Principles of Database Systems (CS307)

Lecture 7: Advanced SQL

Zhong-Qiu Wang

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.
- The slides are largely based on the slides provided by Dr. Yuxin Ma

Announcements

- Deadline of Project I is extended to 16^{th} Nov., Sunday, 10pm
  - Do not miss the deadline, or you will receive reduced scores

Trigger（触发器）

Trigger （触发器）- Actions When Changing Tables

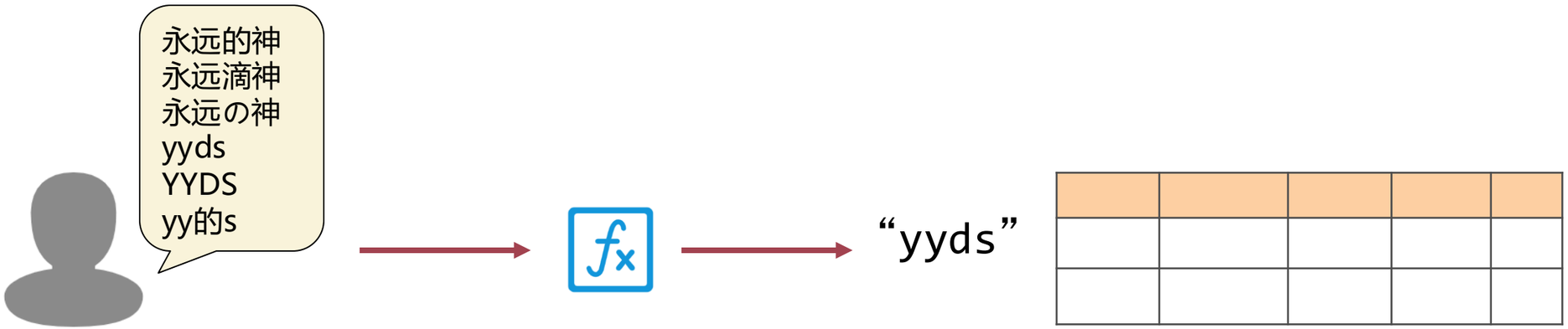
> A trigger is a specification that the database should automatically execute a particular function whenever a certain type of operation is performed.
>
> -- Chapter 39, PostgreSQL Documentation
>
> A trigger is a statement that the system executes automatically as a side effect of a modification to the database.
>
> -- Chapter 5.3, Database System Concepts, 7th

- We can attach “actions” to a table
  - They will be executed automatically whenever the data in the table changes
- Purpose of using triggers
  - Validating data
  - Checking complex rules
  - Managing data redundancy

Purpose of Using Triggers

- Validating data
  - Some data are badly formatted in programs before sending to the database
  - We need to validate such data before inserting them into the database

  - “On-the-fly” modification
    - Change the input directly when the input arrives



Purpose of Using Triggers

- Validating data
  - Example: insert a row in the movies table
    - In the JDBC program, an `insert` request is written like the following:

```
insert into movies(title, country, year_released)
values ('Laura', 'US', 1944);
```

Need to update it to 'us' before inserting

  - Although,
    - Such validation or transformation should be better handled by the application programs

Purpose of Using Triggers

- Checking complex rules
  - Sometimes, the business rules are so complicated
    - They CANNOT be checked via declarative integrity constraints

Purpose of Using Triggers

- Managing data redundancy
  - Some data redundancy issues cannot be avoided by simply adding constraints
  - For example: inserted the same movie but in different languages

```sql
-- US
insert into movies(title, country, year_released)
values ('The Matrix', 'us', 1999);

-- China (Mainland)
insert into movies(title, country, year_released)
values ('黑客帝国', 'us', 1999);

-- Hongkong
insert into movies(title, country, year_released
values ('22世紀殺人網絡', 'us', 1999);
```

It satisfies the unique constraint on (`title, country, year_released`)

- … but they represent the same movie

Trigger Activation

- Two key points:
  - When to fire a trigger?
  - What (command) fires a trigger?

Trigger Activation

- When to fire a trigger?
  - In general: “During the change of data”
    - … but we need a detailed discussion
  - Note: “During the change” means `select` queries won’t fire a trigger.

Trigger Activation: When

- Example: Insert a set of rows with "`insert into select`"
  - One statement, multiple rows

```
insert into movies(title, country, year_released)
select titre, 'fr', annee
from films_francais;
```

  - Option 1: Fire a trigger only once for the statement
    - Before the first row is inserted, or after the last row is inserted
  - Option 2: Fire a trigger for each row
    - Before or after the row is inserted

Trigger Activation: When

- Different options between DBMS products

- Before statement
  - Before each row
  - After each row
- After statement

- ~~Before statement~~
  - Before each row
  - After each row
- ~~After statement~~

- ~~Before statement~~
  - ~~Before each row~~
  - ~~After each row~~
- After statement

Trigger Activation: What

- What (command) fires a trigger?
  - `insert`
  - `update`
  - `delete`

Example of Triggers

- An Example
  - For the `people_1` table, when updating a person, count the number of movies and save the result in the `num_movies` column

```sql
-- auto-generated definition
create table people_1
(
    peopleid   integer,
    first_name varchar(30),
    surname    varchar(30),
    born       integer,
    died       integer,
    gender     bpchar,
    num_movies integer
);
```

	peopleid	first_name	surname	born	died	gender	num_movies
1	13	Hiam	Abbass	1960	<null>	F	<null>
2	559	Aleksandr	Askoldov	1932	<null>	M	<null>
3	572	John	Astin	1930	<null>	M	<null>
4	585	Essence	Atkins	1972	<null>	F	<null>
5	598	Antonella	Attili	1963	<null>	F	<null>
6	611	Stéphane	Audran	1932	<null>	F	<null>
7	624	William	Austin	1884	1975	M	<null>
8	637	Tex	Avery	1908	1980	M	<null>
9	650	Dan	Aykroyd	1952	<null>	M	<null>
10	520	Zackary	Arthur	2006	<null>	M	<null>
11	533	Oscar	Asche	1871	1936	M	<null>
12	546	Elizabeth	Ashley	1939	<null>	F	<null>

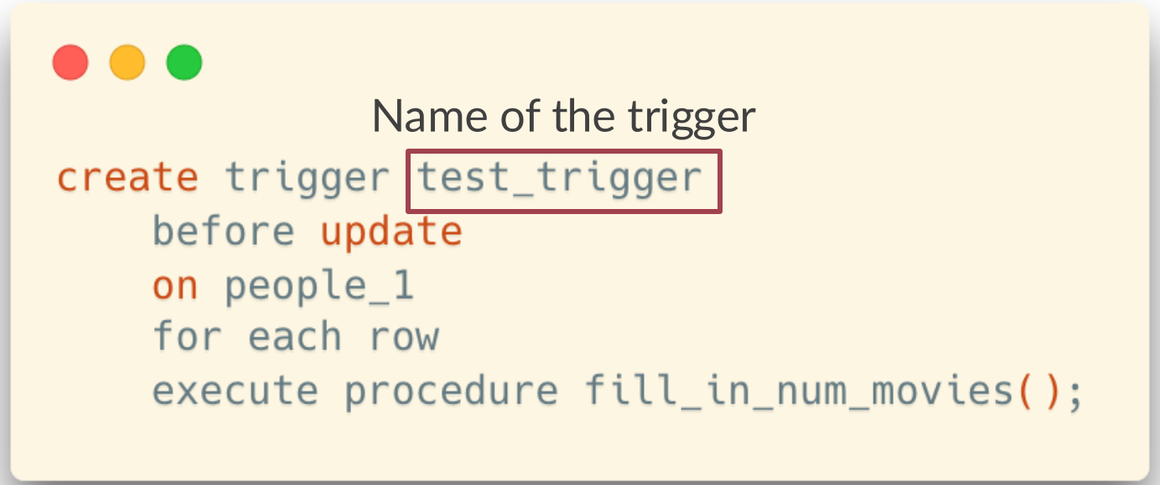
Example of Triggers

- Create a trigger

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

Example of Triggers

- Create a trigger



Example of Triggers

- Create a trigger

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

{ BEFORE | AFTER | INSTEAD OF } { event [OR ...] }

- Specify when the trigger will be executed
  - `before` | `after`
- … and on what operations the trigger will be executed
  - `insert [or update [or delete]]`

Example of Triggers

- Create a trigger

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

The table name

“for each row”
or
“for each statement”
(default)

Example of Triggers

- Create a trigger

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

The actual procedure for the trigger

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

"trigger" is the return type

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

”new” and “old” are two internal variables that represents the row before and after the changes

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

Remember to return the result which will be used in the `update` statement

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well
    - Remember to create the procedure before creating the trigger

- Run test updates

```
-- create the procedure fill_in_num_movies() first

-- then, create the trigger

-- finally, we can run some test update statements
update people_1 set num_movies = 0 where people_1.peopleid <= 100;
```

Before and After Triggers

- Differences between before and after triggers
  - “Before” and “after” the operation is done (`insert`, `update`, `delete`)
  - If we want to update the incoming values in an `update` statement (e.g., “US” → “us”), “before trigger” should be used since the incoming values have not been written to the table yet

Before and After Triggers

- Typical usage scenarios for trigger settings
  - Modify input on the fly
    - before insert / update
    - for each row
  - Check complex rules
    - before insert / update / delete
    - for each row
  - Manage data redundancy
    - after insert / update / delete
    - for each row

Example: Auditing

- One good example of managing some data redundancy is keeping an audit trail
  - Not really care about people who read data
    - (remember that `select` cannot fire a trigger – although with the big products you can trace all queries)
  - … but it may be useful for checking people who modify data that they aren't supposed to modify

Example: Auditing

- Trace the insertions and updates to employees in a company

```
create table company(
   id int primary key     not null,
   name           text    not null,
   age            int     not null,
   address        char(50),
   salary         real
);


create table audit(
   emp_id int not null,
   change_type char(1) not null,
   change_date text not null
);
```

Example: Auditing

- Trace the insertions and updates to employees in a company

```
create trigger audit_trigger
    after insert or update
    on company
    for each row
execute procedure auditlogfunc();


create or replace function auditlogfunc() returns trigger as
$example_table$
begin
    insert into audit(emp_id, change_type, change_date)
    values (new.id,
            case
                when tg_op = 'UPDATE' then 'U'
                when tg_op = 'INSERT' then 'I'
                else 'X'
            end,
            current_timestamp);
    return new;
end ;
$example_table$ language plpgsql;
```

Example: Auditing

- Trace the insertions and updates to employees in a company

```
insert into company (id, name, age, address, salary)
values (2, 'Mike', 35, 'Arizona', 30000.00);
```

company

	id	name	age	address	salary
1	2	Mike	35	Arizona	30000

audit

	emp_id	change_type	change_date
1	2	I	2022-04-25 18:37:35.515151+00

View

Recall: Function

- Used for returning numbers, strings, dates, etc.
  - Or, return a single value

```
select full_name(first_name, surname)
from people
where surname like '%(von)';
```

... returns a string of the full name

View

- Reuse of the relational operations (as we reuse codes in a program)
  - ... which returns relations (tables) instead of simple values

```
create view viewname as
select ...
from ...
where ...
```

View

- Example: Create a view like this:
  - … where the result of the inner `select` query looks like this:

```
create view vmovies as
select m.movieid,
       m.title,
       m.year_released,
       c.country_name
from movies m join countries c
    on c.country_code = m.country;
```

	movieid	title	year_released	country_name
1	1	12 stulyev	1971	Russia
2	2	Al-mummia	1969	Egypt
3	3	Ali Zaoua, prince de la rue	2000	Morocco
4	4	Apariencias	2000	Argentina
5	5	Ardh Satya	1983	India
6	6	Armaan	2003	India
7	7	Armaan	1966	Pakistan
8	8	Babettes gæstebud	1987	Denmark
9	9	Banshun	1949	Japan
10	10	Bidaya wa Nihaya	1960	Egypt
11	11	Variety	2008	United States
12	12	Bon Cop, Bad Cop	2006	Canada
13	13	Brilliantovaja ruka	1969	Russia
14	14	C'est arrivé près de chez vous	1992	Belgium
15	15	Carlota Joaquina - Princesa d…	1995	Brazil
16	16	Cicak-man	2006	Malaysia
17	17	Da Nao Tian Gong	1965	China
18	18	Das indische Grabmal	1959	Germany
19	19	Das Leben der Anderen	2006	Germany
20	20	Den store gavtyv	1956	Denmark

View vs. Table

- In practice, there isn't much to a view
  - It’s basically a named query
  - And, why not just consider it as a “virtual table”?

```
create view vmovies(v_id, v_title, v_year, v_country) as
select m.movieid,
       m.title,
       m.year_released,
       c.country_name
from movies m
         join countries c
                    on c.country_code = m.country;
```

View vs. Table

- In practice, there isn't much to a view
  - It’s basically a named query
  - And, why not just consider it as a “virtual table”?

The columns can be renamed

- Otherwise, the original names in the tables will be used

```
create view vmovies(v_id, v_title, v_year, v_country) as
select m.movieid,
       m.title,
       m.year_released,
       c.country_name
from movies m
         join countries c
                    on c.country_code = m.country;
```

View vs. Table

- In practice, there isn't much to a view
  - It’s basically a named query
  - And, why not just consider it as a “virtual table”?

The columns can be renamed

- Otherwise, the original names in the tables will be used

```
create view vmovies(v_id, v_title, v_year, v_country) as
select m.movieid,
       m.title,
       m.year_released,
       c.country_name
from movies m
         join countries c
                      on c.country_code = m.country;
```

Drop the existing view before creating a new one with the same name:

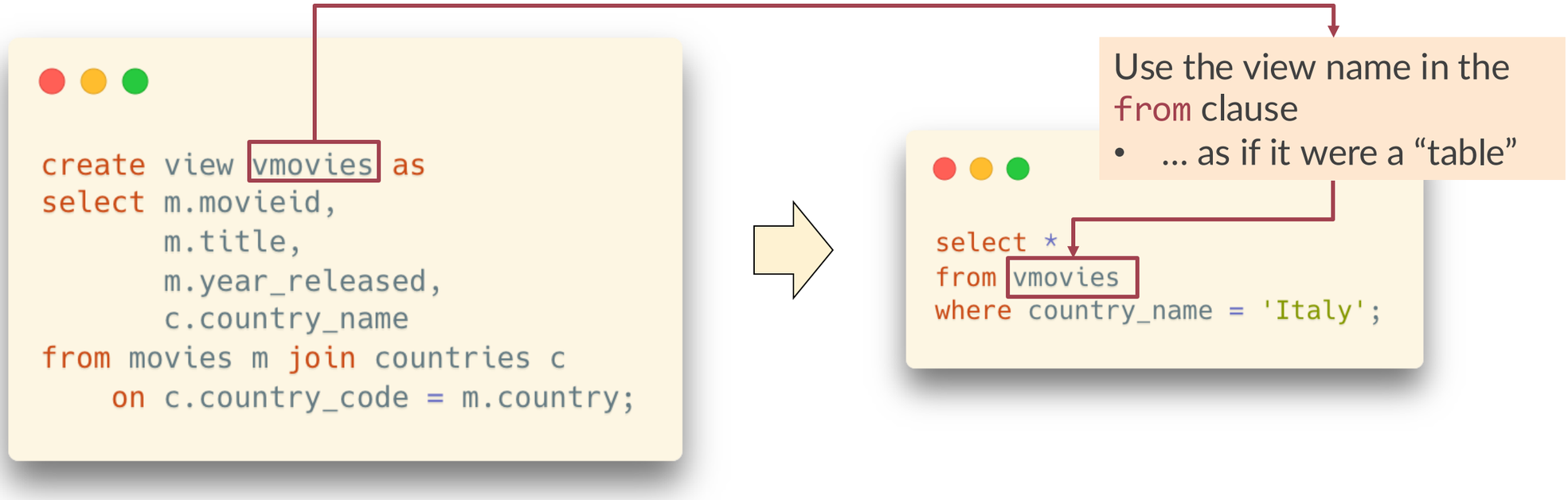
```
drop view vmovies;
```

View vs. Table

- A view can be as a complicated query as you want
  - It will usually return something that isn't as normalized as your tables, but easier to understand.
  - Imagine: a function of a `select` query
  - From a programming language's view:
    - Tables -> variables
    - Rows -> values

View vs. Table

- Once the view is created,
  - … we can query the view exactly as if it were a table



Dynamic Content in Views

- When the rows change in the tables where a view relies on, the result of a view will change as well
  - Think it like a table variable, or a query result
  - * Or, think it like a relational function which returns a relation

Dynamic Content in Views

- Beware that columns are the ones in tables when the view was created
  - Columns added later to tables in the view won't be added even if the view was created with "`select *`"
    - And, "`select *`" is a bad practice in view. We will not know what exact columns are selected once the columns in the tables are changed.

```
-- Create the view first
create view vmovies as
select *  -- IT IS A BAD PRACTICE OF USING * HERE
from movies m join countries c
    on c.country_code = m.country;

-- Alter the table then
alter table movies add column test_col int not null default 0;

-- There won't be a column "test_col" in the query result
select * from vmovies;
```

View vs. Table (Cont.)

- View looks like Table, tastes like Table
  - Sometimes it is used as a table

  - Usage Scenario 1: Simplify Complex Queries
    - Simplify a complicated query result into a single named query
    - E.g. Many business reports are based on the same set of joins, with just variations on the columns that you aggregate or order by
      - One command, same query

View vs. Table (Cont.)

- View looks like Table, tastes like Table
  - Sometimes it is used as a table
  - Usage Scenario 2: An alternative way to implement E-R models
    - Sometimes, we may not be able to use tables to model entities and relationships
      - Access control
      - Dirty and messy original data
    - Since views look like tables, we can use views to represent entities or relationships
      - … based on some existing objects (like tables)

Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as relation schemas that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
  - For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
  - Each schema has a number of columns (generally corresponding to attributes), which have unique names.

There are more than one way to implement E-R models

Drawback of Views

- View looks like Table, tastes like Table
  - But it is still not a table.
  - In some cases, it may cause performance issues or unnecessary operations
    - * … since the users of the views don’t know the details of the views

Drawback of Views

- View looks like Table, tastes like Table
  - Example: A refined way to display the credit information with a view

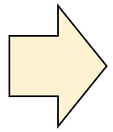
```
create view vmovie_credits
as
select m.title,
       m.year_released release,
       case c.credited_as
           when 'A' then 'Actor'
           when 'D' then 'Director'
           else '?'
           end duty,
       full_name(p.first_name, p.surname) name
from movies m
         inner join credits c
                    on c.movieid = m.movieid
         inner join people p
                    on p.peopleid = c.peopleid;
```

	title	release	duty	name
1	"Erogotoshitachi" yori Jinruigaku nyūmon	1966	Director	Shohei Imamura
2	"Erogotoshitachi" yori Jinruigaku nyūmon	1966	Actor	Shoichi Ozawa
3	"Erogotoshitachi" yori Jinruigaku nyūmon	1966	Actor	Sumiko Sakamoto
4	'76	2016	Actor	Ramsey Nouah
5	'76	2016	Actor	Ibinabo Fiberesima
6	(T)Raumschiff Surprise	2004	Actor	Michael Herbig
7	(T)Raumschiff Surprise	2004	Director	Michael Herbig
8	(T)Raumschiff Surprise	2004	Actor	Til Schweiger
9	(T)Raumschiff Surprise	2004	Actor	Anja Kling
10	(T)Raumschiff Surprise	2004	Actor	Rick Kavanian
11	(T)Raumschiff Surprise	2004	Actor	Christian Tramitz
12	(T)Raumschiff Surprise	2004	Actor	Sky du Mont
13	002 operazione luna	1965	Actor	Linda Sini
14	002 operazione luna	1965	Director	Lucio Fulci

Drawback of Views

- View looks like Table, tastes like Table
  - Example: A refined way to display the credit information with a view
    - However, if we only want to get all distinct movie titles, it will return the correct result, but there are a lot of useless works in the internal query of the view

```
select distinct title
from vmovie_credits
```



```
create view vmovie_credits
as
select m.title,
       m.year_released release,
       case c.credited_as
           when 'A' then 'Actor'
           when 'D' then 'Director'
           else '?'
           end duty,
       full_name(p.first_name, p.surname) name
from movies m
         inner join credits c
                    on c.movieid = m.movieid
         inner join people p
                    on p.peopleid = c.peopleid;
```

Do we really need the joins?

View vs. Materialized View

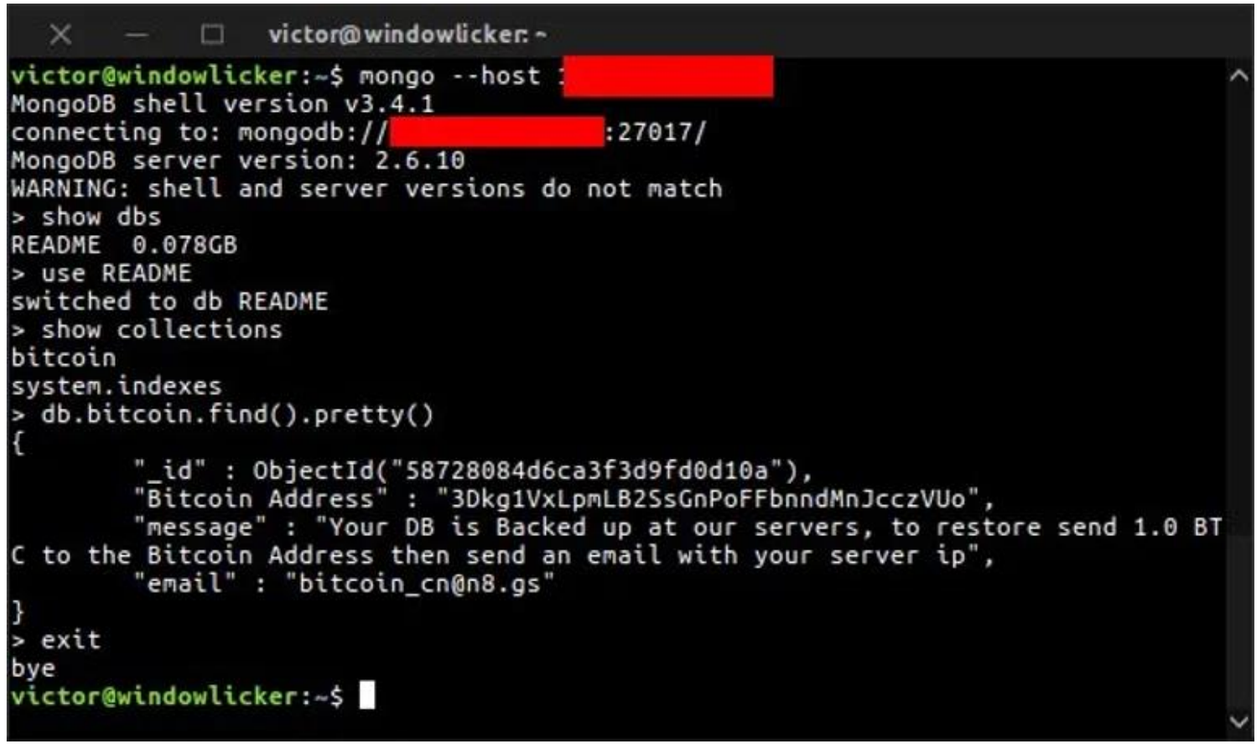
- Normal views dynamically retrieve data from the original tables
  - However, the performance won't be good

- Materialized View: A cached result of a view
  - When the corresponding tables are not changing rapidly, we can cache the view results (i.e., materialize the results)
    - Better performance than dynamic retrieval
  - The results are not updated each time the view is used
    - Manual update, or use the help of triggers

Access Control

Authentication

- To access a database, you must be authenticated, which often means entering a username and a password.
  - If you don’t set a proper password, you will put your database system in danger



https://www.zdnet.com/article/mongodb-ransacked-now-27000-databases-hit-in-mass-ransom-attacks/

Privileges

- Besides, DBMS usually provide another layer of access control on objects in the system
  - Operations (select, update, insert, delete, etc.)
  - Objects (table, database, views, trigger, etc.)

Grant and Revoke Access

- `grant` and `revoke`
  - Can be used on the table or the database level

```
-- grant <right> to <account>
grant select on movies to test_user;

-- revoke <right> from <account>
revoke select on movies from test_user;
```

The possible privileges are:

```
SELECT
INSERT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
CREATE
CONNECT
TEMPORARY
EXECUTE
USAGE
```

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

How Can Views Help

- User access control
  - By creating a view that only returns rows associated with the user name, we can control the privileges for a specific user in this table

```
create view my_stuff
as
select * from stuff
where username = user
```

```
-- E.g., restrict the users to only access
-- the movies assigned to their user names
create view user_view as
select movieid,
       title,
       country,
       year_released,
       runtime
from movies
where user_name = user;
```

How Can Views Help

- User access control
  - By creating a view that only returns rows associated with the user name, we can control the privileges for a specific user in this table

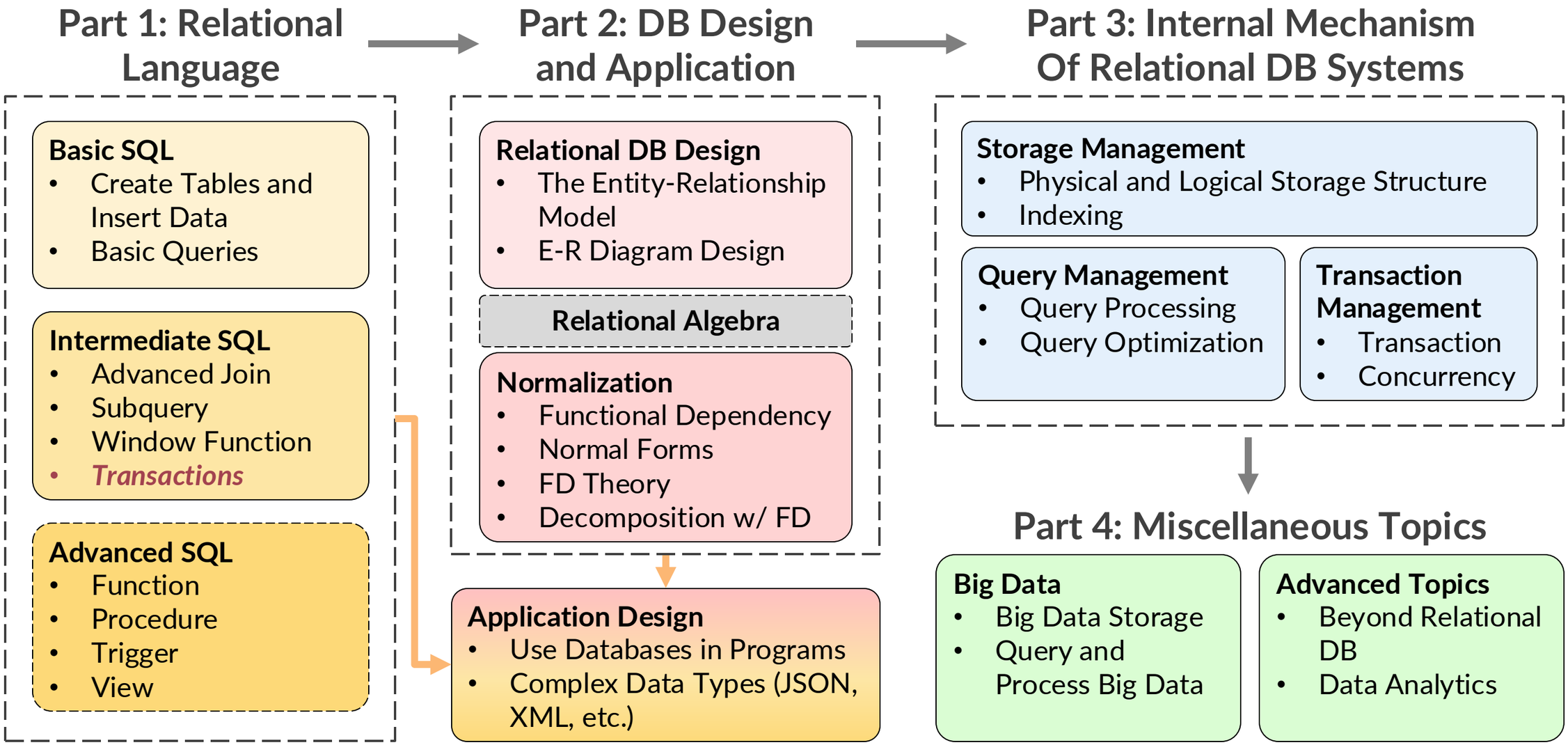
```
create view my_stuff
as
select * from stuff
where username = user
```

```
-- E.g., restrict the users to only access
-- the movies assigned to their user names
create view user_view as
select movieid,
       title,
       country,
       year_released,
       runtime
from movies
where user_name = user;
```

`user`: An internal variable that returns the current user name

- Remember `old` and `new` in triggers?

Outline


Part 1: Relational Language
Basic SQL
• Create Tables and Insert Data
• Basic Queries
Intermediate SQL
• Advanced Join
• Subquery
• Window Function
• Transactions
Advanced SQL
• Function
• Procedure
• Trigger
• View
Part 2: DB Design and Application
Relational DB Design
• The Entity-Relationship Model
• E-R Diagram Design
Relational Algebra
Normalization
• Functional Dependency
• Normal Forms
• FD Theory
• Decomposition w/ FD
Application Design
• Use Databases in Programs
• Complex Data Types (JSON, XML, etc.)
Part 3: Internal Mechanism Of Relational DB Systems
Storage Management
• Physical and Logical Storage Structure
• Indexing
Query Management
• Query Processing
• Query Optimization
Transaction Management
• Transaction
• Concurrency
Part 4: Miscellaneous Topics
Big Data
• Big Data Storage
• Query and Process Big Data
Advanced Topics
• Beyond Relational DB
• Data Analytics