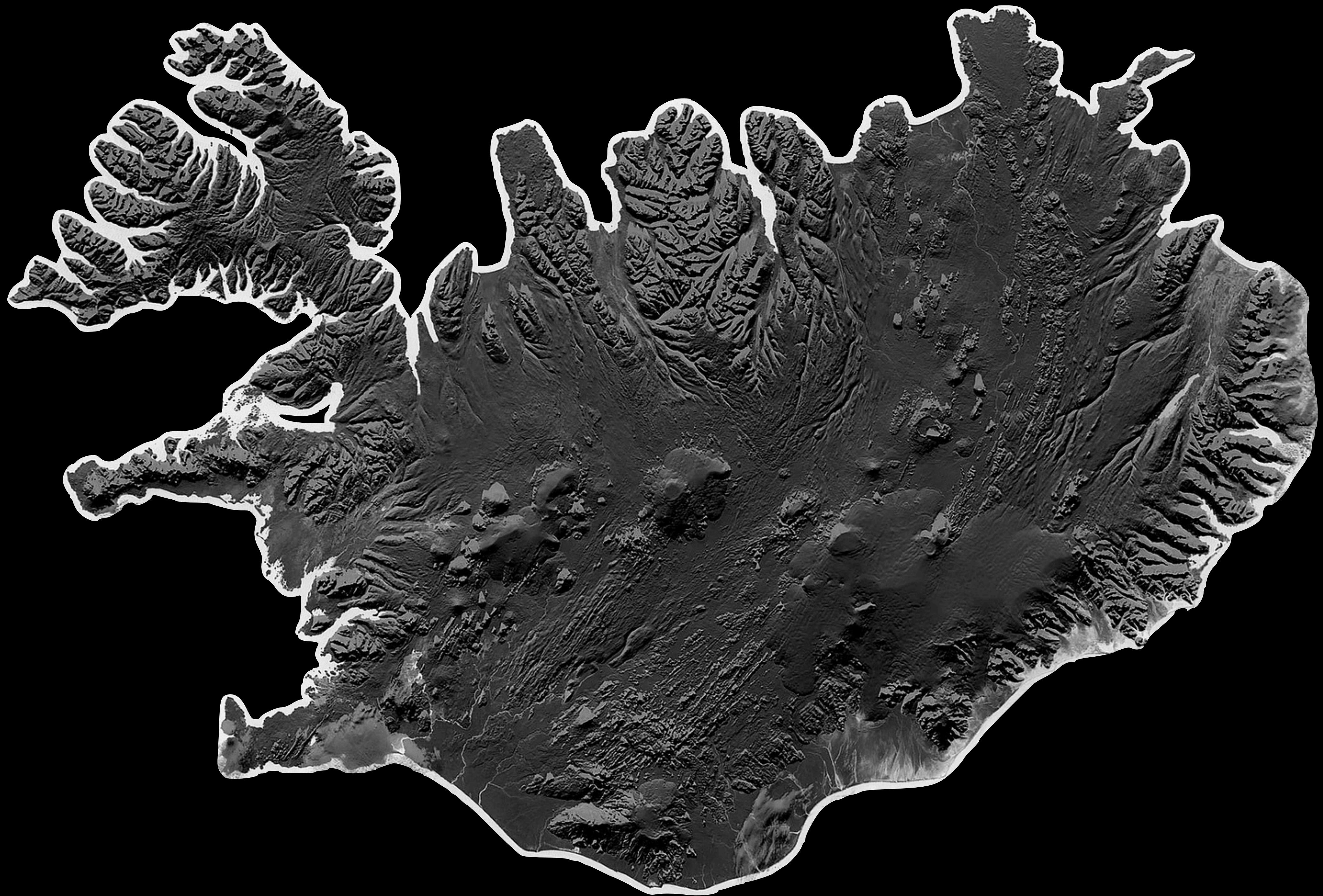




Brighton Data Forum

connect grow learn



from zero to query

a sql primer

oskar 2025-07-04

a note on sqlite

- small (<2mb)
- open source
- self-contained
- fast
- complete
- in-memory
- cross-platform
- ubiquitous
- to start it up: > sqlite3



sqlite commands



- these are **not SQL** commands
- they start with a ‘.’
- they operate on the application, not the data
- examples:
 - `.quit`
 - `.open <path-to-database>`
 - `.show`
 - `.help`
 - `.cd <directory>`
 - `.shell CMD ARGs...`

sqlite commands

.open data/sqlite-sakila.db

.tables

.header ON

.mode qbox

.tables

```
sqlite> .tables
actor
address
category
city
country
customer
customer_list
sqlite> ■
```

	film	payment
	film_actor	rental
	film_category	sales_by_film_category
	film_list	sales_by_store
	film_text	staff
	inventory	staff_list
	language	store

sql - a fundamental data tool

- database management
- data pipeline engineering
- data modeling
- data designing
- **big** data (parallel, distributed)
- data querying
- data analytics

data definition	data management	data querying	data control	transaction control
to operate on entire tables	to operate on table values, rows, columns	to fetch data from tables	to control access to schemas + tables	for transactional atomicity, dev
CREATE	INSERT	SELECT	GRANT	COMMIT
DROP	UPDATE		REVOKE	ROLLBACK
ALTER	DELETE			SAVE POINT
TRUNCATE				

data definition	data management	data querying	data control	transaction control
to operate on entire tables	to operate on table cells, rows, columns	to fetch data from tables	to control access to schemas + tables	for transactional atomicity, dev
CREATE	INSERT	SELECT	GRANT	COMMIT
DROP	UPDATE		REVOKE	ROLLBACK
ALTER	DELETE			SAVE POINT
TRUNCATE				

sql SELECT statements

- run on a database
- operate on data tables
- output a table
- start with **SELECT ... clause**
- contain a **FROM ... clause**
- end with a ‘;’
- can have comments with ‘-- a comment’
- example:
SELECT name FROM category; -- film categories

the sakila training data

- classic, fictional dataset
- dvd rental company
- normalised (no repetition)
- 20 relational tables:
 - stores
 - inventory
 - films
 - film casting
 - actors
 - film ratings

The screenshot shows the MySQL Documentation website with a red border around the main content area. The top navigation bar includes the MySQL logo, a search bar, and links for Contact MySQL, Login, and Register. Below the navigation is a secondary menu with links for MySQL Server, MySQL Enterprise, Workbench, InnoDB Cluster, MySQL NDB Cluster, Connectors, and More. The main content area is titled "Sakila Sample Database". On the left, there's a sidebar with a "Documentation Home" link and a "Sakila Sample Database" section containing a table of contents with numbered links from 1 to 11. A "Download this Manual" button is also present. The right side of the content area contains the actual documentation text, which describes the Sakila sample database installation, structure, usage, and history. It includes links for legal notices, forums, and a note about document generation.

The world's most popular open source database

Contact MySQL | Login | Register

MySQL.COM DOWNLOADS DOCUMENTATION DEVELOPER ZONE

MySQL Server MySQL Enterprise Workbench InnoDB Cluster MySQL NDB Cluster Connectors More

Documentation Home

Sakila Sample Database

Table of Contents

1 Preface and Legal Notices
2 Introduction
3 History
4 Installation
5 Structure
6 Usage Examples
7 Known Issues
8 Acknowledgments
9 License for the Sakila Sample Database
10 Note for Authors
11 Sakila Change History

Download this Manual

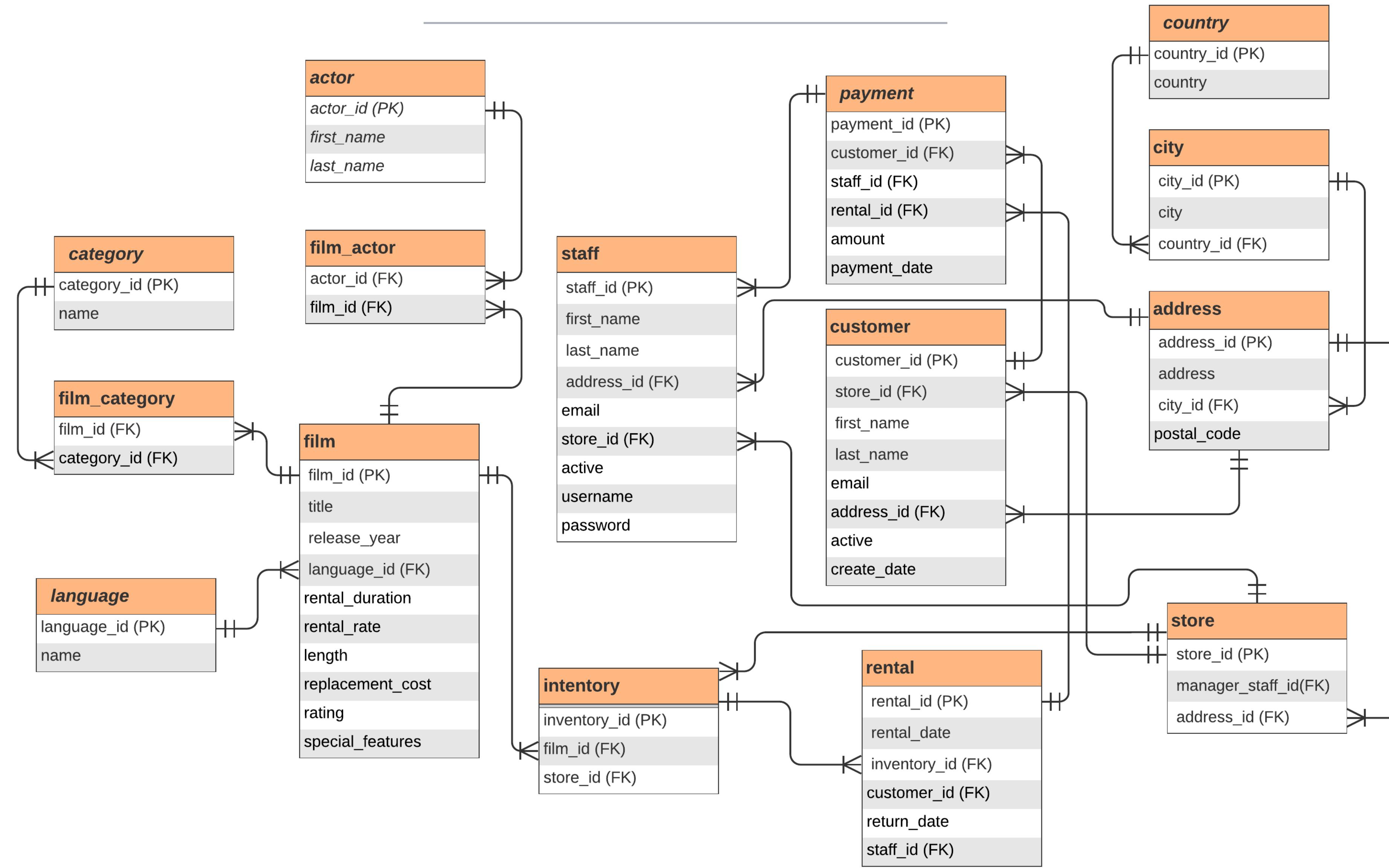
This document describes Sakila sample database installation, structure, usage, and history.

For legal information, see the [Legal Notices](#).

For help with using MySQL, please visit the [MySQL Forums](#), where you can discuss your issues with other MySQL users.

Document generated on: 2023-02-09 (revision: 74968)

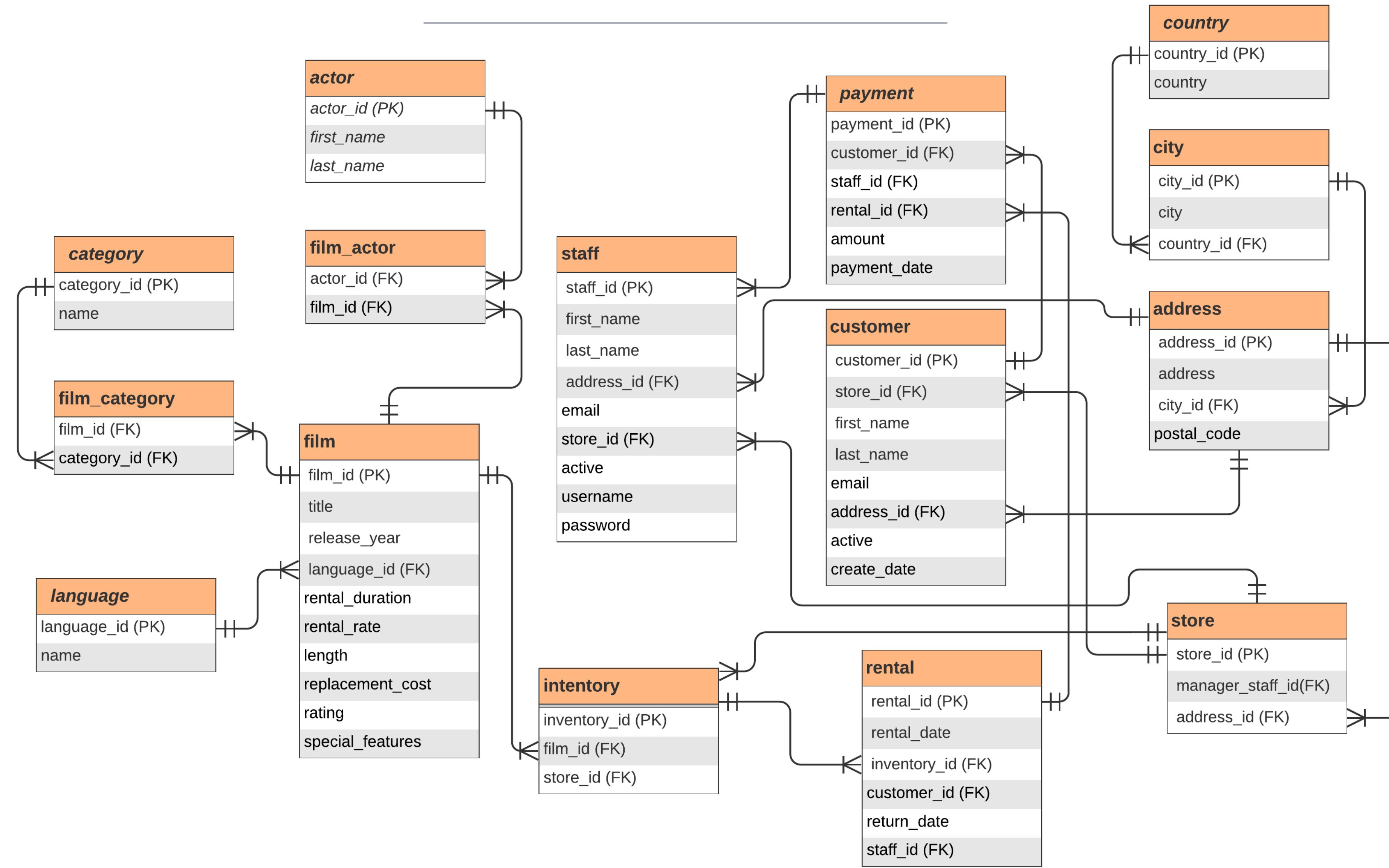
SQLite3 Sakila Sample Database ERD



today's objective:

“which **top 10** actors were *rented* out
the greatest number of times, counting
only ‘R’ rated films made in **2006?**”

SQLite3 Sakila Sample Database ERD



today's plan:

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table}  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {condition}  
GROUP BY {columns}  
HAVING {condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```

what do the tables contain?

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

- `SELECT {columns} FROM {table};
INNER JOIN {table_2} ON {col1}={col2}
WHERE {condition}
GROUP BY {columns}
HAVING {condition}
ORDER BY {columns}
LIMIT {num}`

SELECT {columns} FROM {table};

- `SELECT * FROM staff;`
-- returns all columns and all rows from the staff table
- `SELECT title, rating FROM film;`
-- returns title and rating (in order) from the table film
- `SELECT c.first_name AS customer_name FROM customer c;`
-- sets an alias for table customer, renames column to 'customer_name'
- `SELECT title, replacement_cost/rental_rate AS break_even_count FROM film;`
-- returns the number of rentals a film needs to break even
- `SELECT DISTINCT a.last_name FROM actor a;`
-- returns all the first names in the actor table, with no duplicates

exercise

- show all the columns of the **category** table
- rename the **name** column to **category_name** in the output

SELECT {aggregate function} FROM {table};

- SELECT COUNT(*) AS num_records FROM actor;
-- returns the number of rows in table actor, names the output ‘num_records’
- SELECT COUNT(DISTINCT rating) FROM film;
-- returns a count of distinct values in the rating column
- SELECT AVG(replacement_cost) AS avg_cost FROM film;
-- returns the average replacement cost of a film
- SELECT AVG(rental_rate) AS average_rental_cost FROM film;
-- returns the average rate of rental from film table
- SELECT MAX(rental_rate) AS highest_rental_rate FROM film;
-- returns the most expensive rental_rate from film
- SELECT MIN(length) AS shortest_length FROM film;
-- returns the length of the shortest film

exercises

- what is the average number of times that a **film** needs to be rented out to break even?
- what is the maximum number of times that a **film** needs to be rented out to break even?

that's too many rows!

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table}  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {condition}  
GROUP BY {columns}  
HAVING {condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```

SELECT ... FROM ... LIMIT ...;

- `SELECT * FROM film LIMIT 5;`
-- returns all columns of 5 unspecified rows from table film
- `SELECT * FROM category LIMIT 5;`
-- returns 5 unspecified rows of all columns from table category
- `SELECT title, release_year FROM film LIMIT 15;`
-- returns 15 unspecified rows of two columns from table film
- `SELECT r.rental_id, r.rental_date FROM rental r LIMIT 10;`
-- returns rental id and date of rental for 10 unspecified rows
- `SELECT first_name || ' ' || last_name AS fullname FROM actor LIMIT 10;`
-- returns the full names of 10 unspecified actors

exercises

- show the first 5 records of any table, using a **LIMIT** clause
- output the full name of 5 **customers**, using **||** to paste strings

but i only want the most extreme rows!

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table}  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {condition}  
GROUP BY {columns}  
HAVING {condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```

`SELECT ... FROM ... ORDER BY ... LIMIT ...;`

- `SELECT * FROM payment ORDER BY payment_date LIMIT 7;`
-- return the earliest 7 payments in the payment table
- `SELECT * FROM payment ORDER BY payment_date DESC LIMIT 7;`
-- return the latest 7 payments in the payment table
- `SELECT * FROM payment ORDER BY amount DESC LIMIT 5;;`
-- return only the top 5 highest payment amounts from the payment table

but i only want specific rows!

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table}  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {condition}  
GROUP BY {columns}  
HAVING {condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```

comparison operators

comparison syntax	meaning
{column} = {expression}	column value is equal to expression value
{column} <> {expression}	column value is not equal to expression value
{column} != {expression}	column value is not equal to expression value
{column} < {expression}	column value is less than expression value
{column} <= {expression}	column value is less than or equal to expression value
{column} > {expression}	column value is greater than expression value
{column} >= {expression}	column value is greater than or equal to expression value
{column} IN ({exp1,exp2,...})	column value is one of exp1, exp2, ...
{column} LIKE '%expr%'	(string) column contains substring 'expr'
{column} BETWEEN {exp1} AND {exp2}	{exp1} <= column value <= {exp2}

`SELECT {column} FROM {table} WHERE {cond}`

- `SELECT * FROM actor WHERE LENGTH(last_name) = 3;`
-- returns only records of actors whose last name is three characters
- `SELECT title AS name FROM film f WHERE rating <>'R' AND;`
-- returns only rows where the value in column1 is not {expression}
- `SELECT title AS film_name, rental_rate FROM film WHERE rental_rate<=1.0;`
-- returns titles of films whose rental price is at most £1
- `SELECT first_name FROM staff WHERE store_id=2;`
-- returns the first names of staff at store with id 2

exercises

- show the `first_names` of *inactive customers* (`active = '0'`)
- how many `payments` have `amounts` greater than \$10.00?

comparison operators

comparison syntax	meaning
{column} = {expression}	column value is equal to expression value
{column} <> {expression}	column value is not equal to expression value
{column} != {expression}	column value is not equal to expression value
{column} < {expression}	column value is less than expression value
{column} <= {expression}	column value is less than or equal to expression value
{column} > {expression}	column value is greater than expression value
{column} >= {expression}	column value is greater than or equal to expression value
{column} IN ({exp1,exp2,...})	column value is one of exp1, exp2, ...
{column} LIKE '%expr%'	(string) column contains substring 'expr'
{column} BETWEEN {exp1} AND {exp2}	{exp1} <= column value <= {exp2}

SELECT ... FROM ... WHERE ...;

- **SELECT * FROM rental WHERE rental_date BETWEEN '2005-08-16' AND '2005-08-17';**
-- returns only rentals occurring in
- **SELECT * FROM payment WHERE amount IN (7.98, 8.97);**
-- returns info on all payments of a specific amount
- **SELECT * FROM city WHERE city LIKE '0k%';**
-- returns info on all cities whose name begins with '0k'
- **SELECT last_name AS full_name FROM customer WHERE first_name LIKE 'AL%';**
-- returns the last name of all customers whose first name begins with 'AL'

exercises

- what `city` names begin with a 'Q'?
- which `actors`' `first_names` end with 'K'?

how can i aggregate groups of rows into a single row?

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table}  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {a_condition}  
GROUP BY {columns}  
HAVING {a_condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```

```
SELECT {col} FROM {tab} GROUP BY {col};
```

- `SELECT city_id, COUNT(*) AS num_address FROM address GROUP BY city_id;`
-- return number of addresses in each city id in table address
- `SELECT rating, AVG(length) AS avg_len FROM film GROUP BY rating ORDER BY avg_len;`
-- returns the average length of a movie in each rating category
- `SELECT country_id, COUNT(*) AS num_cities
FROM city
GROUP BY country_id
ORDER BY num_cities DESC
LIMIT 5;`
-- return top 5 country ids, by number of cities assigned to each

how do i report only some aggregated groups?

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table}  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {condition}  
GROUP BY {column}  
HAVING {condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```

SELECT ... FROM ... GROUP BY ... HAVING ...;

- **SELECT col1, COUNT(*) AS num FROM table GROUP BY col1 HAVING num>9;**
-- count instances of each value of col1, but only output rows with count>9
- **SELECT rating, AVG(length) AS len FROM film GROUP BY rating HAVING len<115;**
-- the film rating categories with average length of film under 115 minutes
- **SELECT actor_id, COUNT(*) AS n FROM film_actor GROUP BY actor_id HAVING n<15;**
-- which actor ids have appeared in fewer than 15 films?

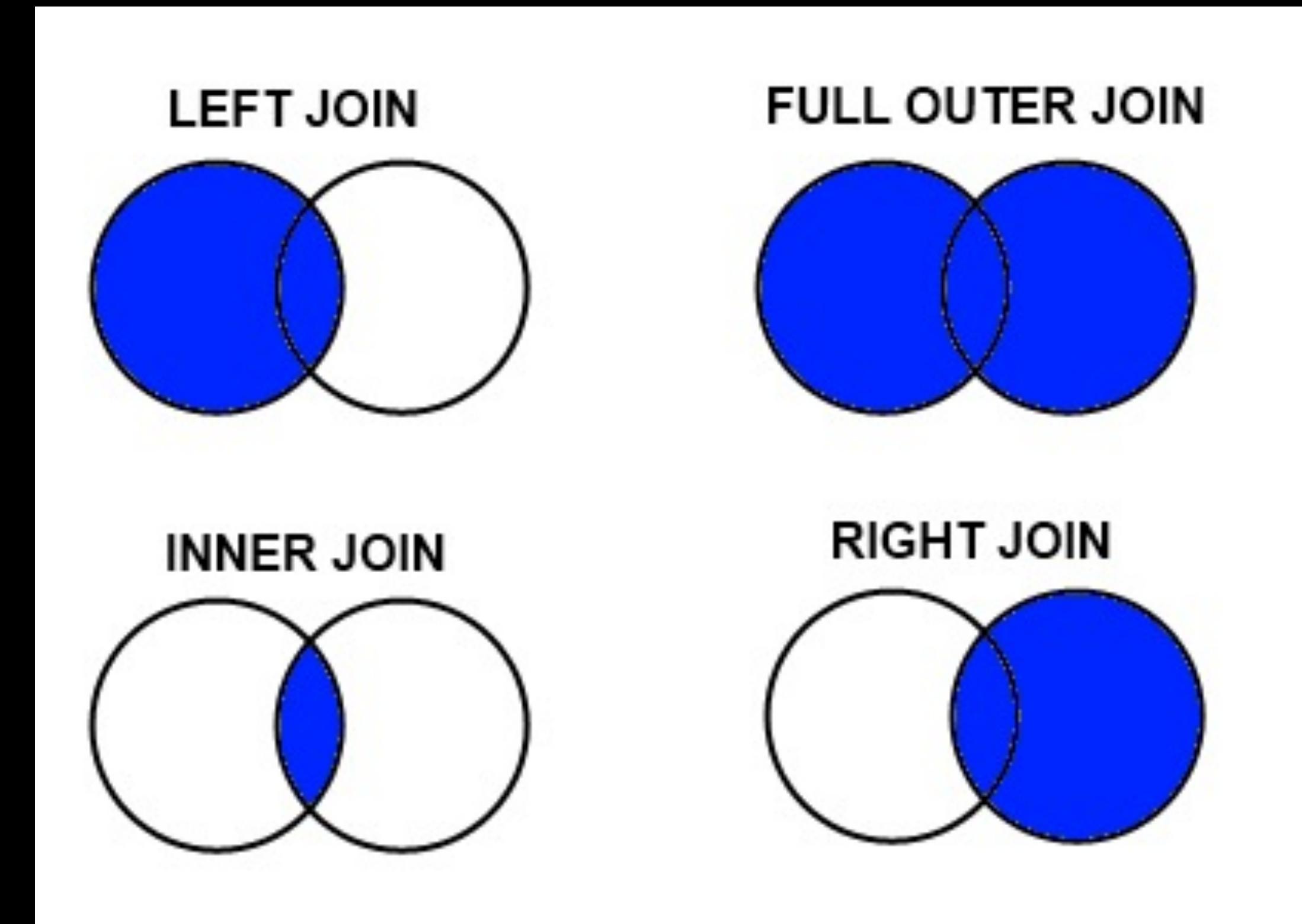
but my information is spread over two tables!

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table};  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {condition}  
GROUP BY {columns}  
HAVING {condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```



@DATAWITHDANNY



JOIN

city

city_id	city	country_id	last_update
1	A Corua (La Corua)	87	2021-03-06 15:51:49
2	Abha	82	2021-03-06 15:51:49
3	Abu Dhabi	101	2021-03-06 15:51:49
4	Acua	60	2021-03-06 15:51:49
5	Adana	97	2021-03-06 15:51:49
6	Addis Abeba	31	2021-03-06 15:51:49
7	Aden	107	2021-03-06 15:51:49
8	Adoni	44	2021-03-06 15:51:49

...

country

country_id	country	last_update
1	Afghanistan	2021-03-06 15:51:49
2	Algeria	2021-03-06 15:51:49
3	American Samoa	2021-03-06 15:51:49
4	Angola	2021-03-06 15:51:49
5	Anguilla	2021-03-06 15:51:49
6	Argentina	2021-03-06 15:51:49
7	Armenia	2021-03-06 15:51:49
8	Australia	2021-03-06 15:51:49
9	Austria	2021-03-06 15:51:49

...

JOIN

city

city_id	city	country_id	last_update
1	A Corua (La Corua)	87	2021-03-06 15:51:49
2	Abha	82	2021-03-06 15:51:49
3	Abu Dhabi	101	2021-03-06 15:51:49
4	Acua	60	2021-03-06 15:51:49
5	Adana	97	2021-03-06 15:51:49
6	Addis Abeba	31	2021-03-06 15:51:49
7	Aden	107	2021-03-06 15:51:49
8	Adoni	44	2021-03-06 15:51:49

...

country

country_id	country	last_update
1	Afghanistan	2021-03-06 15:51:49
2	Algeria	2021-03-06 15:51:49
3	American Samoa	2021-03-06 15:51:49
4	Angola	2021-03-06 15:51:49
5	Anguilla	2021-03-06 15:51:49
6	Argentina	2021-03-06 15:51:49
7	Armenia	2021-03-06 15:51:49
8	Australia	2021-03-06 15:51:49
9	Austria	2021-03-06 15:51:49

...

JOIN

city

city_id	city	country_id	last_update
1	A Corua (La Corua)	87	2021-03-06 15:51:49
2	Abha	82	2021-03-06 15:51:49
3	Abu Dhabi	101	2021-03-06 15:51:49
4	Acua	60	2021-03-06 15:51:49
5	Adana	97	2021-03-06 15:51:49
6	Addis Abeba	31	2021-03-06 15:51:49
7	Aden	107	2021-03-06 15:51:49
8	Adoni	44	2021-03-06 15:51:49

...

Turkey

country

country_id	country	last_update
1	Afghanistan	2021-03-06 15:51:49
2	Algeria	2021-03-06 15:51:49
3	American Samoa	2021-03-06 15:51:49
4	Angola	2021-03-06 15:51:49
5	Anguilla	2021-03-06 15:51:49
6	Argentina	2021-03-06 15:51:49
7	Armenia	2021-03-06 15:51:49
8	Australia	2021-03-06 15:51:49
9	Austria	2021-03-06 15:51:49

...

JOIN

city

city_id	city	country_id	last_update
1	A Corua (La Corua)	87	2021-03-06 15:51:49
2	Abha	82	2021-03-06 15:51:49
3	Abu Dhabi	101	2021-03-06 15:51:49
4	Acua	60	2021-03-06 15:51:49
5	Adana	97	2021-03-06 15:51:49
6	Addis Abeba	31	2021-03-06 15:51:49
7	Aden	107	2021-03-06 15:51:49
8	Adoni	44	2021-03-06 15:51:49

...

country

country_id	country	last_update
1	Afghanistan	2021-03-06 15:51:49
2	Algeria	2021-03-06 15:51:49
3	American Samoa	2021-03-06 15:51:49
4	Angola	2021-03-06 15:51:49
5	Anguilla	2021-03-06 15:51:49
6	Argentina	2021-03-06 15:51:49
7	Armenia	2021-03-06 15:51:49
8	Australia	2021-03-06 15:51:49
9	Austria	2021-03-06 15:51:49

...

Turkey
Yemen

we want this

city-and-country

city_id	city	country
1	?	?
2	?	?
3	?	?
4	?	?
5	?	?
6	?	?
7	?	?
8	?	?

...

we want this

city-and-country

city_id	city	country
1	A Corua (La Corua)	Spain
2	Abha	Saudi Arabia
3	Abu Dhabi	United Arab Emirates
4	Acua	Mexico
5	Adana	Turkey
6	Addis Abeba	Ethiopia
7	Aden	Yemen
8	Adoni	India

...

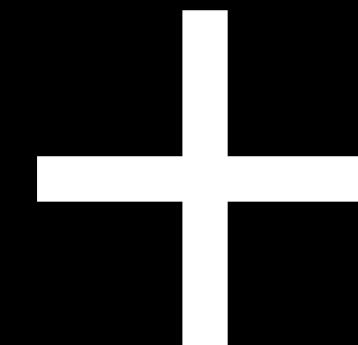
so we add a JOIN to the WHERE clause

city

city_id	city	country_id	last_update
1	A Corua (La Corua)	87	2021-03-06 15:51:49
2	Abha	82	2021-03-06 15:51:49
3	Abu Dhabi	101	2021-03-06 15:51:49
4	Acua	60	2021-03-06 15:51:49
5	Adana	97	2021-03-06 15:51:49
6	Addis Abeba	31	2021-03-06 15:51:49
7	Aden	107	2021-03-06 15:51:49
8	Adoni	44	2021-03-06 15:51:49

country

country_id	country	last_update
1	Afghanistan	2021-03-06 15:51:49
2	Algeria	2021-03-06 15:51:49
3	American Samoa	2021-03-06 15:51:49
4	Angola	2021-03-06 15:51:49
5	Anguilla	2021-03-06 15:51:49
6	Argentina	2021-03-06 15:51:49
7	Armenia	2021-03-06 15:51:49
8	Australia	2021-03-06 15:51:49
9	Austria	2021-03-06 15:51:49



...

...

SELECT

 city.city_id, city.city, country.country

FROM

city

INNER JOIN country ON city.country_id=country.country_id

;

so we add a JOIN to the WHERE clause

city-and-country

city_id	city	country
1	A Corua (La Corua)	Spain
2	Abha	Saudi Arabia
3	Abu Dhabi	United Arab Emirates
4	Acua	Mexico
5	Adana	Turkey
6	Addis Abeba	Ethiopia
7	Aden	Yemen
8	Adoni	India

...

```
SELECT
    city.city_id, city.city, country.country
FROM
    city
INNER JOIN country ON city.country_id=country.country_id
;
```

```
SELECT ... FROM a INNER JOIN b ON a.key=b.key;
```

- ```
SELECT a.city, b.country
FROM
 city a
 INNER JOIN country b ON a.country_id=b.country_id
LIMIT 10
; -- output a table with city-country names
```
- ```
SELECT f.title, f.length, l.name
FROM film f
    INNER JOIN language l ON f.language_id=l.language_id
WHERE f.rating='R'
LIMIT 10
; -- output a sample of films and the name of the language it is in
```

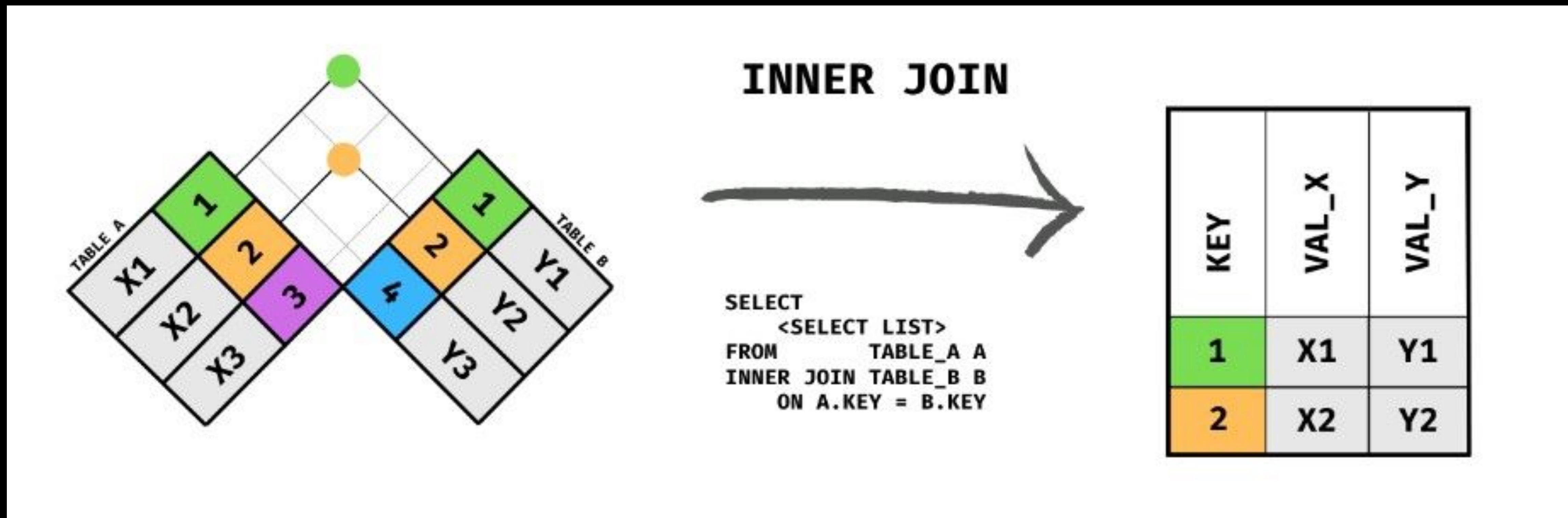
```
SELECT ... FROM a INNER JOIN b ON a.key=b.key;
```

```
SELECT
    f.title AS film_title,
    c.name AS category
FROM film f
    INNER JOIN film_category fc ON f.film_id=fc.film_id
    INNER JOIN category c ON fc.category_id=c.category_id
WHERE f.rating IN ('G', 'PG') AND f.length BETWEEN 85 AND 90
; 
```

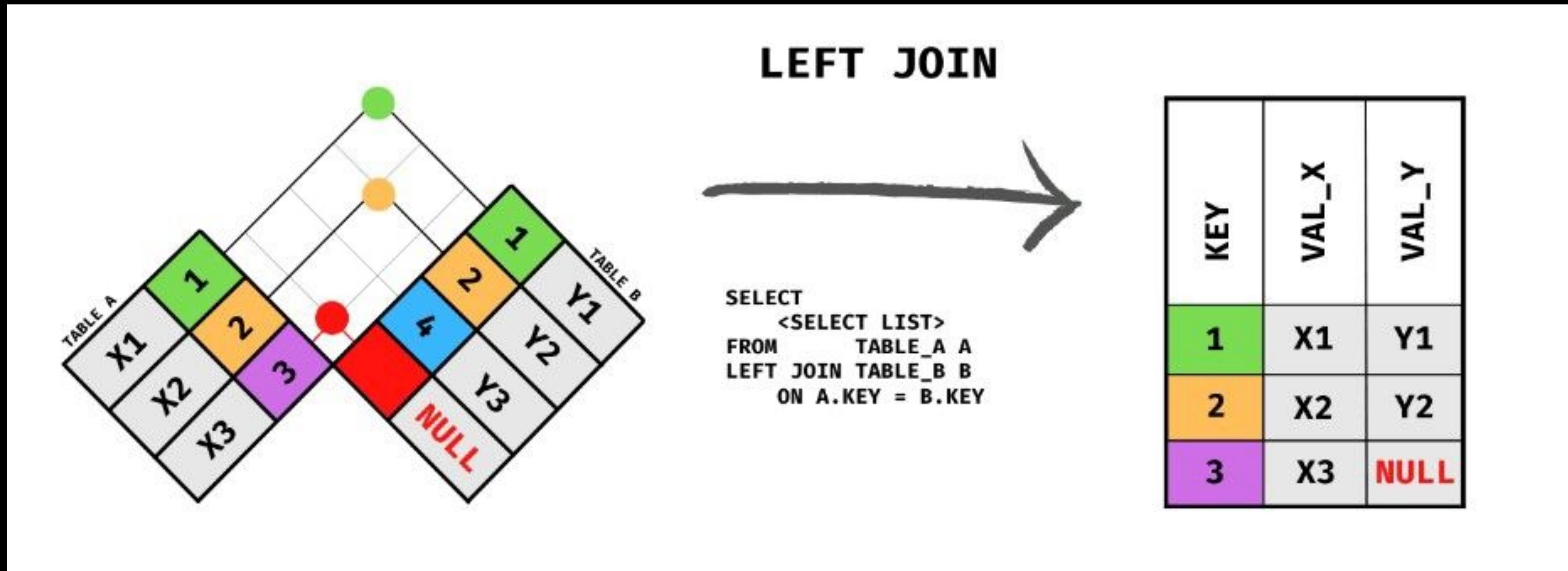
exercises

- what countries are the cities, whose name starts with 'Q', in?

FROM a INNER JOIN b



LEFT OUTER JOIN





@DATAWITHDANNY

Table 1

A
B
C

Table 2

A
B
D

INNER JOIN: show all matching records in both tables.

A	A
B	B

LEFT JOIN: show all records from left table, and any matching records from right table.

A	A
B	B
C	

RIGHT JOIN: show all records from right table, and any matching records from left table.

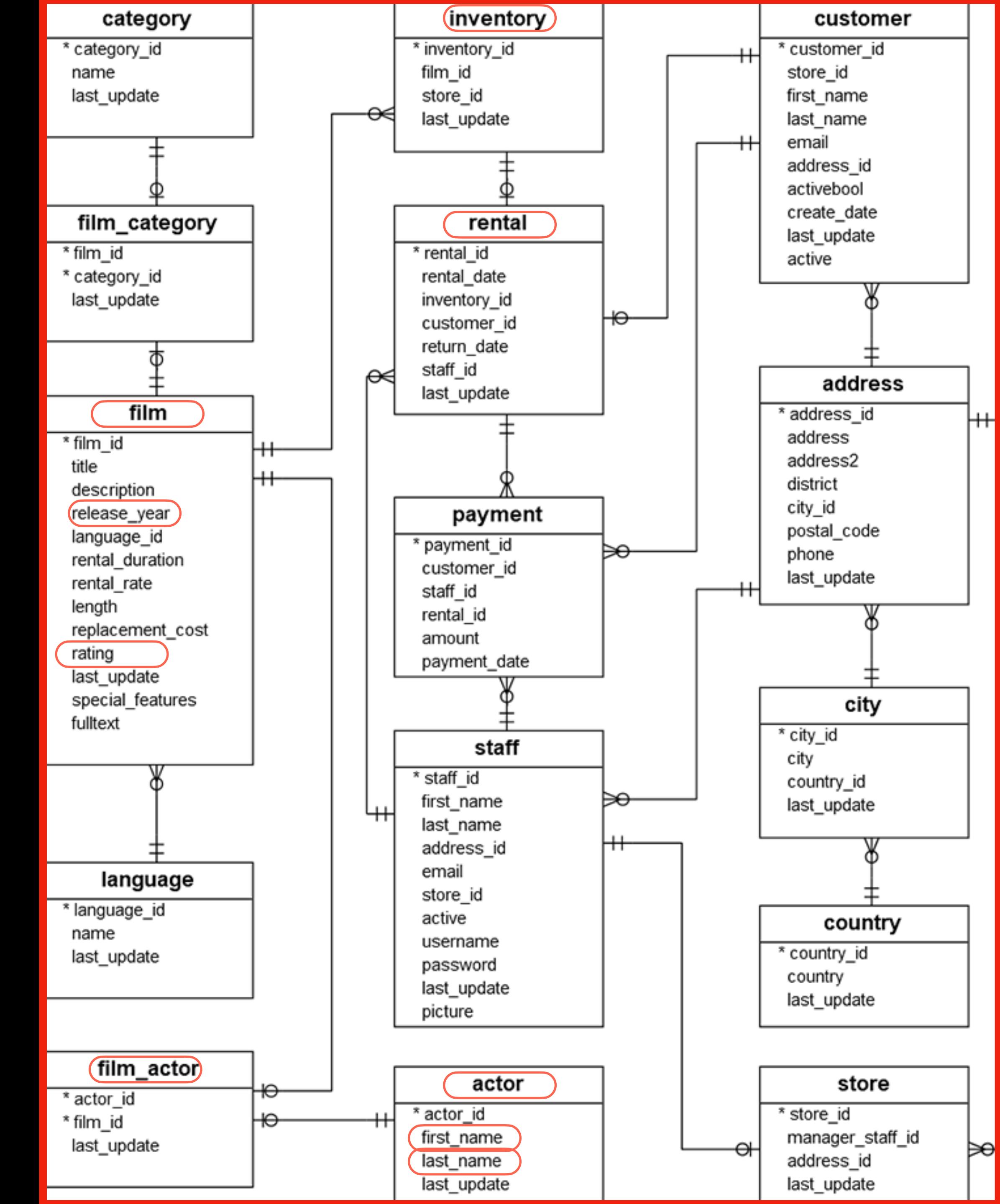
A	A
B	B
	D

FULL JOIN: show all records from both tables, whether there is a match or not.

A	A
B	B
C	
	D

today's objective:

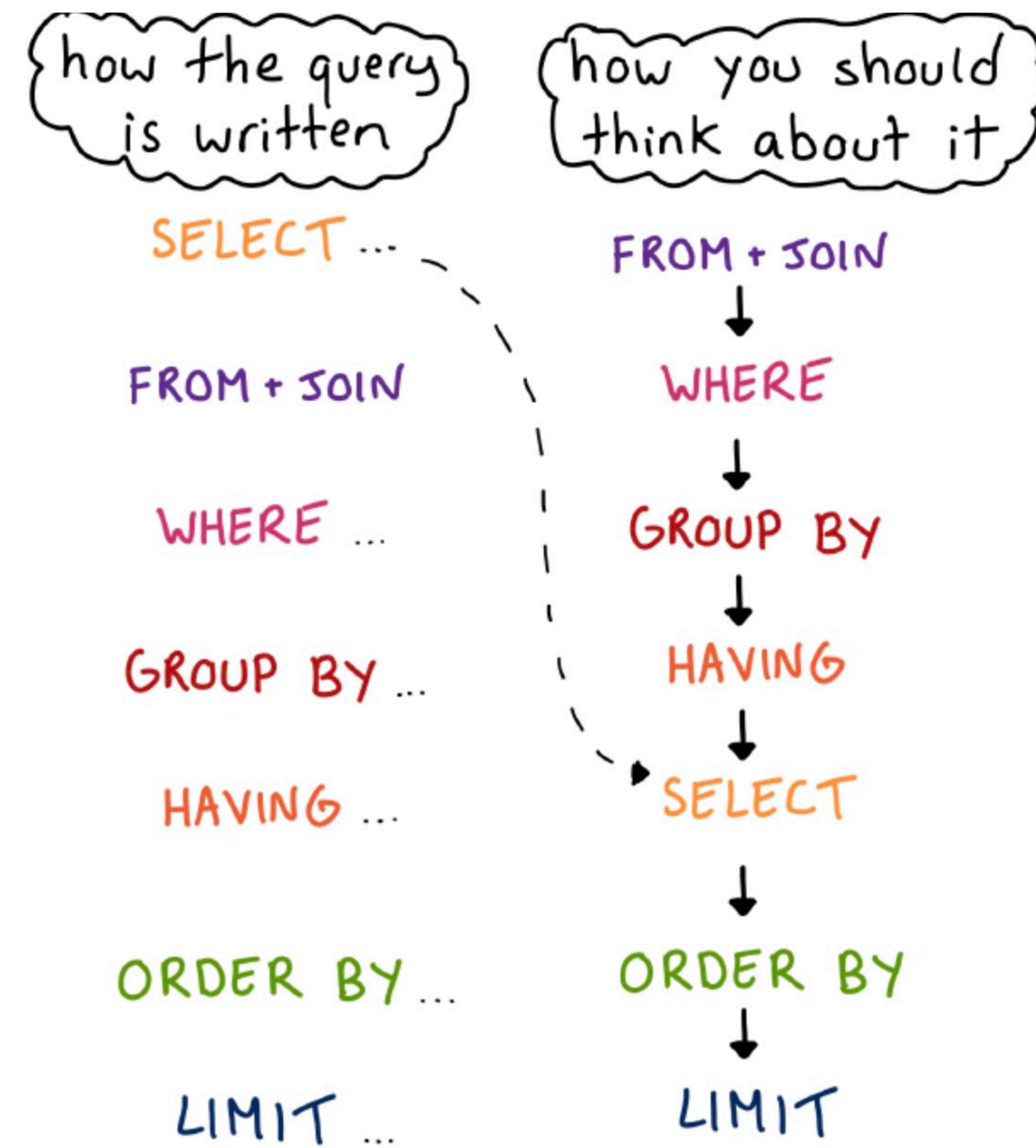
“which **top 10** actors were
rented out the greatest number
of times, counting only ‘R’ rated
films made in **2006**? ”



how do i combine the components of a SELECT?

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

```
- SELECT {columns} FROM {table};  
INNER JOIN {table_2} ON {col1}={col2}  
WHERE {condition}  
GROUP BY {columns}  
HAVING {condition}  
ORDER BY {columns}  
LIMIT {num}  
;
```



your turn! compose a query to answer:

“which top 10 actors were rented out
the greatest number of times, counting
only ‘R’ rated films made in 2006?”

hint: structure of the solution

```
SELECT
    {}          AS actor_name,
    COUNT({}) AS num_rentals
FROM {table1}
    INNER JOIN {table2} ON {join-condition}
    INNER JOIN {table3} ON {join-condition}
    INNER JOIN {table4} ON {join-condition}
    INNER JOIN {table5} ON {join-condition}
WHERE {row condition1}
    AND {row condition2}
GROUP BY {columns}
ORDER BY {column} DESC
LIMIT {num}
;
```



ORMs	data types	ORDER BY	CREATE TABLE	foreign keys	
GROUP BY		SELECT / INSERT / UPDATE / DELETE			
LIMIT and OFFSET		NUL	indexes	JOIN	
inverted indexes	query plans and EXPLAIN	ACID	keyset pagination		
window functions	computed columns		transactions		
outer joins	CTEs	stored columns	ORDER BY in aggregates	normal forms	
connection pools	the DUAL table				
ORMs create bad queries	LATERAL joins		recursive CTEs		
there are no non-nullable types	stored procedures		cursors		
MVCC garbage collection	plan hints	optimizers don't work without table statistics			
COUNT(*) vs COUNT(1)	isolation levels	generator functions zip when cross joined		sharding	
serializable restarts require retry loops on all statements	zigzag join	phantom reads	triggers	MERGE	
grouping sets, cube, rollup		write skew	partial indexes		
denormalization	SELECT FOR UPDATE	NULLs in CHECK constraints are truthy			
transaction contention	sargability	timestamptz doesn't store a timezone	star schemas		
ascending key problem	ambiguous network errors		utf8mb4		
cost models don't reflect reality	'null'::jsonb IS NULL = false	TPCC requires wait times			
EXPLAIN approximates SELECT COUNT(*)	DEFERRABLE INITIALLY IMMEDIATE		causal reverse		
vectorized doesn't mean SIMD	NULLs are equal in DISTINCT but unequal in UNIQUE			volcano model	
join ordering is NP hard	database cracking			WCOJ	
learned indexes		XID exhaustion			
the halloween problem	dee and dum	SERIAL is non-transactional			
fsyncgate	allballs	NULL		every sql operator is actually a join	

ORMs

data types

GROUP BY

LIMIT and OFFSET

ORDER BY

SELECT

CREATE TABLE

foreign keys

NULL

indexes

JOIN

inverted indexes

query plans and EXPLAIN

ACID

keyset pagination

computed columns

transactions

CTEs

stored columns

ORDER BY in aggregates

normal forms

window functions

outer joins

connection pools

the DUAL table

ORMs create bad queries

LATERAL joins

recursive CTEs

there are no non-nullable types

stored procedures

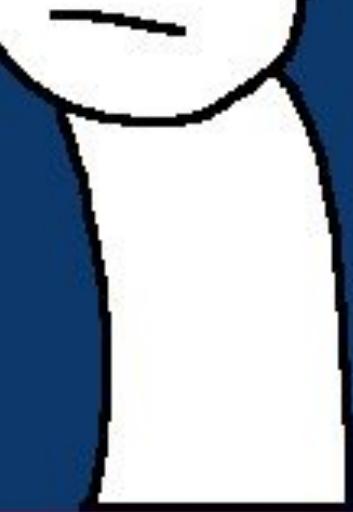
cursors

MVCC garbage collection

plan hints

optimizers don't work without
table statistics





there are no non-nullable types

MVCC garbage collection

plan hints

optimizers don't work without
table statistics

cursors

COUNT(*) vs COUNT(1)

isolation levels

generator functions zip
when cross joined

sharding

serializable restarts require
retry loops on all statements

zigzag join

phantom reads

triggers

MERGE

grouping sets, cube, rollup

write skew

partial indexes



denormalization

SELECT FOR UPDATE

NULLs in CHECK constraints
are truthy



transaction contention

sargability

timestamptz doesn't
store a timezone

star schemas

ascending key problem

ambiguous network errors

utf8mb4

cost models don't
reflect reality

'null'::jsonb IS NULL = false

TPCC requires wait times

DEFERRABLE INITIALLY IMMEDIATE



cost models don't reflect reality

EXPLAIN approximates
SELECT COUNT(*)

'null'::jsonb IS NULL = false

DEFERRABLE INITIALLY IMMEDIATE

TPCC requires wait times

causal reverse

MATCH PARTIAL foreign keys

vectorized doesn't mean SIMD

join ordering is NP hard

learned indexes

NULLs are equal in DISTINCT but unequal in UNIQUE

database cracking

volcano model

WCOJ

XID exhaustion

the halloween problem

fsyncgate

allballs

dee and dum

NULL

SERIAL is non-transactional

every sql operator is actually a join





Brighton Data Forum

connect grow learn

further learning

- refresher:
<https://www.youtube.com/watch?v=kbKty5VKMY>
- pandas experts note:
<https://www.youtube.com/watch?v=fmrmwFPMMaM>
- more discussion:
<https://www.youtube.com/watch?v=OV6Mh2Jl9zQ>
- deeper learning:
<https://app.datacamp.com/learn/career-tracks/data-analyst-in-sql>

