from zero to query

a sql primer

oskar 2023-03-06

sql - a fundamental data tool

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

sql - a fundamental data tool

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

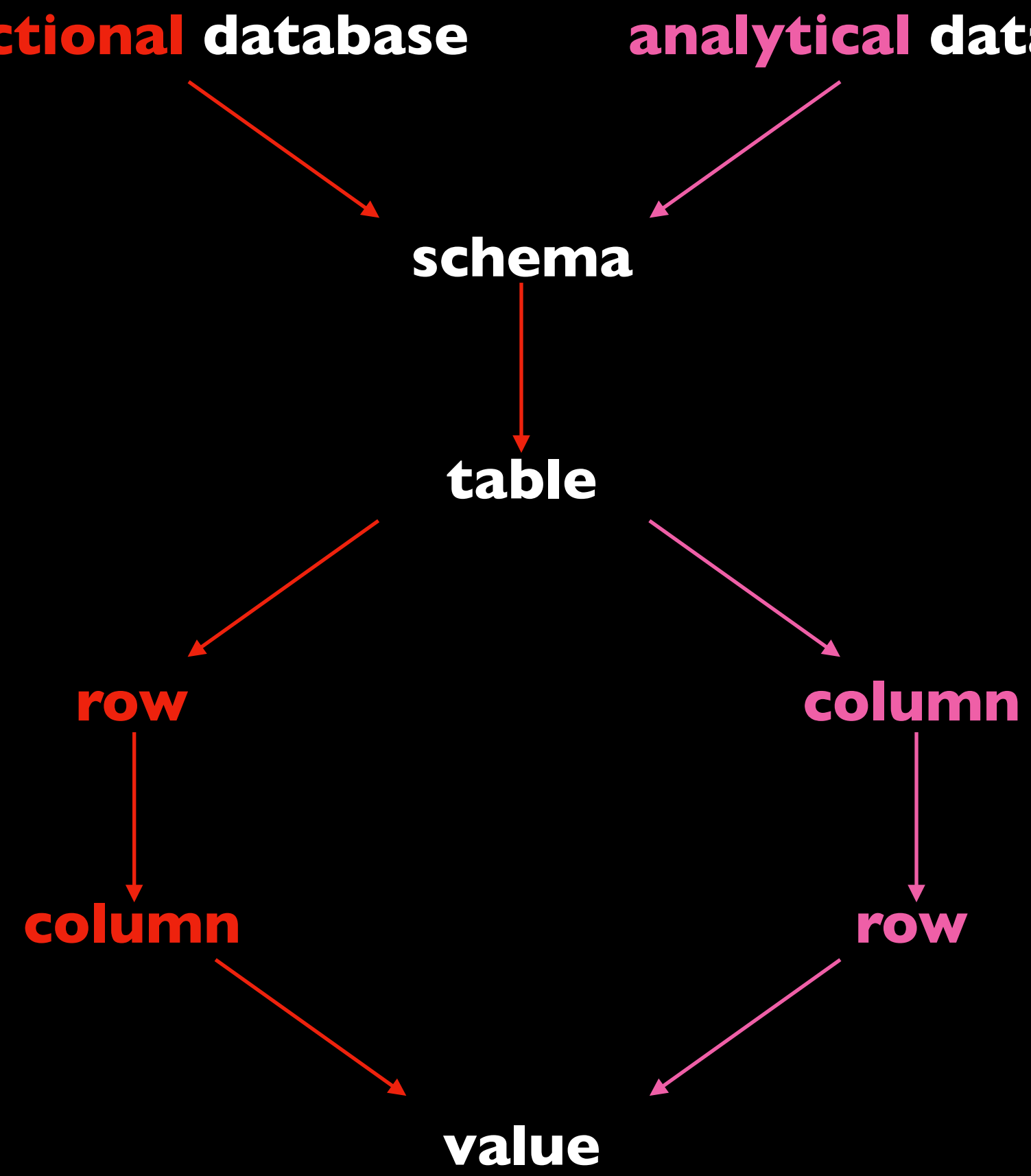transactional database   analytical database

schema

table

row   column

column   row

value

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				

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

.header ON
.mode qbox

.tables
```

.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>
```
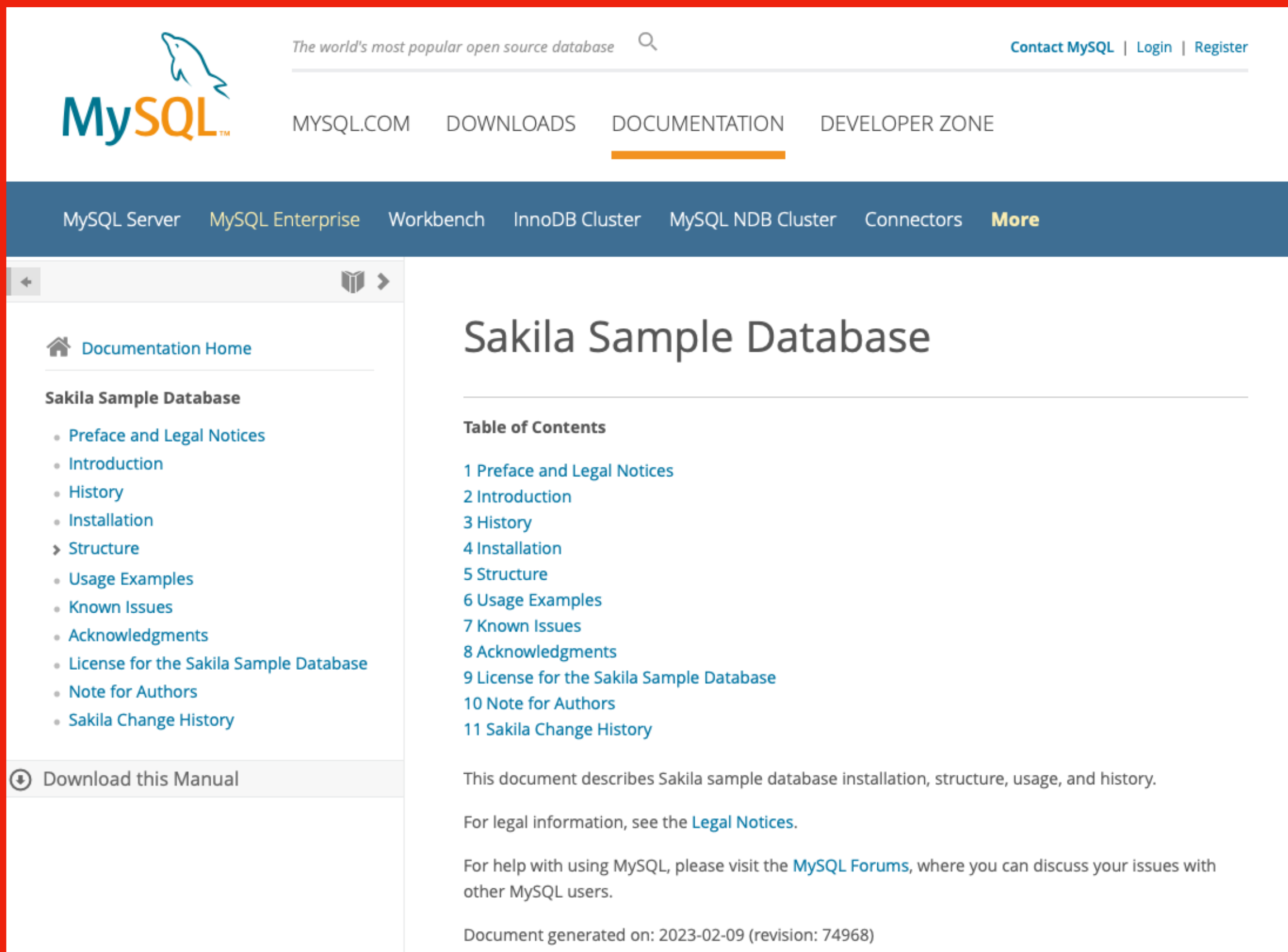
sql commands

- these run on the database
- they end with a ';'
- you can add comments with '-- a comment'
- they operate on the data tables
- example:
  - SELECT {columns} FROM {table}; -- a&b

the sakila training data

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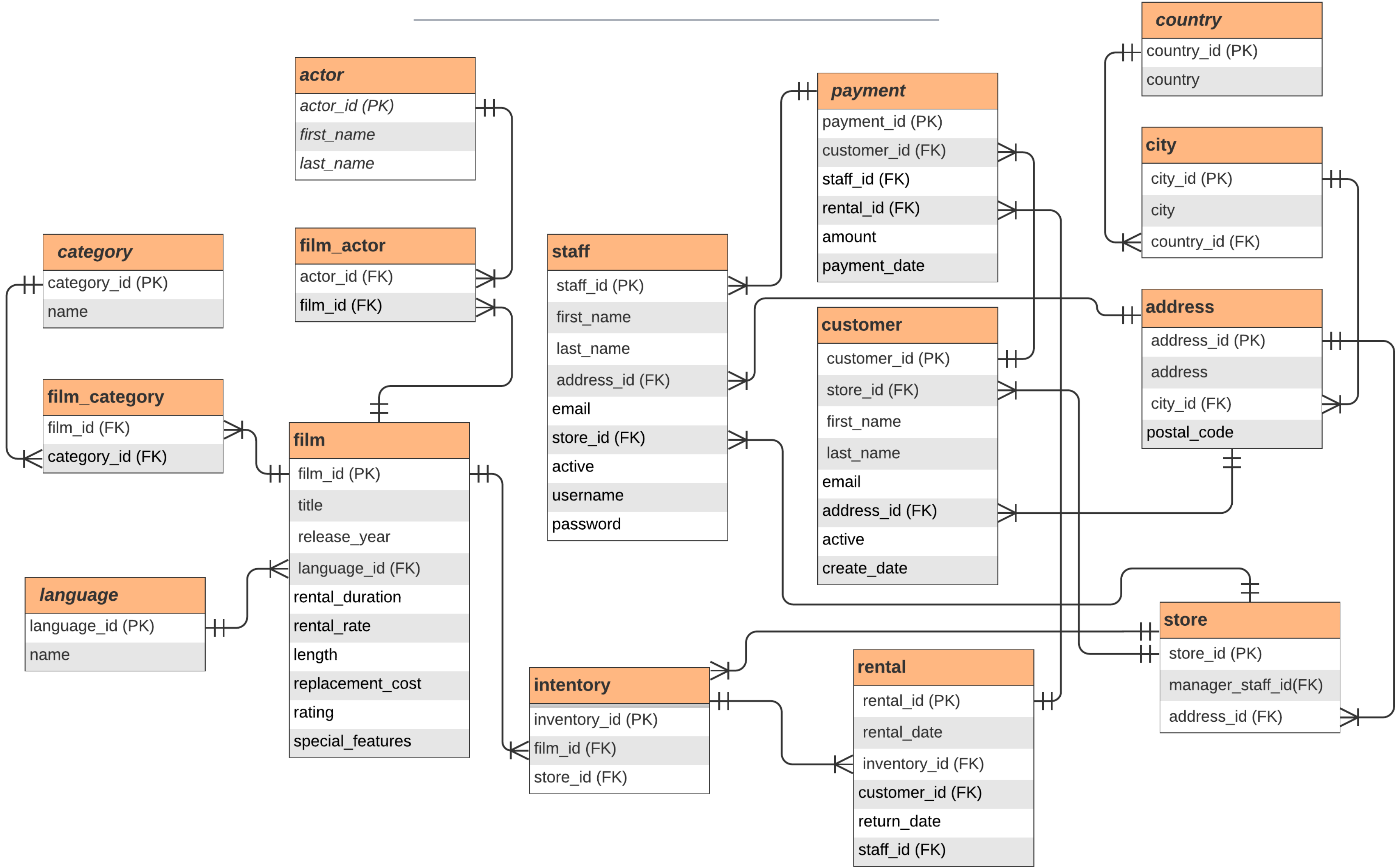 - films
  - film casting
  - actors
  - film ratings

SQLite3 Sakila Sample Database ERD

actor
- actor_id (PK)
- first_name
- last_name

film_actor
- actor_id (FK)
- film_id (FK)

category
- category_id (PK)
- name

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

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

country
- country_id (PK)
- country

city
- city_id (PK)
- city
- country_id (FK)

address
- address_id (PK)
- address
- city_id (FK)
- postal_code

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

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

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:

```
- 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
```

what do the tables contain?

```
- 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 … FROM …;

- `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 name FROM customer c;`
  `-- sets an alias for table customer, renames column to 'name'`

- `SELECT rental_rate + replacement_cost AS total_cost FROM film;`
  `-- returns the 'total_cost' of renting, from table film`

- `SELECT DISTINCT first_name FROM actor;`
  `-- returns all the first names in the actor table, no duplicates`

SELECT (aggregate function) FROM … ;

- 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_rate 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

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 {a_condition}
- + GROUP BY {columns}
- + HAVING {a_condition}
- + ORDER BY {columns}
- + LIMIT num
```

SELECT … FROM … LIMIT …;

- SELECT * FROM {table} LIMIT {n};
  -- returns {n} unspecified rows of all columns from {table}

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

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

- SELECT rental_id, rental_date FROM rental LIMIT 10;
  -- returns region id and region name for 10 unspecified rows

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 {a_condition}
- + GROUP BY {columns}
- + HAVING {a_condition}
- + ORDER BY {columns}
- + LIMIT num

SELECT … FROM … WHERE … [LIMIT n];

- `SELECT * FROM {table} WHERE {column}={expression};`
  `-- returns only rows where the value in {column} equals {expression}`

- `SELECT title AS name FROM film f WHERE rating <>'R' LIMIT 10;;`
  `-- 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 name of staff at store with id 2`

comparison operators

operator 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}

comparison operators

operator 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 '2005-11-01' AND '2005-01-01';
  -- returns only rentals occurring in december 2005


- 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'
```

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 {a_condition}
– + GROUP BY {columns}
– + HAVING {a_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

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
```
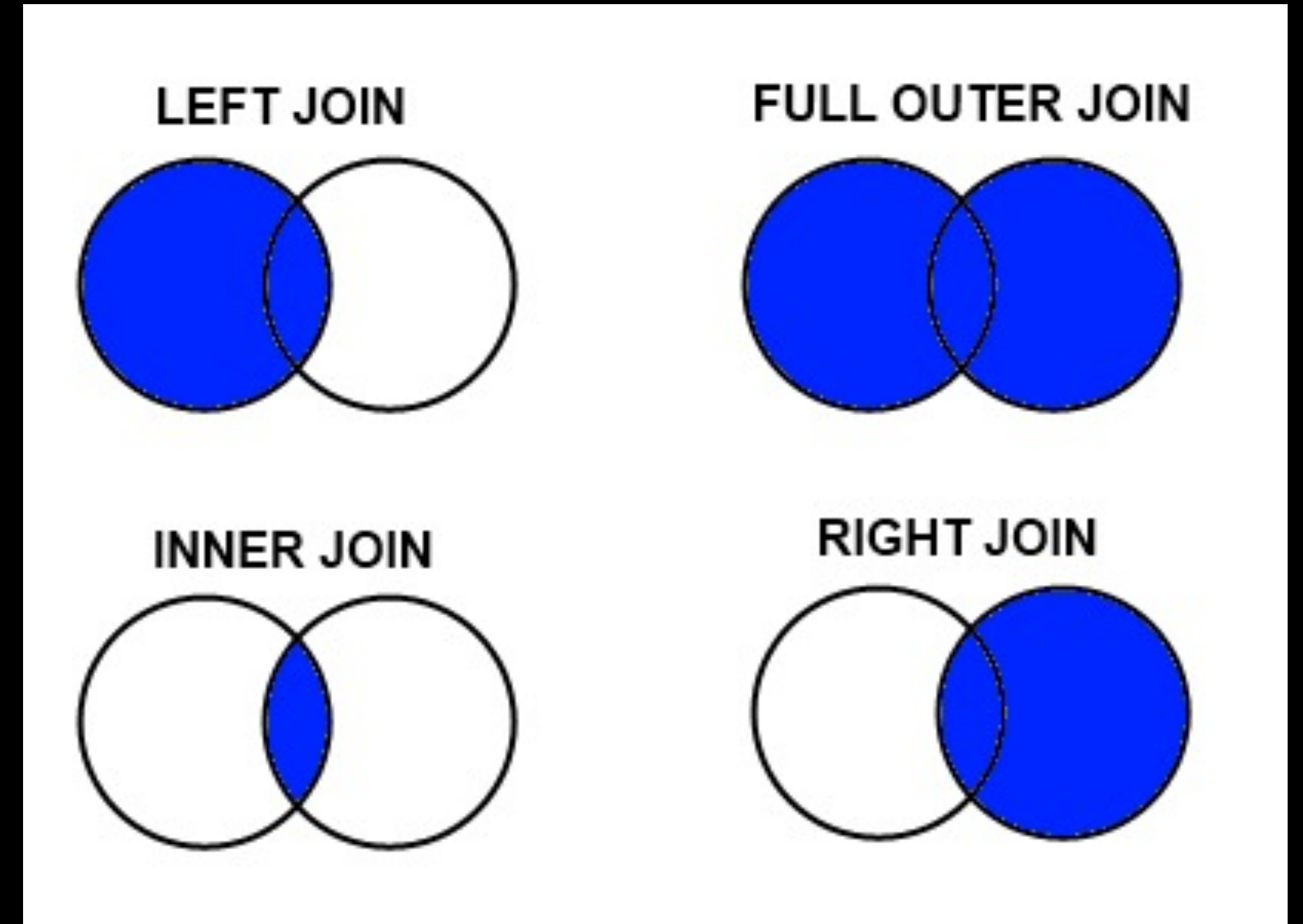
SELECT {col}, … FROM … GROUP BY {col};

- 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 {a_condition}
– + GROUP BY {columns}
– + HAVING {a_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 {a_condition}
- + GROUP BY {columns}
- + HAVING {a_condition}
- + ORDER BY {columns}
- + LIMIT num

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

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	
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	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
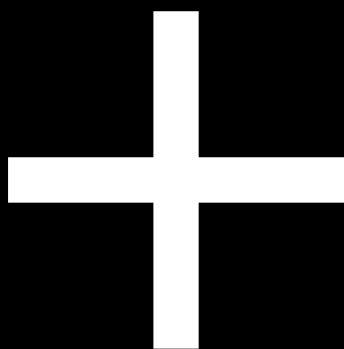
country

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 …;

- SELECT a.city, b.country
  FROM city a
    INNER JOIN country b ON a.country_id=b.country_id
  ; -- 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 rating='R'
  LIMIT 10; -- output a sample of films and the name of the language it is in

FROM a INNER JOIN b
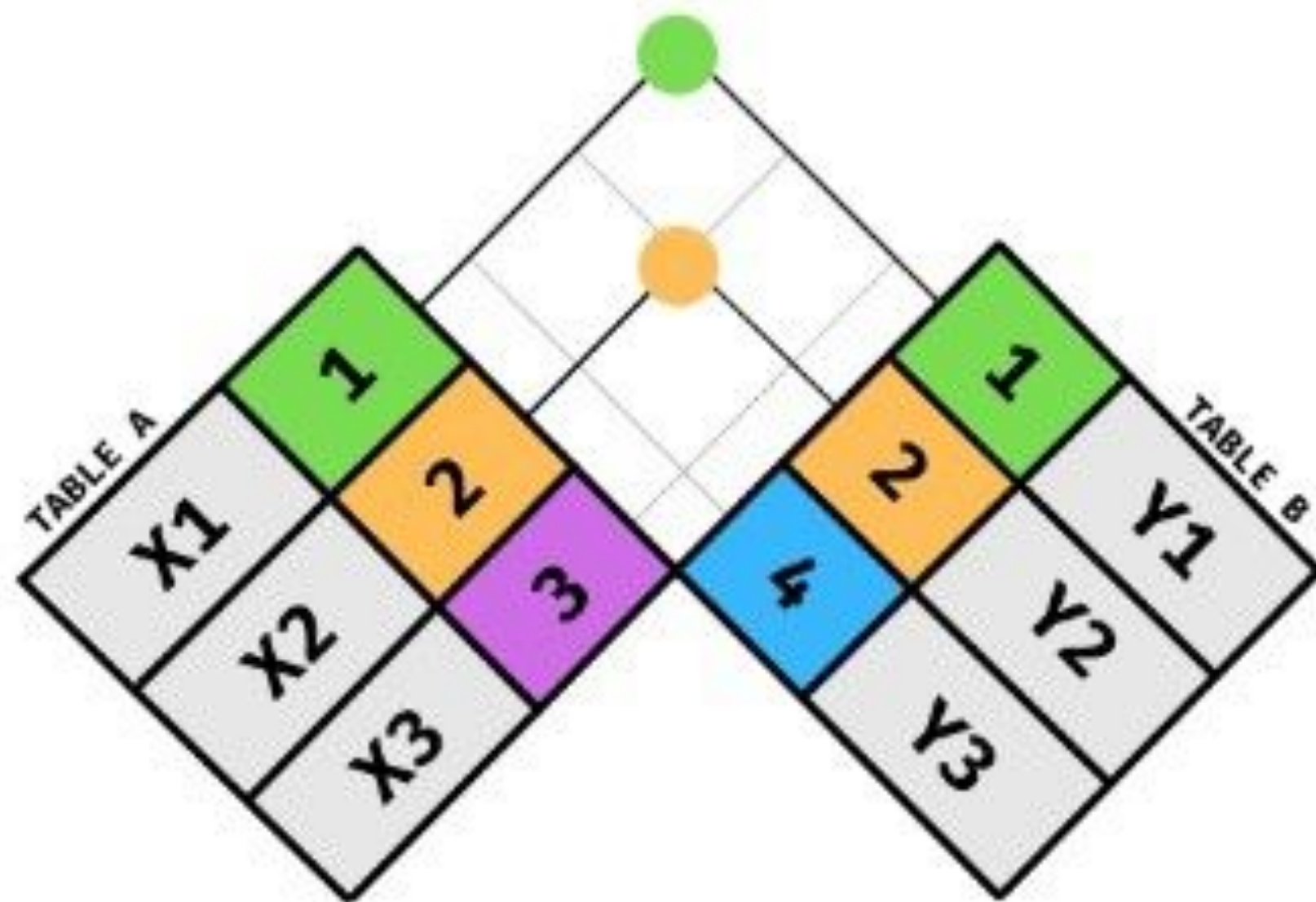
LEFT OUTER JOIN

All data professionals need to understand table joins

@DATAWITHDANNY
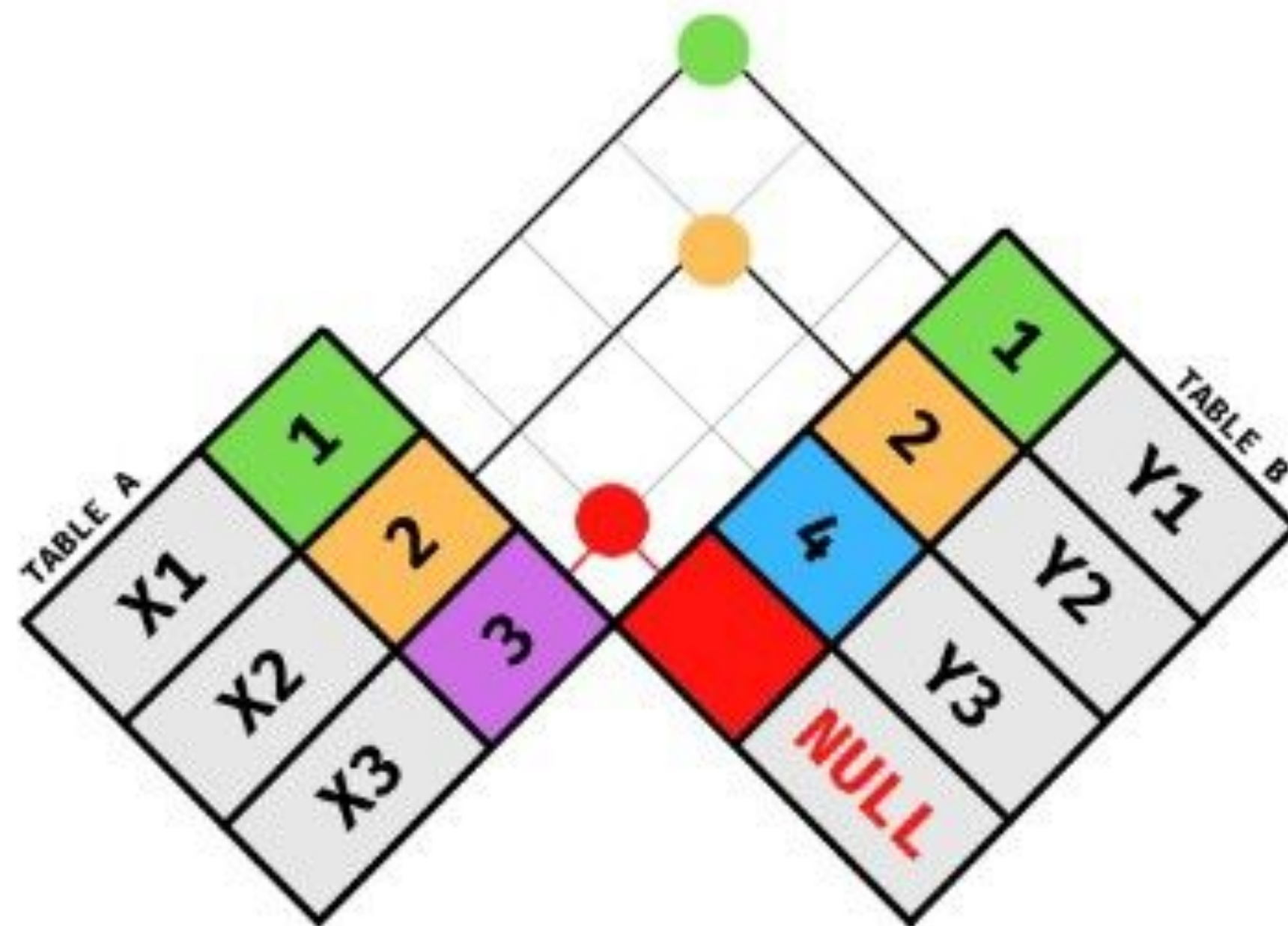
Table 1

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
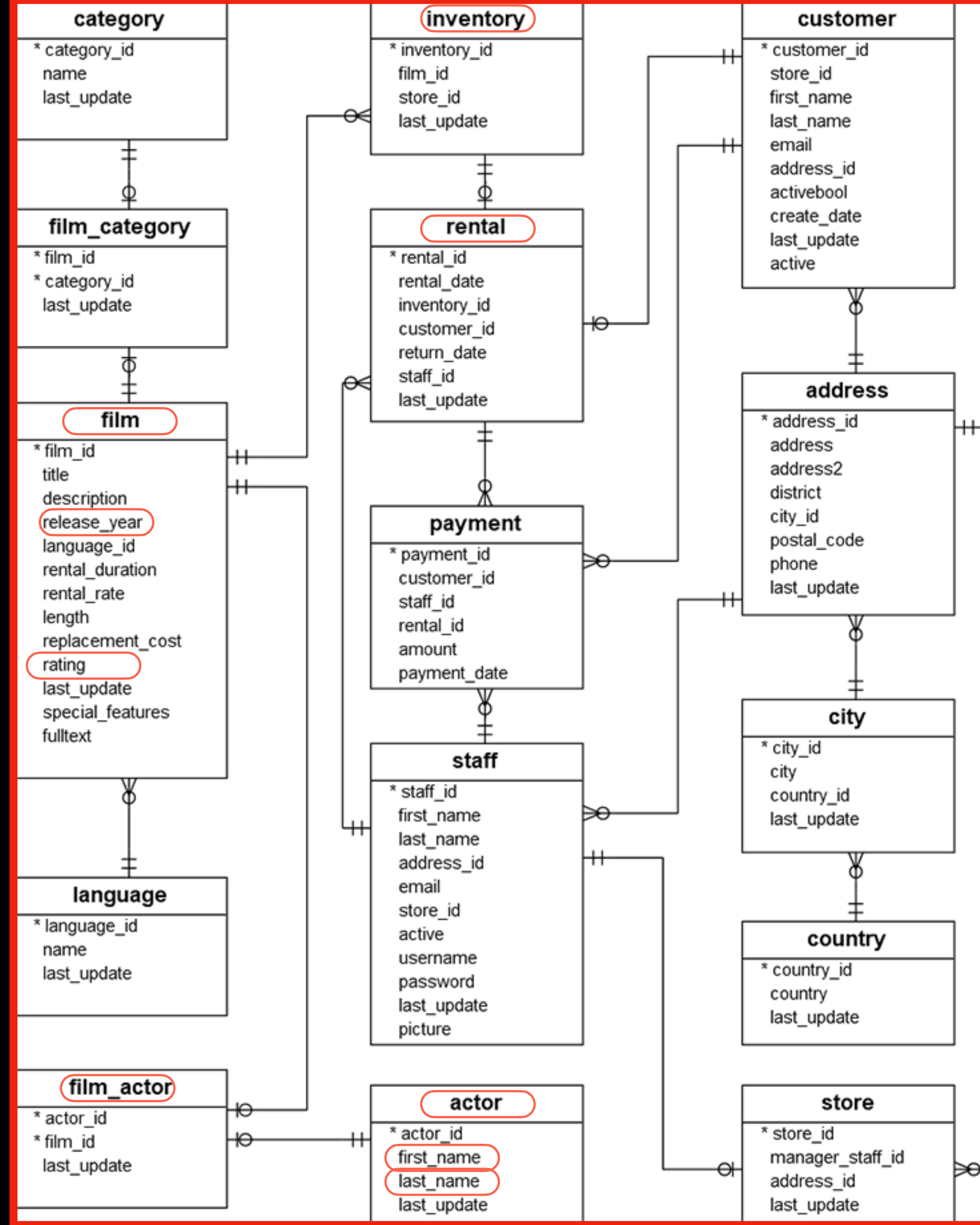
how do i combine the components of a SELECT?

"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 {a_condition}
– + GROUP BY {columns}
– + HAVING {a_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 {column1}
ORDER BY {column} DESC
LIMIT {num}
```

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

query plans and EXPLAIN
inverted indexes
ACID
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
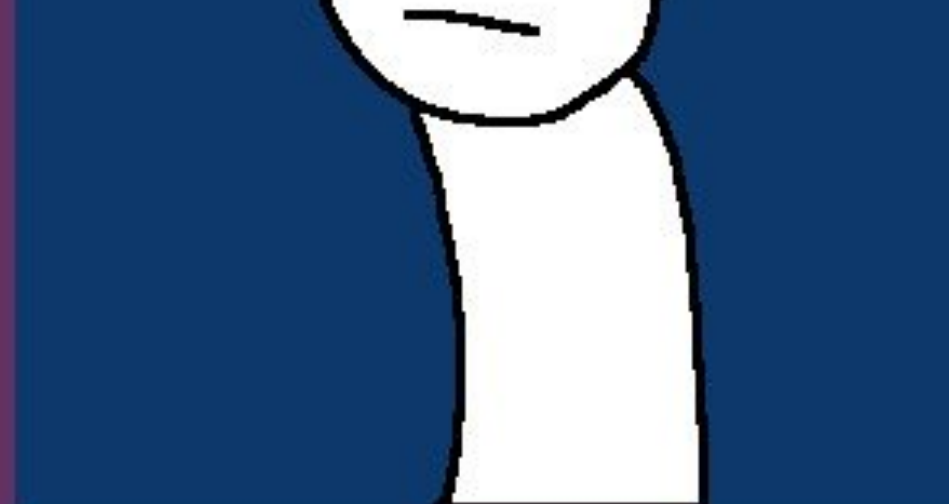recursive CTEs
ORMs create bad queries
stored procedures
cursors
there are no non-nullable types
plan hints
optimizers don't work without
table statistics
MVCC garbage collection

COUNT(*) vs COUNT(1)
generator functions zip
when cross joined
sharding
isolation levels
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
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
join ordering is NP hard
database cracking
WCOJ
XTID exhaustion
learned indexes

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

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

transaction contention

star schemas

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

DEFERRABLE INITIALLY IMMEDIATE

TPCC requires wait times

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

join ordering is NP hard

database cracking

WCOJ

learned indexes

XTID exhaustion

the halloween problem

dee and dum

SERIAL is non-transactional

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