ASSIGNMENT 1 REPORT

Exercise 1

1 .& 2. (Primary and foreign keys)

Here we have all the tables with their attributes. In bold we can see the primary keys. Foreign keys are indicated with arrows.

actor(**actor_id**, first_name, last_name, last_update)

address(**address_id**, address, address2, district, city_id→ city.city_id, postal_code, phone, last_update)

category(**category_id**, name, last_update)

city(**city_id,** city, country_id→ country.country_id, last_update)

country(**country_id**, country, last_update)

customer(**customer_id**, store_id, first_name, last_name, email, address_id→ address.address_id, activebool, create_date, last_update, active)

film(**film_id**, title, description, release_year, language_id→ language.language_id, rental_duration, rental_rate, length, replacement_cost, rating, last_update, special_features, fulltext)

film_actor(**actor_id**→actor.actor_id, **film_id**→ film.film_id, last_update)

film_category(**film_id**→film.film_id, **category_id**→ category.category_id, last_update)

inventory(**inventory_id**, film_id→film.film_id, store_id, last_update)

language(**language_id**, name, last_update)

payment(**payment_id**, customer_id→ customer.customer_id, staff_id→ staff.staff_id, rental_id→ rental.rental_id, amount, payment_date)

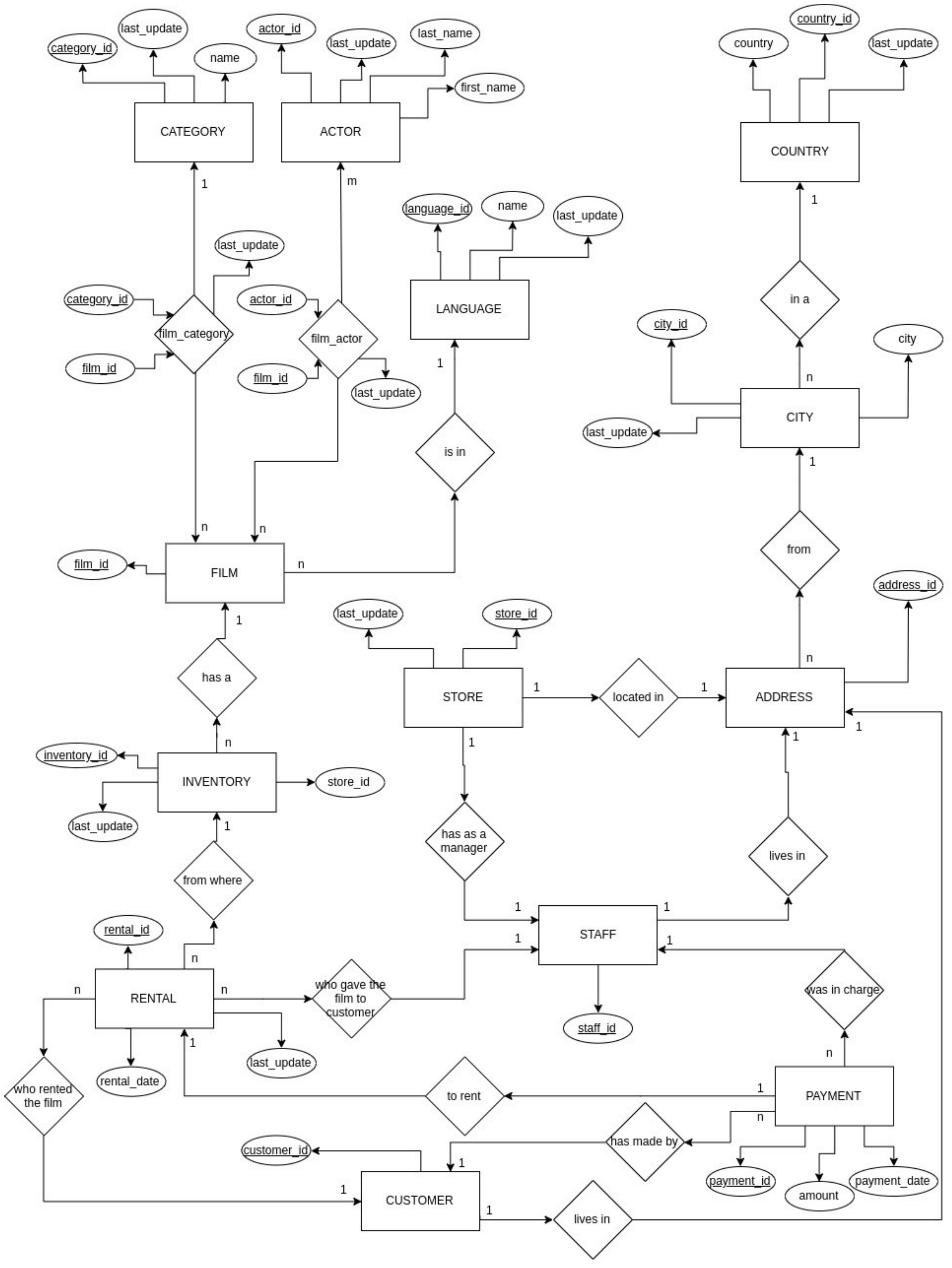
rental(**rental_id**, rental_date, inventory_id→inventory.inventory_id, customer_id→ customer.customer_id, return_date, staff_id→ staff.staff_id, last_update)

staff(**staff_id**, first_name, last_name, address_id→address.address_id, email, store_id, active, username, password, last_update, picture)

store(**store_id**, manager_staff_id→staff.staff_id, address_id→address.address_id, last_update)

3 (Database relational schema)

We have done this entity relational diagram in order to see how the database is created. Tables like film_category and film_actor are considered relations as they join one table with other one.



Exercise 2

In this exercise we had to implement some queries using SQL.

QUERY 1

```
SELECT Extract(year FROM rental_date),
   Count(*)
FROM  rental
GROUP BY Extract(year FROM rental_date)
```

In this query we need to extract the date from the rental_date attribute as they ask us to count the films rented each year.

QUERY 2

```
SELECT customer.customer_id,
   customer.first_name,
   customer.last_name,
   Count(*)
FROM  rental,
   customer
WHERE rental.customer_id = customer.customer_id
GROUP BY customer.customer_id,
    customer.first_name
HAVING Count(*) IN (SELECT Count(*)
         FROM  rental
         GROUP BY customer_id
         ORDER BY Count(*) DESC
         LIMIT 1)
```

First we need to select the maximum films that the users have watched. Then we compare that number with the count of the films ordered by customer_id, in order to have the customer with the biggest number of rentals. And with the id we get the first and last name.

QUERY 3

```
SELECT DISTINCT city.city_id,
      city.city
FROM  city,
   address,
   customer,
   rental,
   inventory,
   film,
   film_actor,
   actor
WHERE city.city_id = address.city_id
```

```
    AND address.address_id = customer.address_id
    AND customer.customer_id = rental.customer_id
    AND rental.inventory_id = inventory.inventory_id
    AND inventory.film_id = film.film_id
    AND film_actor.film_id = film.film_id
    AND film_actor.actor_id = actor.actor_id
    AND actor.first_name = 'Bob'
    AND actor.last_name = 'Fawcett'
ORDER BY city.city
```

In this query we have first compared the first name and last name of the actor and then we have done some joins between tables to get to the cities from where the clients, who rented Bob Fawcett films, are, as it was indicated in the Spanish assignment (The English one cause misunderstanding, it can be interpreted as the stores that have rented Bob Fawcett films).

QUERY 4

```
SELECT name
FROM  language
WHERE language_id IN(SELECT language_id
         FROM  film
         GROUP BY language_id
         HAVING Count(*) IN(SELECT Count(*)
                 FROM  film
                 GROUP BY language_id
                 LIMIT 1))
```

For this query we have counted all the languages and we have taken the maximum. Then we compare this number to the same count, to take the language_id that is the most used. We do it this way because if there was a tie between languages then both will appear, as both have the same number of films.

QUERY 5

```
SELECT language.name
FROM  language
WHERE language.language_id IN(SELECT film.language_id
             FROM  rental,
                inventory,
                film
             WHERE rental.inventory_id =
                inventory.inventory_id
                AND inventory.film_id = film.film_id
             GROUP BY film.language_id
             HAVING Count(film.language_id) IN
                (SELECT
                Count(film.language_id)
                            FROM  rental,
                inventory,
                film
                          WHERE
                rental.inventory_id =
   inventory.inventory_id
   AND inventory.film_id = film.film_id
   GROUP BY film.language_id
   ORDER BY Count(film.language_id) DESC
   LIMIT 1))
```

For this query first we need to count the films that have been rented grouped by the languages just to take the maximum. Then we compare that maximum with the count itself to get the language_id (in case there is a tie this will give us all languages in which a greater number of rentals have been done). Finally we just get the name from that language_id.

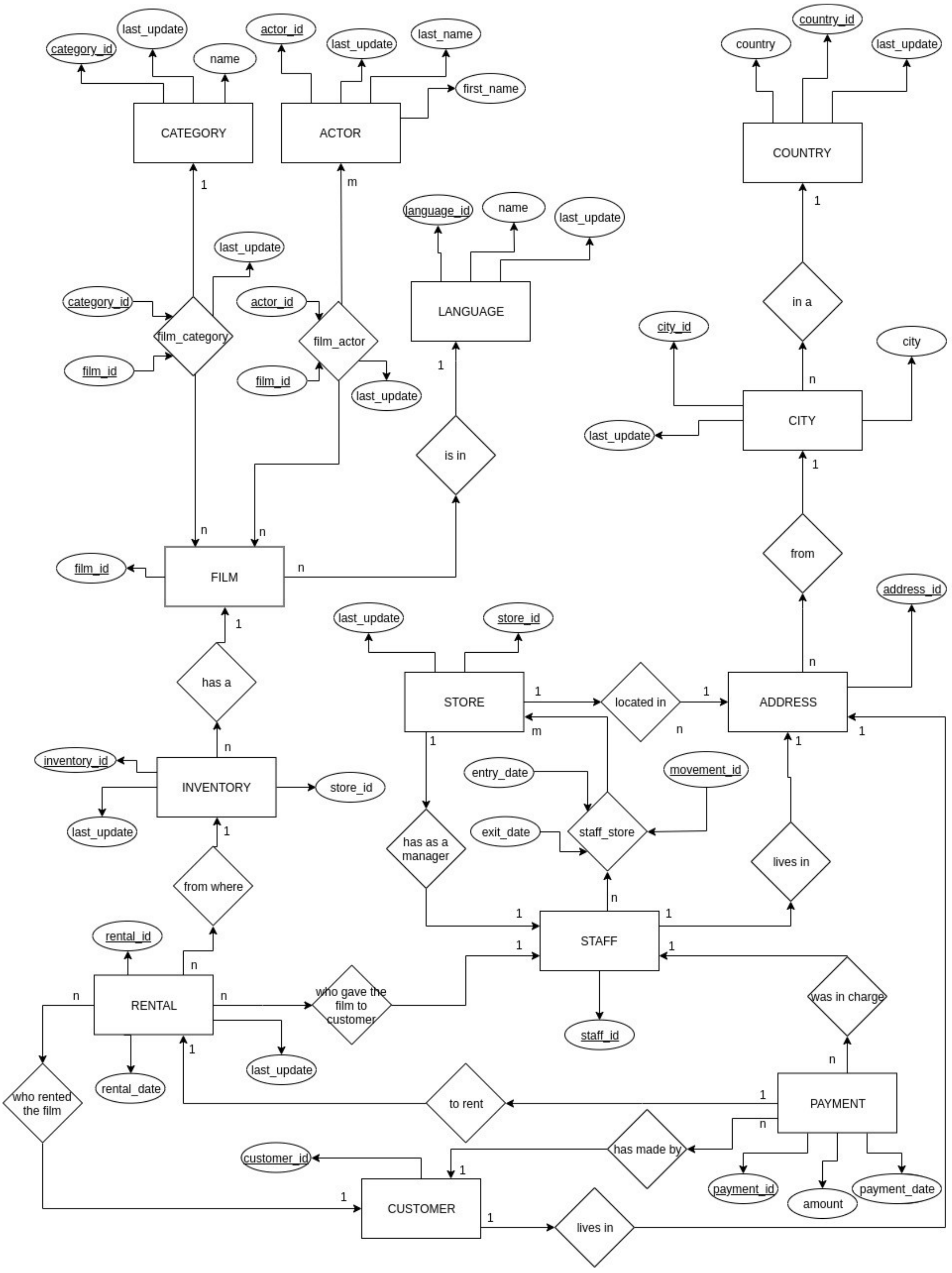
QUERY 6

```
SELECT rental.customer_id,
   category.name,
   Count(*)
FROM  inventory,
   rental,
   film,
   film_category,
   category
WHERE rental.customer_id IN(SELECT customer_id
            FROM  rental
            GROUP BY customer_id
            HAVING Count(*) IN (SELECT Count(*)
                  FROM  rental
                  GROUP BY customer_id
                  ORDER BY Count(*) DESC
                  LIMIT 1))
   AND rental.inventory_id = inventory.inventory_id
   AND inventory.film_id = film.film_id
   AND film.film_id = film_category.film_id
   AND film_category.category_id = category.category_id
GROUP BY category.category_id,
    rental.customer_id,
    category.name
HAVING ( customer_id, Count(*) ) IN (SELECT customer_id,
                    Max(x)
              FROM  (SELECT rental.customer_id,
                    category.category_id,
                    Count(*) x,
                    category.name
                FROM  inventory,
                    rental,
                    film,
                    film_category,
                    category
                    WHERE
      rental.customer_id IN(SELECT customer_id
               FROM  rental
               GROUP BY customer_id
               HAVING Count(*) IN (SELECT Count(*)
                        FROM  rental
                        GROUP BY customer_id
                        ORDER BY Count(*)
                        DESC
                        LIMIT 1))
      AND rental.inventory_id = inventory.inventory_id
      AND inventory.film_id = film.film_id
      AND film.film_id = film_category.film_id
      AND film_category.category_id = category.category_id
                    GROUP BY category.category_id,
                        rental.customer_id,
                        category.name
                    ORDER BY Count(*) DESC) tmp
                GROUP BY customer_id)
ORDER BY Count(*) DESC
```

This is the longest query because of the repetition of the same pattern, as we want to select two clients or more if there is a tie. First we have to count how many films have rented each customer, and select the maximum. Then we compare that maximum, with the count itself to get the customer_id who has made more rentals. Then we do the same for the categories, we count the maximum number of films in a category grouped by the customer_id, and then we compare that number with the count itself to get the favorite category of the customer (or customers) who has (or have) rented more movies.

Exercise 4

We have done a modification on the ER diagram from the first exercise. We have included a new table, staff_store, that works as a relation between the staff and the store. We have defined in this new table two foreign keys (staff_id→ staff.staff_id, store_id→ store.store_id) and then two attributes to determine from which date to other one, an employer have been working on a store. It has also a primary key which is **movement_id** to have a number that orders all movements between staff and stores.



CATEGORY
category_id
last_update
name
ACTOR
actor_id
last_update
last_name
first_name
COUNTRY
country
country_id
last_update
LANGUAGE
language_id
name
last_update
film_category
film_actor
in a
CITY
city_id
city
FILM
film_id
is in
from
STORE
store_id
located in
ADDRESS
address_id
has a
INVENTORY
inventory_id
store_id
staff_store
entry_date
exit_date
movement_id
has as a manager
lives in
from where
STAFF
staff_id
RENTAL
rental_id
rental_date
who gave the film to customer
was in charge
to rent
PAYMENT
payment_id
amount
payment_date
has made by
who rented the film
CUSTOMER
customer_id

The instructions needed to create this new table are:

```
CREATE SEQUENCE PUBLIC.staff_store_id_seq START WITH 1 INCREMENT BY
1 NO
MINVALUE NO MAXVALUE CACHE 1;

ALTER TABLE PUBLIC.staff_store_id_seq owner TO postgres;


CREATE TABLE public.staff_store
  (
     movement_id INTEGER DEFAULT NEXTVAL('public.staff_store_id_seq
'::regclass)
     NOT
     NULL,
     staff_id    INT NOT NULL,
     store_id    INT NOT NULL,
     entry_date  DATE NOT NULL,
     exit_date   DATE NOT NULL
  );

ALTER TABLE PUBLIC.staff_store owner TO postgres;

ALTER TABLE only PUBLIC.staff_store ADD CONSTRAINT staff_id_fkey FO
REIGN KEY (staff_id) REFERENCES PUBLIC.staff(staff_id)
ON
UPDATE CASCADE
ON
DELETE RESTRICT;

ALTER TABLE only PUBLIC.staff_store ADD CONSTRAINT store_id_fkey FO
REIGN KEY (store_id) REFERENCES PUBLIC.store(store_id)
ON
UPDATE CASCADE
ON
DELETE RESTRICT;
```