from zero to query

a sql primer

oskar 2023-02-16

sql - a fundamental tool for the data professional

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

sql - a fundamental tool for the data professional

- 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 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				

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...

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 column1, column2 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
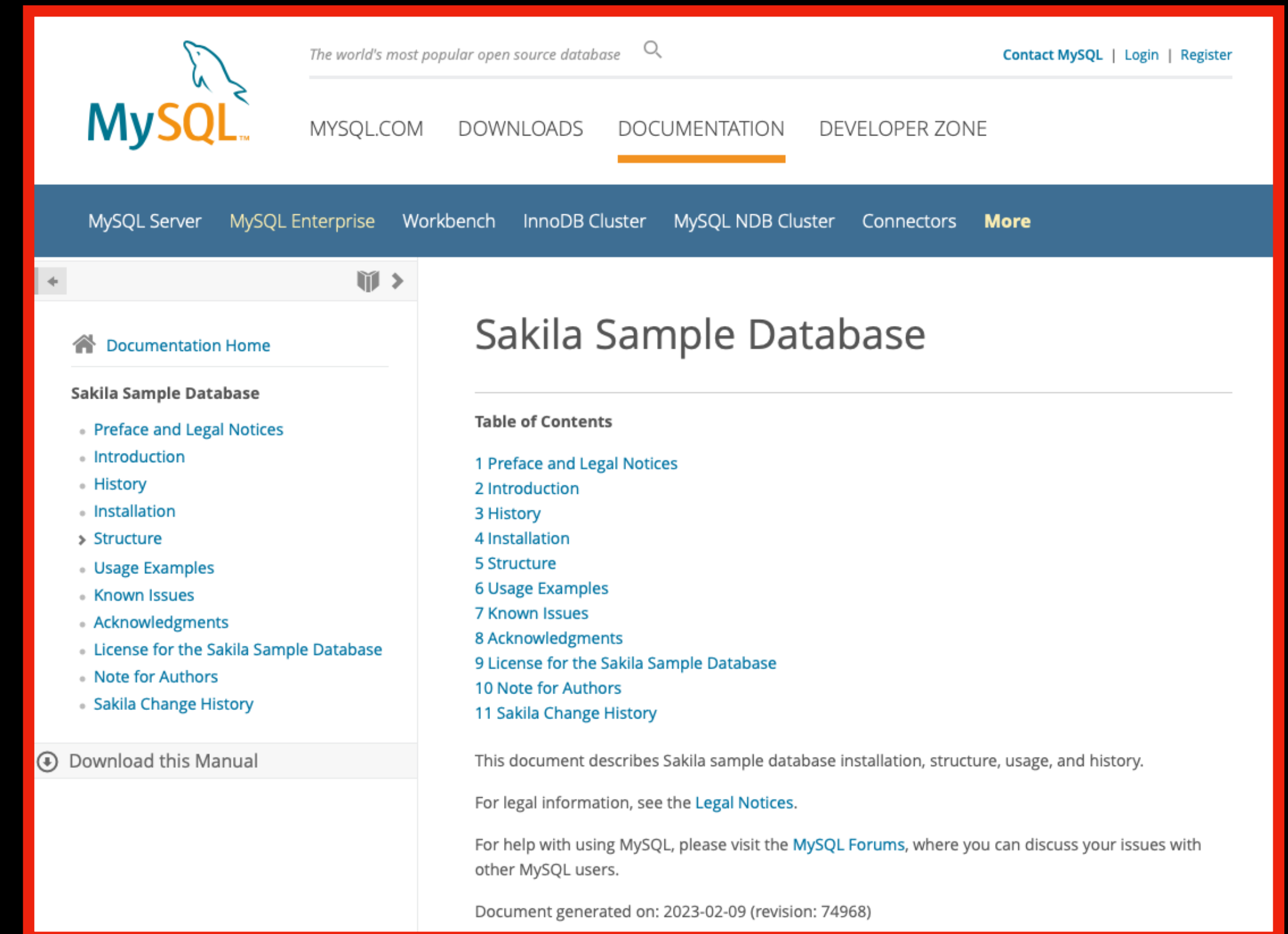
.tables

```
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> █
```

what tables are there?

"which top 10 actors were rented out
the greatest number of times, counting
only 'R' rated films made in 2006?"

```
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>
```
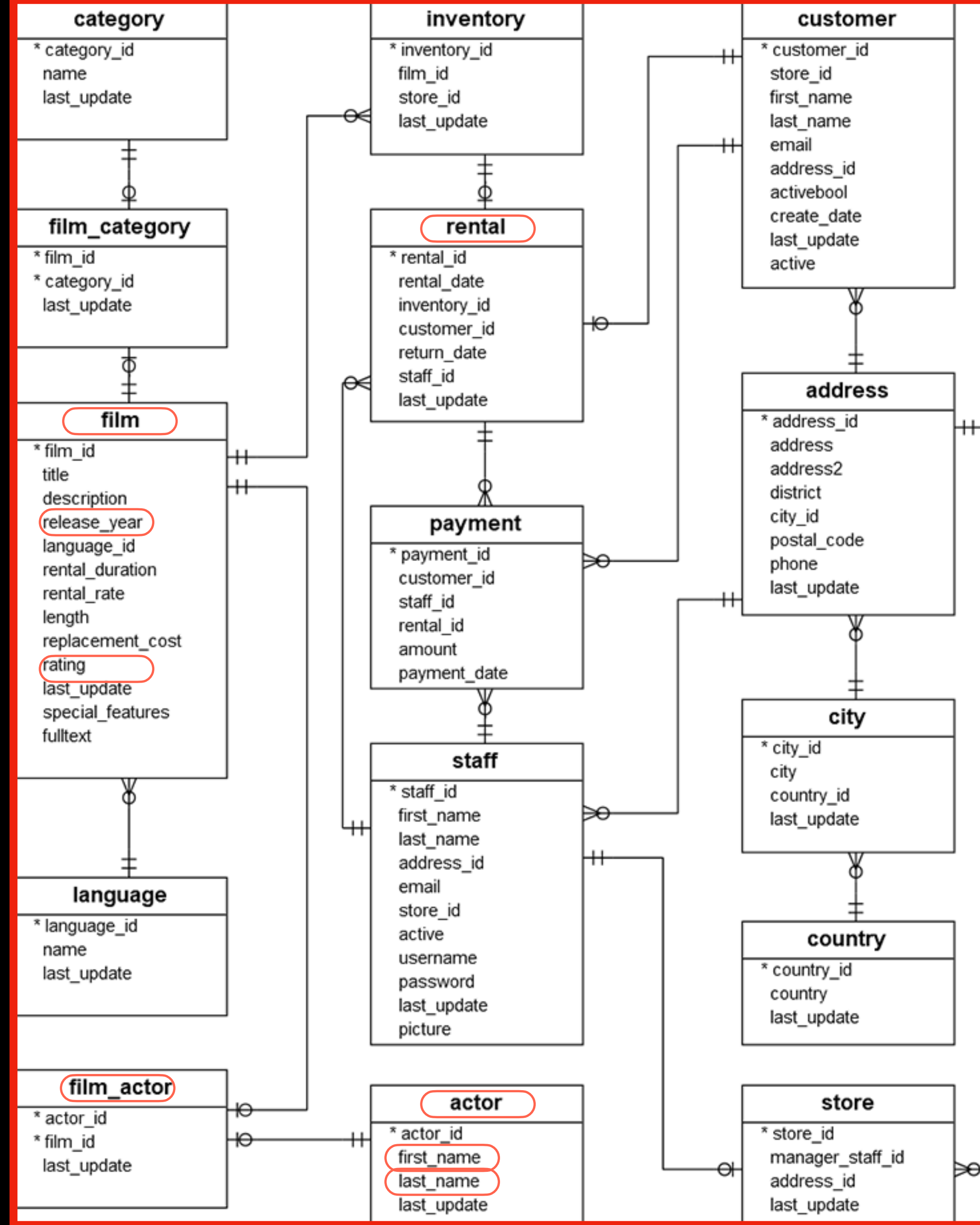
the sakila training data

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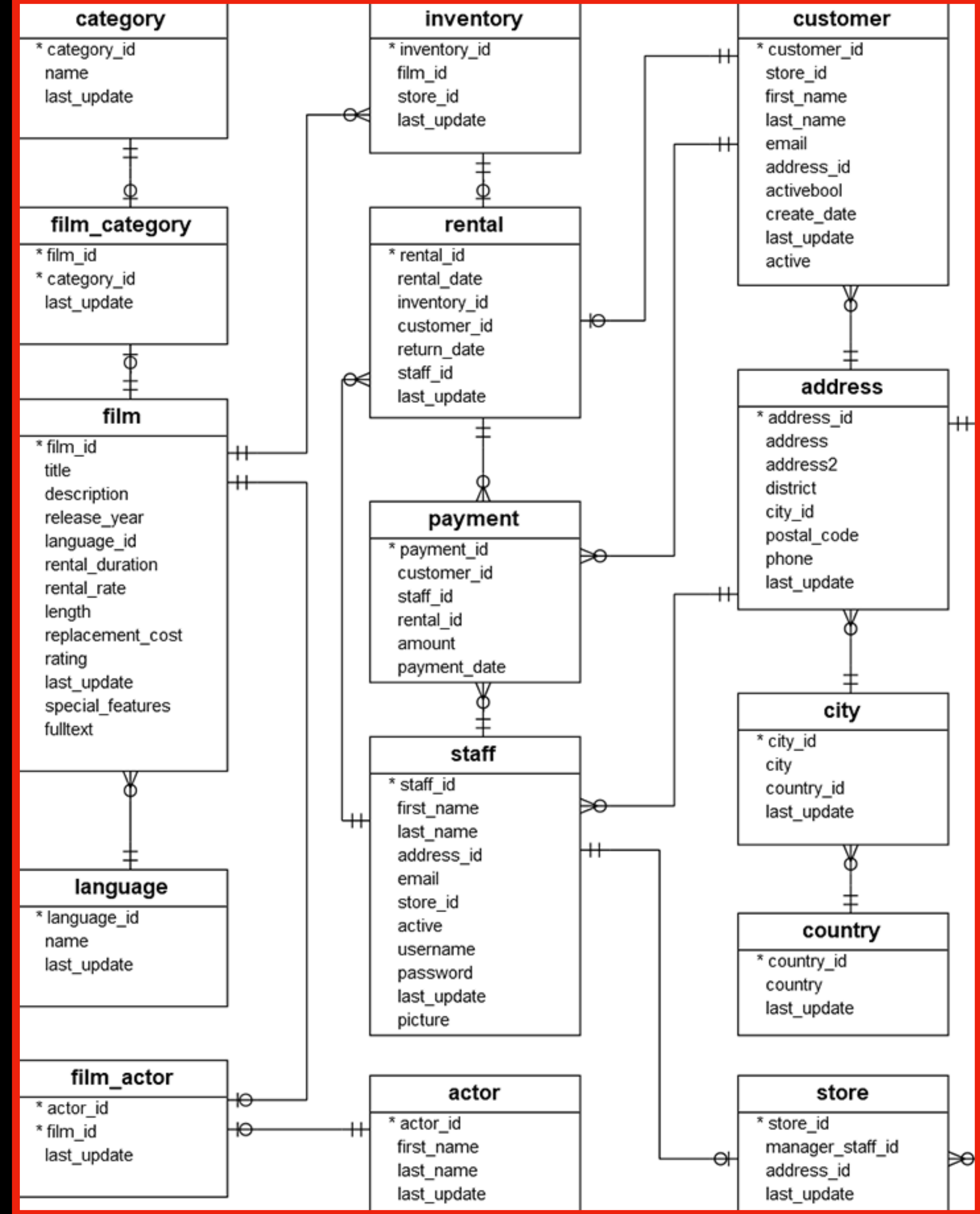
today's objective:

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

what do the tables contain?

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

SELECT … FROM …;

- SELECT * FROM {table};
  -- returns all columns and all rows from {table}

- SELECT col2, col1 FROM example_table;
  -- returns columns 'col2' and 'col1' (in that order) from example_table

- SELECT t.col2, t.col1 FROM table_name t;
  -- introduces an alias for table_name, refers to its columns, 'col2', 'col1'

- SELECT price + tax AS total_cost FROM sales;
  -- returns a single column, sum of price+tax, calls the output 'total_cost'

that's too many rows!

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

SELECT … FROM … LIMIT …;

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

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

- SELECT sale_date, sale_cost FROM sales LIMIT 15;
  -- returns 15 unspecified rows of two columns from sales table

- SELECT id AS region_id, name AS region_name FROM regions LIMIT 10;
  -- returns region id and region name for 10 unspecified rows

SELECT (aggregate function) FROM … ;

- SELECT COUNT(*) AS num_records FROM table_name;
  -- returns the number of rows in table_name, names the output 'num_records'

- SELECT SUM(s.sale_cost) AS total_sales FROM sales s;
  -- returns the sum of the sale cost column from the sales table

- SELECT AVG(s.sale_cost) AS average_sales FROM sales s;
  -- returns the average of the sale cost column from sales

- SELECT MAX(s.sale_cost) AS highest_value_sale FROM sales s;
  -- returns the highest value sale from sales

- SELECT MIN(s.sale_date) AS earliest_sale FROM sales s;
  -- returns the date of the earliest sale from sales

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

SELECT … FROM … WHERE … [LIMIT n];

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

- SELECT * FROM table_name WHERE column1<>{expression};
  -- returns only rows where the value in column1 is not {expression}

- SELECT name AS item_name FROM items WHERE item_price>=10;
  -- returns names of items whose price is greater than or equal to £10

- SELECT name FROM items WHERE item_price>=10 LIMIT 8;
  -- returns 8 of the items whose price is greater or equal to £10

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 sales WHERE sale_date BETWEEN '2023–02-01' AND '2023-02-04';
  -- returns only sales occurring between feb 1st and feb 4th, inclusive

- SELECT * FROM sales WHERE region_id IN (14,56,43) ;
  -- returns only sales in regions with id 14, 56, or 43

- SELECT * FROM region WHERE region_name LIKE '%new%';
  -- returns only regions whose name contains 'new'

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

SELECT … FROM … ORDER BY … LIMIT …;

```
- SELECT * FROM items ORDER BY item_cost LIMIT 10;
  -- return only the top 10 least expensive items in the catalog


- SELECT * FROM items ORDER BY item_cost DESC LIMIT 10;
  -- return only the top 10 most expensive items in the catalog
```

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

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
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

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-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, country
FROM
    city
    INNER JOIN country ON city.country_id=country.country_id
;
```
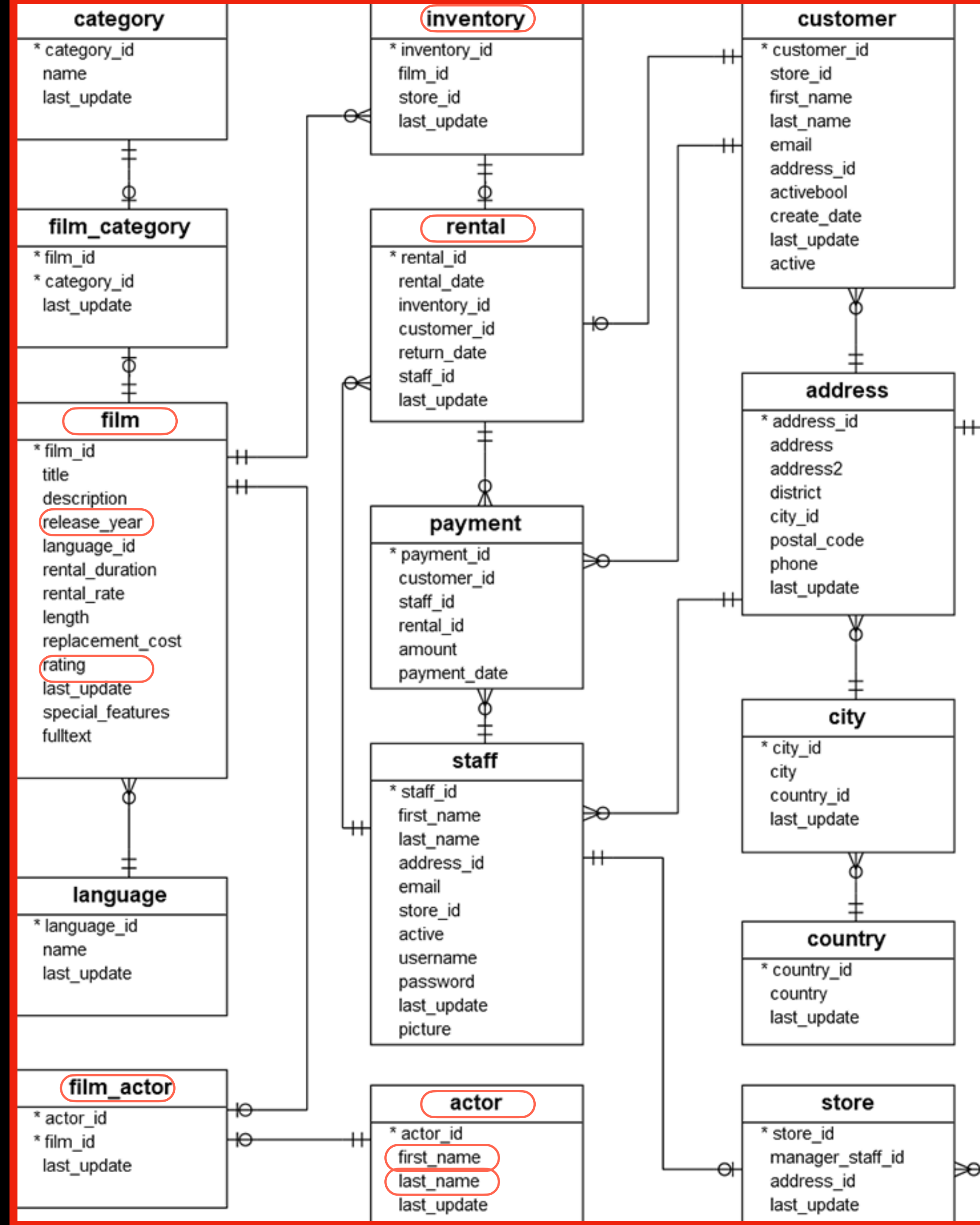
FROM a <type> JOIN b ON a.col=b.col
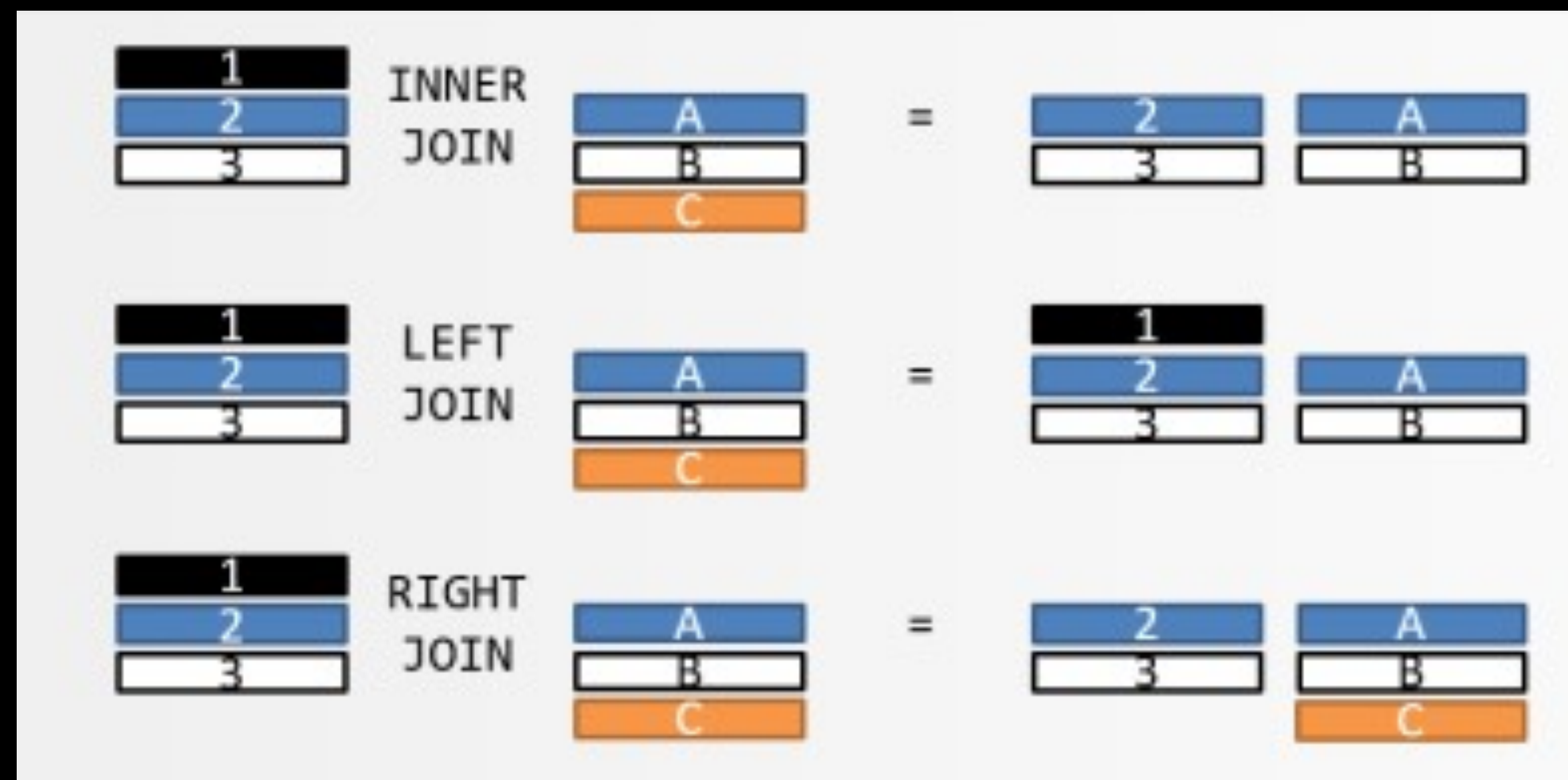
FROM a INNER JOIN b
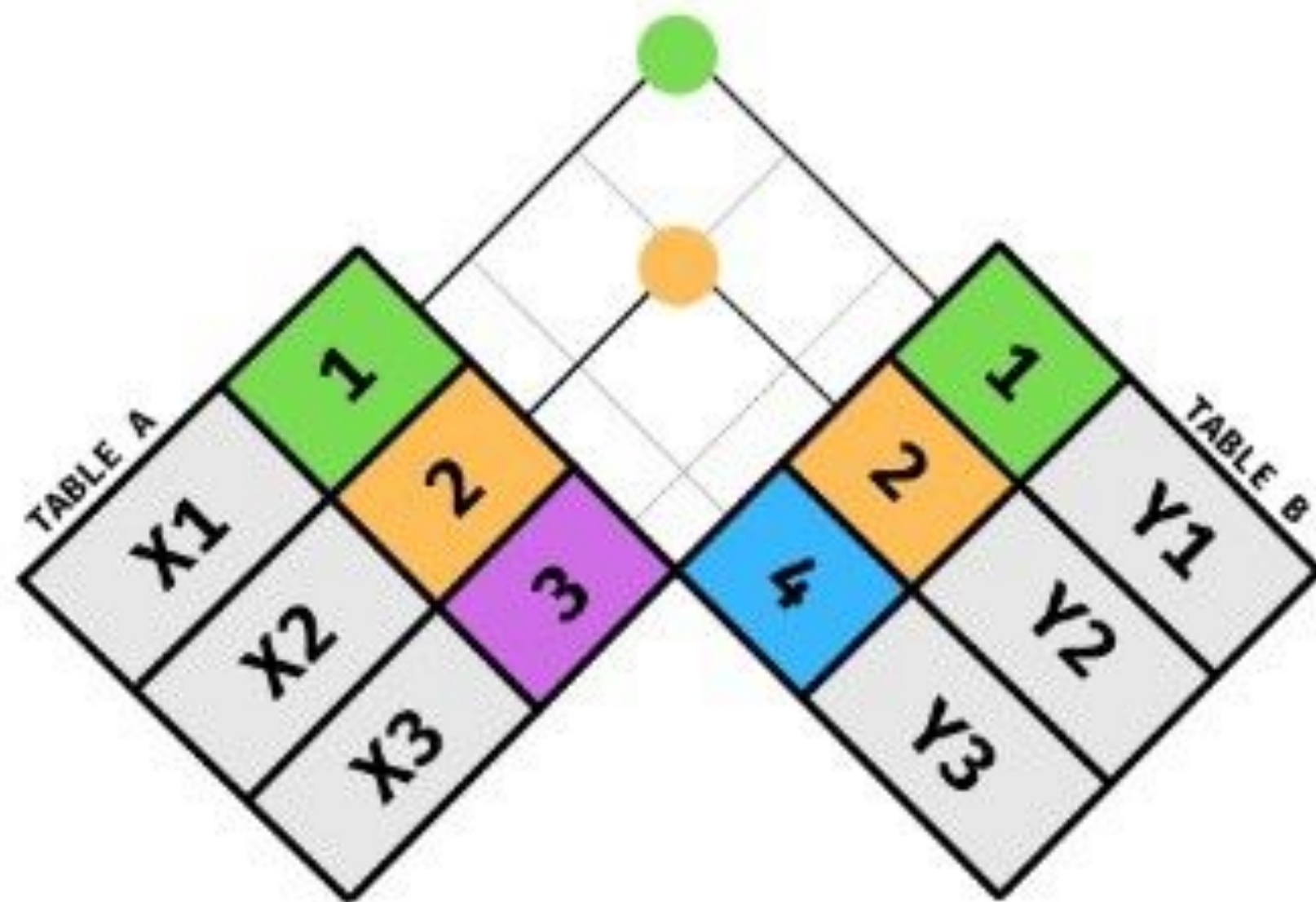
SELECT … FROM a INNER JOIN b ON …;

- 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

-

LEFT OUTER JOIN

how can i aggregate select 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};
- + LIMIT num
- + WHERE {a_condition}
- + ORDER BY {columns}
- + INNER JOIN {table_2} ON {col1}={col2}
- + GROUP BY {columns}
- + HAVING {a_condition}

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

- SELECT region_id, COUNT(*) FROM sales GROUP BY region_id;
  -- return each region's number of records from the sales table

- SELECT region_id, AVG(item_price) FROM items GROUP BY item_type;
  -- return the average price of items of each type from the items table

- SELECT item_type, MAX(item_price) FROM items GROUP BY item_type;
  -- returns the price of the priciest item of each type from the item table

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

how do i combine the components of a SELECT?

- SELECT {columns} FROM {table};
- + LIMIT num
- + WHERE {a_condition}
- + ORDER BY {columns}
- + INNER JOIN {table_2} ON {col1}={col2}
- + GROUP BY {columns}
- + HAVING {a_condition}
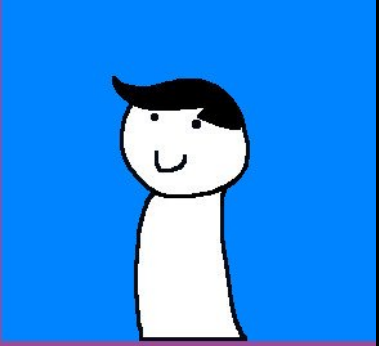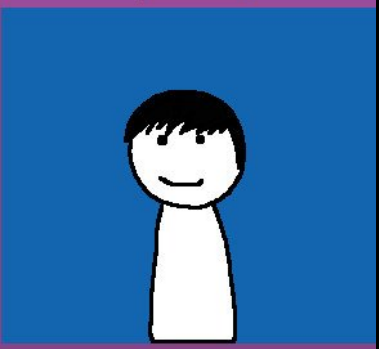
hint: structure of the solution

```sql
SELECT
    {}          AS actor_name,
    COUNT({}) AS num_rentals
FROM {table1}
    LEFT JOIN {table2} ON {join-condition}
    LEFT JOIN {table3} 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

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

inverted indexes
query plans and EXPLAIN
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
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
timestamptz doesn't store a timezone
sargability
ascending key problem
ambiguous network errors
utf8mb4

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

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

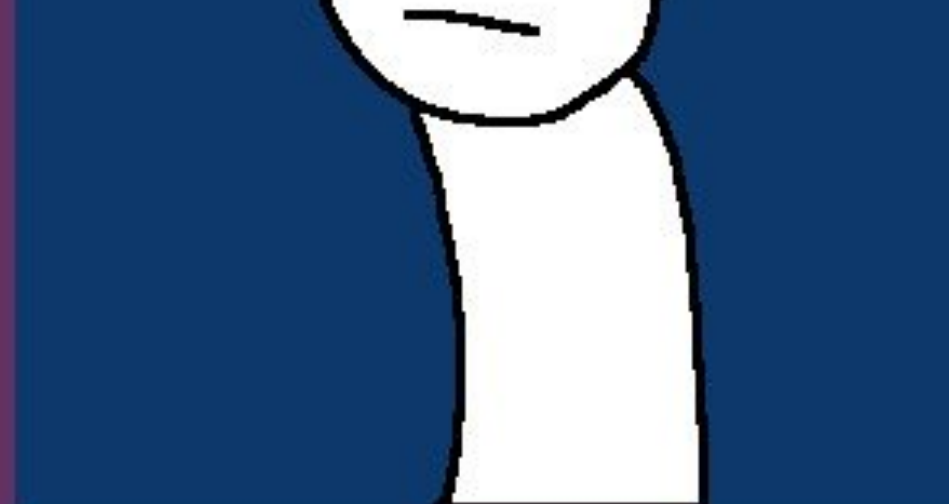there are no non-nullable types   plan hints   optimizers don't work without
                                                                  table statistics
MVCC garbage collection

COUNT(*) vs COUNT(1)   generator functions zip
                                    isolation levels   when cross joined   sharding

                                zigzag join
serializable restarts require   triggers
retry loops on all statements   phantom reads   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
                    sargability   store a timezone

ascending key problem   ambiguous network errors   utf8mb4

                        'null'::jsonb IS NULL = false
cost models don't   TPCC requires wait times
reflect reality
                        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

join ordering is NP hard

database cracking

WCOJ

learned indexes

XTID exhaustion

the halloween problem

SERIAL is non-transactional

dee and dum

allballs

every sql operator is
actually a join

fsyncgate

NULL

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

- two week free course online starting 2023-02-20:
https://corise.com/course/sql-crash-course