surname
birthday_date
Employee
name
ssn
isA
isA
Back Office
Front Office
desk
The-Back-Office
date_authorization
Authorize
The-Book
registration_number
name
Customer
surname
birthday_date
ending_date
Rent
starting_date
ISBN
title
Book
year_publ
oid
Order
order_date
n_copies
Collection
Part
date_of_publication
name
The-Dealer
PublishC
PublishB
Write
Dealers
Publisher
Author
name
did
address
fiscal_code
name
head_quarters_address
birthday_date
name
nationality

```
-- Publishers

create table Publishers (
	fiscal_code varchar,
	publisher_name varchar,
	head_quarters_address varchar,

	primary key (fiscal_code)
);

-- Books

create table Books (
	ISBN varchar,
	title varchar,
	year_of_publication integer,

	publisher_fiscal_code varchar not null,

	primary key (ISBN),

	foreign key (publisher_fiscal_code) references Publishers(fiscal_code)
);

-- Authors

create table Authors (
	author_name varchar,
	birthday_date date,
	nationality varchar,

	primary key (author_name, birthday_date)
);

-- Write (Authors - Books)

create table Write (
	author_name varchar,
	author_birthday_date date,
	book_ISBN varchar,

	primary key (author_name, author_birthday_date, book_ISBN),

	foreign key (author_name, author_birthday_date) references Authors(author_name, birthday_date),
	foreign key (book_ISBN) references Books(ISBN)
);

-- Collections

create table Collections (
	collection_name varchar,
	date_of_publication date,

	publisher_fiscal_code varchar,

	primary key (collection_name, publisher_fiscal_code),

	foreign key (publisher_fiscal_code) references Publishers(fiscal_code) on delete cascade
);

-- Part (Books- Collections)

create table Part (
	book_ISBN varchar,
	collection_name varchar,
	publisher_fiscal_code varchar,

	primary key (book_ISBN, collection_name, publisher_fiscal_code),

	foreign key (book_ISBN) references Books(ISBN),
	foreign key (collection_name, publisher_f6iscal_code) references Collections(collection_name, publisher_fiscal_code)
);
```

```
-- Customers

create table Customers (
	registration_number varchar,
	customer_name varchar,
	customer_surname varchar,
	birthday_date date,

	primary key (registration_number)
);

-- Employees

create type emp_role as enum('front_office', 'back_office', 'none');

create table Employees (
	ssn varchar,
	employee_name varchar,
	employee_surname varchar,
	birthday_date date,

	employee_type emp_role,
	desk varchar,

	primary key (ssn),

	check (((employee_type='back_office' or employee_type='none') and desk is null) or employee_type='front_office')
);

-- Rent (Customers - Books)

create table Rents (

	starting_date date,
	ending_date date,

	book_ISBN varchar,
	customer_registration_number varchar,

	employee_ssn varchar not null,

	date_of_authorization date,

	primary key (book_ISBN, customer_registration_number),

	foreign key (book_ISBN) references Books(ISBN),
	foreign key (customer_registration_number) references Customers(registration_number),
	foreign key (employee_ssn) references Employees(ssn)
);

-- Dealers

create table Dealers (
	did varchar,
	dealer_name varchar,
	dealer_address varchar,

	primary key (did)
);

-- Orders

create table Orders (
	order_id varchar,
	n_copies integer,
	order_date date,

	dealer_id varchar not null,
	employee_ssn varchar not null,
	book_ISBN varchar not null,

	primary key (order_id),
```

```
        foreign key (dealer_id) references Dealers(id),
        foreign key (employee_ssn) references Employees(ssn),
        foreign key (book_ISBN) references Books(ISBN)
);



-- Trigger to check employee type before Authorizations
CREATE OR REPLACE FUNCTION check_front_office_authorization()
RETURNS TRIGGER AS $$
DECLARE
        employee_role emp_role;
BEGIN

        IF TG_TABLE_NAME = 'rents' THEN

                SELECT employee_type INTO employee_role
                FROM employees
                WHERE ssn = NEW.employee_ssn;

                -- Check if the employee has 'front_office' type
                IF employee_role <> 'front_office' THEN
                        RAISE EXCEPTION 'Front-Office employees are only allowed to perform Authorizations.';
                END IF;
        END IF;

        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER front_office_authorization_trigger
BEFORE INSERT ON rents
FOR EACH ROW
EXECUTE FUNCTION check_front_office_authorization();


-- Trigger to check employee type before Orders
CREATE OR REPLACE FUNCTION check_back_office_orders()
RETURNS TRIGGER AS $$
DECLARE
        employee_role emp_role;
BEGIN

        IF TG_TABLE_NAME = 'orders' THEN

                SELECT employee_type INTO employee_role
                FROM employees
                WHERE ssn = NEW.employee_ssn;

                -- Check if the employee has 'front_office' type
                IF employee_role <> 'back_office' THEN
                        RAISE EXCEPTION 'Back-Office employees are only allowed to perform Orders.';
                END IF;
        END IF;

        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER back_office_authorization_trigger
BEFORE INSERT ON orders
FOR EACH ROW
EXECUTE FUNCTION check_back_office_orders();
```