from zero to query

a sql primer

oskar 2023-09-13

a note on sqlite

- small (<2mb)
- open source
- serverless
- self-contained
- fast
- complete
- in-memory
- cross-platform
- ubiquitous

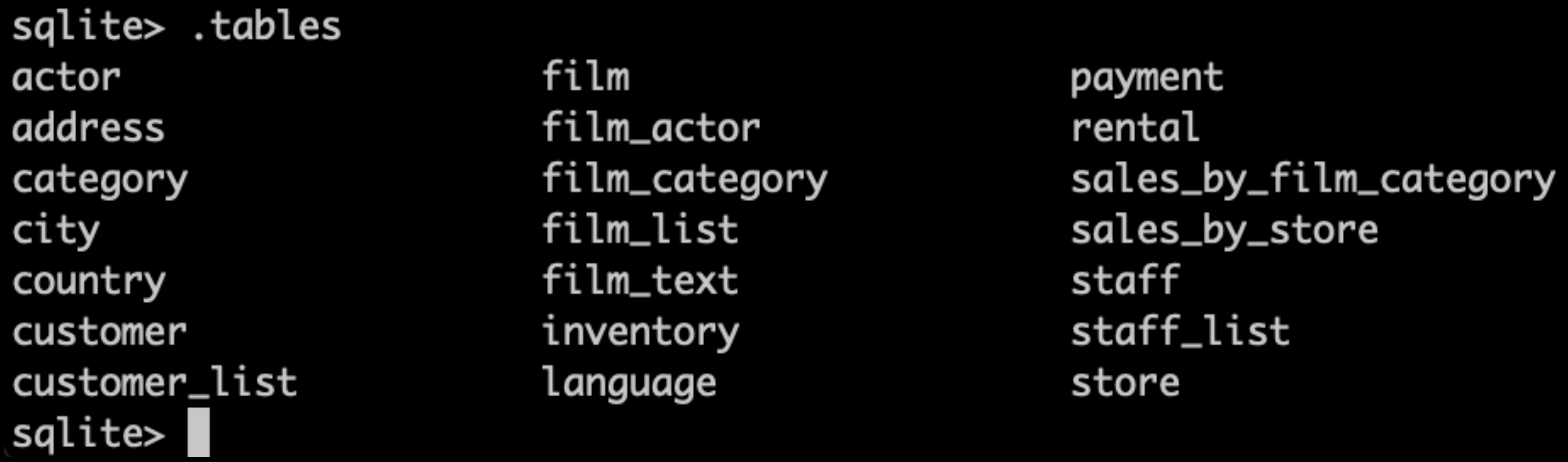
sqlite commands

- these are not sql commands!
- they start with a ‘.’
- they operate on the environment, not the data
- examples:
  - .quit
  - .open <path-to-database>
  - .show
  - .help
  - .cd <directory>
  - .shell CMD ARGS…

```
.open data/sqlite-sakila.db
.tables
```

```
.header ON
.mode qbox
```

.tables



```
sqlite> .tables
actor                   film                    payment
address                 film_actor              rental
category                film_category           sales_by_film_category
city                    film_list               sales_by_store
country                 film_text               staff
customer                inventory               staff_list
customer_list           language                store
sqlite> 
```

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				

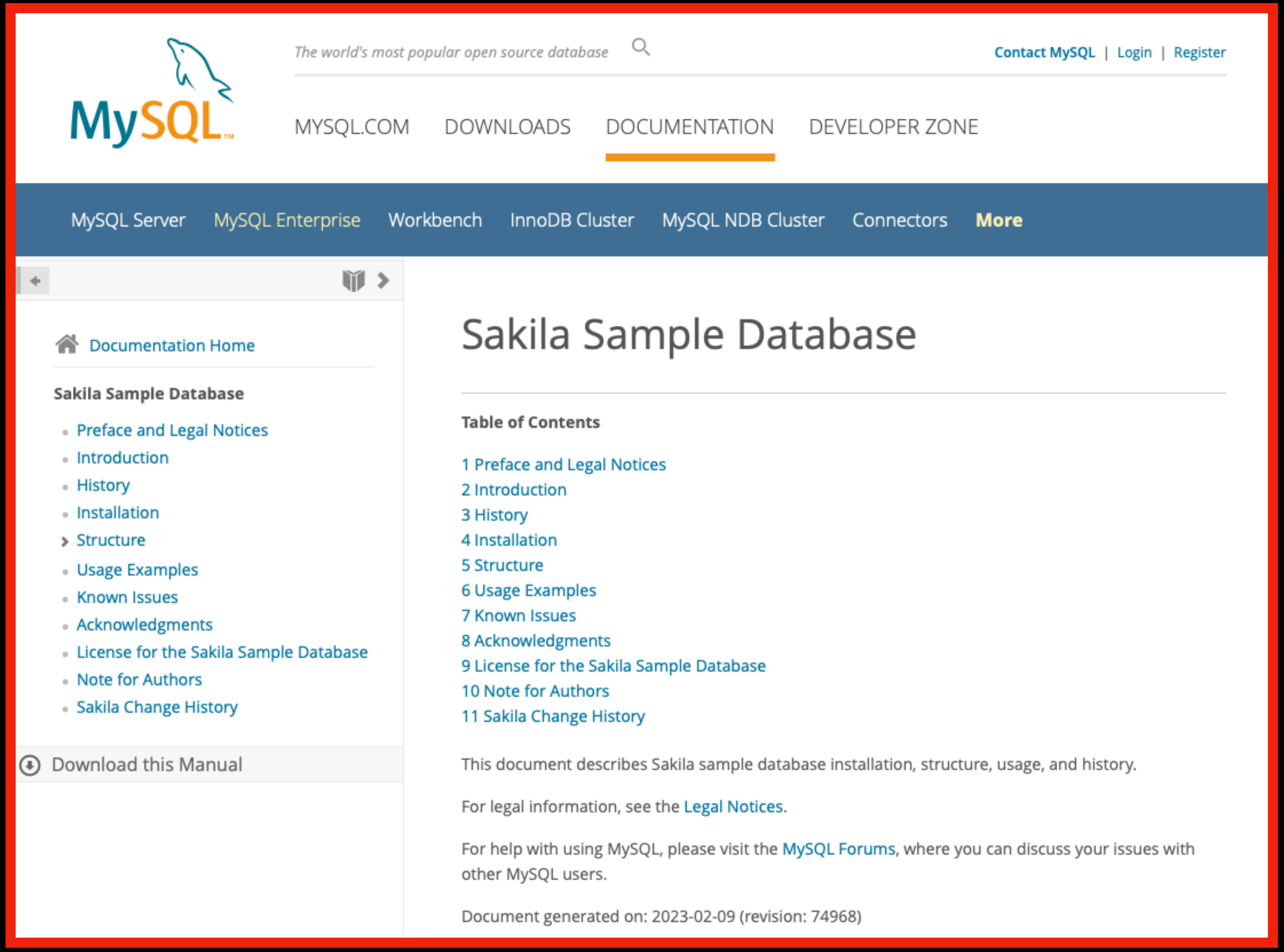
sql commands

- run on the database
- operate on the data tables
- output a table
- start with SELECT … clause
- contain a FROM … clause
- end with a ‘;’
- you can add comments with ‘-- a comment’
- example:

```
SELECT name FROM category; -- film categories
```

the sakila training data

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



Sakila Sample Database

- Preface and Legal Notices
- Introduction
- History
- Installation
- Structure
- Usage Examples
- Known Issues
- Acknowledgments
- License for the Sakila Sample Database
- Note for Authors
- Sakila Change History

Sakila Sample Database

Table of Contents

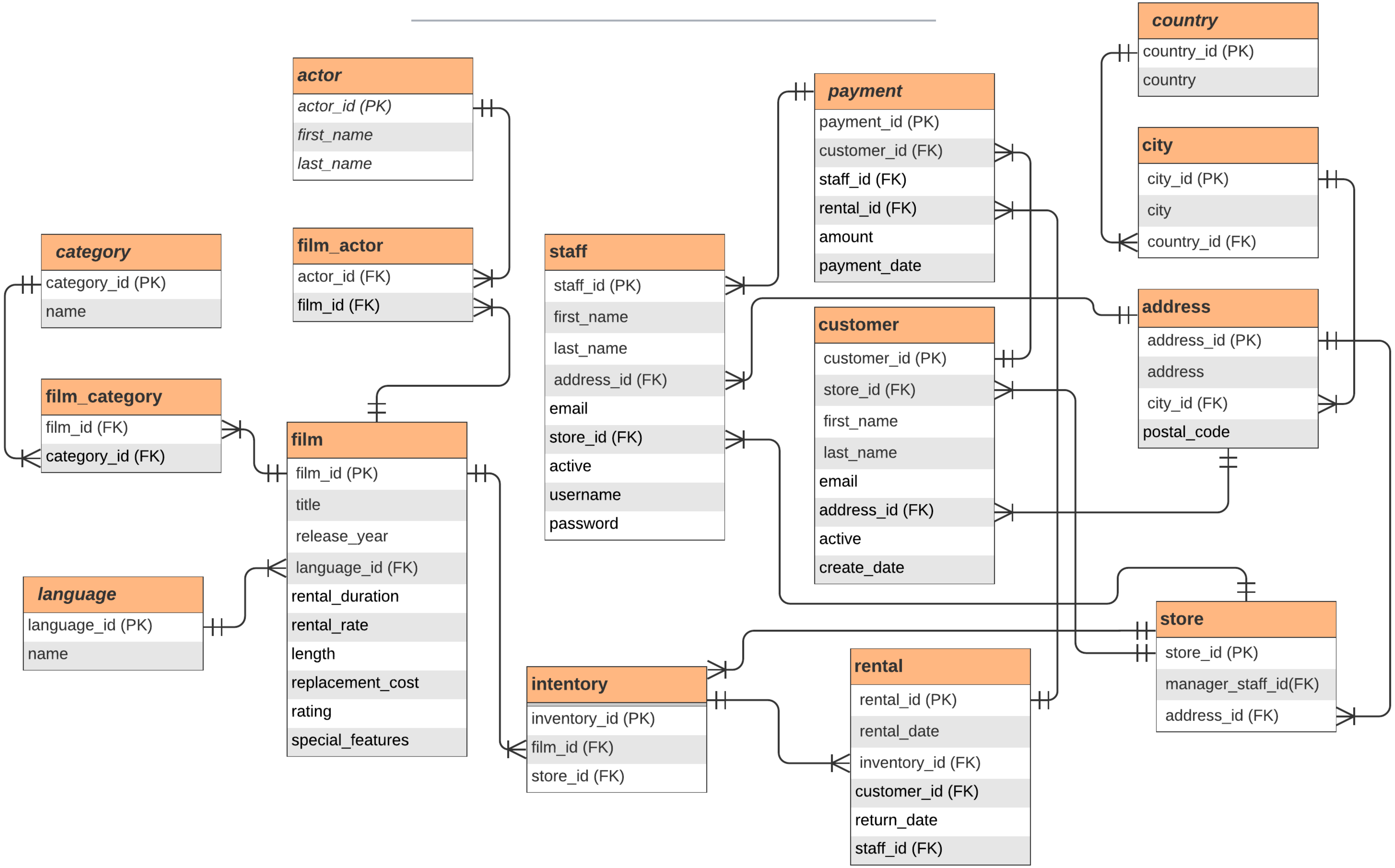
This document describes Sakila sample database installation, structure, usage, and history.

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



actor
actor_id (PK)
first_name
last_name

category
category_id (PK)
name

film_actor
actor_id (FK)
film_id (FK)

film_category
film_id (FK)
category_id (FK)

film
film_id (PK)
title
release_year
language_id (FK)
rental_duration
rental_rate
length
replacement_cost
rating
special_features

language
language_id (PK)
name

staff
staff_id (PK)
first_name
last_name
address_id (FK)
email
store_id (FK)
active
username
password

intentory
inventory_id (PK)
film_id (FK)
store_id (FK)

payment
payment_id (PK)
customer_id (FK)
staff_id (FK)
rental_id (FK)
amount
payment_date

customer
customer_id (PK)
store_id (FK)
first_name
last_name
email
address_id (FK)
active
create_date

rental
rental_id (PK)
rental_date
inventory_id (FK)
customer_id (FK)
return_date
staff_id (FK)

country
country_id (PK)
country

city
city_id (PK)
city
country_id (FK)

address
address_id (PK)
address
city_id (FK)
postal_code

store
store_id (PK)
manager_staff_id(FK)
address_id (FK)

today’s objective:

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

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
```

exercises

- 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 + replacement_cost) AS average_total_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 maximum number of times that a film needs to be rented out to break even?

- what is the average 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 film

- SELECT * FROM category LIMIT 5;
  -- returns 5 unspecified rows of all columns from category

- SELECT title, release_year FROM film LIMIT 15;
  -- returns 15 unspecified rows of two columns from film table

- 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 actor
```

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 'Ok%';
  -- returns info on all cities whose name begins with ‘Ok’


- 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 in address table
  ```

-
  ```
  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 …;

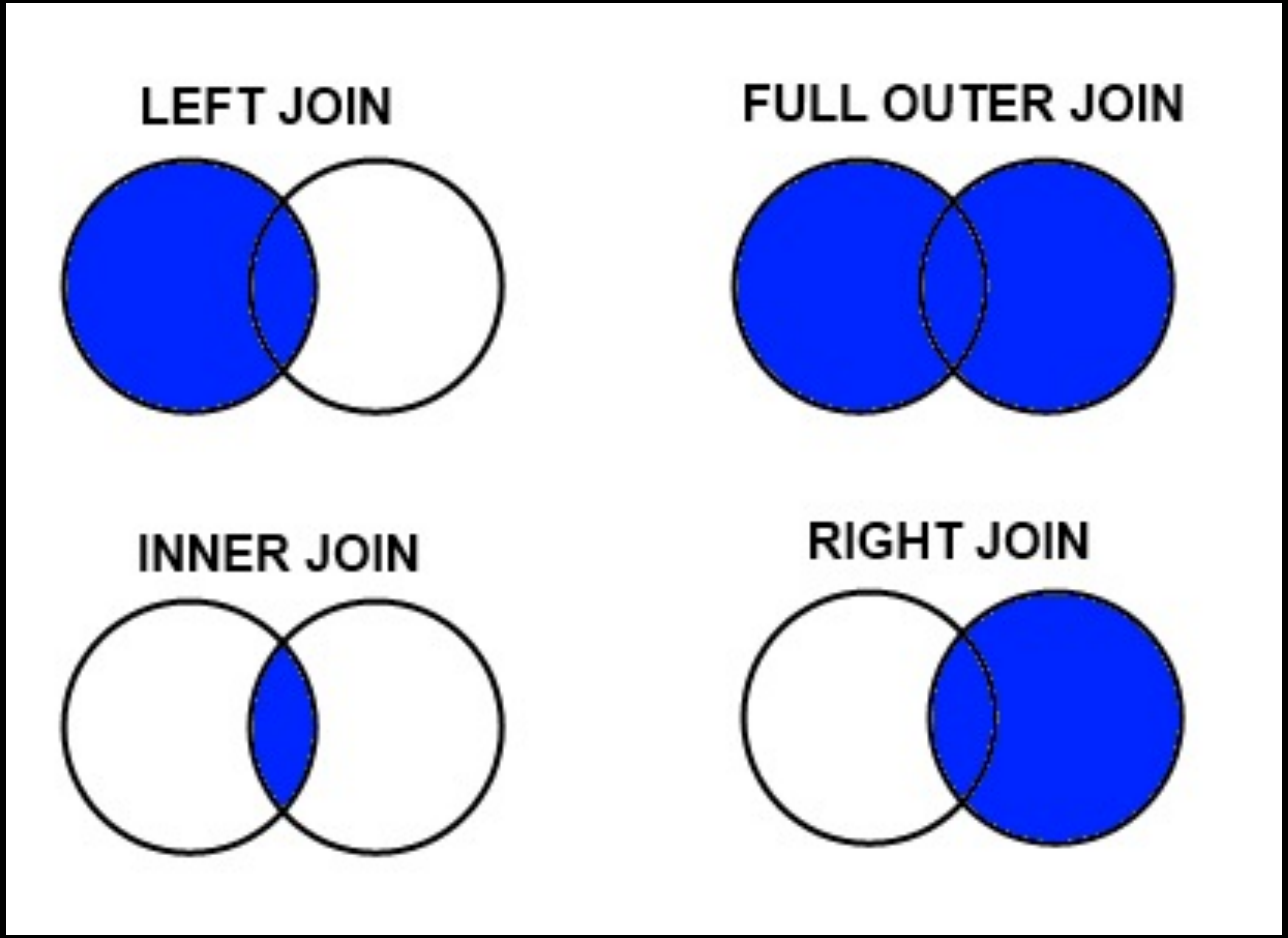
```
- SELECT col1, COUNT(*) AS num FROM table GROUP BY col1 HAVING num>9;
  -- count instances of each value of col1, but only output rows with count>9

- 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}
    ;
```

All data professionals need to understand table joins
@DATAWITHDANNY


LEFT JOIN
FULL OUTER JOIN
INNER JOIN
RIGHT JOIN

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	Turkey
6	Addis Abeba	31	2021-03-06 15:51:49	
7	Aden	107	2021-03-06 15:51:49	Yemen
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

...

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

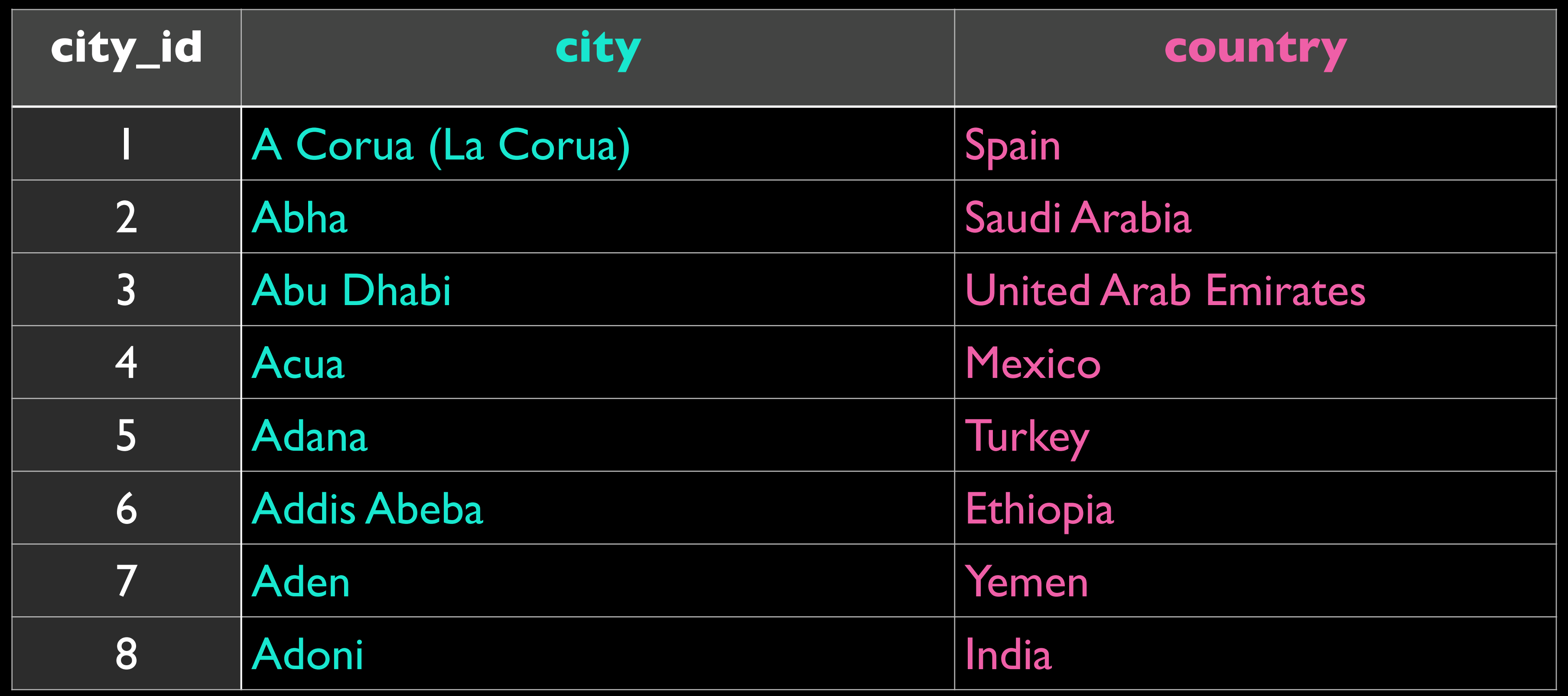
...

```
SELECT
    city_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_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
```

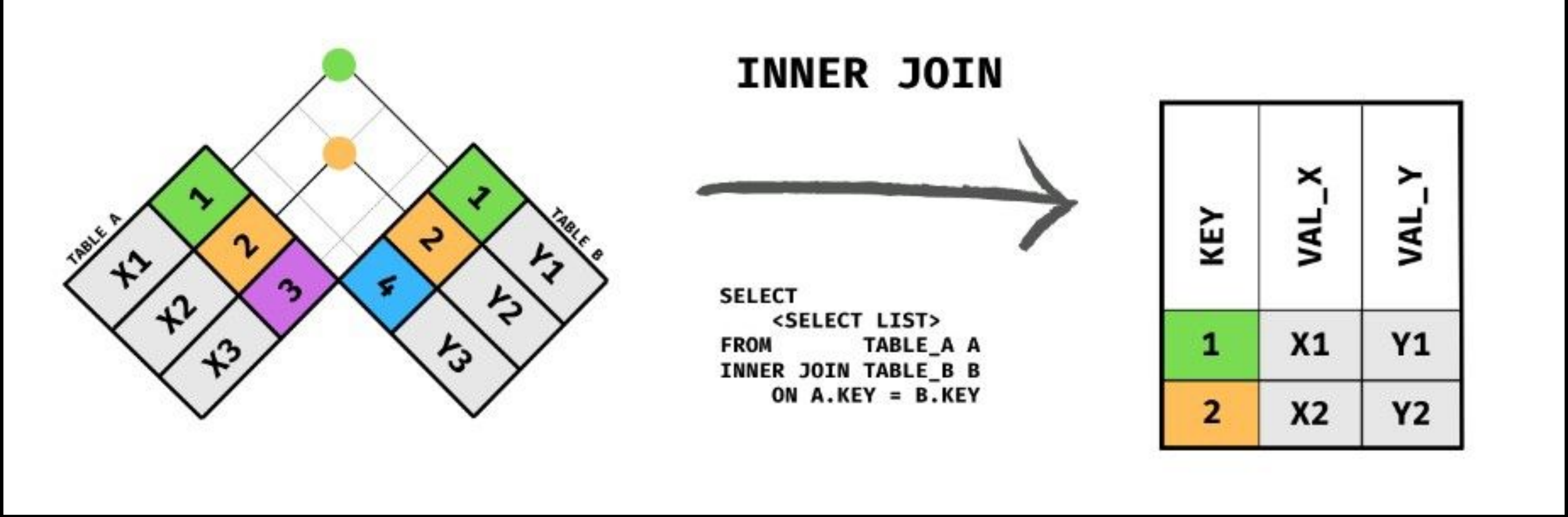
SELECT … FROM a INNER JOIN b ON a.key=b.key;

```sql
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?

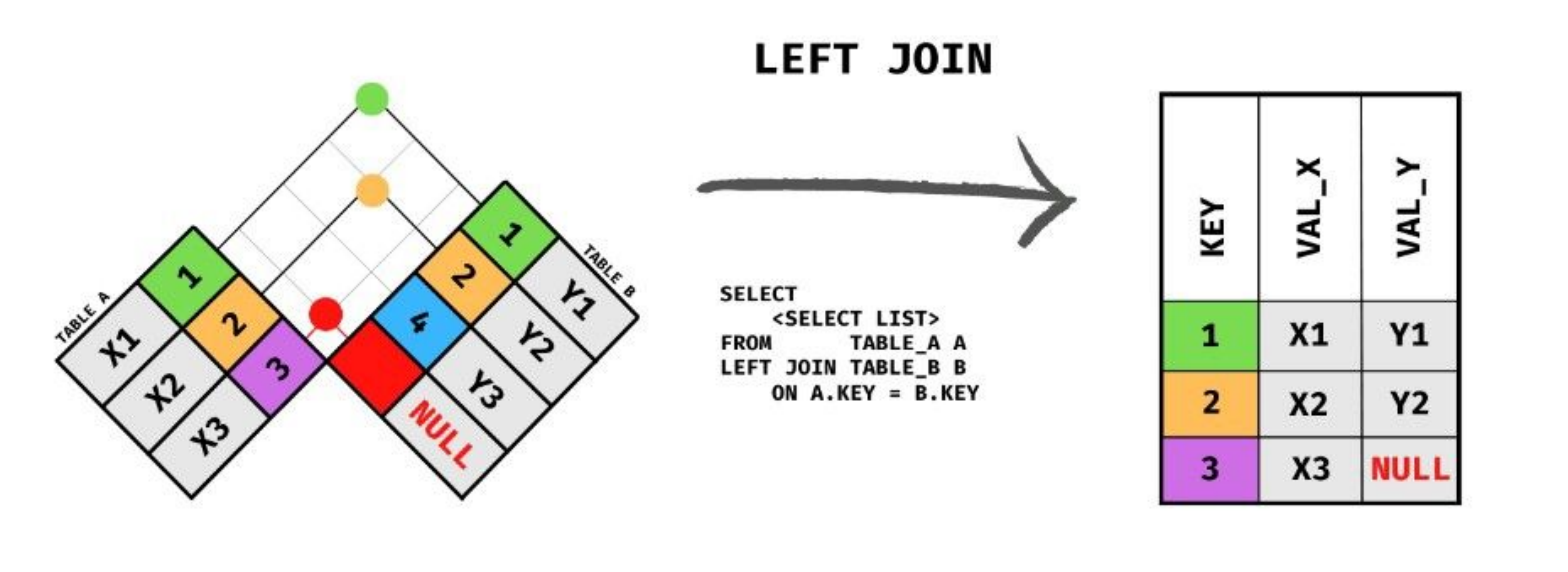
FROM a INNER JOIN b



```
SELECT
    <SELECT LIST>
FROM        TABLE_A A
INNER JOIN TABLE_B B
    ON A.KEY = B.KEY
```

KEY	VAL_X	VAL_Y
1	X1	Y1
2	X2	Y2

LEFT OUTER JOIN



LEFT JOIN

```
SELECT
    <SELECT LIST>
FROM       TABLE_A A
LEFT JOIN TABLE_B B
    ON A.KEY = B.KEY
```

KEY	VAL_X	VAL_Y
1	X1	Y1
2	X2	Y2
3	X3	NULL

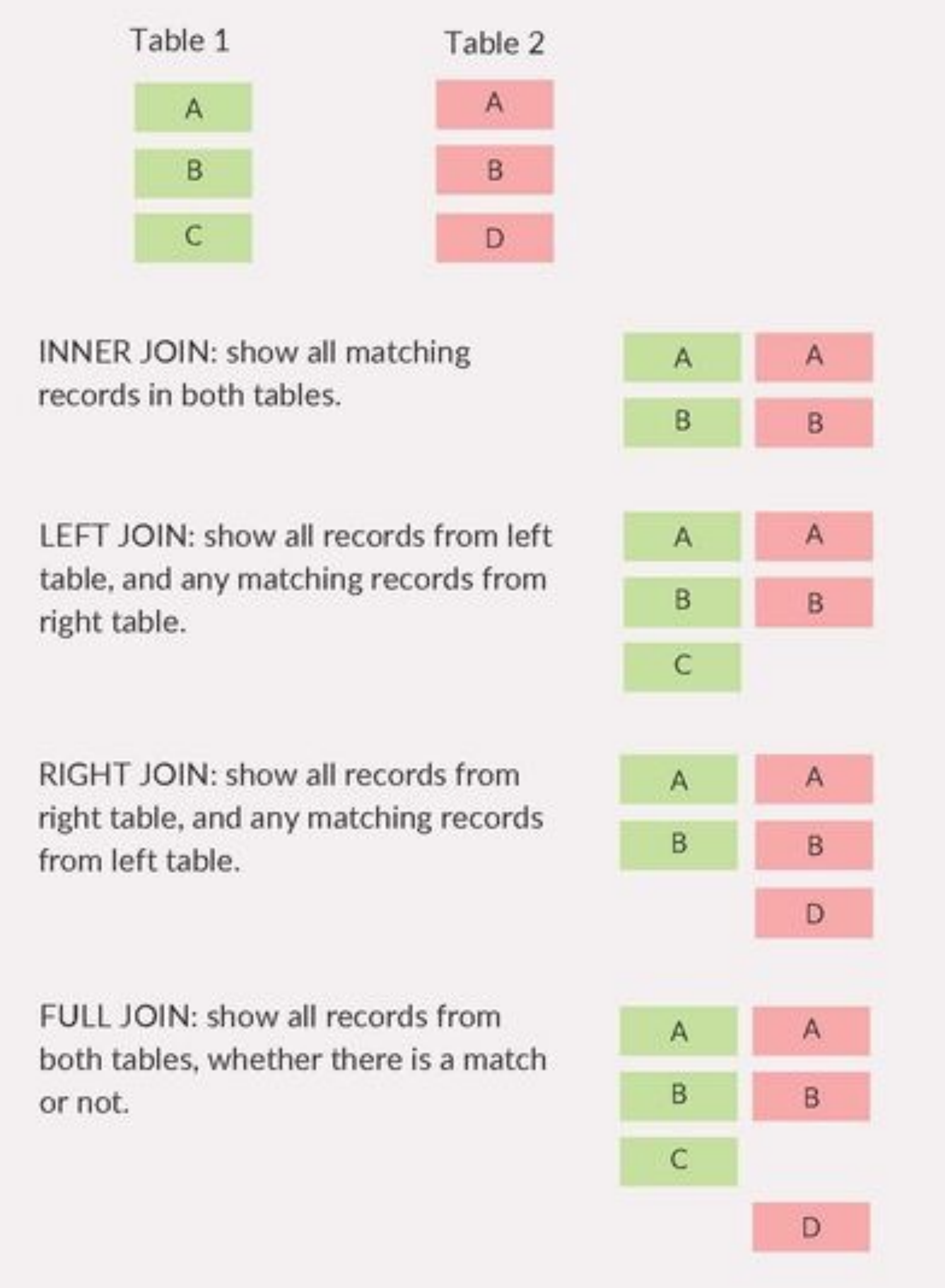


Table 1	Table 2
A	A
B	B
C	D

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.

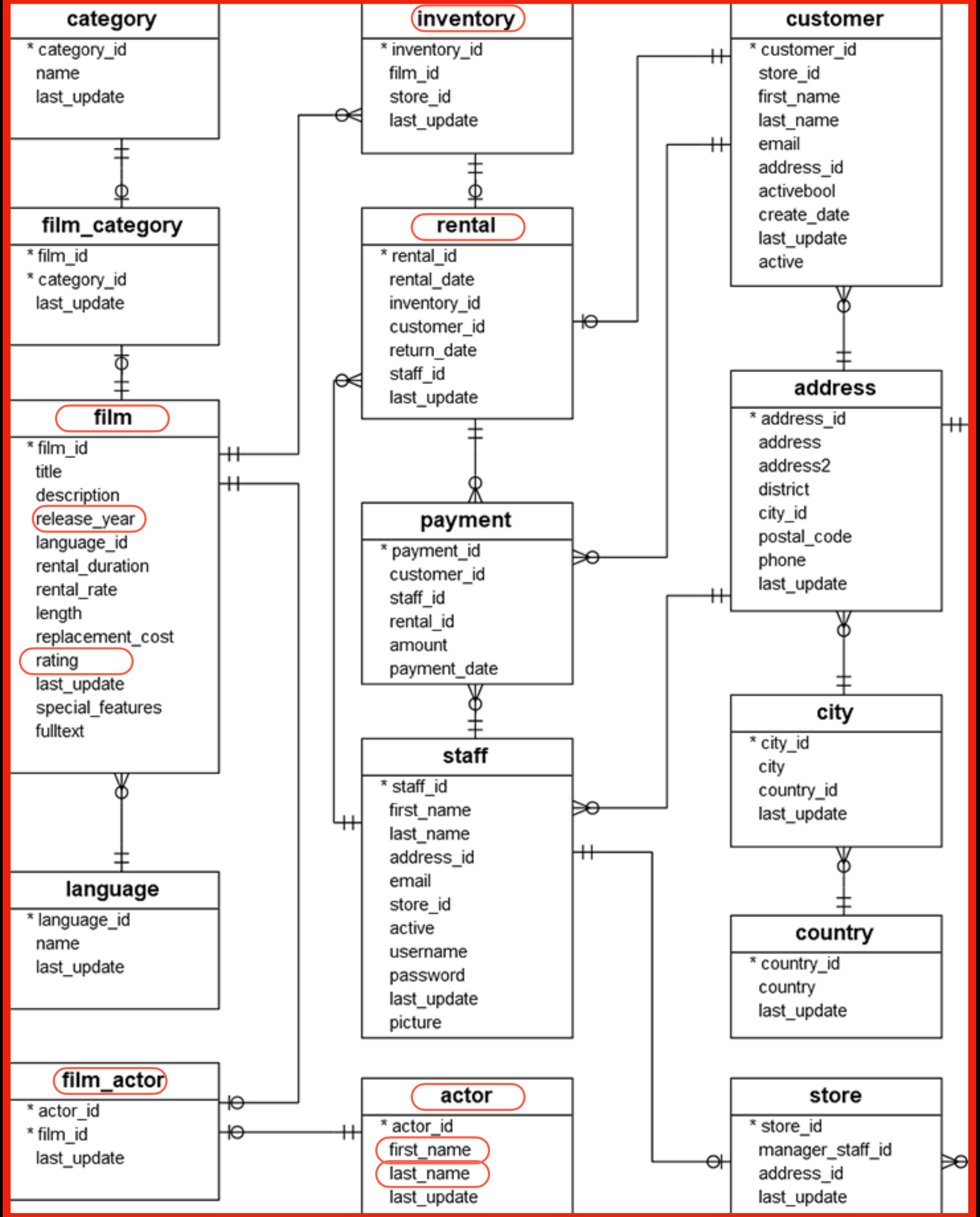
A	A
B	B
	D

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**?”



category
* category_id
name
last_update

inventory
* inventory_id
film_id
store_id
last_update

customer
* customer_id
store_id
first_name
last_name
email
address_id
activebool
create_date
last_update
active

film_category
* film_id
* category_id
last_update

rental
* rental_id
rental_date
inventory_id
customer_id
return_date
staff_id
last_update

address
* address_id
address
address2
district
city_id
postal_code
phone
last_update

film
* film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
rating
last_update
special_features
fulltext

payment
* payment_id
customer_id
staff_id
rental_id
amount
payment_date

staff
* staff_id
first_name
last_name
address_id
email
store_id
active
username
password
last_update
picture

city
* city_id
city
country_id
last_update

language
* language_id
name
last_update

country
* country_id
country
last_update

film_actor
* actor_id
* film_id
last_update

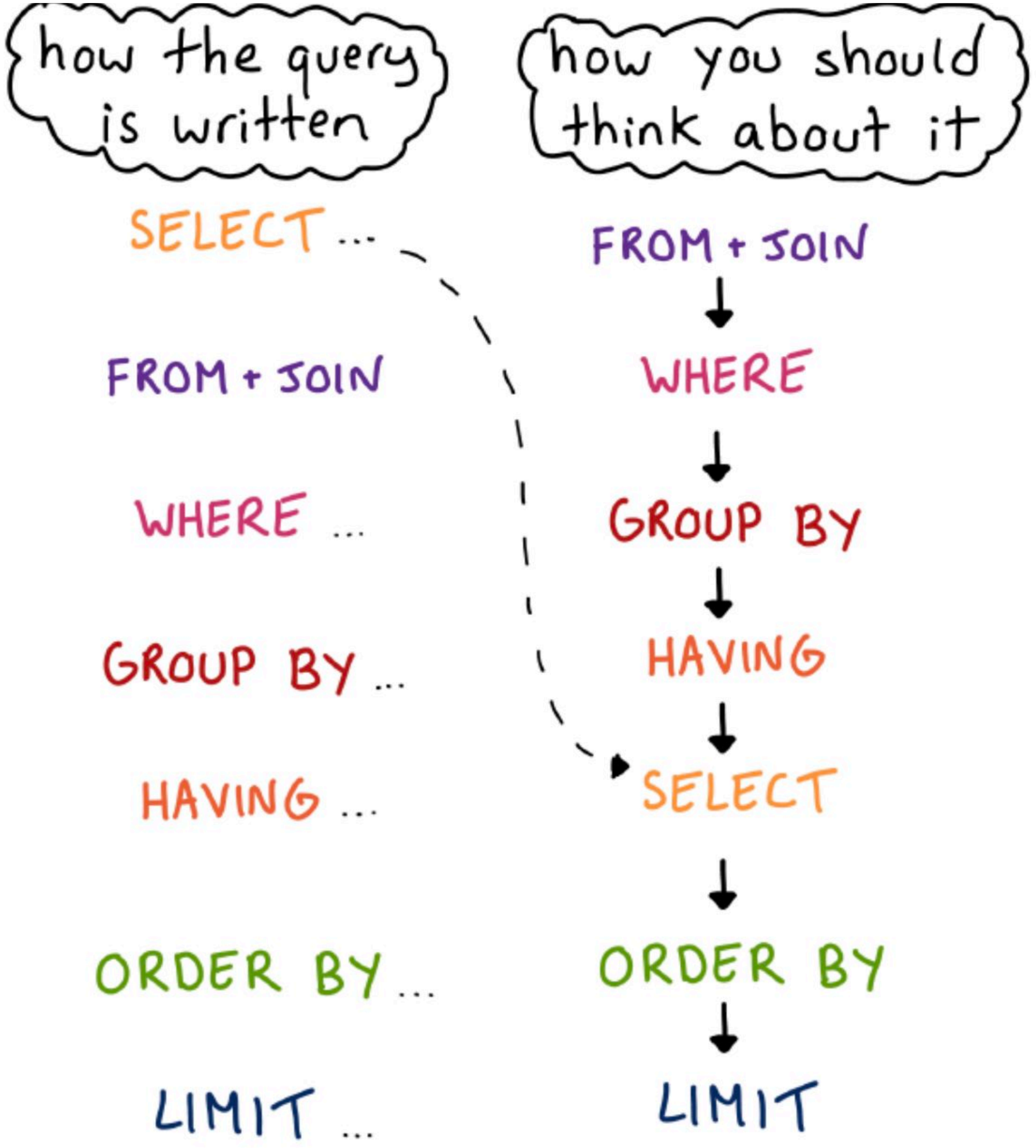
actor
* actor_id
first_name
last_name
last_update

store
* store_id
manager_staff_id
address_id
last_update

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}
    ;
```



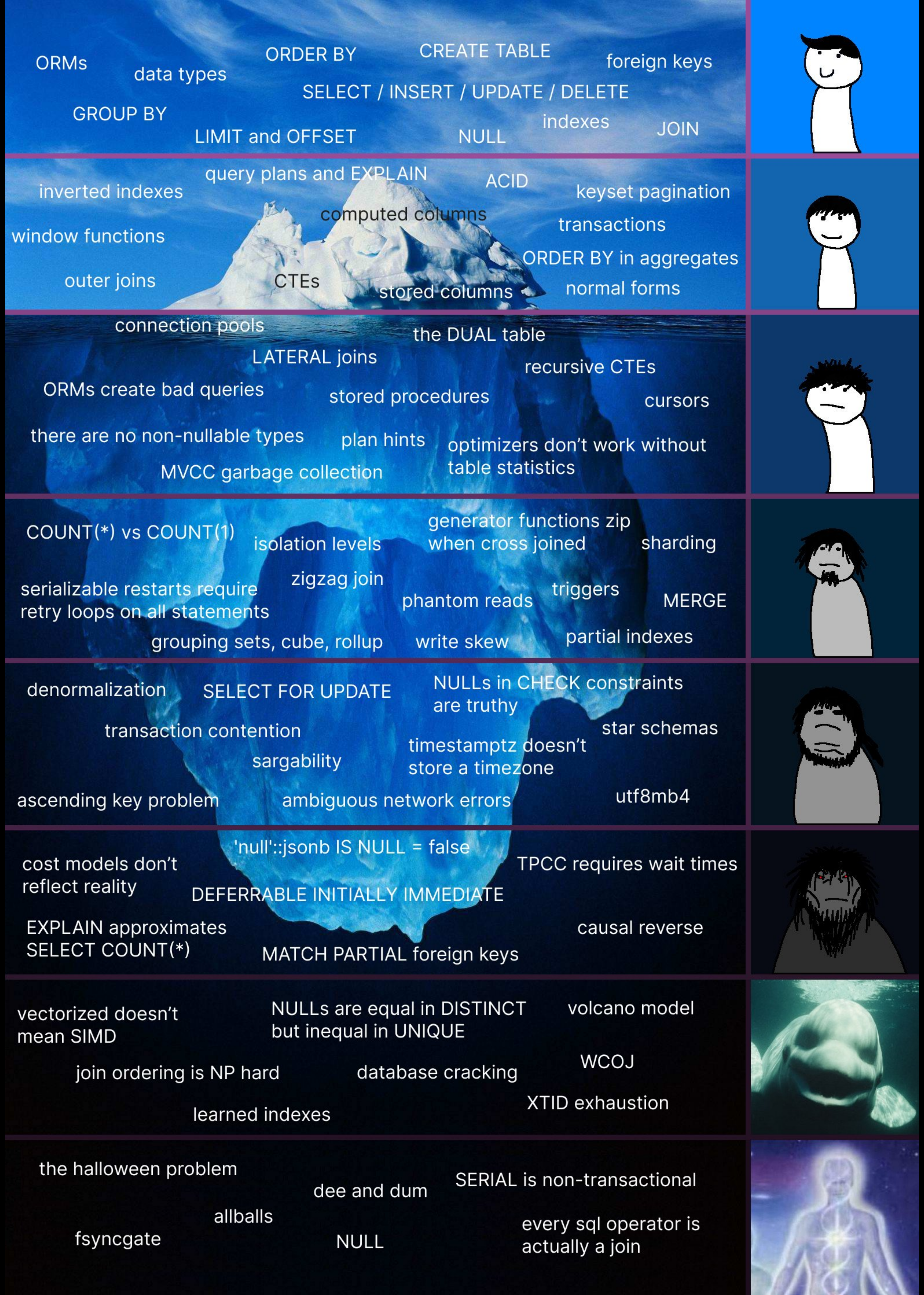
JULIA EVANS
@b0rk
wizardzines.com

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 {column1}
ORDER BY {column} DESC
LIMIT {num}
;
```



ORMs
data types
ORDER BY
CREATE TABLE
foreign keys
SELECT / INSERT / UPDATE / DELETE
GROUP BY
LIMIT and OFFSET
NULL
indexes
JOIN

query plans and EXPLAIN
ACID
inverted indexes
keyset pagination
computed columns
transactions
window functions
ORDER BY in aggregates
outer joins
CTEs
stored columns
normal forms

connection pools
the DUAL table
LATERAL joins
recursive CTEs
ORMs create bad queries
stored procedures
cursors
there are no non-nullable types
plan hints
optimizers don't work without table statistics
MVCC garbage collection

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
star schemas
sargability
timestamptz doesn't store a timezone
ascending key problem
ambiguous network errors
utf8mb4

'null'::jsonb IS NULL = false
cost models don't reflect reality
TPCC requires wait times
DEFERRABLE INITIALLY IMMEDIATE
EXPLAIN approximates SELECT COUNT(*)
causal reverse
MATCH PARTIAL foreign keys

vectorized doesn't mean SIMD
NULLs are equal in DISTINCT but inequal in UNIQUE
volcano model
WCOJ
join ordering is NP hard
database cracking
XTID exhaustion
learned indexes

the halloween problem
dee and dum
SERIAL is non-transactional
allballs
fsyncgate
NULL
every sql operator is actually a join

ORMs

data types

ORDER BY

CREATE TABLE

foreign keys

SELECT / INSERT / UPDATE / DELETE

GROUP BY

LIMIT and OFFSET

NULL

indexes

JOIN

query plans and EXPLAIN

ACID

inverted indexes

keyset pagination

computed columns

transactions

window functions

ORDER BY in aggregates

outer joins

CTEs

stored columns

normal forms

connection pools

the DUAL table

LATERAL joins

recursive CTEs

ORMs create bad queries

stored procedures

cursors

there are no non-nullable types

plan hints

optimizers don't work without table statistics

MVCC garbage collection

stored procedures

cursors

there are no non-nullable types

plan hints

optimizers don’t work without table statistics

MVCC garbage collection

COUNT(*) vs COUNT(1)

isolation levels

generator functions zip when cross joined

sharding

zigzag join

serializable restarts require retry loops on all statements

phantom reads

triggers

MERGE

grouping sets, cube, rollup

write skew

partial indexes

denormalization

SELECT FOR UPDATE

NULLs in CHECK constraints are truthy

star schemas

transaction contention

timestamptz doesn’t store a timezone

sargability

ascending key problem

ambiguous network errors

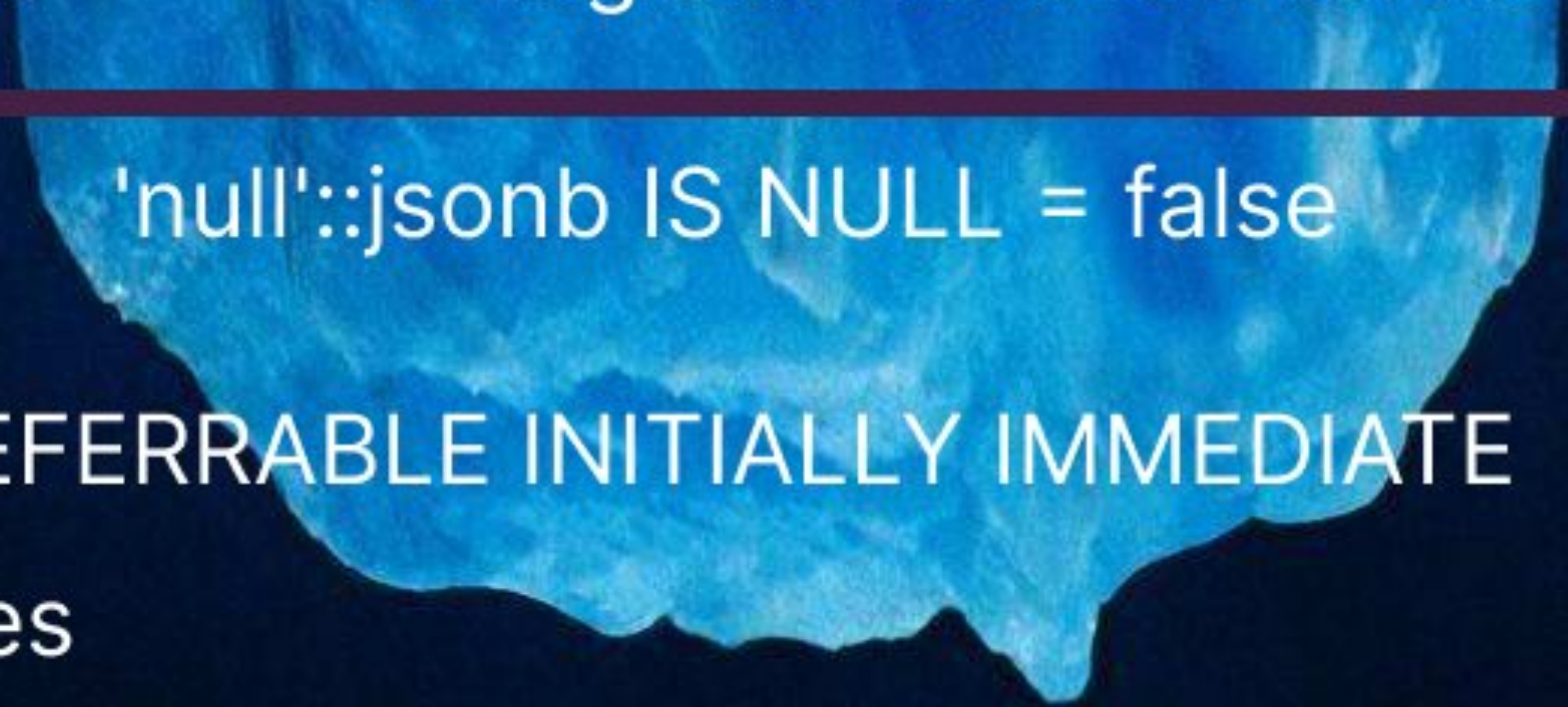
utf8mb4

'null'::jsonb IS NULL = false

cost models don’t reflect reality

TPCC requires wait times

DEFERRABLE INITIALLY IMMEDIATE



'null'::jsonb IS NULL = false

cost models don't reflect reality

TPCC requires wait times

DEFERRABLE INITIALLY IMMEDIATE

EXPLAIN approximates SELECT COUNT(*)

causal reverse

MATCH PARTIAL foreign keys

vectorized doesn't mean SIMD

NULLs are equal in DISTINCT but inequal in UNIQUE

volcano model

WCOJ

join ordering is NP hard

database cracking

XTID exhaustion

learned indexes

the halloween problem

SERIAL is non-transactional

dee and dum

allballs

fsyncgate

NULL

every sql operator is actually a join

further learning

- refresher:
  https://www.youtube.com/watch?v=kbKty5ZVKMY

- 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