DATA BASES

CAL3

LAURA MAMBRILLA MORENO
ISABEL BLANCO MARTÍNEZ
JESÚS GONZÁLEZ SÁNCHEZ

Universidad de Alcalá

INTRODUCTION

In order to make this practice workable, it was necessary to change the relationships *cashier-worker* and *repletener-worker* from a relation of heritage to a relation 1:1. That means that some of the queries of the CAL2 needed to be changed in order to select the information properly.

We have also changed the option of NotNull (we have checked it out) in priceVat of *product* and in pointsAccumulated of *member*.

1. NORMALIZATION

supermarket (city, district) BCNF ok

worker (ID, name, phone, timeTable, type, city_supermarket, district_supermarket) BCNF ok

phone (ID_worker, phone) BCNF ok

member (membershipCode, name, email, address, telephoneNumber, pointsAccumulated) BCNF ok

telephoneNumber (telephoneNumber, membershipCode_member) BCNF ok

coupon (ID, percentage) BCNF ok

opinion (date, time, score, membershipCode_member) BCNF ok

product (barCode, name_product, priceVAT, priceNotVAT, week_discount, startDate, endDate) BCNF ok

cashier (ID, ID_worker) BCNF ok

repletener (ID, ID_worker, numberHours) BCNF ok

ticket (ID, date, timeIssuance, coupon_used, totalPrice, ID_cashier, membershipCode_member, usesAccNum, quantity) BCNF ok

discount (week, percentage) BCNF ok

When making the normalization we realised that after doing the PgModeler part, all the tables had to be on BCNF (that means that all of them accomplish 1NF, 2NF and 3NF) because all of the multivalued attributes have its own table created.
After doing the normalization, you can see that all the relations are attributes dependents on all the superkeys, that is BCNF.

There is an exception in the entity supermarket because it has no relations, so we considered that it accomplishes BCNF.

2. TRIGGERS

- Trigger Functions (7)
  - amount_accumulated()
  - avg_grade()
  - increase_stock()
  - insert_product()
  - reduce_stock()
  - total_price()
  - vat_price()

INSERT PRODUCT

- Trigger used to set the quantity of products in stock for each of the products introduced in the database.

```
CREATE FUNCTION insert_product() returns trigger
as
$$
begin
update product
set stock = 10;
return new;
end
$$

language plpgsql;

CREATE TRIGGER insert_product after insert on product for each row execute procedure insert_product();
```

Data Output Explain Messages Notifications

	barCode [PK] integer	name_product character (40)	priceNotVAT real	priceVAT real	stock integer	week_discount integer	startDate date	endDate date
1	123456789	Bread ...	0.38	0.4598	10	40	2019-02-12	2019-08-12
2	123456701	Serrano ham ...	9.2	11.132	10	4	2019-09-12	2019-12-15
3	123111701	Nougat, ...	5	6.05	10	40	2019-12-13	2019-12-20
4	123121701	Pantene Shampoo ...	1.88	2.2748	10	23	2019-01-12	2019-01-19

REDUCE STOCK (NOT WORKING)

- Trigger used to reduce the quantity of products in stock for each of the products purchased on the ticket

```
CREATE FUNCTION reduce_stock() returns trigger
as
$$
begin
update product
set stock = stock - muchos_ticket_tiene_muchos_product.amount
where "barCode_product" = "barCode".product;
```

```
return new;
end
$$

language plpgsql;

CREATE TRIGGER reduce_stock after insert on muchos_ticket_tiene_muchos_product for each row execute
procedure reduce_stock();
```

INCREASE STOCK (NOT WORKING)

- Trigger used to increase the quantity of products in stock for each of the products returned.

```
CREATE FUNCTION increase_stock() returns trigger
as
$$
begin
update product
set stock = stock + muchos_ticket_tiene_muchos_product.amount
where "barCode_product" = "barCode".product;
return new;
end
$$

language plpgsql;

CREATE TRIGGER increase_stock after insert on muchos_ticket_tiene_muchos_product for each row execute
procedure increase_stock();
```

PRICE VAT

- Trigger used to calculate the total price of each product taking into account the 21% VAT.

```
CREATE FUNCTION VAT_price() returns trigger
as
$$
begin
update product
set "priceVAT" = "priceNotVAT" + "priceNotVAT" * 0.21;
return new;
end
$$

language plpgsql;

CREATE TRIGGER VAT_price after insert on product for each row execute procedure VAT_price();
```

CAL3/postgres@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | Messages | Notifications

```
INSERT INTO public.product(
    "barCode", name_product, "priceNotVAT", "priceVAT", stock, week_discount, "startDate", "endDate")
    VALUES (567943205, 'Salt', 2, NULL, NULL, 18, NULL, NULL);
```

public.product/CAL3/postgres@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | Messages | Notifications

	barCode [PK] integer	name_product character (40)	priceNotVAT real	priceVAT real	stock integer	week_discount integer	startDate date	endDate date
1	567943205	Salt ...	2	2.42	10	18	[null]	[null]
2	123456789	Bread ...	0.38	0.4598	10	40	2019-12-02	2019-12-08
3	123456701	Serrano ham ...	10.4	12.584	10	4	2019-12-09	2019-12-15
4	123111701	Nougat, ...	5	6.05	10	40	2019-12-13	2019-12-20
5	123121701	Pantene Shampoo ...	1.88	2.2748	10	23	2019-01-12	2019-01-19
6	123347701	Goat Cheese ...	4.1	4.961	10	22	2018-05-24	2018-05-31
7	456456789	Nivea cream ...	4.5	5.445	10	40	2019-12-02	2019-12-08
8	789456789	Chicken Pie ...	3.01	3.6421	10	22	2019-05-24	2019-05-30
9	147456789	Barceló 75cl. ...	14.2	17.182	10	18	2019-05-01	2019-05-05
10	123456715	Lamb leg ...	9.2	11.132	10	4	[null]	[null]
11	123456720	Loin Spetec ...	16.34	19.7714	10	23	[null]	[null]
12	123456733	Jack Daniel's 1L ...	25.6	30.976	10	22	[null]	[null]
13	123456731	Prawns 5kg. ...	6.56	7.9376	10	18	[null]	[null]

AVERAGE GRADE

- Trigger used to calculate the average grade of the workers.

```
CREATE OR REPLACE FUNCTION avg_grade() RETURNS TRIGGER AS $avg_grade$
        DECLARE
  BEGIN
   UPDATE worker SET "averageGrade" =
        (SELECT AVG(grade) FROM muchos_worker_tiene_muchos_worker    INNER JOIN worker ON
"ID"="ID_worker1" WHERE ("ID" = new."ID_worker1"))
        WHERE "ID" = new."ID_worker1";

   RETURN NULL;
  END;
$avg_grade$
LANGUAGE plpgsql;

CREATE TRIGGER avg_grade AFTER INSERT or UPDATE
  ON "muchos_worker_tiene_muchos_worker" FOR EACH ROW
  EXECUTE PROCEDURE avg_grade();
```

Data Output Explain Messages Notifications

	ID [PK] character (9)	name character (40)	phone integer	timeTable character (1)	averageGrade double precision	city_supermarket character (40)	district_supermarket character (40)	type character (40)
1	88888888J	Andrea	610425367	B	7.333333333333333	Málaga	Palma-Palmilla	R
2	77777777G	Álvaro	610423567	B	6.5	Alcalá de Henares	Espartales	R
3	45872396L	Ander	610535637	B	5	Granada	Albaicín	R
4	11111111A	Aarón	611111111	A	5	Madrid	Gran Vía	C
5	44444444D	Angy	601234567	A	6.75	Madrid	Carabanchel	C
6	55555555E	Adrián	601243567	A	2.5	Mallorca	Arenal	C
7	05794628M	Daniel	613655637	B	5	Alcalá de Henares	Espartales	R
8	22222222B	Bernardo	622222222	B	6.333333333333333	Madrid	Carabanchel	C
9	99999999K	Ramón	610535367	A	8	Murcia	La Flota-Vista Alegre	R
10	33333333C	Carmen	633333333	A	6	Madrid	Carabanchel	C
11	66666666F	Aitor	610243567	B	5.75	Alcalá de Henares	La Garena	C

TOTAL PRICE (Calcula precio por producto en ticket)

AMOUNT ACCUMULATED (NOT WORKING)

```
CREATE OR REPLACE FUNCTION amount_accumulated() RETURNS TRIGGER AS $amount_accumulated$
  BEGIN
        UPDATE member
        SET "pointsAccumulated" = ticket."totalPrice" * 0.05
        WHERE "ID" = new."ID";
        IF EXISTS (SELECT "membershipCode_member" FROM ticket WHERE ticket."ID" = new."ID_ticket")
THEN UPDATE member
                        SET "pointsAccumulated" = (SELECT sum("pointsAccumulated") FROM ticket
WHERE ticket."ID" = new."ID_ticket");

        END IF;
        RETURN NEW;
END;


$amount_accumulated$
LANGUAGE plpgsql;

CREATE TRIGGER amount_accumulated AFTER INSERT or UPDATE
  ON ticket FOR EACH ROW
  EXECUTE PROCEDURE amount_accumulated();
```

We have tried to make all the triggers work, but finally just 3 of them works. The others have been deleted from the database in order to make no error when we execute the code.

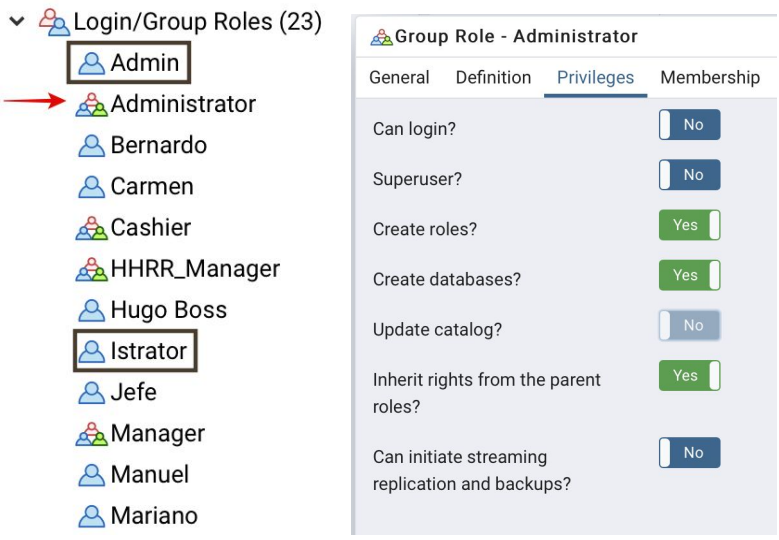
3. ROLES Y USUARIOS

ROLE	PERMISSIONS	ENTITIES	USER	PASSWORD
Administrator	• No restrictions	• Every table	Admin	Admin
			Istrator	Istrator
Manager	• Insert • Select • Update • Delete	• Every table	Manuel	Manuel
			Mariano	Mariano
Cashier	• Insert • Select • Update • Delete	• Product • Purchased • Ticket • Returned	Carmen	Carmen
			Bernardo	Ramon
HHRR_Manager	• Insert • Select • Update • Delete	• Worker • Cashier • Repletener	Hugo Boss	HugoBoss
			Jefe	Jefe

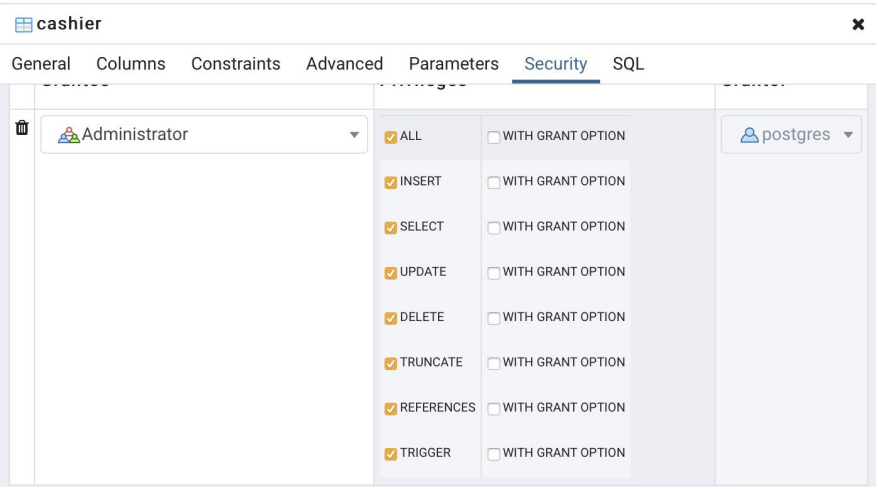
3.1. ADMINISTRATOR

PERMISSIONS	ENTITIES	USER	PASSWORD
• No restrictions	• Every table	Admin	Admin
		Istrator	Istrator

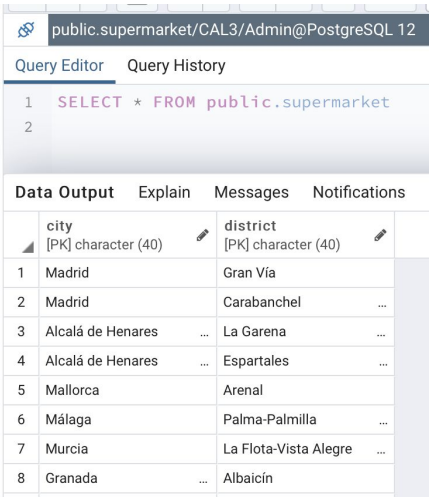
The requirements are that all the users contained in Administrator have all kinds of permissions with all tables.
In order to do that, using PgAdmin we have created a role called Administrator.



We give permissions to Administrator in all the tables that appear in the database. We have included as an example the first entity cashier. The rest of the tables are just the same.



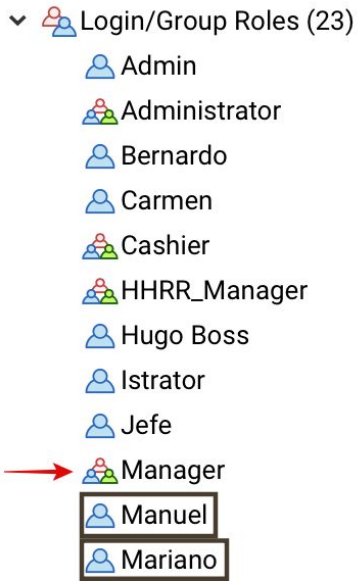
As we can see, if we try to view the information of any table it will be shown because we have the permissions of every table. We included the screenshot of logging in with Admin (that is one of the users enrolled in Administrator) and access information of supermarket.



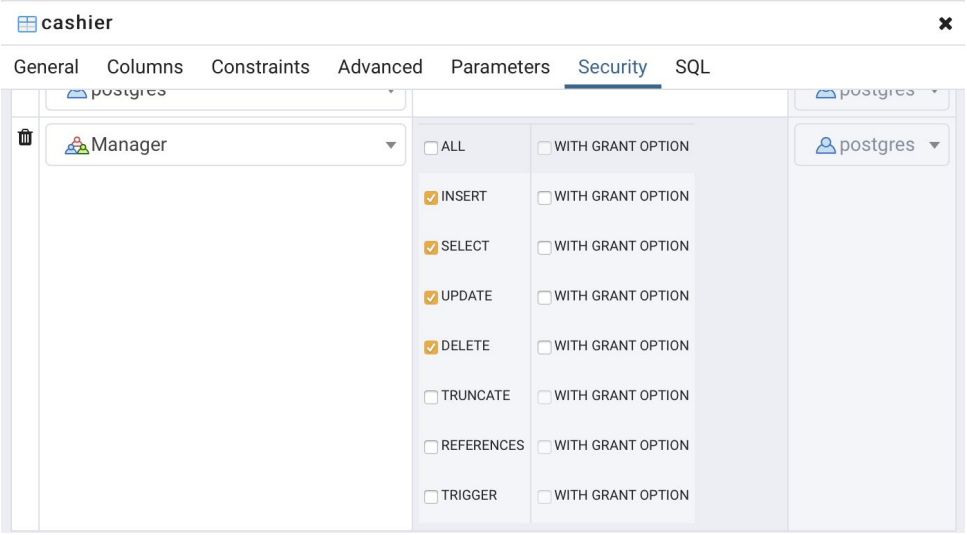
3.2. MANAGER

PERMISSIONS	ENTITIES	USER	PASSWORD
• Insert • Select • Update • Delete	• Every table	Manuel	Manuel
		Mariano	Mariano

The requirements are that all the users contained in Manager have permissions to *insert, update, delete and consult* all the tables.
In order to do that, using PgAdmin we have created a role called Manager.



We have to give permissions to Manager in all the tables but in this case we won't give all the permissions, just *insert, select, update and delete*. We have included cashier as an example, and the rest of the tables are just the same.



The following pictures will corroborate the permissions that Manager has:

- select:

public.supermarket/CAL3/Manuel@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | M

	city [PK] character (40)	district [PK] character (40)
1	Madrid	Gran Vía
2	Madrid	Carabanchel
3	Alcalá de Henares	La Garena
4	Alcalá de Henares	Espartales
5	Mallorca	Arenal
6	Málaga	Palma-Palmilla
7	Murcia	La Flota-Vista Alegre
8	Granada	Albaicín

- insert:

CAL3/Manuel@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | Messages

```
INSERT INTO public.supermarket(
    city, district)
    VALUES ('Palencia', 'Calle Mayor');
```

CAL3/Manuel@PostgreSQL 12

Query Editor | Query History | Data Output | Explain

```
INSERT 0 1

Query returned successfully in 100 msec.
```

- update:

CAL3/Manuel@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | Messages | Notifications

```
UPDATE public.supermarket
    SET city='Palencia', district='Plaza Pequeña'
    WHERE city='Palencia';
```

CAL3/Manuel@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | Messages | Notifications

```
UPDATE 1

Query returned successfully in 152 msec.
```

- delete:

CAL3/Manuel@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | Messages | Notifications

```
DELETE FROM public.supermarket
    WHERE city='Palencia';
```

CAL3/Manuel@PostgreSQL 12

Query Editor | Query History | Data Output | Explain | Messages | Notifications

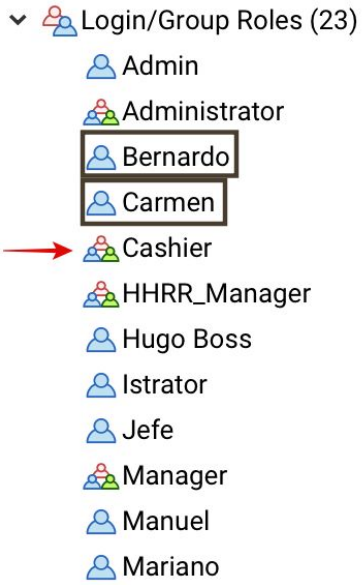
```
DELETE 1

Query returned successfully in 62 msec.
```

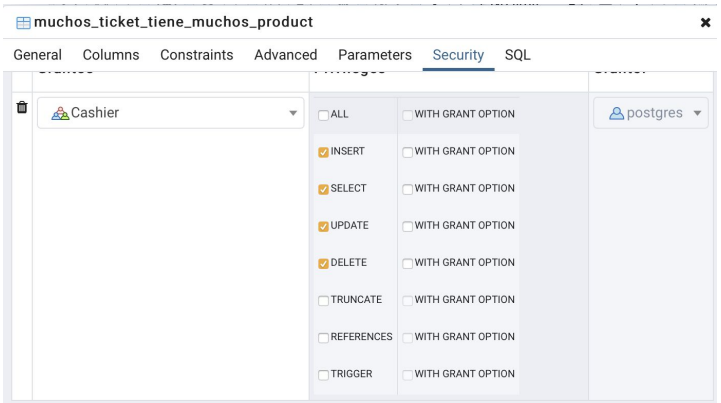
3.3. CASHIER

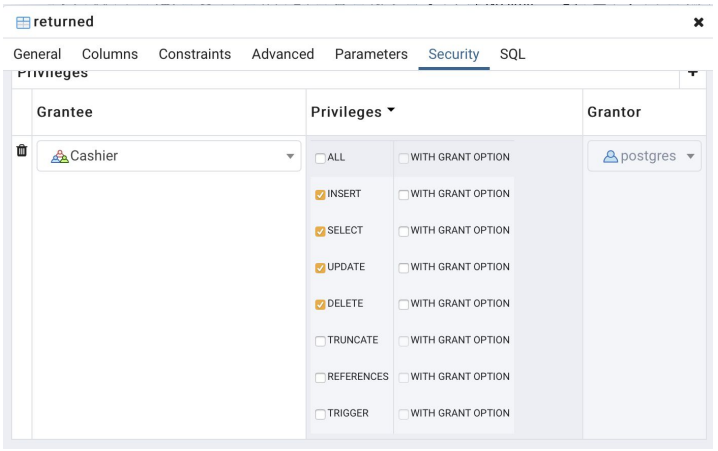
PERMISSIONS	ENTITIES	USER	PASSWORD
• Insert • Select • Update • Delete	• Product • Purchased • Ticket • Returned	Carmen	Carmen
		Bernardo	Bernardo

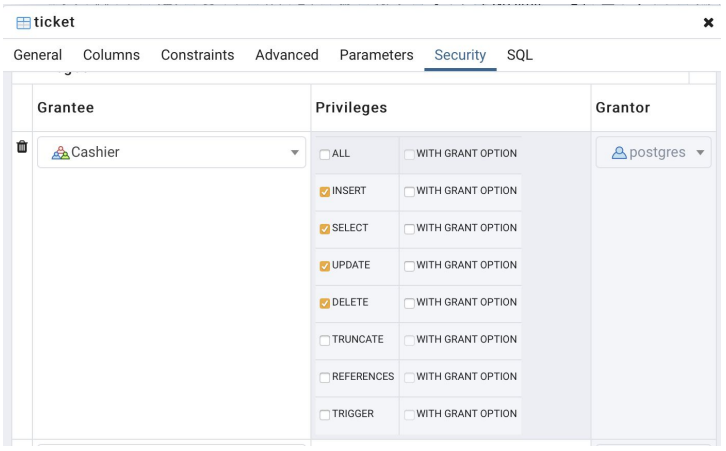
The requirements are that all the users contained in Cashier have permissions to *insert, update, delete and consult* all the tables related to ticket and products, including purchased (muchos_ticket_tiene_muchos_product) and returned.
In order to do that, using PgAdmin we have created a role called Cashier.

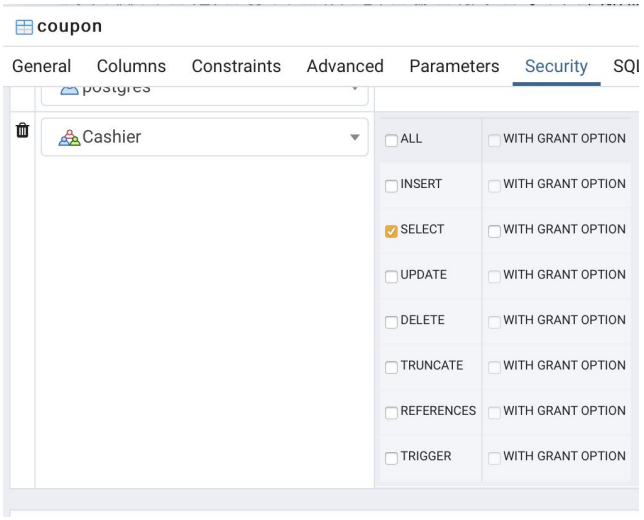


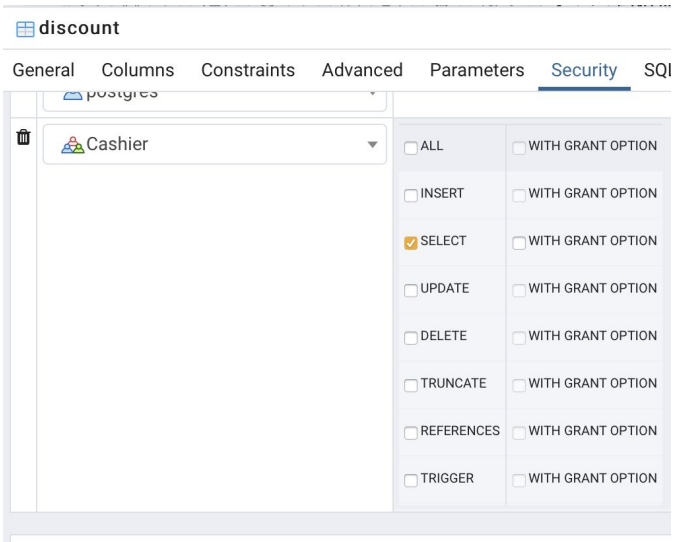
We created two users. Carmen and Bernardo, that are members from Cashier. After enrolling Carmen and Bernardo in Cashier, we have gone to the Security field in each of the tables in which we want all the Cashier users to have permissions and given its appropriate permissions.


muchos_ticket_tiene_muchos_product
General
Columns
Constraints
Advanced
Parameters
Security
SQL
Cashier
ALL
WITH GRANT OPTION
INSERT
SELECT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
postgres


returned
General
Columns
Constraints
Advanced
Parameters
Security
SQL
Grantee
Privileges
Grantor
Cashier
ALL
WITH GRANT OPTION
INSERT
SELECT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
postgres

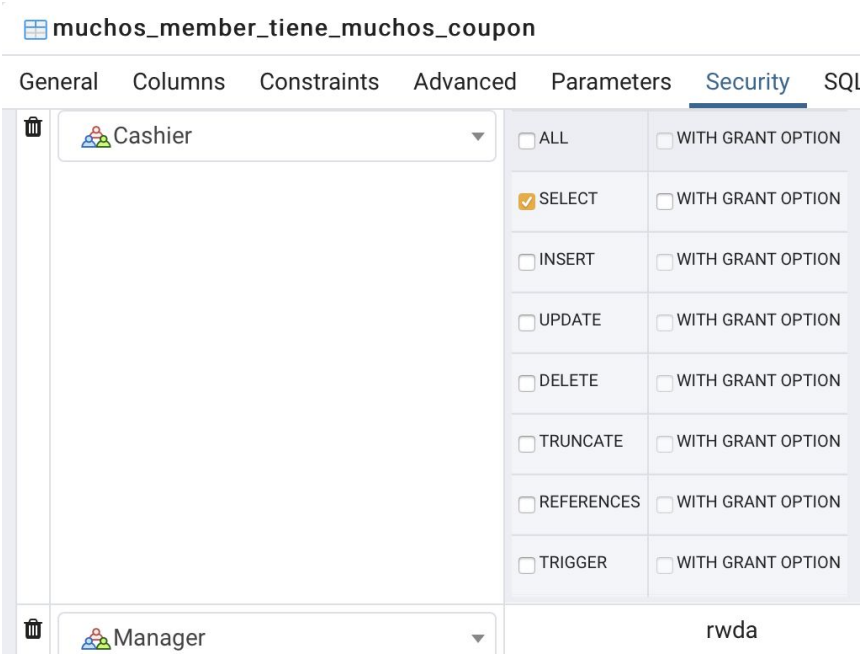

ticket
General
Columns
Constraints
Advanced
Parameters
Security
SQL
Grantee
Privileges
Grantor
Cashier
ALL
WITH GRANT OPTION
INSERT
SELECT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
postgres


coupon
General
Columns
Constraints
Advanced
Parameters
Security
Cashier
ALL
WITH GRANT OPTION
INSERT
SELECT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER

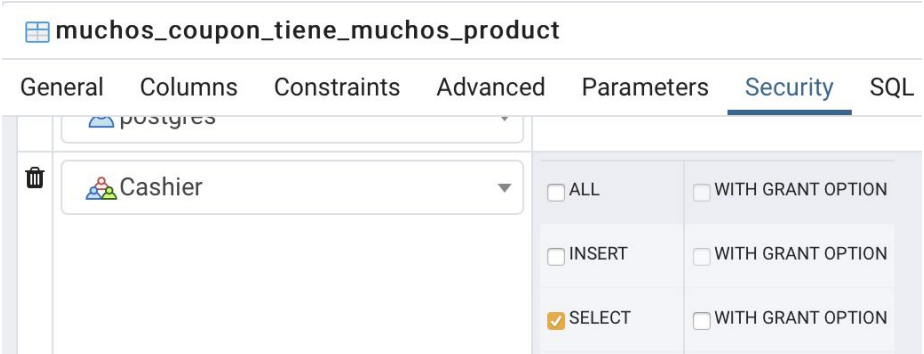


We add permissions in discount to the cashiers but they are only able to view the discount.

We add permissions in coupon to the cashiers but they are only able to view the coupons.



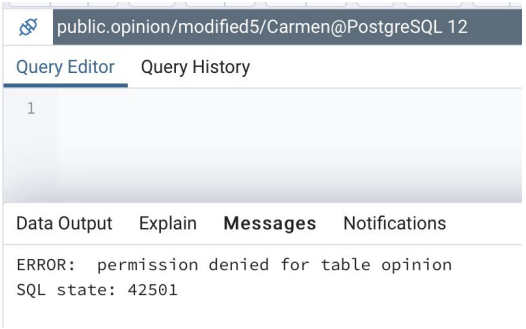
We add permissions in muchos_member_tiene_muchos_coupon to the cashiers but they are only able to view the coupons.



We add permissions in muchos_coupon_tiene_muchos_coupon to the cashiers but they are only able to view the coupons.

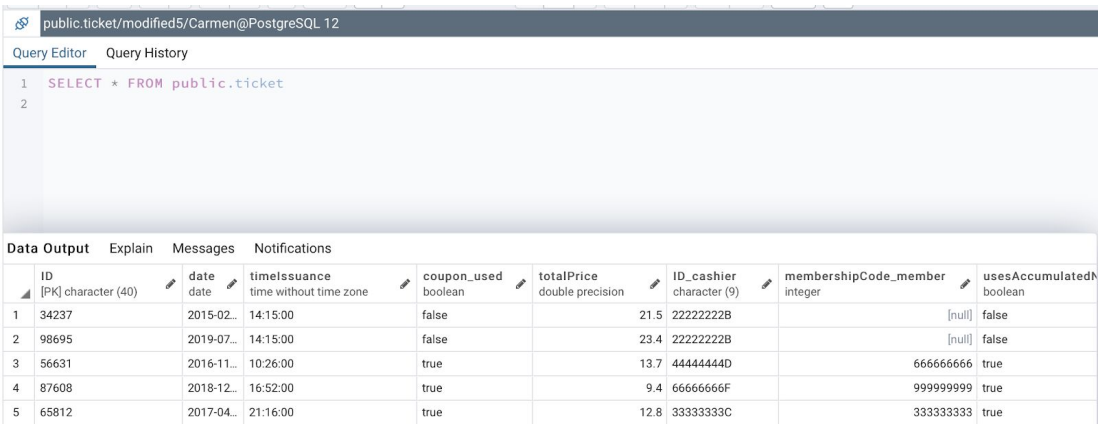
We can see in the point ***3.2. Manager*** the corroboration of the permissions due to the fact that they are the same. The difference is just in the tables of the entities they can access to.

Now, we are going to ensure that when the user logs in has the appropriate permissions. We log in with Carmen (for example) and after connecting with its password to the server we try viewing the table of *opinion*.



As we can see, Carmen does not have permissions to see that table, so the permission is denied for her.

We try viewing the *ticket* table, and as Carmen has permissions the table will be shown:



	ID [PK] character (40)	date date	timeIssuance time without time zone	coupon_used boolean	totalPrice double precision	ID_cashier character (9)	membershipCode_member integer	usesAccumulatedN boolean
1	34237	2015-02...	14:15:00	false	21.5	22222222B	[null]	false
2	98695	2019-07...	14:15:00	false	23.4	22222222B	[null]	false
3	56631	2016-11...	10:26:00	true	13.7	44444444D	666666666	true
4	87608	2018-12...	16:52:00	true	9.4	66666666F	999999999	true
5	65812	2017-04...	21:16:00	true	12.8	33333333C	333333333	true

We try viewing the *returned* table, and as Carmen has permissions the table will be shown:

public.returned/modified5/Carmen@PostgreSQL 12

Query Editor Query History

```
1  SELECT * FROM public.returned
2
```

Data Output Explain Messages Notifications

	barCode_product [PK] integer	ID_ticket [PK] character (40)	amount integer
1	123456733	22001	1
2	789456789	24001	2
3	123456715	21001	1
4	789456789	7	1
5	123347701	6	2
6	123456715	25001	2
7	123456789	4	1
8	123456720	1	1

We try viewing the *muchos_ticket_tiene_muchos_product* table, and as Carmen has permissions the table will be shown:

public.muchos_ticket_tiene_muchos_product/modified5/Carmen@PostgreSQL 12

Query Editor Query History

```
1  SELECT * FROM public.muchos_ticket_tiene_muchos_product
2
```

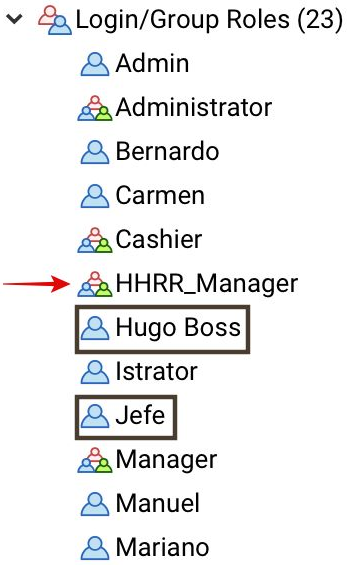
Data Output Explain Messages Notifications

	ID_ticket [PK] character (40)	barCode_product [PK] integer	amount integer
1	34237	123456789	1
2	34237	123456701	3
3	34237	123456720	5
4	98695	123456789	3
5	56631	456456789	3
6	56631	147456789	7
7	87608	456456789	5
8	87608	123111701	2
9	87608	123456789	1

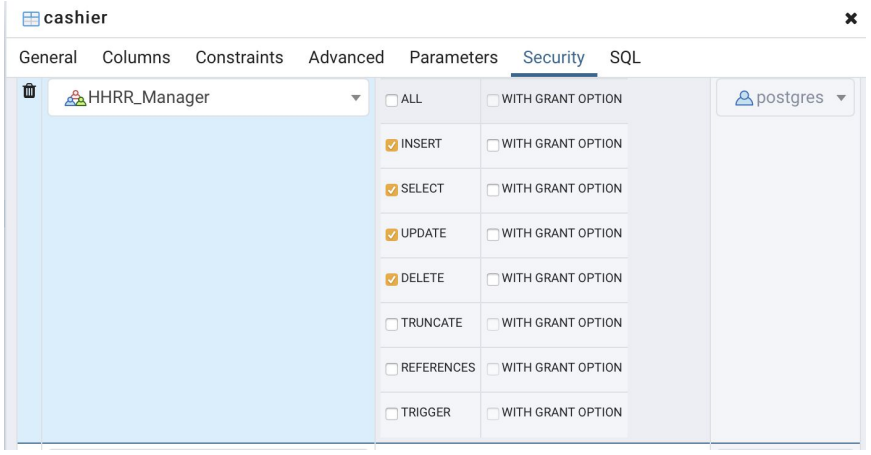
3.4. HHRR_MANAGER

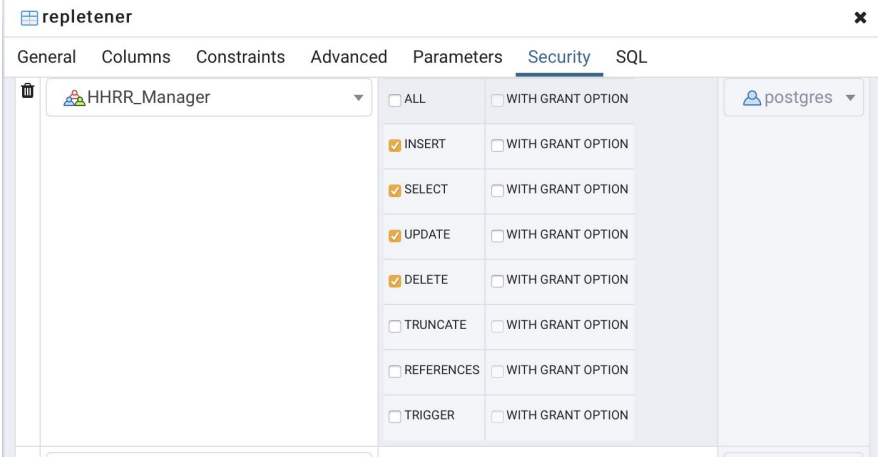
PERMISSIONS	ENTITIES	USER	PASSWORD
• Insert • Select • Update • Delete	• Worker • Cashier • Repletener	Hugo Boss	HugoBoss
		Jefe	Jefe

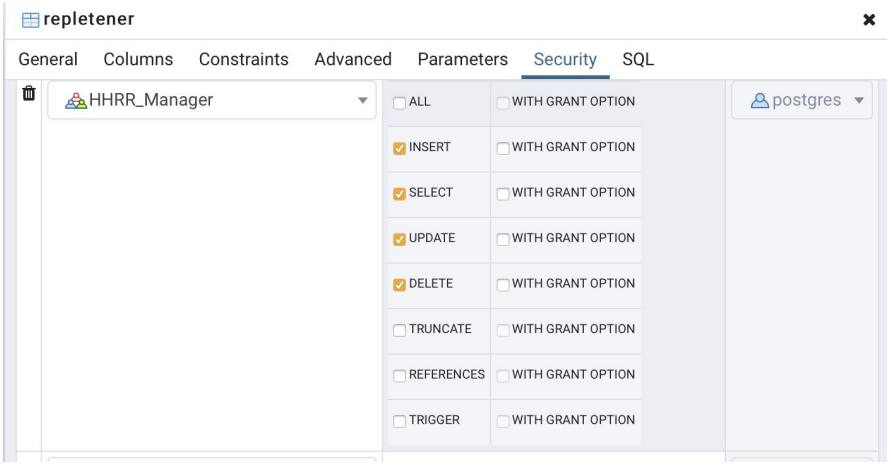
The requirements are that all the users contained in HHRR_Manager have permissions to *insert, update, delete and consult* all the tables related to employees (worker, cashier and repletener).
In order to do that, using PgAdmin we have created a role called HHRR_Manager.

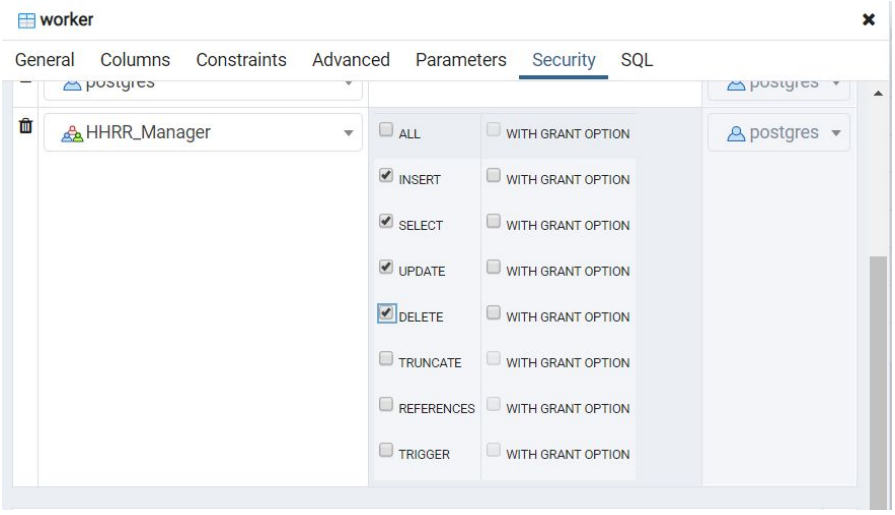


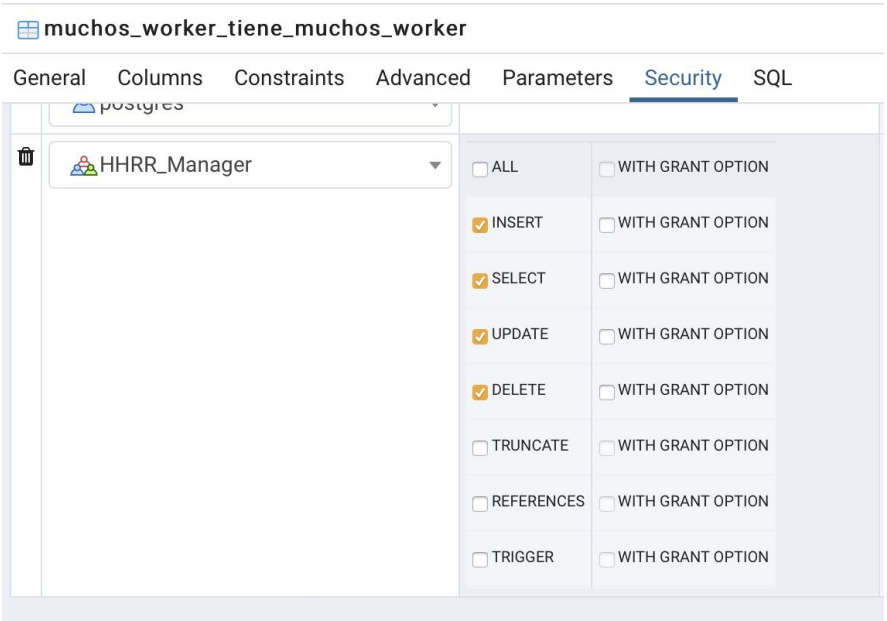
We created two users: Hugo Boss and Jefe, that are members from HHRR_Manager. After enrolling Hugo Boss and Jefe in HHRR_Manager, we have gone to the Security field in each of the tables in which we want all the HHRR_Manager users to have permissions and given its appropriate permissions.


cashier
General
Columns
Constraints
Advanced
Parameters
Security
SQL
HHRR_Manager
ALL
WITH GRANT OPTION
INSERT
SELECT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
postgres


repletener
General
Columns
Constraints
Advanced
Parameters
Security
SQL
HHRR_Manager
ALL
WITH GRANT OPTION
INSERT
SELECT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
postgres

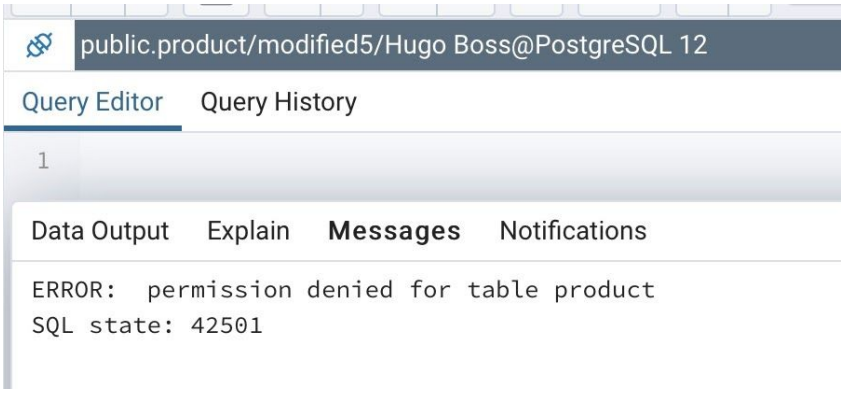

repletener
General
Columns
Constraints
Advanced
Parameters
Security
SQL
HHRR_Manager
ALL
WITH GRANT OPTION
INSERT
SELECT
UPDATE
DELETE
TRUNCATE
REFERENCES
TRIGGER
postgres





We can see in the point ***3.2. Manager*** the corroboration of the permissions due to the fact that they are the same. The difference is just in the tables of the entities they can access to.

Now, we are going to ensure that when the user logs in has the appropriate permissions. We log in with Hugo Boss (for example) and after connecting with its password to the server we try viewing the table of *product*.



This message appears when we try to make with HHRR_Manager an action that is no-allowed to this role.

Beside HHRR_Manager is allowed to consult the information of Cashier, the Cashier's table is shown.

public.cashier/modified5/Hugo Boss@PostgreSQL 12

Query Editor | Query History

```
1 SELECT * FROM public.cashier
2
```

Data Output | Explain | Messages | Notifications

	ID [PK] character (9)	ID_worker character (9)
1	11111111A	11111111A
2	22222222B	22222222B
3	33333333C	33333333C
4	44444444D	44444444D
5	55555555E	55555555E
6	66666666F	66666666F

public.repletener/modified5/Hugo Boss@PostgreSQL 12

Query Editor | Query History

```
1 SELECT * FROM public.repletener
2
```

Data Output | Explain | Messages | Notifications

	ID [PK] character (9)	numberHours integer	ID_worker character (9)
1	77777777G	22	77777777G
2	88888888J	16	88888888J
3	99999999K	38	99999999K
4	45872396L	24	45872396L
5	05794628M	18	05794628M

As HHRR_Manager is allowed to consult the information of Repletener, the Repletener's table is shown.

Beside HHRR_Manager is allowed to consult the information of *worker*, the *worker*'s table is shown:

public.worker/modified5/Hugo Boss@PostgreSQL 12

Query Editor | Query History

```
1 SELECT * FROM public.worker
2
```

Data Output | Explain | Messages | Notifications

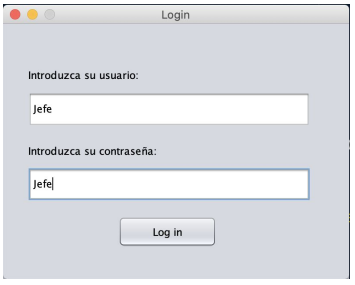
	ID [PK] character (9)	name character (40)	phone integer	timeTable character (1)	city_supermarket character (40)	district_supermarket character (40)	type character (40)
1	11111111A	Aarón	611111111	A	Madrid	Gran Vía	C
2	22222222B	Bernardo	622222222	B	Madrid	Carabanchel	C
3	33333333C	Carmen	633333333	A	Madrid	Carabanchel	C
4	44444444D	Angy Carmen	601234567	A	Madrid	Carabanchel	C
5	55555555E	Adrián	601243567	A	Mallorca	Arenal	C
6	66666666F	Aitor	610243567	B	Alcalá de Henares	La Garena	C
7	77777777G	Álvaro	610423567	B	Alcalá de Henares	Espartales	R
8	88888888J	Andrea	610425367	B	Málaga	Palma-Palmilla	R

4. JAVA CONNECTION

We make the GUI having on the main window the option of logging in among all of the users that have previously been explained.
To make it easier, we are going to simply show the options of the query's that the user selected has permissions to.

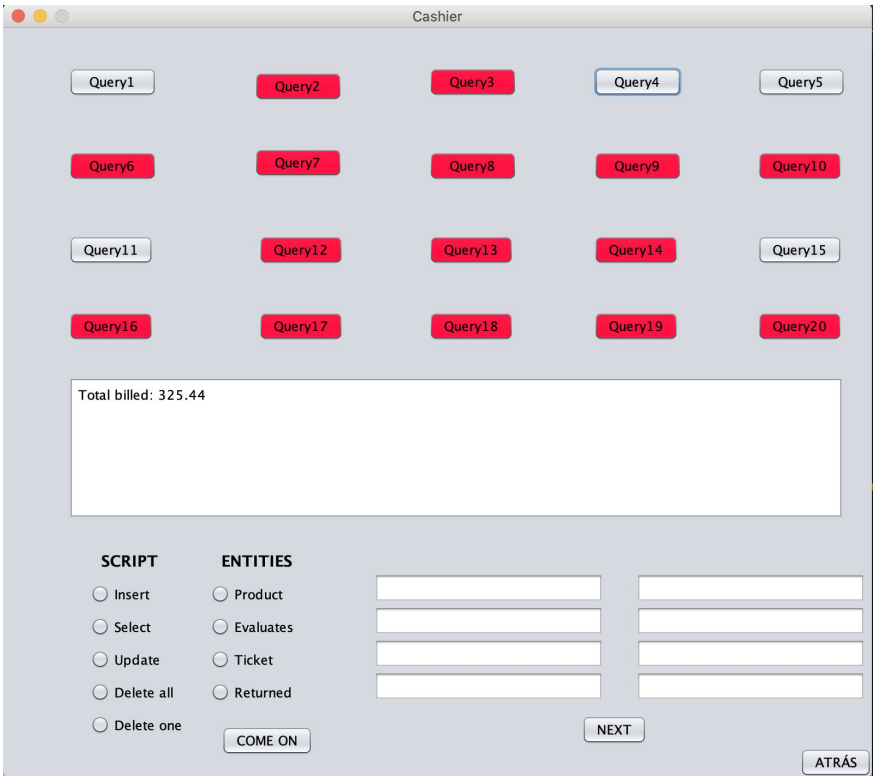
The first thing to do in the GUI Java interface is to log in with any of hour previously mentioned users. For example: Carmen or Bernardo (Cashiers), Manuel or Mariano (Manager), Admin or Istrator (Administrators) and Hugo Boss or Jefe (HHRR_Manager).

Some screenshots of the log in phase:

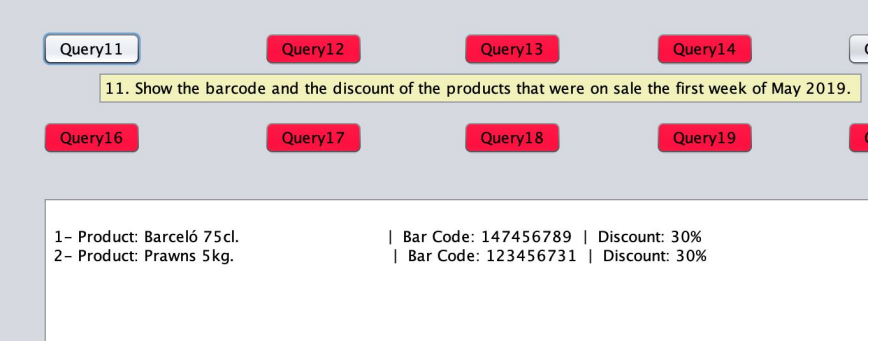


After logging in with a user a specific window will appear matching the role you have selected.

- **CASHIER WINDOW**

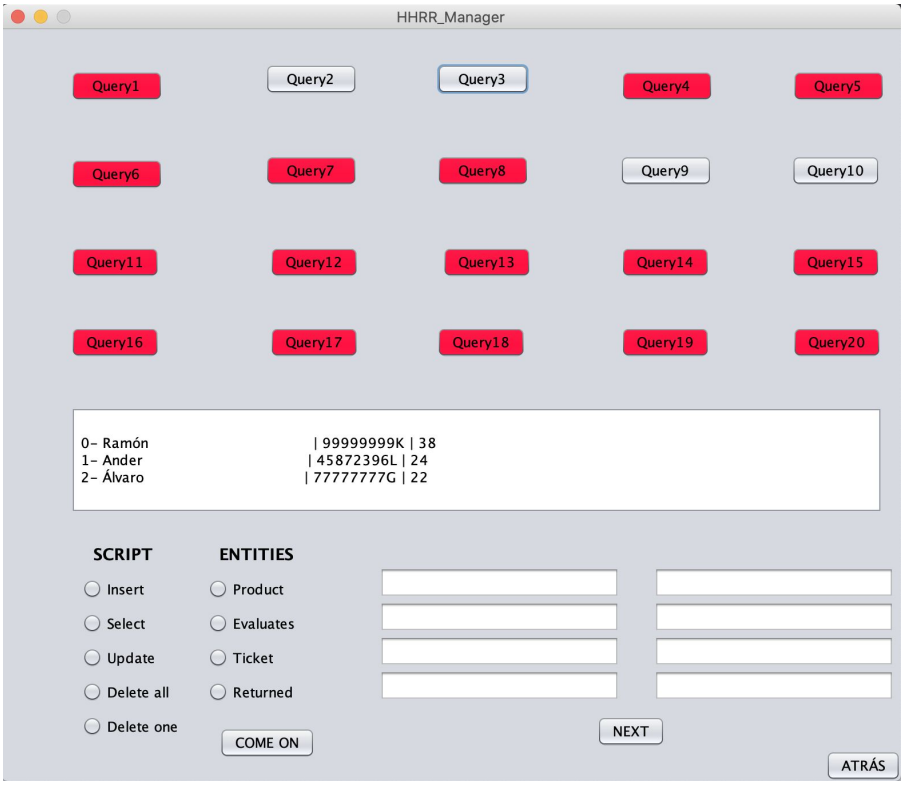


As you can see, we have set up the queries that any user rolled in Cashier can access, taking into account the permissions that it has. Even if you try to see the queries that cashiers can't access, an emergent window will show to warn you that the access is denied. In the example the Query 4 is shown, because cashier has access permitted.



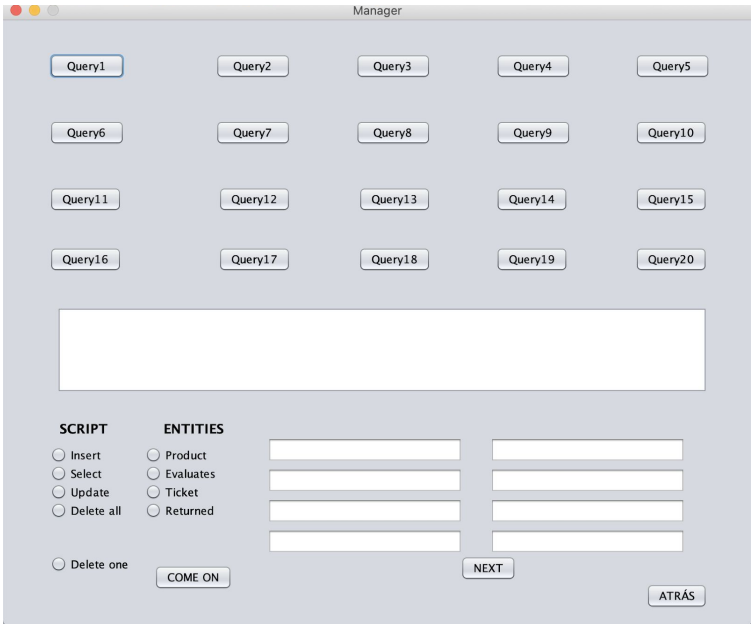
In this screenshot you can see that when you move the pointer above the cells of the queries, the title of the queries will appear for you to know which query are you referring to. In that case, query 11 is shown.

- **HHRR_MANAGER WINDOW**



As well as Cashiers do, HHRR_Manager role can only do some restricted actions. In this case, HHRR_Manager can only execute query2, query3, query9 and query10. In this screenshot Query3 is shown.

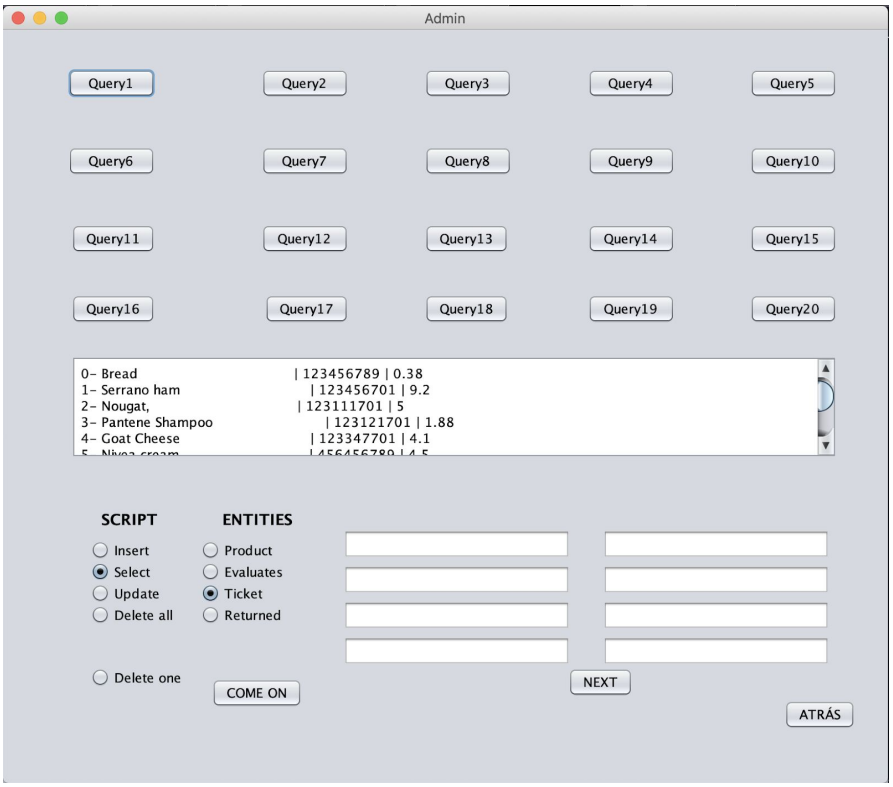
- **MANAGER WINDOW**



As said before, Manager has all the permissions required in order to show all the queries because it can access all the table's information and also insert, update or delete.

In this screenshot query 13 is shown:

- **ADMINISTRATOR WINDOW**

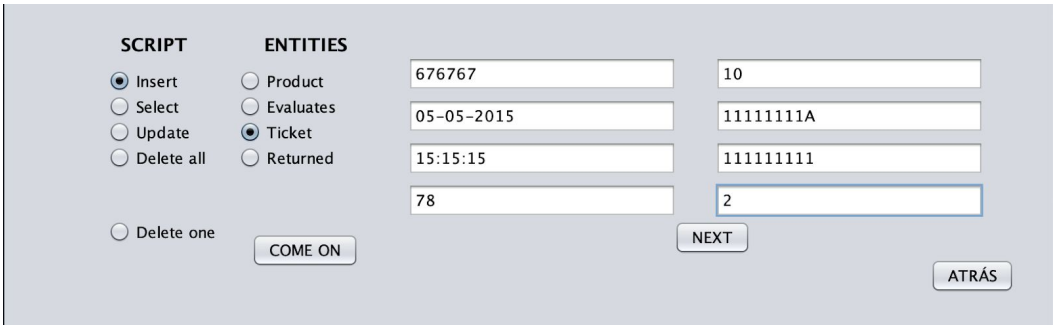


As well as Manager's role does, Administrator is able to select, update, insert or delete among all the database, because it has all the permissions. All the queries can be accessed by them. The screenshot shows query 1.

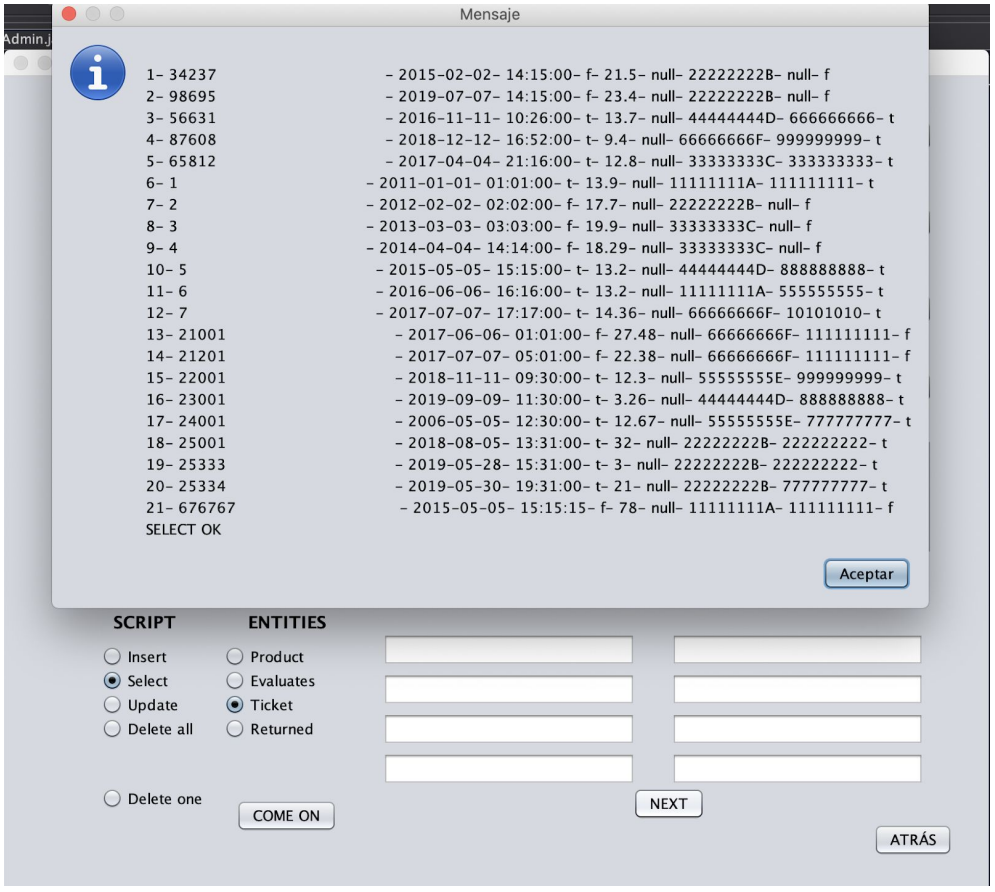
As Administrator has all the permissions, we are going to explain the rest of the functions that our interface has, as you can see on the bottom of the window.

We have added the entities product, evaluates, ticket and returned because we considered that were the most relevant and complex ones that can be found in this database.

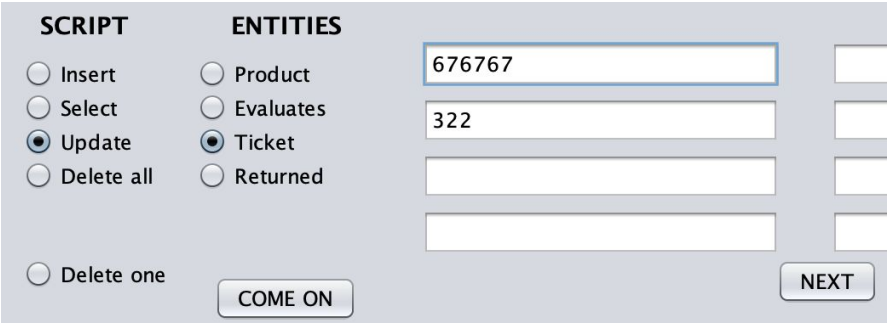
We will start by showing the utility of the *Insert Script*.



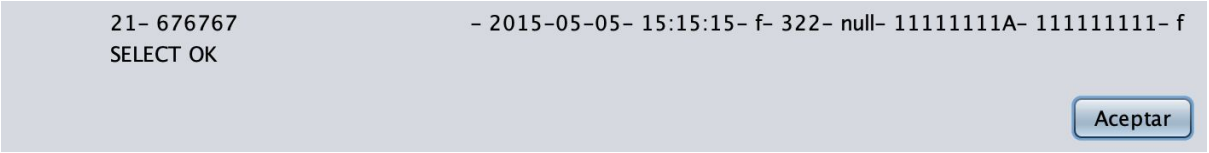
Using the radio buttons you can select any of the entities available, but we have chosen ticket for this example. Then you have to click on come on in order to see the information you have to fill. Now that we have created this ticket using java, we will check that it appears using the *Select Script*.



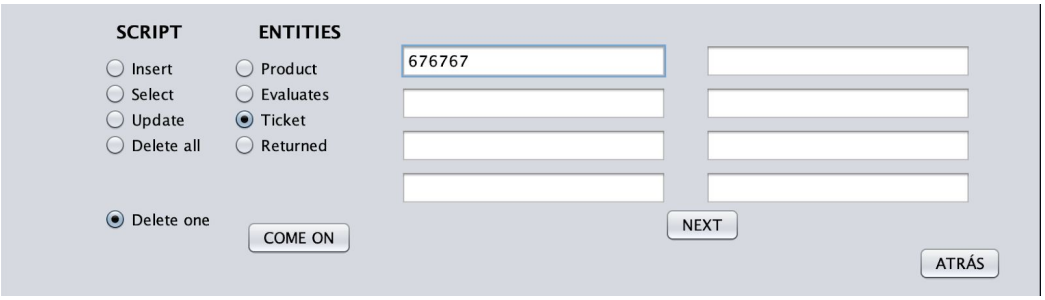
As you can see, the ticket we have just created appears at the end, on line 21. Now, we are going to check the *Update Script*.



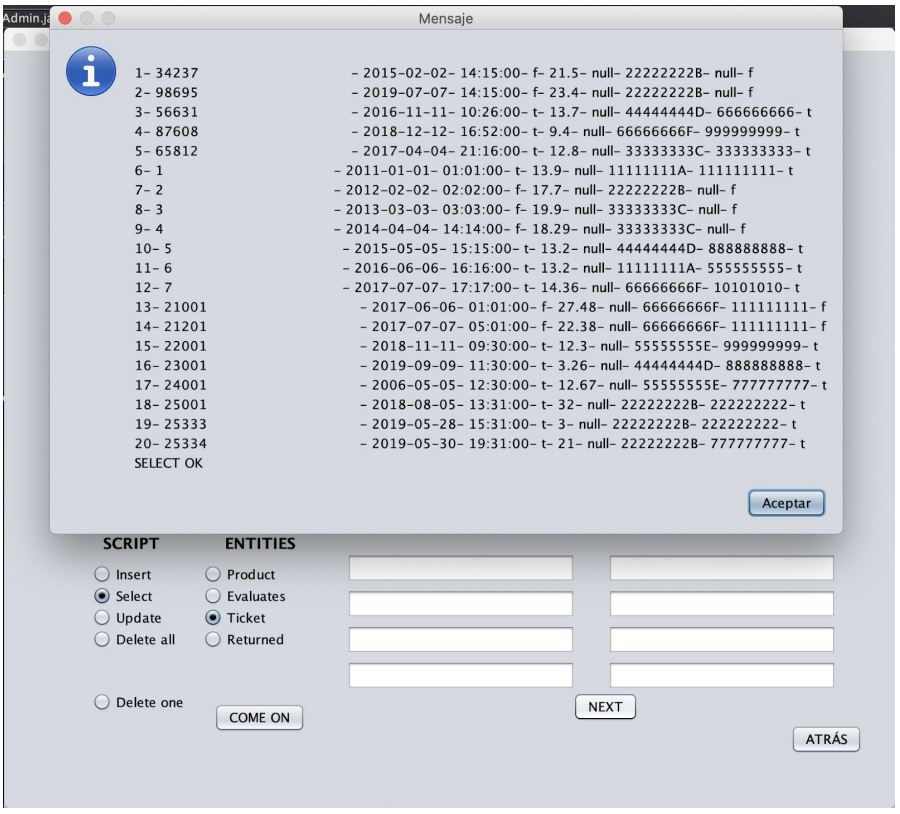
We fill the ID of the ticket previously created and update the total price from 78 to 322. Again, to check if it was correctly performed we will use the *Select script.*



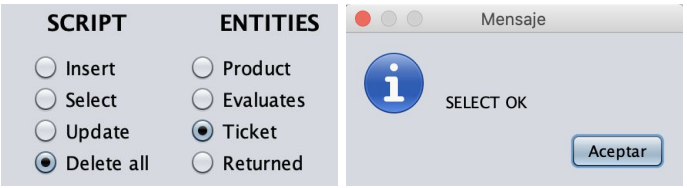
Now that we have seen it works, we will delete it. We select *Delete one* and fill 676767 as the ID of the ticket.



The result, after checking again with the *Select Script* is this:



The ticket is not there anymore. And the only Script left to try is Delete, that will delete all the tickets existing in the database. We try it:



The result shown after trying the result with the Select Script is empty, and we now that because of the message that shows that the Select operation is made in a right way.

This example was made with Ticket, but you can try by yourself using Product, Evaluates or Returned and it will work in the same way.

In the interface, as you can check, we have given the same permissions in the Script field (insert, select, update, delete all, delete one) and the same entities with each role can work with (Product, Evaluates, Ticket, Returned).

We know that each role is able to do an action different than the one that another role can do. You can see in the point 3 of this practice a blue table where we have noted that:

- Administrator does not have any restrictions and it can get access in any table.
- Manager can insert, select, update and delete in any table.
- Cashier can insert, select, update and delete in Product, Purchased, Ticket and Returned.
- HHRR_Manager can insert, select, update and delete in Worker, Cashier and Repletener.

We have not cared about that because every entity gets connected in the same name, just changing its respective code.

To sum up, although it seems that we have made the interface by the fastest way in order to focus on the triggers, that have been the most difficult part of the practice for us.