DATA STRUCTURE- First assignment

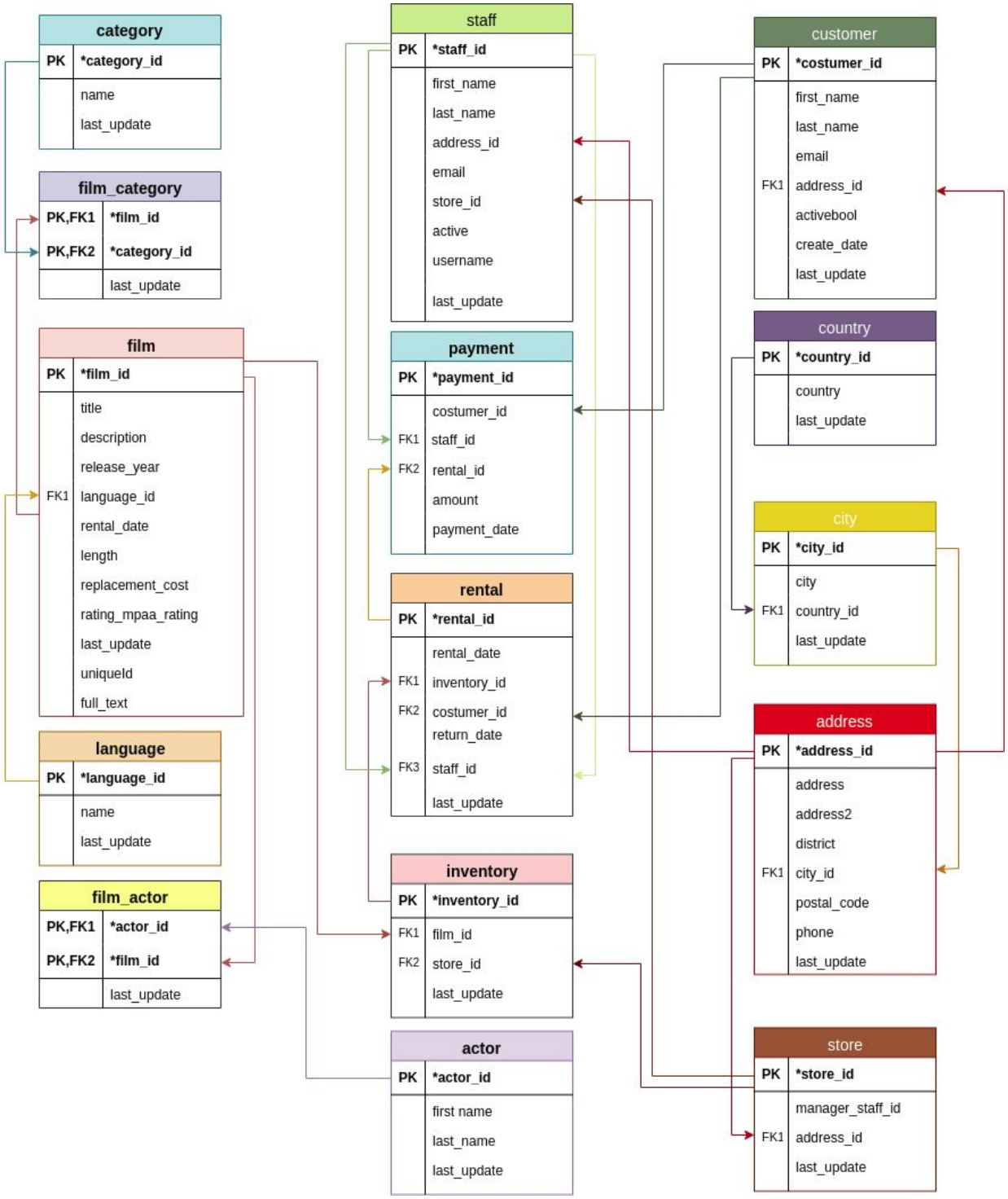
S. Xiaofan Fernández and Samai García (1292)

2. Explore the database

In the following lines we describe the different tables we have and the primary and foreign key for each one of them, in the format previously required:

actor (**actor_id**, first_name, last_name, last_uppdate)
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)
film (**film_id**, title, description, release_year, language_id → language.language_id, rental_duration, rental_rate, length, replacement_cost, rating mpaa_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.retnal_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, last_update)
store (**store_id**, manager_staff_id → staff.staff_id, address_id, last_update)

Then, we created the corresponding relational schema:


category
PK *category_id
name
last_update
film_category
PK,FK1 *film_id
PK,FK2 *category_id
last_update
film
PK *film_id
title
description
release_year
FK1 language_id
rental_date
length
replacement_cost
rating_mpaa_rating
last_update
uniqueId
full_text
language
PK *language_id
name
last_update
film_actor
PK,FK1 *actor_id
PK,FK2 *film_id
last_update
staff
PK *staff_id
first_name
last_name
address_id
email
store_id
active
username
last_update
payment
PK *payment_id
costumer_id
FK1 staff_id
FK2 rental_id
amount
payment_date
rental
PK *rental_id
rental_date
FK1 inventory_id
FK2 costumer_id
return_date
FK3 staff_id
last_update
inventory
PK *inventory_id
FK1 film_id
FK2 store_id
last_update
actor
PK *actor_id
first name
last_name
last_update
customer
PK *costumer_id
first_name
last_name
email
FK1 address_id
activebool
create_date
last_update
country
PK *country_id
country
last_update
city
PK *city_id
city
FK1 country_id
last_update
address
PK *address_id
address
address2
district
FK1 city_id
postal_code
phone
last_update
store
PK *store_id
manager_staff_id
FK1 address_id
last_update

3. Queries

In this part we were required to implement some queries on the database using SQL. All of them have been passed through a SQL formatter so that they are readable (http://www.dpriver.com/pp/ sqlformat.htm).

1. Number of movies rented each year. The query must return two attributes containing the year and the number of films rented. Order the answer by the rental year in ascending order.

```
SELECT Count(*)                          AS number_rented_films,
       Extract (year FROM rental_date) AS year --PICKS THE YEAR OF THE DATE
FROM   rental
GROUP  BY year
ORDER  BY ( year ) ASC --ORDERS THE YEAR ASC. AND COUNTS THE NUMBER OF
FILMS RENTED EACH OF THEM
```

2. Client who has rented more movies. If there is a tie between several clients, all clients with the maximum number of rented movies must appear in the response. The query must return the attributes: customer id, first name, last name and the number of rented movies.

```
SELECT first_name,
       last_name,
       customer.customer_id,
       count
FROM   customer,
       (SELECT customer_id,
               count
        FROM   (SELECT customer_id,
                       Count(*)
                FROM   rental
                GROUP  BY customer_id
                ORDER  BY Count (*) DESC) AS customer
        -- LIST OF HOW MANY MOVIES THE CUSTOMERS RENTED
        WHERE  count IN (SELECT Max(count)
                         FROM   (SELECT customer_id,
                                        Count(*)
                                 FROM   rental
                                 GROUP  BY customer_id
                                 ORDER  BY Count (*) DESC) AS x)) AS
id_client
--X = CUSTOMER; ID_CLIENT SELECTS THE TOP ONE
WHERE  id_client.customer_id = customer.customer_id -- CHOSE THE FIRST NAME,
LAST NAME, ID AND COUNT
```

3. List the cities where movies, in which “Bob Fawcett” appears, have been rented. Each city must appear just once. Sort the output alphabetically by the city name from A to Z. The query must return the city id and the city.name.

```
SELECT city.city_id,
       city.city
FROM   city,
       (SELECT address.address_id,
               address.city_id
        FROM   address,
               (SELECT store.store_id,
                       store.address_id
                FROM   store,
                       (SELECT inventory_id,
                               inventory.store_id
                        FROM   inventory,
                               (SELECT film.film_id
                                FROM   film,
                                       (SELECT film_id
                                        FROM   film_actor,
                                               (SELECT actor_id,
                                                       first_name
                                                FROM   actor
                                                WHERE  first_name = 'Bob'
                                                       AND last_name = 'Fawcett'
                                               ) AS
                                               actor_bob
                                        -- SELECTS THE NAME BOB AND LAST NAME
FAWCETT
                                        WHERE  film_actor.actor_id =
                                       actor_bob.actor_id) AS
                                       film_of_bob
                                --SELECTS ALL THE FILMS IN WHICH HE APPEARS
                                WHERE  film.film_id = film_of_bob.film_id) AS
                               inventory_bob
                        --SELECTS ALL THE MOVIES OF THE INVENTORY HE APPEARS
                        WHERE  inventory.film_id = inventory_bob.film_id) AS
                       store_bob
               WHERE  store_bob.store_id = store.store_id) AS store_id_bob
        --SELECTS ALL THE STORES THAT HAS THE MOVIES HE APPEARS
        WHERE  store_id_bob.address_id = address.address_id) AS city_bob
--SELECTS THE CITY ID IN WHICH THE STORE THAT HAS HIS FILMS ARE
WHERE  city_bob.city_id = city.city_id
GROUP  BY city.city_id
ORDER  BY city.city ASC -- SELECTS THE NAME OF THE CITIES IN ALPHABETICAL ORDER
```

4. Language in which most of the films have been filmed. The query must return the language.name attribute. If there is a tie between several languages, all languages in which the greatest number of films have been filmed should appear in the answer.

```
SELECT NAME
FROM   language,
       (SELECT language_id,
               order_count
        FROM   (SELECT language_id,
                       Count(*) AS order_count
                FROM   film
                GROUP  BY language_id
                ORDER  BY Count(*) DESC) AS aviable_languages
        --SELECTS ALL THE AVAILABLE LANGUAGES
        WHERE  order_count IN (SELECT Max(order_count)
                               FROM   (SELECT language_id,
                                              Count(*) AS order_count
                                       FROM   film
                                       GROUP  BY language_id
                                       ORDER  BY Count(*) DESC) AS x)) AS
       id_movie -- SELECTS THE TOP LANGUAGE IN WHICH MOVIES HAVE BEEN
RECORD
WHERE  language.language_id = id_movie.language_id --SELECTS THE NAME OF THE
TOP LANGUAGE MOVIES HAVE BEEN RECORD
```

5. Language (of the films) in which a greater number of rentals has been done. The query must return the language.name attribute. If there is a tie between several languages, all languages in which a greater number of rentals has been done must appear in the answer.

```
SELECT z.NAME
FROM   (SELECT x.NAME,
               Count(*) AS most_rented_movie
        FROM   (SELECT film_rental.NAME
                FROM   rental,
                       (SELECT inventory_id,
                               film_inventory.NAME
                        FROM   inventory,
                               (SELECT film_id,
                                       film_name.NAME
                                FROM   film,
                                       (SELECT language_id,
                                               NAME
                                        FROM   language
                                        GROUP  BY language_id) AS film_name
                                WHERE  film.language_id = film_name.language_id)
                               AS
                               film_inventory
                       --SELECTS THE FILM ID AND THE NAME OF THE LANGAGE IT HAS
BEEN RECORDED
```

```
                        WHERE  inventory.film_id = film_inventory.film_id) AS
                       film_rental
                --SELECTS THE ID OF THE INVENTORY THAT HAS THAT FILM
                WHERE  film_rental.inventory_id = rental.inventory_id) AS x
        --IF THE MOVIE HAS BEEN RENTED AND IT IS IN THE INVENTORY IT TAKES THE NAME
OF IT
        GROUP  BY x.NAME
        ORDER  BY most_rented_movie DESC) AS z
--SELECTS ALL THE TIMES A FILM HAS BEEN RECORD IN A LANGUAGE
WHERE  most_rented_movie IN (SELECT Max(most_rented_movie)
                             FROM   (SELECT x.NAME,
                                            Count(*) AS most_rented_movie
                                     FROM    (SELECT film_rental.NAME
                                              FROM   rental,
                                                     (SELECT inventory_id,
                                                             film_inventory.NAME
                                                      FROM   inventory,
                                                             (SELECT film_id,
film_name.NAME
FROM   film,
(SELECT language_id,
        NAME
 FROM   language
GROUP  BY language_id) AS film_name
WHERE  film.language_id =
film_name.language_id)
AS
film_inventory
--SELECTS THE FILM ID AND THE NAME OF THE LANGAGE IT HAS BEEN RECORDED
WHERE  inventory.film_id = film_inventory.film_id) AS
film_rental
--SELECTS THE ID OF THE INVENTORY THAT HAS THAT FILM
WHERE  film_rental.inventory_id = rental.inventory_id) AS x
--IF THE MOVIE HAS BEEN RENTED AND IT IS IN THE INVENTORY IT TAKES THE NAME OF IT
GROUP  BY x.NAME
ORDER  BY most_rented_movie DESC)AS y)--Y=SELECTS ALL THE TIMES A FILM HAS BEEN
RECORD IN A LANGUAGE
---PICKS THE TOP LANGUAGE A FILM HAS BEEN RECORDED
```

6. Favorite category (category.name) of the customer who has rented more movies. By favorite category we refer to the category in which the client has made more rentals. If a client rents the same movie twice it should count as two rentals. The query must return the name (category.name) and the identifier (category id) of the category. If there is a tie between several clients, all clients who have rented more films should appear in the response.

```sql
CREATE OR replace VIEW customer_and_num_rented_movies
AS
  SELECT customer_id,
         Max(count) AS num_film
  FROM   (SELECT customer_id,
                 category_id,
                 Count(category_id)
          FROM   (SELECT top_customer.customer_id,
                         film_category.category_id
                  FROM   inventory,
                         rental,
                         film_category,
                         (SELECT customer_id
                          FROM   (SELECT customer_id,
                                         Count(*)
                                  FROM   rental
                                  GROUP  BY customer_id
                                  ORDER  BY Count (*) DESC) AS customer
                          WHERE  count IN (SELECT Max(count)
                                           FROM   (SELECT customer_id,
                                                          Count(*)
                                                   FROM   rental
                                                   GROUP  BY customer_id
                                                   ORDER  BY Count (*) DESC) AS
                                                  x)
                         )AS
                         top_customer --TOP CUSTOMERS
                  WHERE  rental.inventory_id = inventory.inventory_id
                         AND rental.customer_id = top_customer.customer_id
                         AND inventory.film_id = film_category.film_id) AS
                 custumer_and_category
          -- PICKS THE CATEGORY ID OF ALL TH EMOVIES THE TOP CUSTOMERS RENTED
          GROUP  BY category_id,
                    customer_id
          ORDER  BY category_id ASC) AS count_category
  --COUNTS ALL THE TIMES A DIFFERENT CATEGORY HAS BEEN RENTED FROM A DIFFERENT CUSTOMER
  GROUP  BY customer_id; -- IT GIVES YOU THE MOST RENTED CATEGORY OF THE TOP CUSTOMERS
CREATE OR replace VIEW cost_and_category_id_count
AS
  SELECT customer_and_num_rented_movies.customer_id,
         count_category.category_id,
         count_category.count
  FROM   customer_and_num_rented_movies,
         (SELECT customer_id,
                 category_id,
                 Count(category_id)
          FROM   (SELECT top_customer.customer_id,
                         film_category.category_id
                  FROM   inventory,
                         rental,
                         film_category,
                         (SELECT customer_id
                          FROM   (SELECT customer_id,
                                         Count(*)
                                  FROM   rental
                                  GROUP  BY customer_id
                                  ORDER  BY Count (*) DESC) AS customer
```

```
                        WHERE  count IN (SELECT Max(count)
                                       FROM   (SELECT customer_id,
                                                      Count(*)
                                               FROM   rental
                                               GROUP  BY customer_id
                                               ORDER  BY Count (*) DESC) AS
                                              x)
                              )AS
                              top_customer --TOP CUSTOMERS
                       WHERE  rental.inventory_id = inventory.inventory_id
                              AND rental.customer_id = top_customer.customer_id
                              AND inventory.film_id = film_category.film_id) AS
                     custumer_and_category
          -- PICKS THE CATEGORY ID OF ALL TH EMOVIES THE TOP CUSTOMERS RENTED
          GROUP  BY category_id,
                    customer_id
          ORDER  BY category_id ASC) AS count_category
  --COUNTS ALL THE TIMES A DIFFERENT CATEGORY HAS BEEN RENTED FROM A DIFFERENT CUSTOMER
  WHERE  customer_and_num_rented_movies.num_film = count_category.count
         AND customer_and_num_rented_movies.customer_id =
             count_category.customer_id -- PICKS THE CUSTOMER ID AND THE CATEGORY ID AND SHOWS
HOW MANY TIMES A CATEGORY HAS BEEN RENTED

SELECT * FROM customer_and_num_rented_movies
```

4. **DataBase Redesign**

In this exercise we were required to change the database so that if a staff member is transferred from one store to another, the information about the stores he had worked before are kept.

We added the following queries to the newdatabase.sql file:

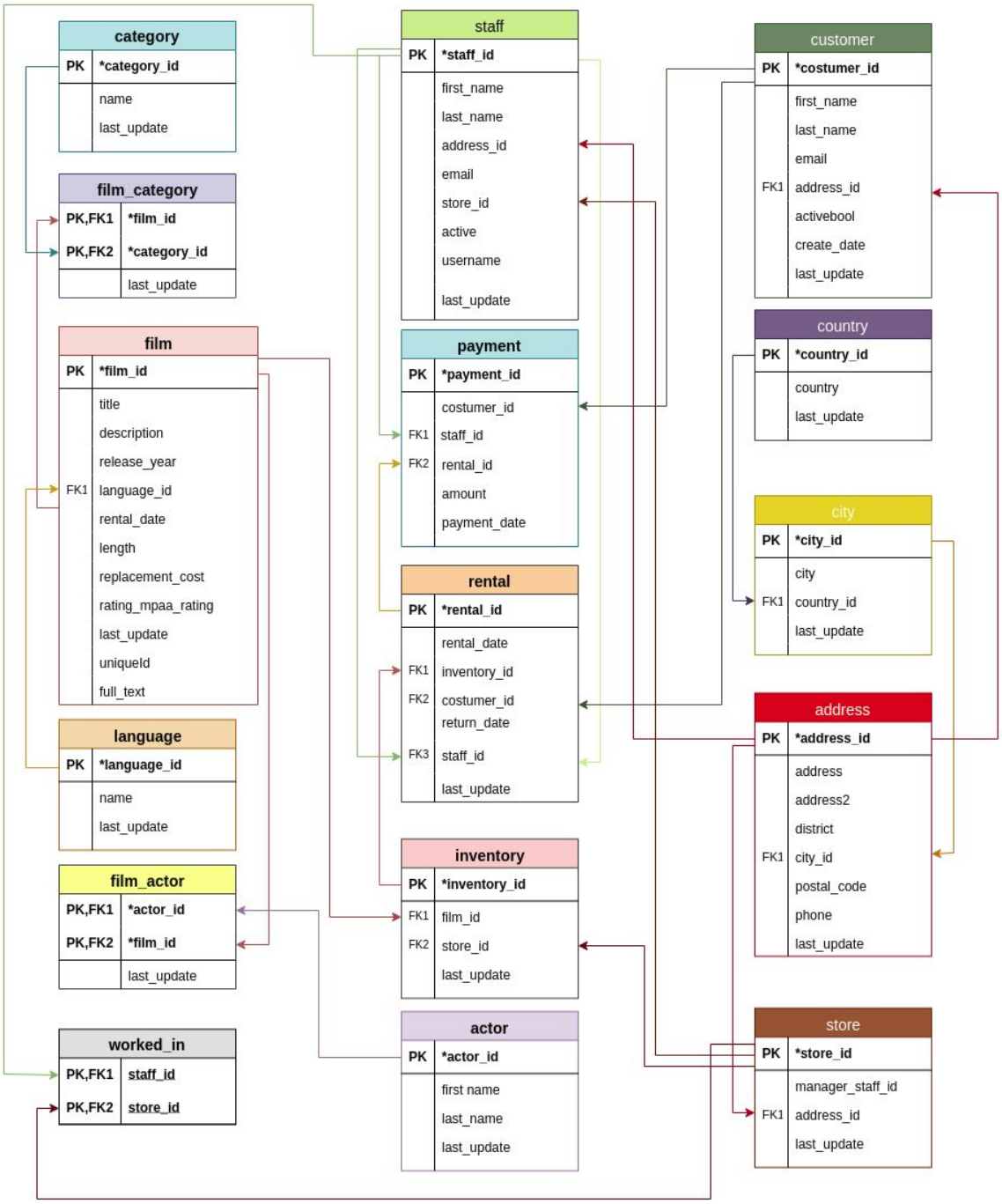
```
CREATE TABLE worked_in (
        staff_id integer NOT NULL REFERENCES staff(staff_id),
        store_id integer NOT NULL REFERENCES store(store_id),
        PRIMARY KEY (staff_id, store_id)
);
```

To create a new table that has as primary key the staff id and the store id. That are foreing keys at the same time.

```
INSERT INTO worked_in VALUES
        (1,     1);
        (1,     2);
        (2,     1);
```

We added those values at the table. The worker 1 has worked in the store 1 and 2. The number 2 has worked only in the first one.

(1) We added the table **worked_in** to store the required information. This table contains foreign keys connected to **staff** and **store**.



(2)