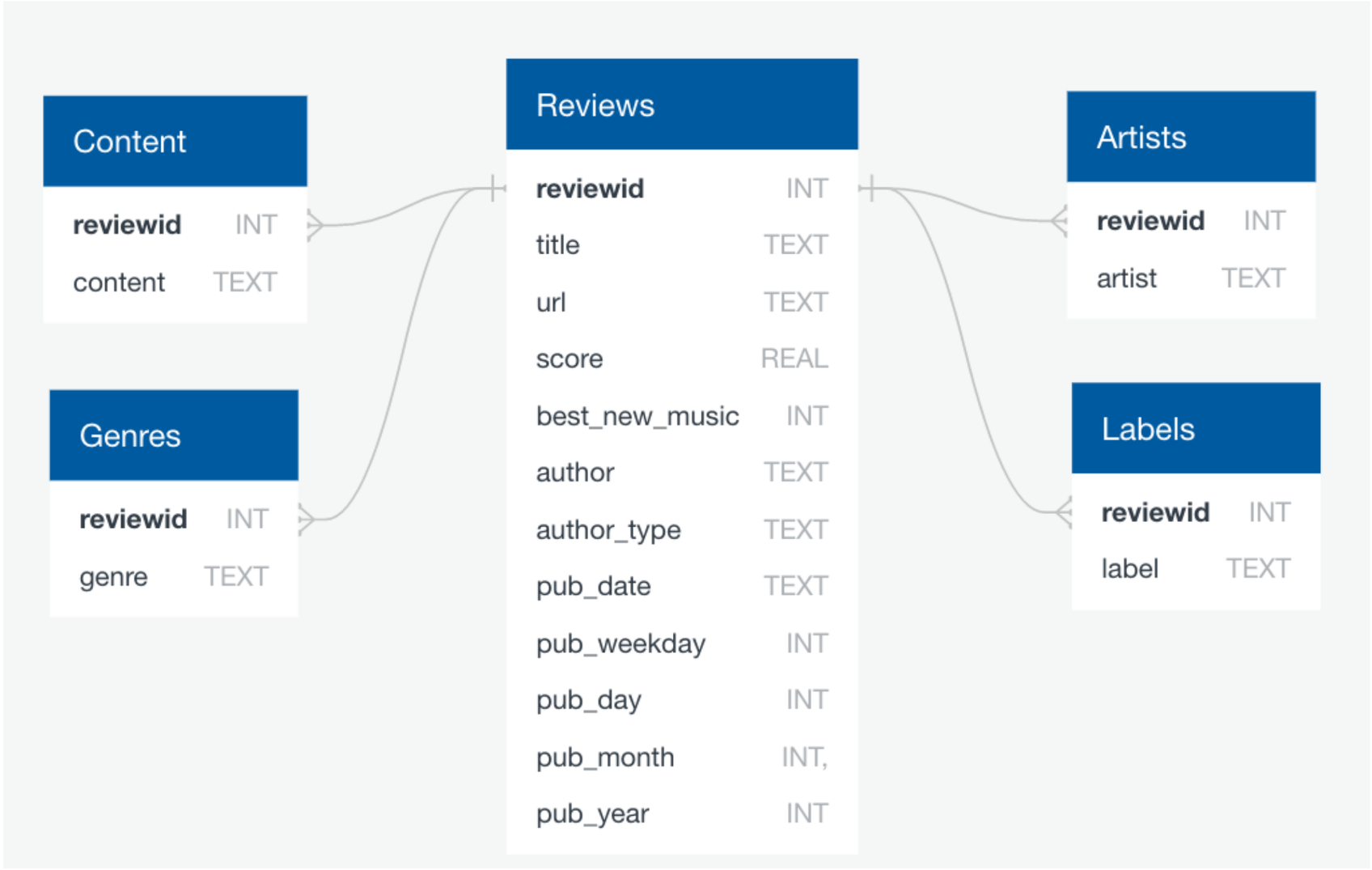
Includes system views

```
SELECT * FROM information_schema.views
WHERE table_schema NOT IN ('pg_catalog', 'information_schema');
```

Excludes system views

Benefits of views

- Doesn't take up storage
- A form of **access control**
  - Hide sensitive columns and restrict what user can see
- Masks complexity of queries
  - Useful for highly normalized schemas



[1] https://www.kaggle.com/nolanbconaway/pitchfork [2] data

Let's practice!

DATABASE DESIGN

Managing views

DATABASE DESIGN

Lis Sulmont
Curriculum Manager

Creating more complex views

- **Aggregation:** `SUM()` , `AVG()` , `COUNT()` , `MIN()` , `MAX()` , `GROUP BY` , etc
- **Joins:** `INNER JOIN` , `LEFT JOIN` . `RIGHT JOIN` , `FULL JOIN`
- **Conditionals:** `WHERE` , `HAVING` , `UNIQUE` , `NOT NULL` , `AND` , `OR` , `>` , `<` , etc

Granting and revoking access to a view

`GRANT privilege(s)` or `REVOKE privilege(s)`

`ON object`

`TO role` or `FROM role`

- **Privileges:** `SELECT` , `INSERT` , `UPDATE` , `DELETE` , etc
- **Objects:** table, view, schema, etc
- **Roles:** a database user or a group of database users

Granting and revoking example

```
GRANT UPDATE ON ratings TO PUBLIC;
```

```
REVOKE INSERT ON films FROM db_user;
```

Updating a view

```
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

Not all views are updatable

- View is made up of one table
- Doesn't use a window or aggregate function

[1] https://www.postgresql.org/docs/9.5/sql [2] update.html

Inserting into a view

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

Not all views are insertable

[1] https://www.postgresql.org/docs/9.5/sql [2] insert.html

Inserting into a view

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

Not all views are insertable

Takeaway: avoid modifying data through views

[1] https://www.postgresql.org/docs/9.5/sql [2] insert.html

Dropping a view

```
DROP VIEW view_name [ CASCADE | RESTRICT ];
```

- `RESTRICT` (default): returns an error if there are objects that depend on the view
- `CASCADE` : drops view and any object that depends on that view

Redefining a view

```
CREATE OR REPLACE VIEW view_name AS new_query
```

- If a view with `view_name` exists, it is replaced
- `new_query` must generate the same column names, order, and data types as the old query
- The column output may be different
- New columns may be added at the end

If these criteria can't be met, drop the existing view and create a new one

[1] https://www.postgresql.org/docs/9.2/sql [2] createview.html

Altering a view

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET DEFAULT expression
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP DEFAULT
ALTER VIEW [ IF EXISTS ] name OWNER TO new_owner
ALTER VIEW [ IF EXISTS ] name RENAME TO new_name
ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema
ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_option_value] [, ... ] )
ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ... ] )
```

[1] https://www.postgresql.org/docs/9.2/sql [2] alterview.html

Let's practice!

DATABASE DESIGN

Materialized views

DATABASE DESIGN

Lis Sulmont
Curriculum Manager

Two types of views

Views

- Also known as **non-materialized views**
- How we've defined views so far

Two types of views

Views

- Also known as **non-materialized views**
- How we've defined views so far

Materialized views

- Physically materialized

Materialized views

- Stores the ***query results***, not the ***query***
- Querying a materialized view means accessing the stored query results
  - Not running the query like a non-materialized view
- Refreshed or rematerialized when prompted or scheduled

When to use materialized views

- Long running queries
- Underlying query results don't change often
- Data warehouses because OLAP is not write-intensive
  - Save on computational cost of frequent queries

Implementing materialized views

(in PostgreSQL)

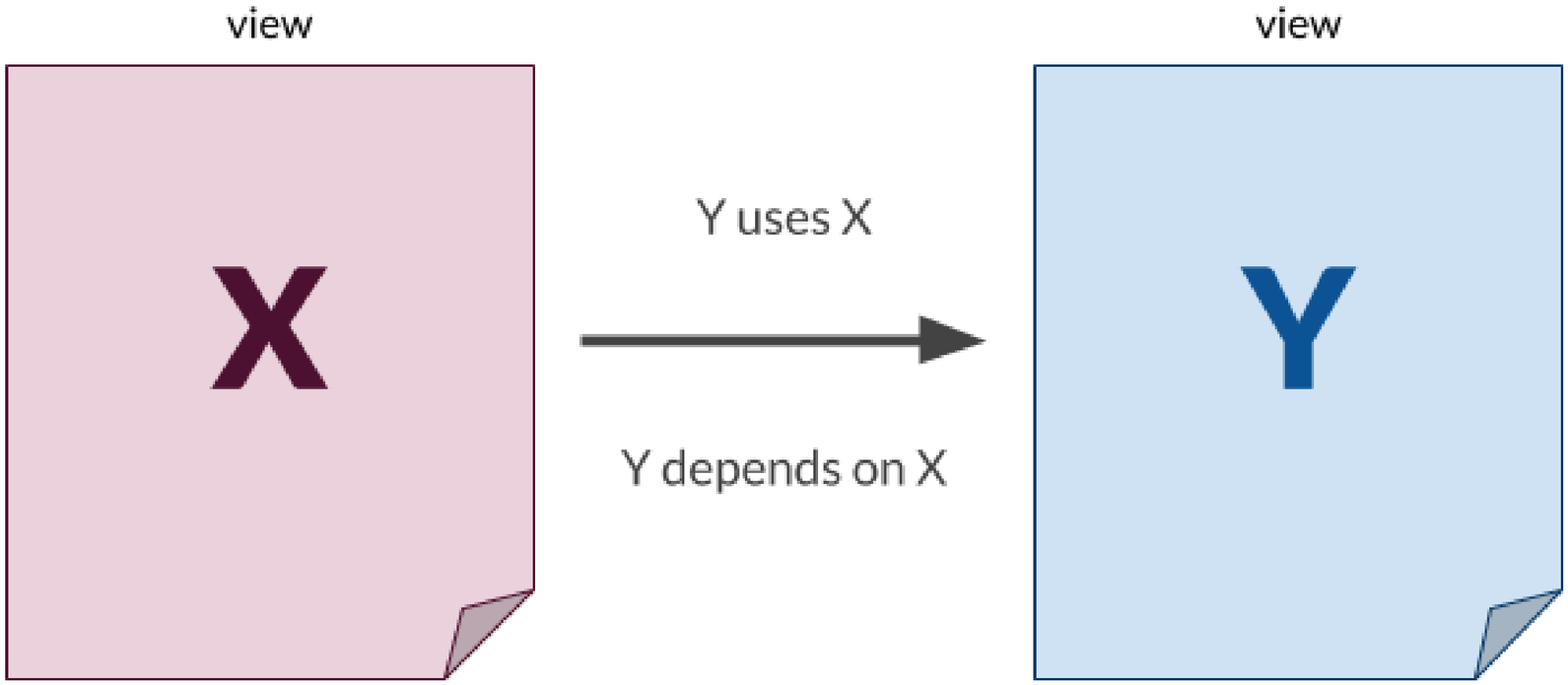
```
CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM existing_table;
```

```
REFRESH MATERIALIZED VIEW my_mv;
```

Managing dependencies

- Materialized views often depend on other materialized views

Dependency example

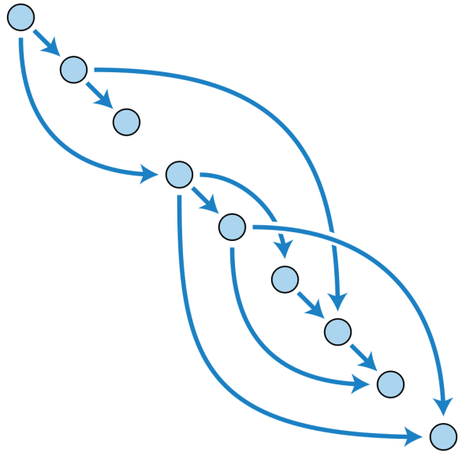


Managing dependencies

- Materialized views often depend on other materialized views
- Creates a **dependency chain** when refreshing views
- Not the most efficient to refresh all views at the same time

Tools for managing dependencies

- Use Directed Acyclic Graphs (**DAGs**) to keep track of views



- Pipeline scheduler tools

Let's practice!

DATABASE DESIGN