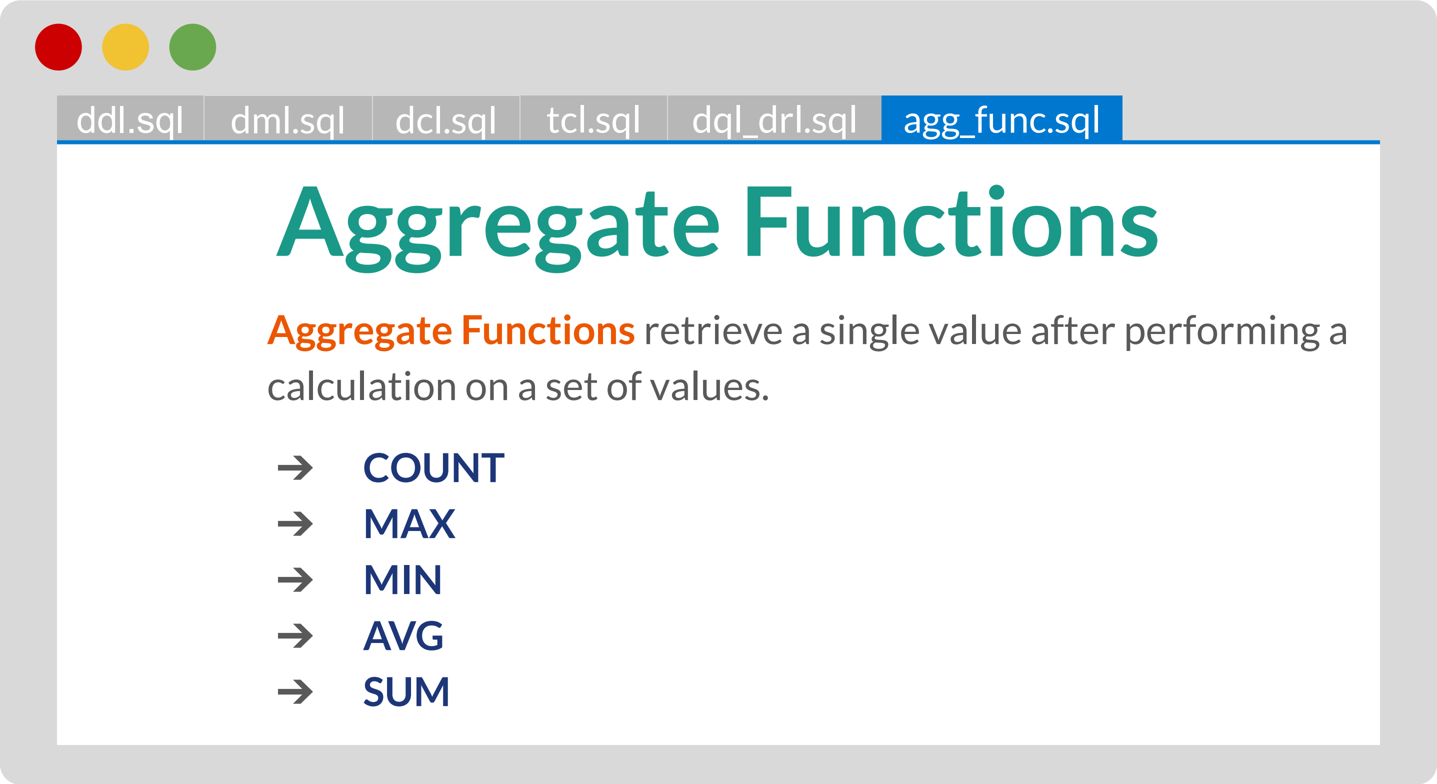


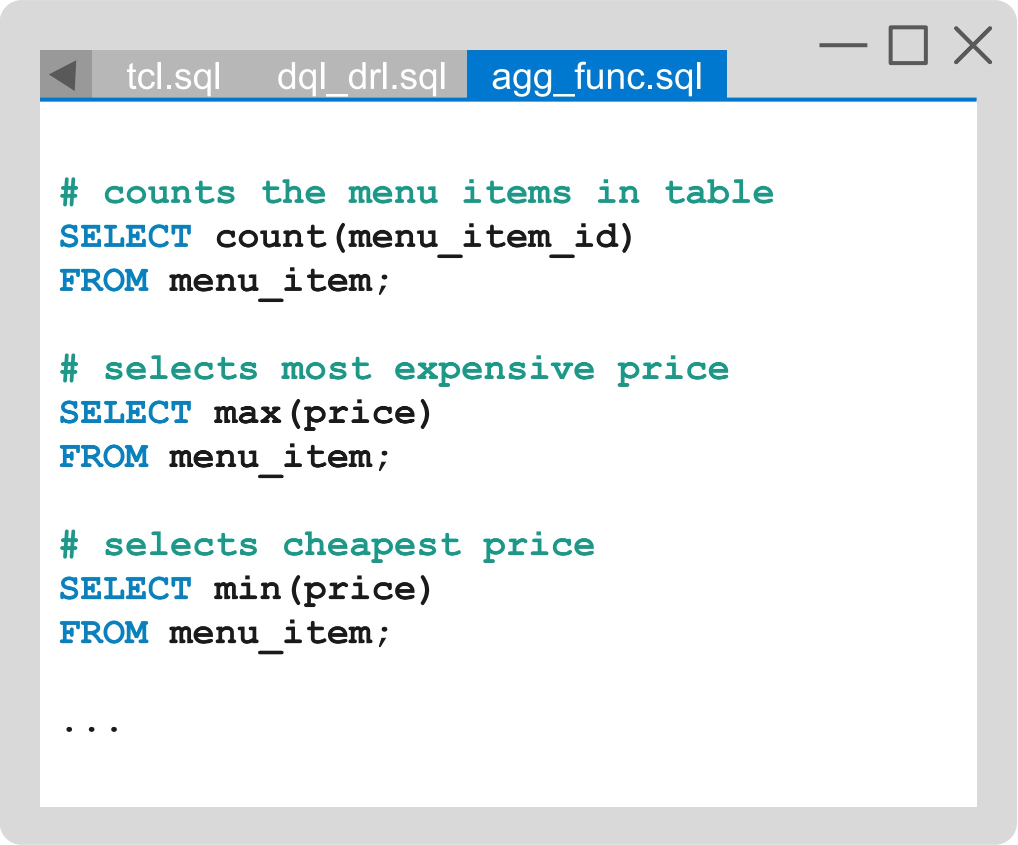
Aggregate Functions

Aggregate Functions retrieve a single value after performing a calculation on a set of values.

- → COUNT
- → MAX
- → MIN
- → AVG
- → SUM

Aggregate Functions

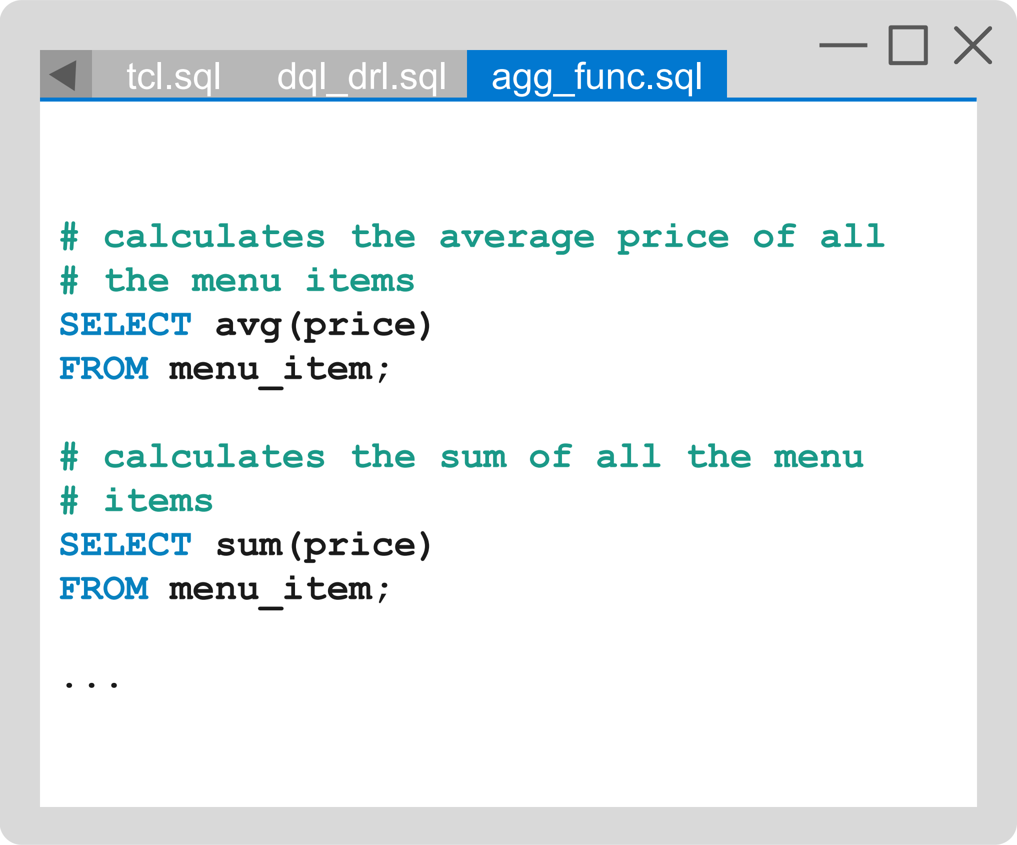
- ➔ COUNT will count records
- ➔ MAX will select the maximum value in a column
- ➔ MIN selects the minimum



```sql
# counts the menu items in table
SELECT count(menu_item_id)
FROM menu_item;

# selects most expensive price
SELECT max(price)
FROM menu_item;

# selects cheapest price
SELECT min(price)
FROM menu_item;

...
```

Aggregate Functions

- ➔ **AVG** calculates the average value for a column
- ➔ **SUM** calculates the sum of all the values in a column



```
# calculates the average price of all
# the menu items
SELECT avg(price)
FROM menu_item;

# calculates the sum of all the menu
# items
SELECT sum(price)
FROM menu_item;

...
```

Save Point & Check In

Sakila Queries Exercise

1. The rental table tracks each film rental by a customer. Select the customer_id and count of the amount of rentals in for that customer. Only select the top 5 customers with the highest rentals amounts.

2. Using the address table, select each district and a count of how many addresses are in each district.

3. Select the title, rental_rate, and replacement_cost from the film table, find the films that have a rental_rate less than a dollar or a replacement_cost less than fifteen dollars.

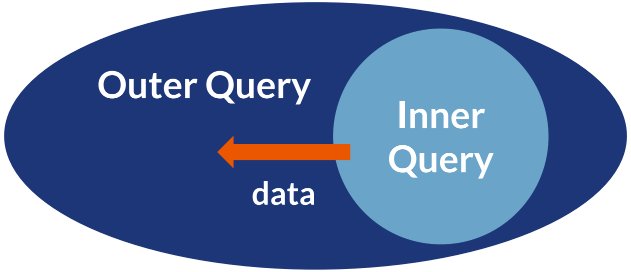
4. Find the sum of the total amount spent by each customer from the payment table. Only select the customers that have a sum above $150. Display the customer_id and the sum total as 'total amount spent'.

5. Select the customer_id and first_name from the customer table where the first_name is at least 3 characters long and ends in an 'o'.

◀ agg_func.sql | str_func.sql | num_func.sql | date_func.sql | subqu.sql

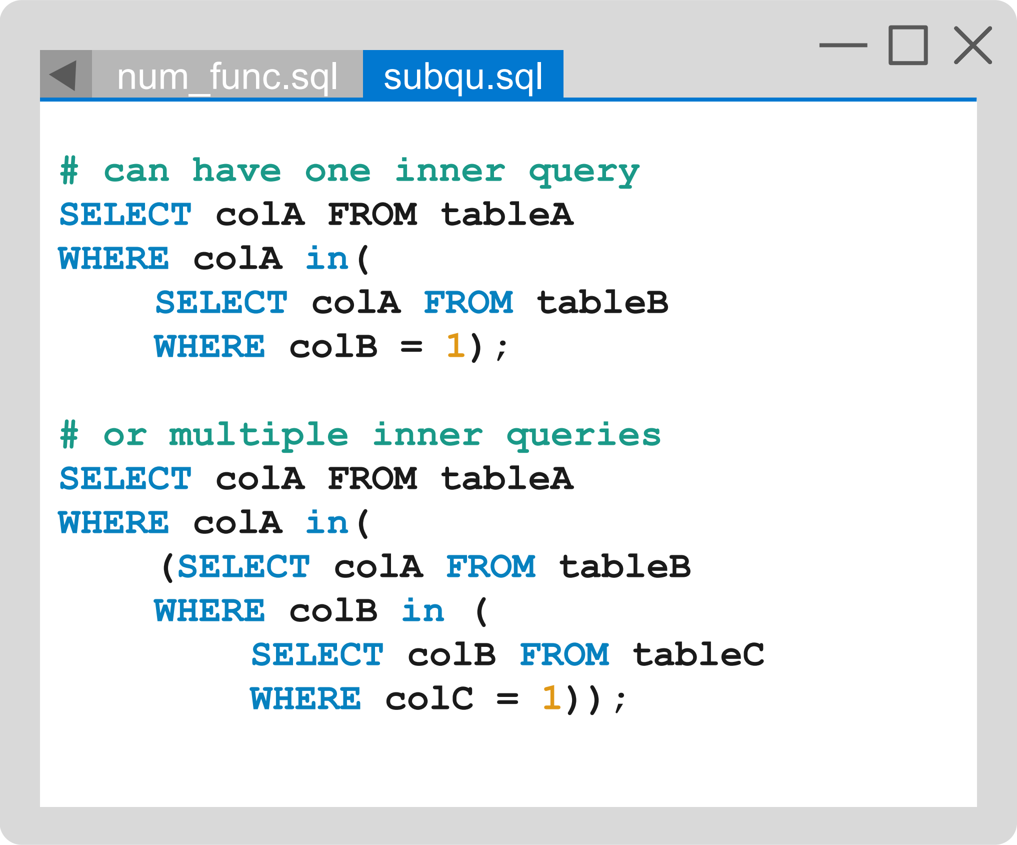
Subqueries

Subqueries (nested queries) are a query within another query and embedded within a WHERE clause.



Subqueries

- ➔ Subqueries can have multiple layers of queries within queries
- ➔ ***Best to use when you need a calculation like an aggregate in one query to do selection in other query***

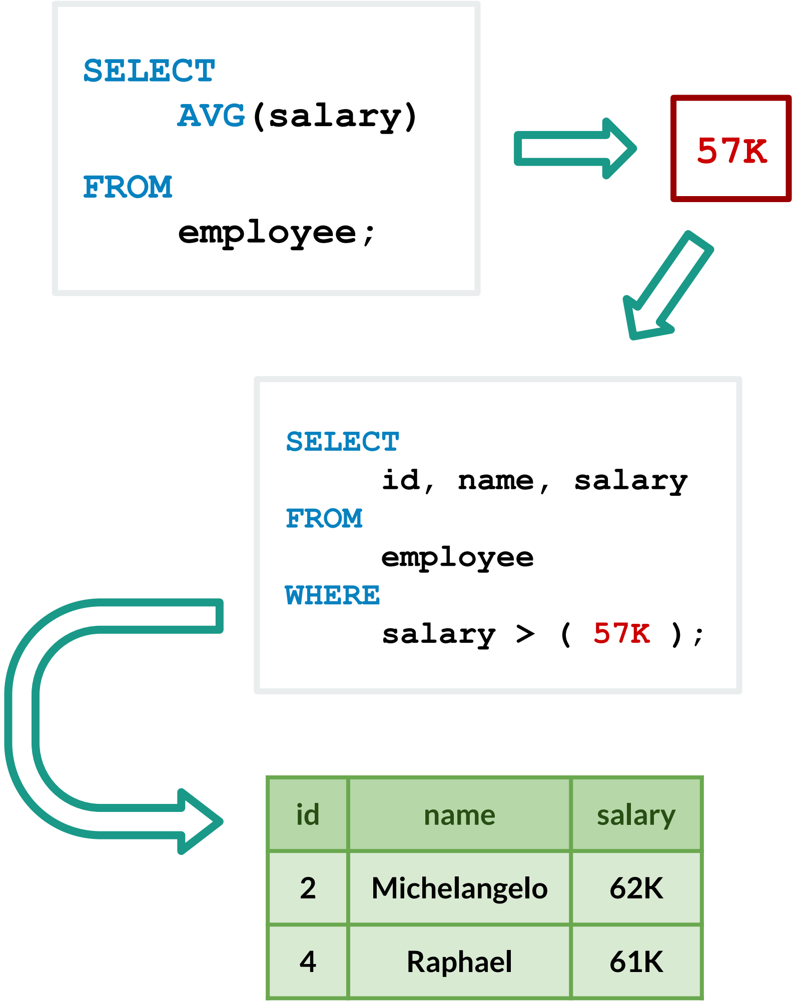


```
# can have one inner query
SELECT colA FROM tableA
WHERE colA in(
    SELECT colA FROM tableB
    WHERE colB = 1);

# or multiple inner queries
SELECT colA FROM tableA
WHERE colA in(
     (SELECT colA FROM tableB
    WHERE colB in (
        SELECT colB FROM tableC
        WHERE colC = 1));
```

```
SELECT
     id, name, salary
FROM
     employee
WHERE
     salary > (
          SELECT
               AVG(salary)

          FROM
               employee
     );
```

Employee Table

id	name	salary
1	Leonardo	50K
2	Michelangelo	62K
3	Donatello	55K
4	Raphael	61K

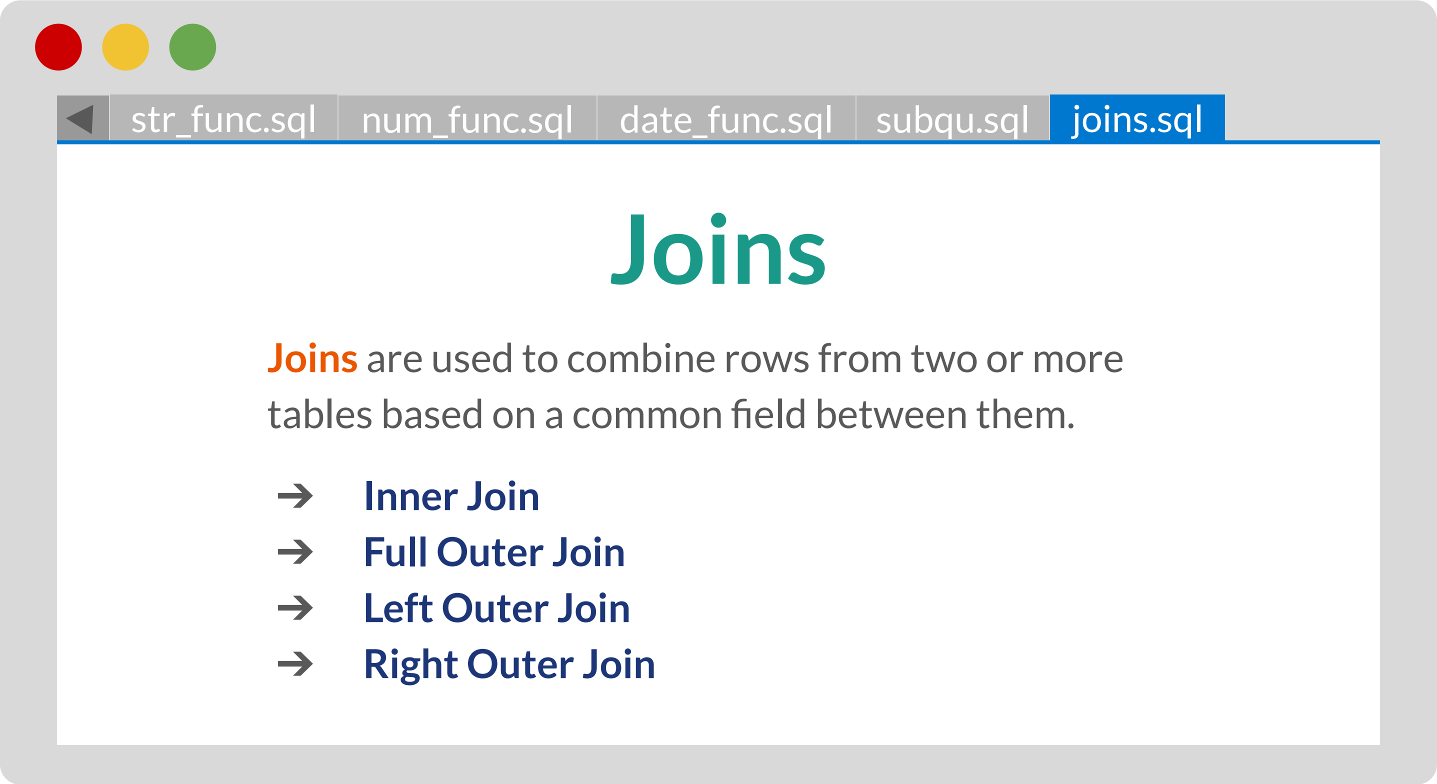


```
SELECT
     AVG(salary)

FROM
     employee;
```

57K

```
SELECT
     id, name, salary
FROM
     employee
WHERE
     salary > ( 57K );
```

id	name	salary
2	Michelangelo	62K
4	Raphael	61K

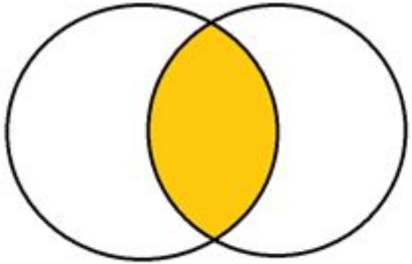


Joins

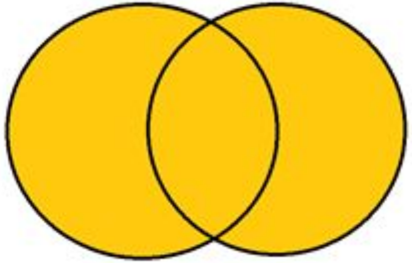
Joins are used to combine rows from two or more tables based on a common field between them.

→ **Inner Join**
→ **Full Outer Join**
→ **Left Outer Join**
→ **Right Outer Join**

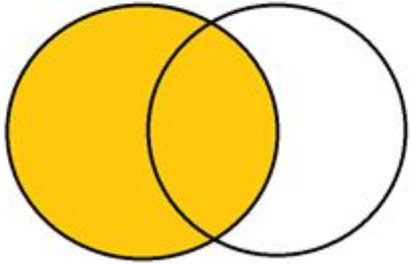
Inner Join



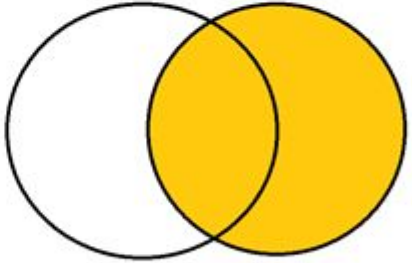
Full Outer Join



Left Outer Join



Right Outer Join



INNER JOIN

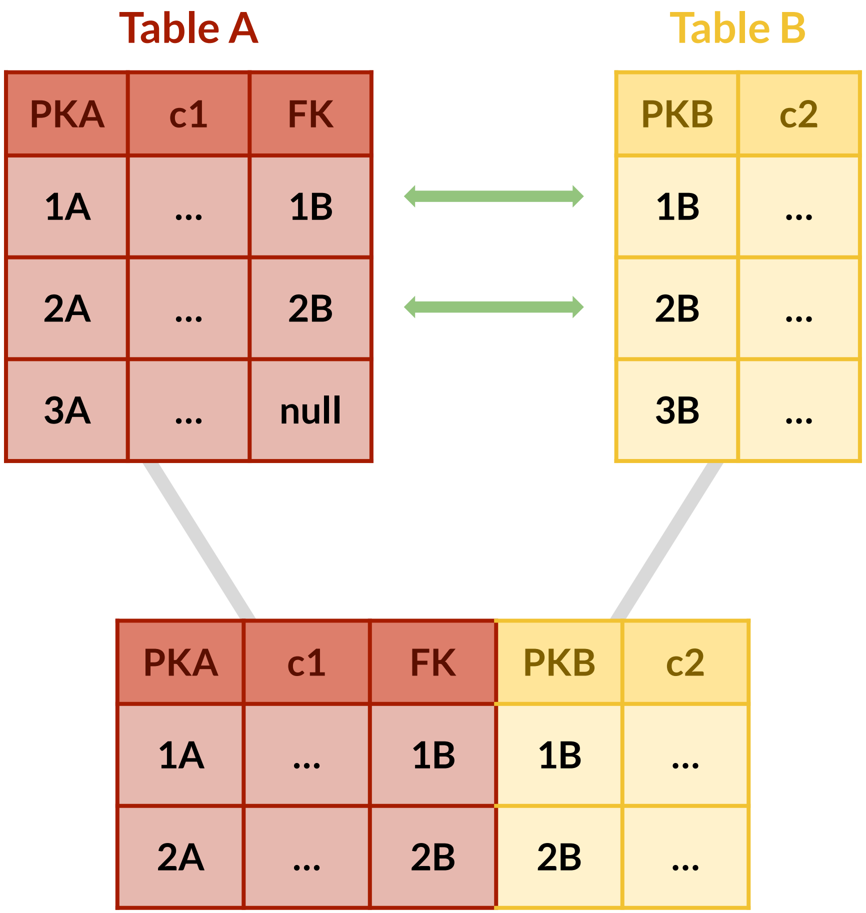


Table A

PKA	c1	FK
1A	...	1B
2A	...	2B
3A	...	null

Table B

PKB	c2
1B	...
2B	...
3B	...

PKA	c1	FK	PKB	c2
1A	...	1B	1B	...
2A	...	2B	2B	...

Combine rows from Table A and Table B that contain a **matching value**. This matching value ***must be present in both tables***.

```
SELECT
    PKA, c1, FK, PKB, c2, c3
FROM
    Table_A
JOIN
    Table_B
ON
    Table_A.FK = Table_B.PKB
```

LEFT JOIN

Combine rows from Table A and Table B that contain a **matching value** that is ***present in the left table***. The left table will be the first table listed in the query (Table A).

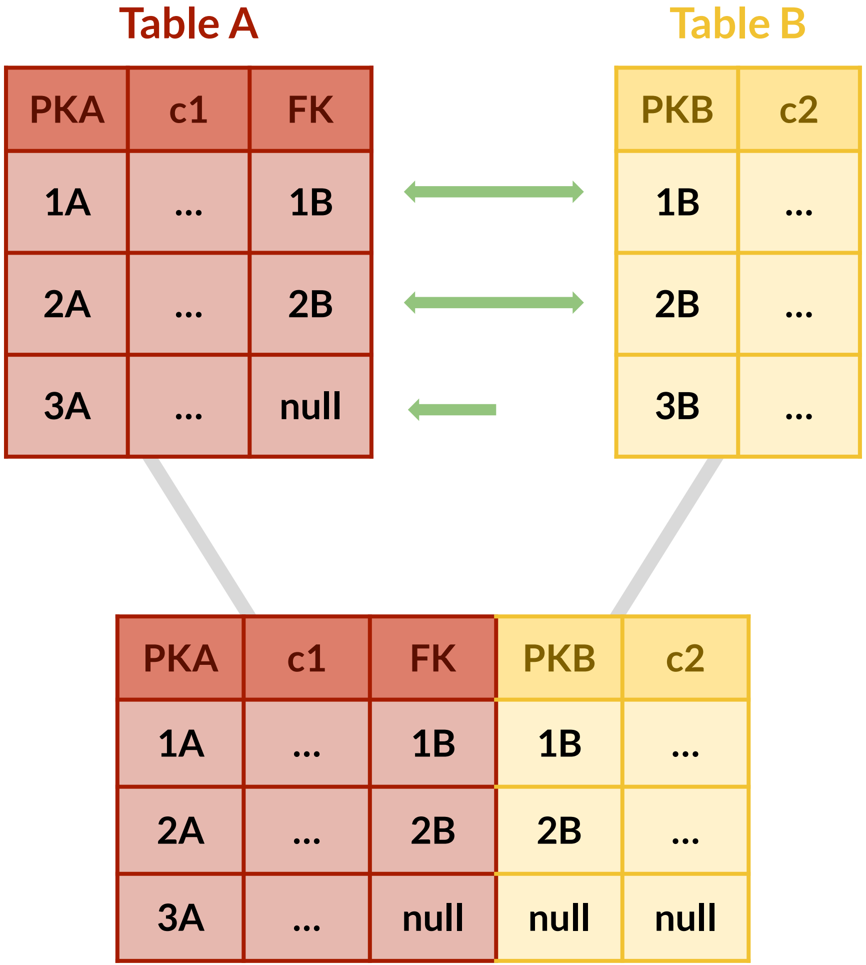


Table A

PKA	c1	FK
1A	...	1B
2A	...	2B
3A	...	null

Table B

PKB	c2
1B	...
2B	...
3B	...

PKA	c1	FK	PKB	c2
1A	...	1B	1B	...
2A	...	2B	2B	...
3A	...	null	null	null

```
SELECT
    PKA, c1, FK, PKB, c2, c3
FROM
    Table_A
LEFT JOIN
    Table_B
ON
    Table_A.FK = Table_B.PKB
```

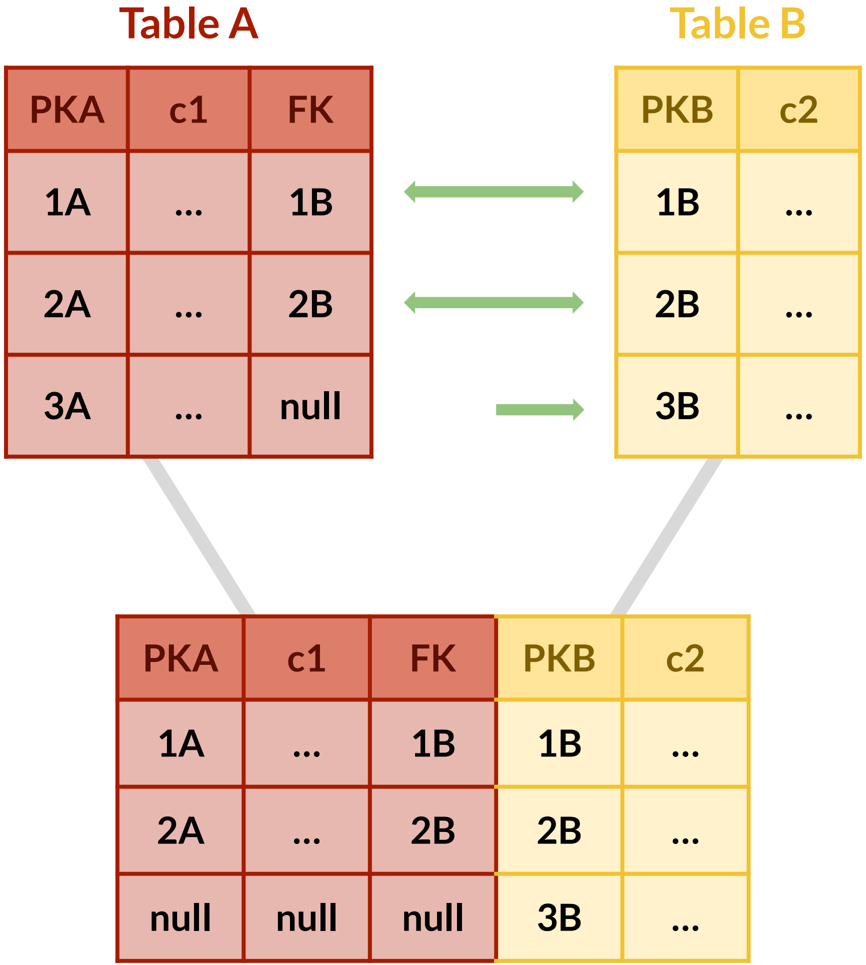


Table A

PKA	c1	FK
1A	…	1B
2A	…	2B
3A	…	null

Table B

PKB	c2
1B	…
2B	…
3B	…

PKA	c1	FK	PKB	c2
1A	…	1B	1B	…
2A	…	2B	2B	…
null	null	null	3B	…

RIGHT JOIN

Combine rows from Table A and Table B that contain a **matching value** that is ***present in the right table***. The right table will be the second table listed in the query (Table B).

```
SELECT
    PKA, c1, FK, PKB, c2, c3
FROM
    Table_A
RIGHT JOIN
    Table_B
ON
    Table_A.FK = Table_B.PKB
```

FULL OUTER JOIN

Combine rows from Table A and Table B so both **matched and unmatched values** are selected. Unmatched rows will have nulls for columns that don't have values.

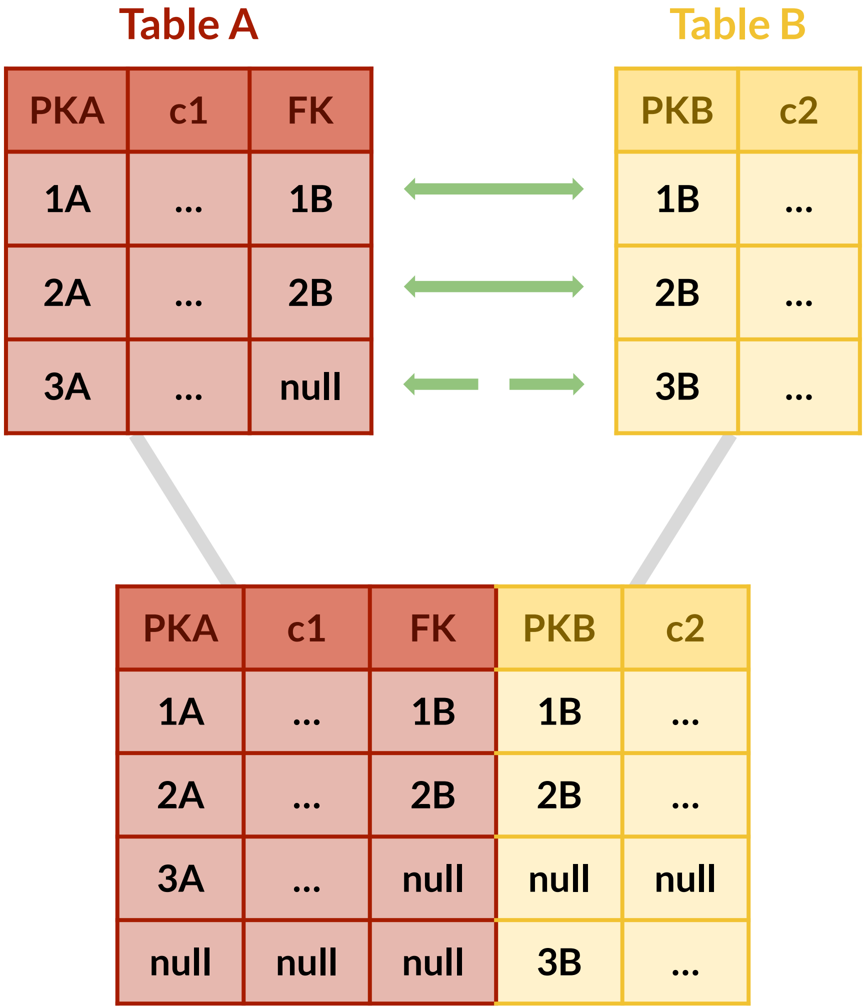


Table A

PKA	c1	FK
1A	...	1B
2A	...	2B
3A	...	null

Table B

PKB	c2
1B	...
2B	...
3B	...

PKA	c1	FK	PKB	c2
1A	...	1B	1B	...
2A	...	2B	2B	...
3A	...	null	null	null
null	null	null	3B	...

```
SELECT
     PKA, c1, FK, PKB, c2, c3
FROM
     Table_A
FULL JOIN
     Table_B
ON
     Table_A.FK = Table_B.PKB
```

*** Note no FULL JOIN keyword in MySQL, need to use UNION on a right join and left join.*

Typical SQL Full Outer Join

```
SELECT
    PKA, c1, FK, PKB, c2, c3
FROM
    Table_A
FULL JOIN
    Table_B
ON
    Table_A.FK = Table_B.PKB
```

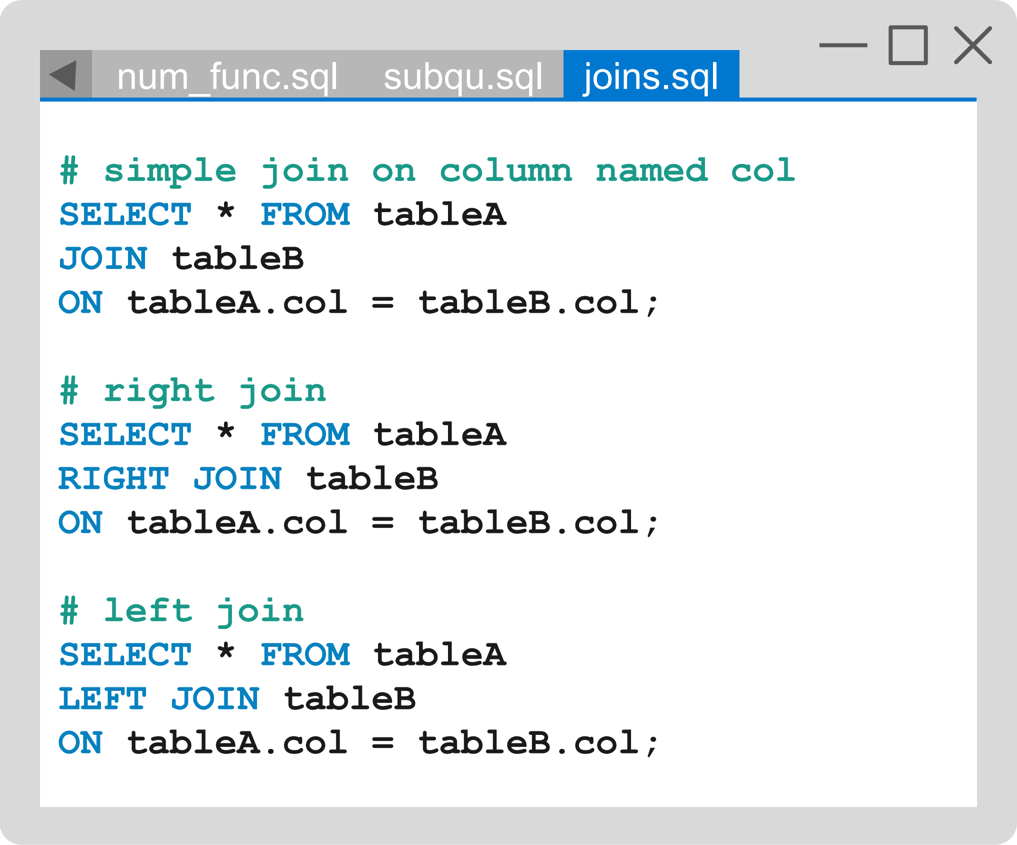
- ➔ Typical full join is written like above
- ➔ MySQL doesn’t have `FULL JOIN` command
- ➔ To mimic this command, create `UNION` between `LEFT JOIN` and `RIGHT JOIN`

MySQL Full Outer Join

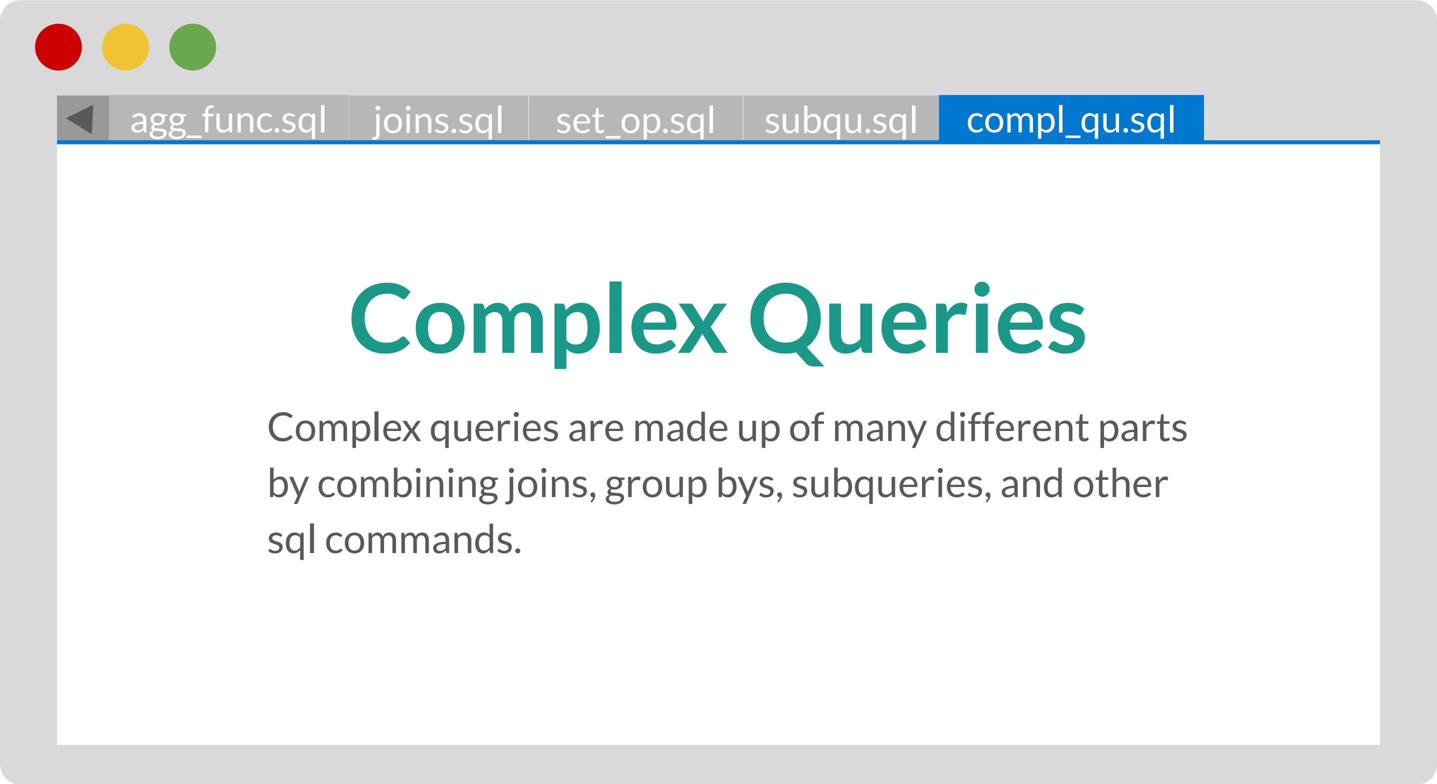
```
SELECT
    PKA, c1, FK, PKB, c2, c3
FROM
    Table_A
LEFT JOIN
    Table_B
ON
    Table_A.FK = Table_B.PKB

UNION

SELECT
    PKA, c1, FK, PKB, c2, c3
FROM
    Table_A
RIGHT JOIN
    Table_B
ON
    Table_A.FK = Table_B.PKB
```

Joins

- ➔ Joins bring data from two tables together
- ➔ There is no full outer join keyword so we must do a double join to do this



```
# simple join on column named col
SELECT * FROM tableA
JOIN tableB
ON tableA.col = tableB.col;

# right join
SELECT * FROM tableA
RIGHT JOIN tableB
ON tableA.col = tableB.col;

# left join
SELECT * FROM tableA
LEFT JOIN tableB
ON tableA.col = tableB.col;
```

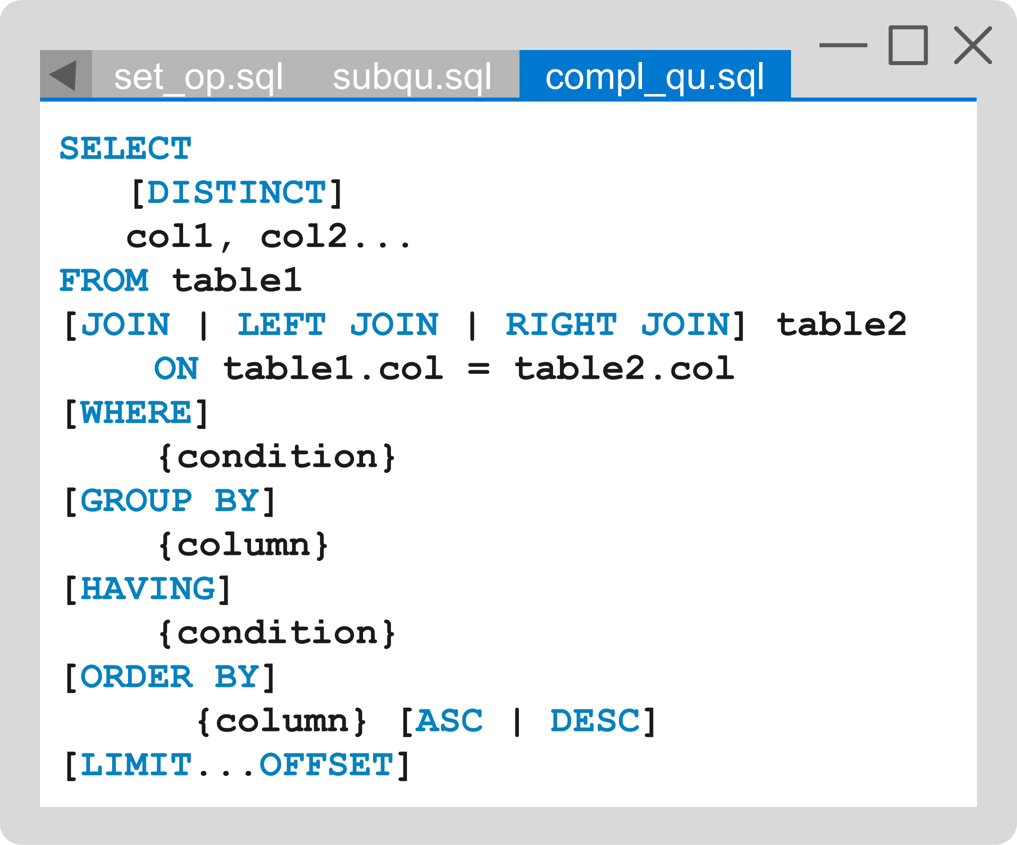


Complex Queries

Complex queries are made up of many different parts by combining joins, group bys, subqueries, and other sql commands.

Complex Queries

- ➔ **Complex queries** are select statements that contain many conditions for selecting specific data
- ➔ Can be made up of many joins, group bys, subqueries, etc.



```
SELECT
    [DISTINCT]
    col1, col2...
FROM table1
[JOIN | LEFT JOIN | RIGHT JOIN] table2
     ON table1.col = table2.col
[WHERE]
     {condition}
[GROUP BY]
     {column}
[HAVING]
     {condition}
[ORDER BY]
       {column} [ASC | DESC]
[LIMIT...OFFSET]
```

Save Point & Check In

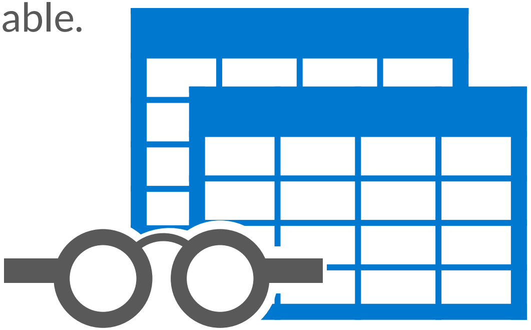
Sakila Joins & Subqueries Exercise

1. List the title of a film, rating and category, don't include any films that have a rating of PG.
2. List the name of every actor and the amount of films they are in.
3. The inventory table stores the films in inventory. Find the title of the film, the amount of that film in inventory. Order it from highest to lowest inventory count.
4. Find the average replacement cost for films that are PG-13. Using subqueries, find the films that have a replacement cost above the average for PG-13 films..

◀ num_func.sql | date_func.sql | subqu.sql | joins.sql | views.sql

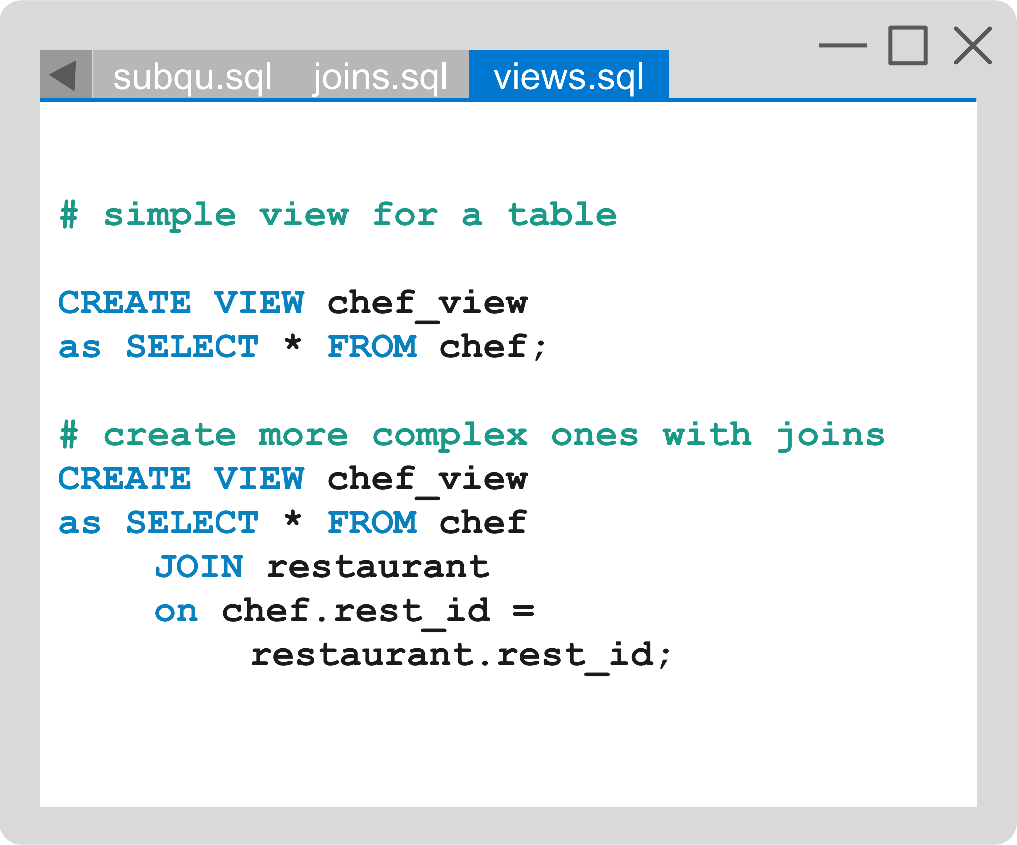
Views

Views display virtual tables. Their contents are based on a base table.

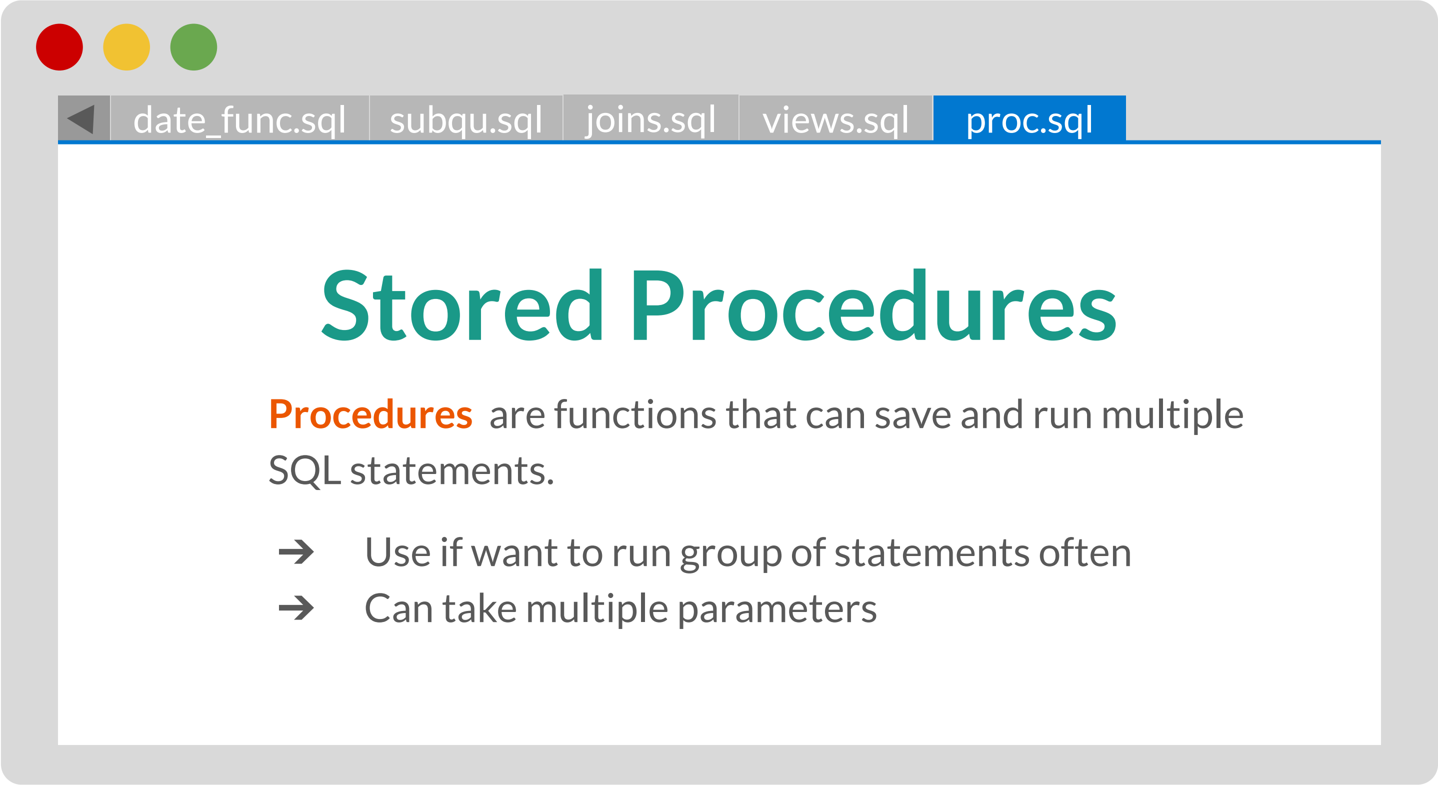


Views

- ➔ They show a “view” of a table
- ➔ Don’t store actual data
- ➔ Just store definition of a base table



```
# simple view for a table

CREATE VIEW chef_view
as SELECT * FROM chef;

# create more complex ones with joins
CREATE VIEW chef_view
as SELECT * FROM chef
     JOIN restaurant
     on chef.rest_id =
          restaurant.rest_id;
```



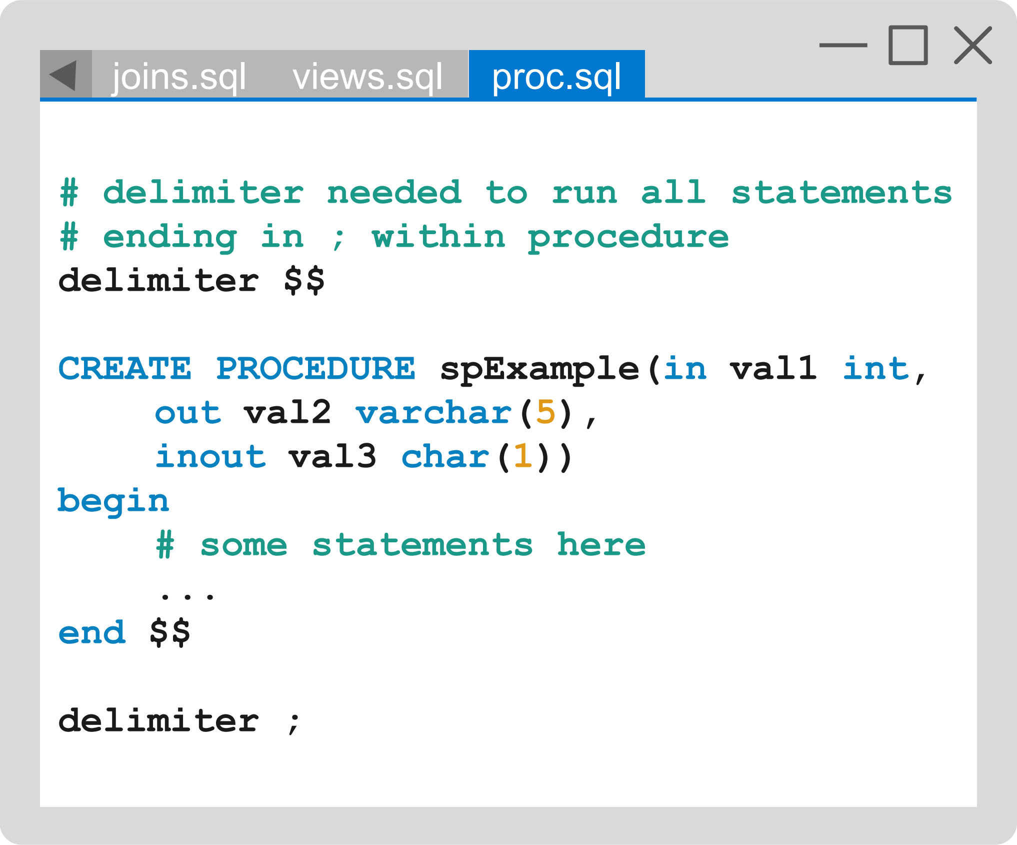
Stored Procedures

Procedures are functions that can save and run multiple SQL statements.

- Use if want to run group of statements often
- Can take multiple parameters

Procedures

- ➔ Procedures can take in three types of parameters:
    - **in** - input, can’t be accessed again once passed
    - **out** - used to pass a value outside the procedure
    - **inout** - combination of in and out



```sql
# delimiter needed to run all statements
# ending in ; within procedure
delimiter $$

CREATE PROCEDURE spExample(in val1 int,
    out val2 varchar(5),
    inout val3 char(1))
begin
    # some statements here
    ...
end $$

delimiter ;
```

◀ subqu.sql joins.sql views.sql proc.sql cursor.sql st_func.sql

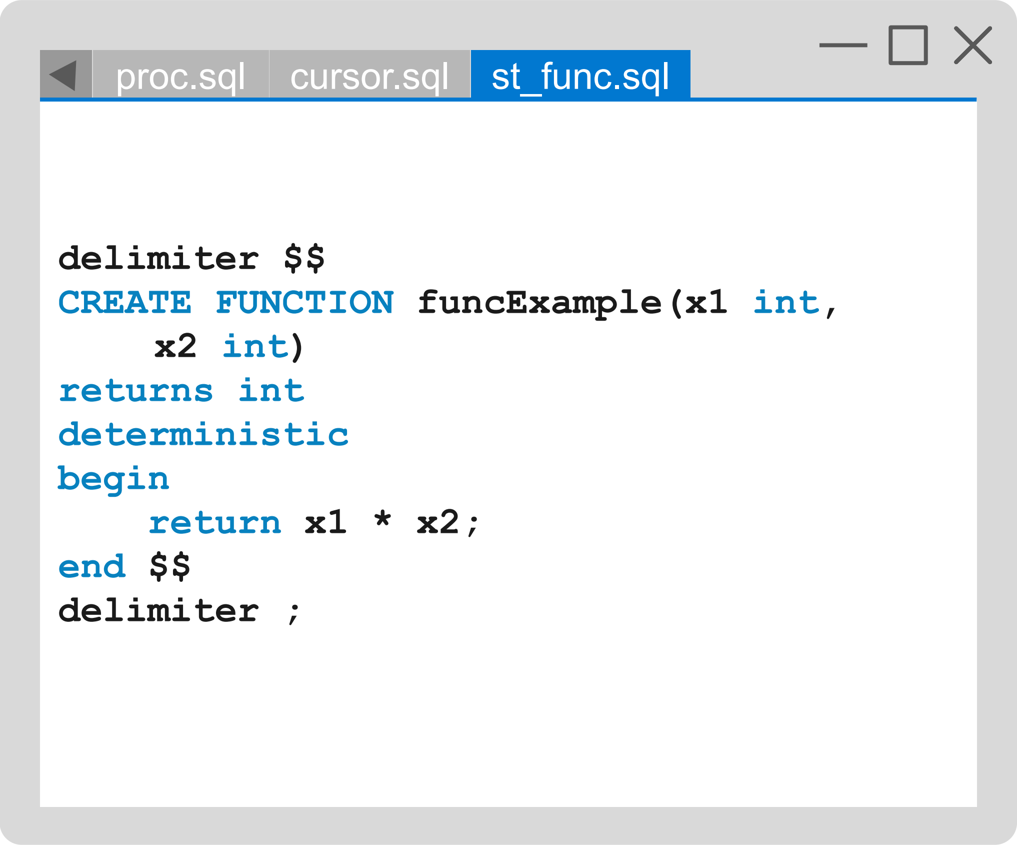
Stored Functions

Functions can be created by users and stored to specific databases.

- ➔ Can take in parameters
- ➔ Return a value

Stored Functions

- ➔ Unlike procedures, functions can actually return values directly



```
delimiter $$
CREATE FUNCTION funcExample(x1 int,
    x2 int)
returns int
deterministic
begin
    return x1 * x2;
end $$
delimiter ;
```

Save Point & Check In

FIN