Project Report
CAMPUS ADMINISTRATION

Contents

1. Project description

1.1. Short overview

The students' dormitory is the main place of university students' daily life, so the students' dormitory management is an important part of university management. With the increasing number of students living in dormitories, the necessity of an application for managing all tasks related with the application, allocation and monitoring of the students' housing facilities becomes clear. Therefore, our team decided to develop Campus Administration System with the aim to cater to the needs of the dormitories and the residence hall managers in terms of easier data input and processing, as well as printing output and manipulation of data. Specifically, the system aims at helping the dormitory administrator managing the information during daily work, since he/she is in charge of all kinds of things in the dormitory. So, by using the system it would make the dormitory administrator work easier, more efficient, and make fewer mistakes. Furthermore, Campus Administration System maintains data of campus personnel including security guard, cleaning service representatives and others.

1.2. List of features

Below key features of the Campus Administration System are presented:

- **Record and management of the students' and employees' information**

The students' information includes student's name, id number, gender, date of birth and scholarship. As for employees, they also have name, id number, gender, date of birth, salary and position. These data can be modified, updated and deleted.

- **Personnel attendance control**

This function gives an opportunity to manage what time the employee checked in and out. Based on the attendance statistics the salary for personnel will be calculated.

- **Keep visitor records**

Anyone who does not live in the dormitory is regarded as a guest and the system records the visit. Personal information including name, date of birth and gender are necessary. The visiting time, reception person are required. When the visitor leaves, the departure time is recorded.

- **Students' and employees' documents maintenance**

Such important documents as passport, academic contract, medical insurance will be stored in the system database.

- **Payment control**

The following feature will help to control payments made by students and guest for room renting.

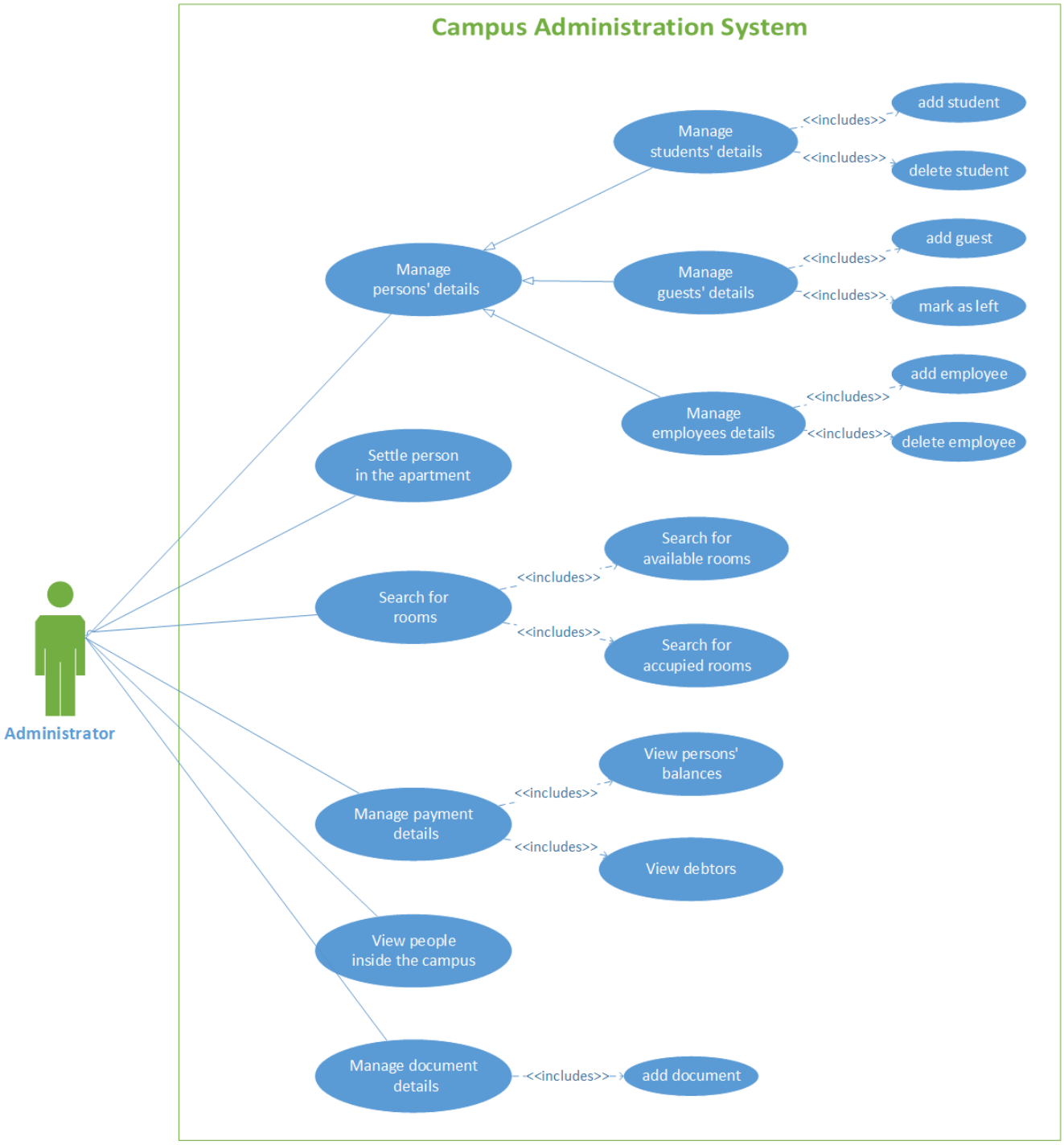
- **Apartments' occupation control**

With the help of implemented queries and functions it will be possible to search rooms which have not been occupied yet. Moreover, the system provide information about rooms by gender compatibility.

- **Access control**

According to the role in the university campus, people have access to rooms within the campus territory. For instance, students can not enter to the staff room. There are entry checking machines which will check an entry permission of the person.

2. Use case


Campus Administration System
Administrator
Manage persons' details
Manage students' details
add student
delete student
Manage guests' details
add guest
mark as left
Manage employees details
add employee
delete employee
Settle person in the apartment
Search for rooms
Search for available rooms
Search for accupied rooms
Manage payment details
View persons' balances
View debtors
View people inside the campus
Manage document details
add document
<<includes>>

№	Use Case name	Actor	Precondition	Postcondition	Event flow
1	Add student	Administrator	Student should provide information to the admin	Student is added	Admin inserts required information about the student into the system
2	Delete student	Administrator	Student should leave the campus	Student is deleted	Admin deletes student's information from the system
3	Add guest	Administrator	Guest should provide information to the admin	Guest is added	Admin inserts required information about the guest into the system
4	Mark guest as left	Administrator	Guest should leave the campus	Guest is marked as left	Admin deletes guest's information from the system
5	Add employee	Administrator	Employee should provide information to the admin	Employee is added	Admin inserts required information about the employee into the system
6	Delete employee	Administrator	Employee should leave his job	Employee id deleted	Admin deletes employee's information from the system
7	Search for available room	Administrator	The list of all existing rooms should be available	Available rooms are displayed	
8	Search occupied rooms	Administrator	The list of all existing rooms should be available	Occupied rooms are displayed	
9	View person's balance	Administrator	The list of all persons' balance should be available	Person's balance is displayed	Admin
10	View debtors	Administrator	The list of transactions should be available	Debtors are displayed	
11	View people inside the campus	Administrator		People inside the campus is displayed	
12	Add document	Administrator	Person should provide the document to the administrator	Document is added	Admin scans the document and upload into the system

3. Overview of technologies used

3.1. PostgeSQL

As a database server, we used PostgeSQL, since it stores data securely, and to allows for retrieval at the request of other software applications. Moreover, PostgreSQL manages concurrency through a system known as multiversion concurrency control (MVCC), which gives each transaction a "snapshot" of the database, allowing changes to be made without being visible to other transactions until the changes are committed. This largely eliminates the need for read locks, and ensures the database maintains the ACID (atomicity, consistency, isolation, durability) principles in an efficient manner.

3.2. JavaFX

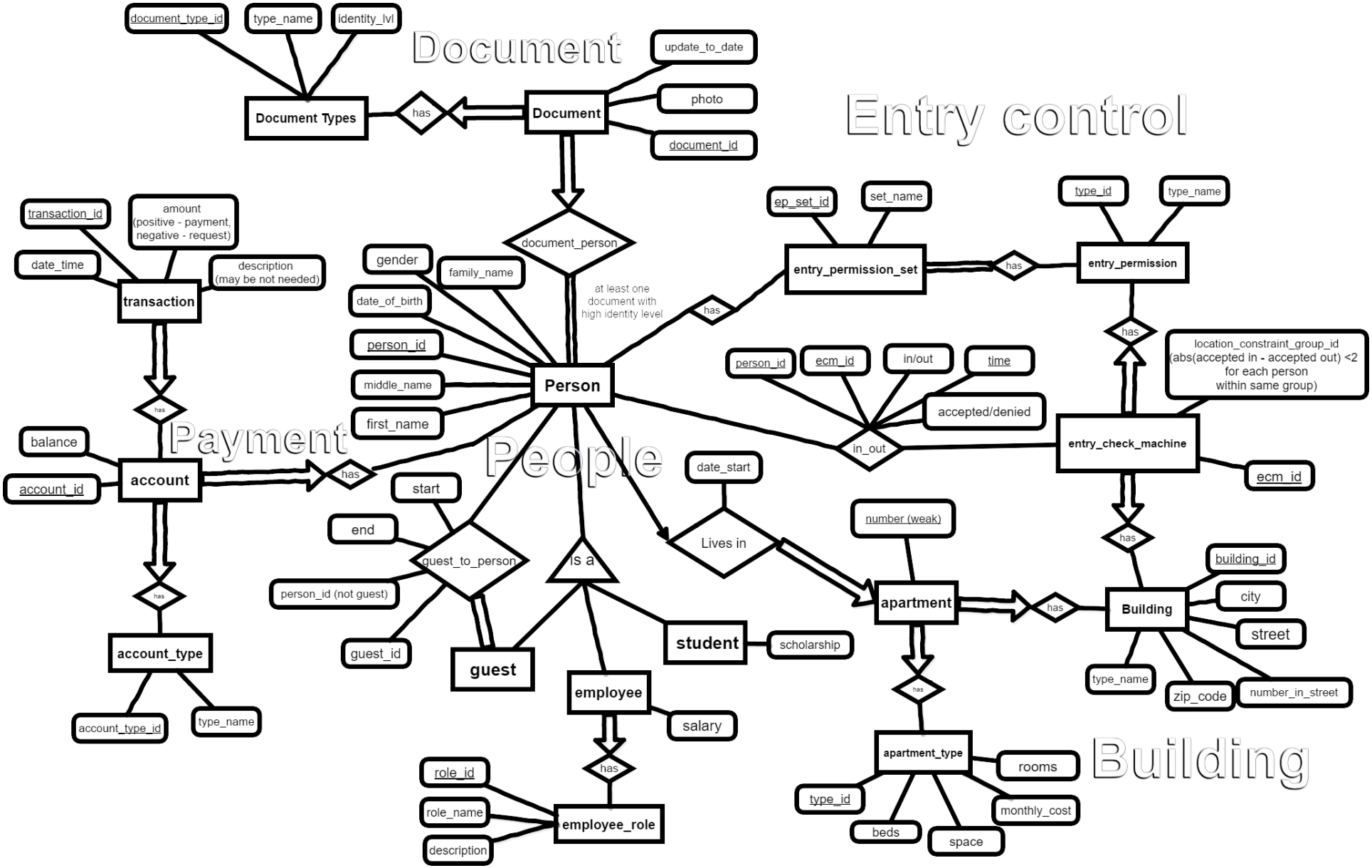
As a software platform for creating and delivering client side application we chose JavaFX. It provides a clear and clean architecture and features many enhancements such as styling, event management, transitions. Furthermore, it provides all the professional Java tooling required to debug, analyze, profile, and log a client application.

Another advantage of JavaFX is that it enables a simple app-like installation on the client side, without any prerequisites.

3.3. JDBC

Java Database Connectivity (JDBC) was implemented as an application programming interface (API) which defines how a client may access a database. The combination of the Java API and the JDBC API made our application development easy and cost effective. Also, The JDBC API includes a way to identify and connect to a data source, using a DataSource object. This made code even more portable and easier to maintain.

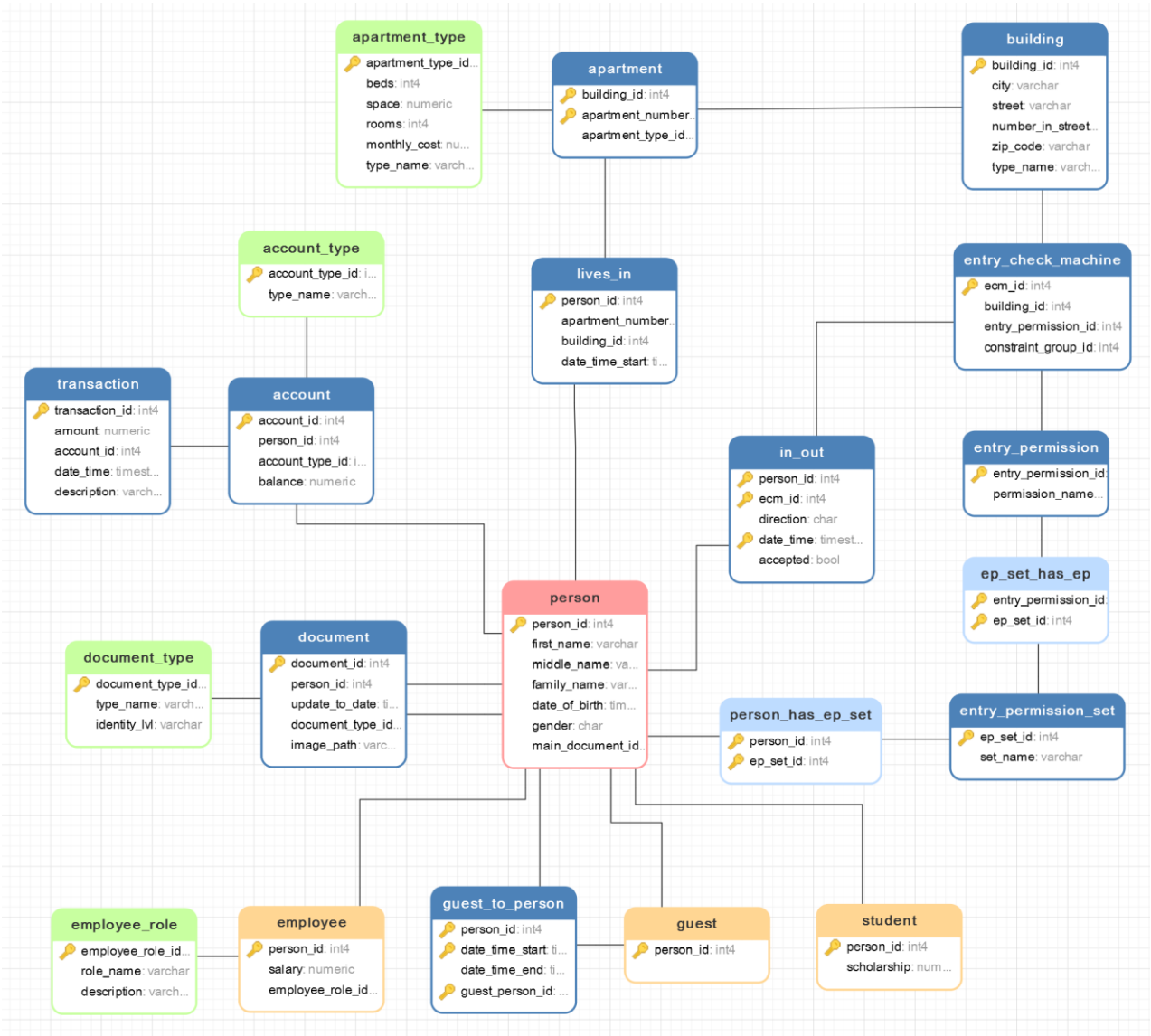
4. ER-diagram


Document
Entry control
Payment
People
Building
document_type_id
type_name
identity_lvl
Document Types
has
Document
update_to_date
photo
document_id
document_person
at least one document with high identity level
transaction_id
amount (positive - payment, negative - request)
date_time
description (may be not needed)
transaction
gender
family_name
date_of_birth
person_id
middle_name
first_name
Person
ep_set_id
set_name
entry_permission_set
has
type_id
type_name
entry_permission
location_constraint_group_id (abs(accepted in - accepted out) <2 for each person within same group)
person_id
ecm_id
in/out
time
accepted/denied
in_out
entry_check_machine
ecm_id
balance
account_id
account
has
start
end
guest_to_person
person_id (not guest)
guest_id
guest
is a
date_start
Lives in
number (weak)
apartment
has
Building
building_id
city
street
type_name
zip_code
number_in_street
student
scholarship
employee
salary
account_type
account_type_id
type_name
role_id
role_name
description
employee_role
apartment_type
rooms
type_id
monthly_cost
beds
space

Above illustrated Entity-Relationship (ER) model, visually represents the structure of Campus Administration database, where data equates to entities and objects, which are linked by defined relationships expressing dependencies and requirements.

The ER diagram of Campus Administration consists of five parts, which describe roles of building, people, payment, entry control and documents. Specifically, the campus consists of several dormitory buildings. Our campus offers maximum one apartment for each person. Apartments (weak entity – depends on building) differ by such parameters as monthly cost, number of beds etc. Moreover, each building can have an entry checking machines, which control all check ins/outs. An entry-checking machine has an entry permission, and if entry permission sets of the person contains this permission then he can access the building or some specific zone inside the building. In addition, entry-checking machines could form groups; inside each group there is a constraint, that person cannot go out/into this group more than one time consecutively (for example: if person have entered campus, he must out from campus, to enter again, because entry-checking machines on the perimeter of campus form a group). There are three types of person: student, guest and employee. Each guest should be invited by a person (who are not guest) and attached to that person (each guest should have at least one guest_to_person relation). As for employee, he has a role that describes his position in the campus. Student and employee can have accounts like scholarship, rent payment, salary etc. with the help of which he can make transactions according to his need. Balance of each account calculated by trigger. Each transaction must be attached to the specific account. Furthermore, each person must have at least one document with high identity level.

5. Database design schema



Many-to-many relations becomes separate tables, other constraints from ER implemented by PK, FK and Unique:

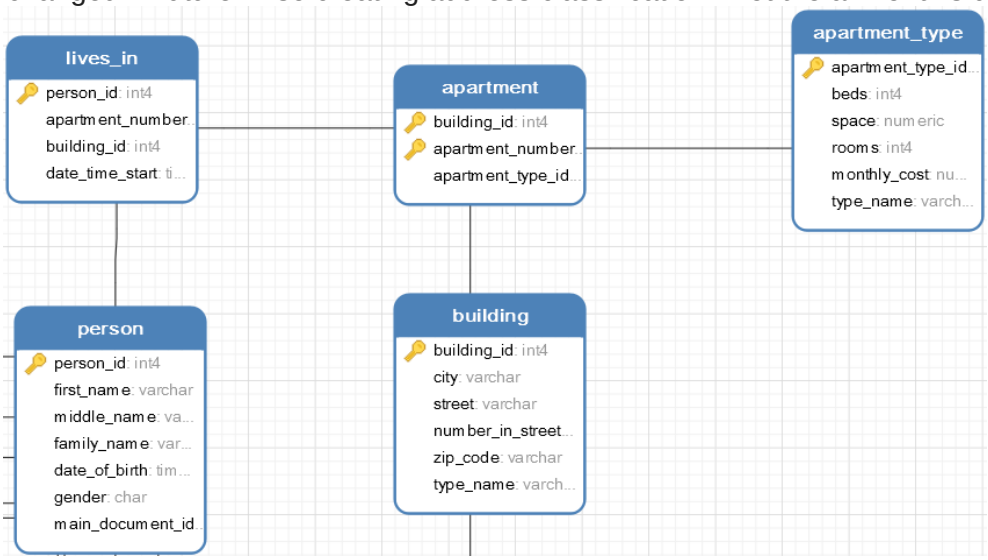
```
-- Uniques structure for table account
ALTER TABLE "public"."account" ADD UNIQUE ("person_id", "account_type_id");
-- Primary Key structure for table account
ALTER TABLE "public"."account" ADD PRIMARY KEY ("account_id");
-- Uniques structure for table account_type
ALTER TABLE "public"."account_type" ADD UNIQUE ("type_name") DEFERRABLE;
-- Primary Key structure for table account_type
ALTER TABLE "public"."account_type" ADD PRIMARY KEY ("account_type_id");
-- Uniques structure for table apartment
ALTER TABLE "public"."apartment" ADD UNIQUE ("building_id", "apartment_number");
-- Primary Key structure for table apartment
ALTER TABLE "public"."apartment" ADD PRIMARY KEY ("building_id", "apartment_number");
-- Uniques structure for table apartment_type
ALTER TABLE "public"."apartment_type" ADD UNIQUE ("type_name") DEFERRABLE;
-- Primary Key structure for table apartment_type
ALTER TABLE "public"."apartment_type" ADD PRIMARY KEY ("apartment_type_id");
-- Uniques structure for table building
ALTER TABLE "public"."building" ADD UNIQUE ("city", "street", "number_in_street");
-- Primary Key structure for table building
ALTER TABLE "public"."building" ADD PRIMARY KEY ("building_id");
-- Primary Key structure for table document
ALTER TABLE "public"."document" ADD PRIMARY KEY ("document_id");
-- Uniques structure for table document_type
ALTER TABLE "public"."document_type" ADD UNIQUE ("type_name") DEFERRABLE;
-- Primary Key structure for table document_type
ALTER TABLE "public"."document_type" ADD PRIMARY KEY ("document_type_id");
-- Primary Key structure for table employee
ALTER TABLE "public"."employee" ADD PRIMARY KEY ("person_id");
-- Uniques structure for table employee_role
ALTER TABLE "public"."employee_role" ADD UNIQUE ("role_name") DEFERRABLE;
-- Primary Key structure for table employee_role
ALTER TABLE "public"."employee_role" ADD PRIMARY KEY ("employee_role_id");
-- Primary Key structure for table entry_check_machine
ALTER TABLE "public"."entry_check_machine" ADD PRIMARY KEY ("ecm_id");
-- Uniques structure for table entry_permission
ALTER TABLE "public"."entry_permission" ADD UNIQUE ("permission_name") DEFERRABLE;
-- Primary Key structure for table entry_permission
ALTER TABLE "public"."entry_permission" ADD PRIMARY KEY ("entry_permission_id");
-- Uniques structure for table entry_permission_set
ALTER TABLE "public"."entry_permission_set" ADD UNIQUE ("set_name") DEFERRABLE;
-- Primary Key structure for table entry_permission_set
ALTER TABLE "public"."entry_permission_set" ADD PRIMARY KEY ("ep_set_id");
-- Primary Key structure for table ep_set_has_ep
ALTER TABLE "public"."ep_set_has_ep" ADD PRIMARY KEY ("entry_permission_id", "ep_set_id");
-- Primary Key structure for table guest
ALTER TABLE "public"."guest" ADD PRIMARY KEY ("person_id");
-- Primary Key structure for table guest_to_person
ALTER TABLE "public"."guest_to_person" ADD PRIMARY KEY ("person_id", "date_time_start", "guest_person_id");
-- Primary Key structure for table in_out
ALTER TABLE "public"."in_out" ADD PRIMARY KEY ("date_time", "person_id", "ecm_id");
-- Indexes structure for table lives_in
CREATE INDEX "apartment_clustered_index" ON "public"."lives_in" USING btree ("apartment_number", "building_id");
ALTER TABLE "public"."lives_in" CLUSTER ON "apartment_clustered_index";
-- Primary Key structure for table lives_in
ALTER TABLE "public"."lives_in" ADD PRIMARY KEY ("person_id");
-- Uniques structure for table person
ALTER TABLE "public"."person" ADD UNIQUE ("main_document_id") DEFERRABLE;
ALTER TABLE "public"."person" ADD UNIQUE ("first_name", "family_name", "date_of_birth", "middle_name");
-- Primary Key structure for table person
ALTER TABLE "public"."person" ADD PRIMARY KEY ("person_id");
-- Primary Key structure for table person_has_ep_set
ALTER TABLE "public"."person_has_ep_set" ADD PRIMARY KEY ("person_id", "ep_set_id");
-- Primary Key structure for table student
ALTER TABLE "public"."student" ADD PRIMARY KEY ("person_id");
-- Indexes structure for table transaction
CREATE INDEX "transaction_account_id_index" ON "public"."transaction" USING btree ("account_id");
ALTER TABLE "public"."transaction" CLUSTER ON "transaction_account_id_index";
```

```
-- Uniques structure for table transaction
ALTER TABLE "public"."transaction" ADD UNIQUE ("account_id", "date_time");
-- Primary Key structure for table transaction
ALTER TABLE "public"."transaction" ADD PRIMARY KEY ("transaction_id");
-- Foreign Key structure for table "public"."account"
ALTER TABLE "public"."account" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."account" ADD FOREIGN KEY ("account_type_id") REFERENCES "public"."account_type"
("account_type_id") ON DELETE RESTRICT ON UPDATE CASCADE;
-- Foreign Key structure for table "public"."apartment"
ALTER TABLE "public"."apartment" ADD FOREIGN KEY ("building_id") REFERENCES "public"."building" ("building_id")
ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."apartment" ADD FOREIGN KEY ("apartment_type_id") REFERENCES "public"."apartment_type"
("apartment_type_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."document"
ALTER TABLE "public"."document" ADD FOREIGN KEY ("document_type_id") REFERENCES "public"."document_type"
("document_type_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."document" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE CASCADE ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."employee"
ALTER TABLE "public"."employee" ADD FOREIGN KEY ("employee_role_id") REFERENCES "public"."employee_role"
("employee_role_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."employee" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."entry_check_machine"
ALTER TABLE "public"."entry_check_machine" ADD FOREIGN KEY ("entry_permission_id") REFERENCES
"public"."entry_permission" ("entry_permission_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."entry_check_machine" ADD FOREIGN KEY ("building_id") REFERENCES "public"."building"
("building_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."ep_set_has_ep"
ALTER TABLE "public"."ep_set_has_ep" ADD FOREIGN KEY ("ep_set_id") REFERENCES "public"."entry_permission_set"
("ep_set_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."ep_set_has_ep" ADD FOREIGN KEY ("entry_permission_id") REFERENCES
"public"."entry_permission" ("entry_permission_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."guest"
ALTER TABLE "public"."guest" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON DELETE
RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."guest_to_person"
ALTER TABLE "public"."guest_to_person" ADD FOREIGN KEY ("guest_person_id") REFERENCES "public"."guest"
("person_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."guest_to_person" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id")
ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."in_out"
ALTER TABLE "public"."in_out" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON DELETE
RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."in_out" ADD FOREIGN KEY ("ecm_id") REFERENCES "public"."entry_check_machine" ("ecm_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."lives_in"
ALTER TABLE "public"."lives_in" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."lives_in" ADD FOREIGN KEY ("building_id", "apartment_number") REFERENCES
"public"."apartment" ("building_id", "apartment_number") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."person"
ALTER TABLE "public"."person" ADD FOREIGN KEY ("main_document_id") REFERENCES "public"."document"
("document_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."person_has_ep_set"
ALTER TABLE "public"."person_has_ep_set" ADD FOREIGN KEY ("ep_set_id") REFERENCES
"public"."entry_permission_set" ("ep_set_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."person_has_ep_set" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person"
("person_id") ON DELETE CASCADE ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."student"
ALTER TABLE "public"."student" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."transaction"
ALTER TABLE "public"."transaction" ADD FOREIGN KEY ("account_id") REFERENCES "public"."account" ("account_id")
ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
```

6. Description of normalization

In our database schema in 3rd NF, except building table.

Here building table not in 3rd NF, because it has transitive dependencies, but we make a decision not to separate this table because it has few number of rows (5), in future their count likely will be less than 10 and likely will not be changed in future. Also creating address classification - not the aim of this database.



Other tables has no partial and transitive dependencies.

7. List of queries

As queries in our project we used views for several reasons:

- name of view explains what exactly happens in this query;
- this allows to separate work with queries and user interface, because if query inside view was changed – no need to change code inside app;
- this allows to create complex queries in more simple way.

```
-- 1)View structure for apartment_occupation
CREATE OR REPLACE VIEW "public"."apartment_occupation" AS
 SELECT count(li.person_id) AS beds_occupied,
    a.apartment_number,
    a.building_id,
    at.beds,
    (at.beds - count(li.person_id)) AS free_beds
   FROM ((apartment a
     LEFT JOIN lives_in li ON (((li.building_id = a.building_id) AND (li.apartment_number =
a.apartment_number))))
     JOIN apartment_type at ON ((a.apartment_type_id = at.apartment_type_id)))
  GROUP BY a.apartment_number, a.building_id, at.beds;
-- 2)View structure for apartments_with_employee
CREATE OR REPLACE VIEW "public"."apartments_with_employee" AS
 SELECT a.apartment_number,
    a.building_id
   FROM ((apartment a
     JOIN lives_in li ON (((a.building_id = li.building_id) AND (a.apartment_number =
li.apartment_number))))
     JOIN employee e ON ((li.person_id = e.person_id)));
-- 3)View structure for apartments_with_female_persons
CREATE OR REPLACE VIEW "public"."apartments_with_female_persons" AS
 SELECT a.apartment_number,
    a.building_id
   FROM ((apartment a
```

```
     JOIN lives_in li ON (((a.building_id = li.building_id) AND (a.apartment_number =
li.apartment_number))))
     JOIN person p ON ((li.person_id = p.person_id)))
  WHERE (p.gender = 'F'::bpchar);
-- 4)View structure for apartments_with_free_beds
CREATE OR REPLACE VIEW "public"."apartments_with_free_beds" AS
 SELECT apartment_occupation.beds_occupied,
    apartment_occupation.apartment_number,
    apartment_occupation.building_id,
    apartment_occupation.beds,
    apartment_occupation.free_beds
   FROM apartment_occupation
  WHERE (apartment_occupation.free_beds > 0);
-- 5)View structure for apartments_with_male_persons
CREATE OR REPLACE VIEW "public"."apartments_with_male_persons" AS
 SELECT a.apartment_number,
    a.building_id
   FROM ((apartment a
     JOIN lives_in li ON (((a.building_id = li.building_id) AND (a.apartment_number =
li.apartment_number))))
     JOIN person p ON ((li.person_id = p.person_id)))
  WHERE (p.gender = 'M'::bpchar);
-- 6)View structure for apartments_with_students
CREATE OR REPLACE VIEW "public"."apartments_with_students" AS
 SELECT a.apartment_number,
    a.building_id
   FROM ((apartment a
     JOIN lives_in li ON (((a.building_id = li.building_id) AND (a.apartment_number =
li.apartment_number))))
     JOIN student s ON ((li.person_id = s.person_id)));
-- 7)View structure for apartments_for_female_employee
CREATE OR REPLACE VIEW "public"."apartments_for_female_employee" AS
 SELECT awfb.beds_occupied,
    awfb.apartment_number,
    awfb.building_id,
    awfb.beds,
    awfb.free_beds
   FROM apartments_with_free_beds awfb
  WHERE ((NOT (EXISTS ( SELECT 1
           FROM apartments_with_male_persons awmp
          WHERE ((awfb.apartment_number = awmp.apartment_number) AND (awfb.building_id =
awmp.building_id))))) AND (NOT (EXISTS ( SELECT 1
           FROM apartments_with_students aws
          WHERE ((awfb.apartment_number = aws.apartment_number) AND (awfb.building_id =
aws.building_id))))));
-- often when need to find apartment for person there is a constraint that female couldn't live with male
and employee couldn't live with student in one apartment. However, if they want, we could settle them
together - there is no restriction in our database, it is just recommendation and help views for
administrator.
-- 8)View structure for apartments_for_female_students
CREATE OR REPLACE VIEW "public"."apartments_for_female_students" AS
 SELECT awfb.beds_occupied,
    awfb.apartment_number,
    awfb.building_id,
    awfb.beds,
    awfb.free_beds
   FROM apartments_with_free_beds awfb
  WHERE ((NOT (EXISTS ( SELECT 1
           FROM apartments_with_male_persons awmp
          WHERE ((awfb.apartment_number = awmp.apartment_number) AND (awfb.building_id =
awmp.building_id))))) AND (NOT (EXISTS ( SELECT 1
           FROM apartments_with_employee awe
          WHERE ((awfb.apartment_number = awe.apartment_number) AND (awfb.building_id =
awe.building_id))))));
-- 9)View structure for apartments_for_male_employee
CREATE OR REPLACE VIEW "public"."apartments_for_male_employee" AS
```

```
 SELECT awfb.beds_occupied,
    awfb.apartment_number,
    awfb.building_id,
    awfb.beds,
    awfb.free_beds
   FROM apartments_with_free_beds awfb
  WHERE ((NOT (EXISTS ( SELECT 1
           FROM apartments_with_female_persons awfp
          WHERE ((awfb.apartment_number = awfp.apartment_number) AND (awfb.building_id =
awfp.building_id))))) AND (NOT (EXISTS ( SELECT 1
           FROM apartments_with_students aws
          WHERE ((awfb.apartment_number = aws.apartment_number) AND (awfb.building_id =
aws.building_id))))));
-- 10)View structure for apartments_for_male_students
CREATE OR REPLACE VIEW "public"."apartments_for_male_students" AS
 SELECT awfb.beds_occupied,
    awfb.apartment_number,
    awfb.building_id,
    awfb.beds,
    awfb.free_beds
   FROM apartments_with_free_beds awfb
  WHERE ((NOT (EXISTS ( SELECT 1
           FROM apartments_with_female_persons awfp
          WHERE ((awfb.apartment_number = awfp.apartment_number) AND (awfb.building_id =
awfp.building_id))))) AND (NOT (EXISTS ( SELECT 1
           FROM apartments_with_employee awe
          WHERE ((awfb.apartment_number = awe.apartment_number) AND (awfb.building_id =
awe.building_id))))));
-- 11)View structure for employees_without_apartments
CREATE OR REPLACE VIEW "public"."employees_without_apartments" AS
 SELECT p.person_id,
    p.first_name,
    p.middle_name,
    p.family_name,
    p.date_of_birth,
    p.gender
   FROM ((person p
     JOIN employee e ON ((p.person_id = e.person_id)))
     LEFT JOIN lives_in li ON ((p.person_id = li.person_id)))
  WHERE (li.apartment_number IS NULL);
-- next views search for guests who are staying in campus more than 24 hours. Often host needs to pay for
such guests
-- 12)View structure for guest_control
CREATE OR REPLACE VIEW "public"."guest_control" AS
 SELECT guest.person_id AS guest_id,
    person.first_name,
    person.family_name,
    (g.date_time_end - g.date_time_start) AS stay_time
   FROM ((guest
     JOIN person ON ((guest.person_id = person.person_id)))
     JOIN guest_to_person g ON ((guest.person_id = g.guest_person_id)))
  WHERE ((g.date_time_end - g.date_time_start) > '24:00:00'::interval);
-- 13)View structure for guest_control_with_host
CREATE OR REPLACE VIEW "public"."guest_control_with_host" AS
 SELECT guest.person_id AS guest_person_id,
    p1.first_name AS guest_name,
    p1.family_name AS guest_family_name,
    (g.date_time_end - g.date_time_start) AS stay_time,
    p2.person_id AS host_person_id,
    p2.first_name AS host_first_name,
    p2.family_name AS host_family_name
   FROM (((guest
     JOIN person p1 ON ((guest.person_id = p1.person_id)))
     JOIN guest_to_person g ON ((guest.person_id = g.guest_person_id)))
     JOIN person p2 ON ((g.person_id = p2.person_id)))
  WHERE ((g.date_time_end - g.date_time_start) > '24:00:00'::interval);
```

```
-- next view show all the information of passing the entrance guard machine when person have a permission
(interesting thing here is "not accepted" entries, because this means that person tries to enter/escape
to/from campus territory twice or more times consecutively. This means that person gave his pass card to
someone else, that is against the rules)
-- 14)View structure for have_entry_permissions
CREATE OR REPLACE VIEW "public"."have_entry_permissions" AS
 SELECT in_out.ecm_id,
    in_out.person_id,
    in_out.direction AS in_out,
    in_out.date_time,
    in_out.accepted,
    entry_check_machine.building_id,
    entry_check_machine.entry_permission_id
   FROM (in_out
     JOIN entry_check_machine USING (ecm_id))
  WHERE (entry_check_machine.entry_permission_id IN ( SELECT ep.entry_permission_id
           FROM ((((person
             JOIN person_has_ep_set USING (person_id))
             JOIN entry_permission_set USING (ep_set_id))
             JOIN ep_set_has_ep USING (ep_set_id))
             JOIN entry_permission ep USING (entry_permission_id))
          WHERE (in_out.person_id = person.person_id)));
```

ecm_id	person_id	in_out	date_time	accepted	building_id	entry_permission_id
1	1	o	2016-09-02 09:26:33.000000	☑	1	1
1	1	o	2016-10-22 06:59:38.000000	☑	1	1
1	1	o	2016-11-19 00:59:39.758146	☑	1	1
1	1	o	2016-11-19 01:00:38.608803	☐	1	1
1	1	i	2016-11-19 01:00:51.649714	☑	1	1
3	1	i	2016-09-05 21:07:23.000000	☑	1	9
8	1	i	2016-09-01 00:08:41.000000	☑	2	2
8	1	i	2016-11-16 00:59:59.000000	☑	2	2
9	1	i	2016-10-02 01:04:02.000000	☑	2	10
9	1	o	2016-10-09 16:02:48.000000	☑	2	10
9	1	i	2016-10-12 02:35:27.000000	☑	2	10
15	1	i	2016-10-10 13:01:55.000000	☑	3	3
23	1	i	2016-09-01 00:00:03.000000	☑	4	4
23	1	o	2016-09-01 00:05:36.000000	☑	4	4
33	1	i	2016-10-03 12:54:26.000000	☐	5	5

```
-- select the last time a person get through a entrance guard machine. It will be useful for checking if a
person didn't active for a long time.
-- 15)View structure for last_time_person_accepted_in_out
CREATE OR REPLACE VIEW "public"."last_time_person_accepted_in_out" AS
 SELECT in_out.person_id,
    max(in_out.date_time) AS last_accepted_entry
   FROM in_out
  WHERE (in_out.accepted = true)
  GROUP BY in_out.person_id;
-- select the information about the unsuccessful passing (when person tries to enter a place and hi haven't
got permissions for this)
-- 16)View structure for no_entry_permission_for_in_out
CREATE OR REPLACE VIEW "public"."no_entry_permission_for_in_out" AS
 SELECT in_out.ecm_id,
    in_out.person_id,
    in_out.direction AS in_out,
    in_out.date_time,
    in_out.accepted,
    entry_check_machine.building_id,
    entry_check_machine.entry_permission_id
   FROM (in_out
     JOIN entry_check_machine USING (ecm_id))
  WHERE (NOT (entry_check_machine.entry_permission_id IN ( SELECT ep.entry_permission_id
           FROM ((((person
             JOIN person_has_ep_set USING (person_id))
             JOIN entry_permission_set USING (ep_set_id))
             JOIN ep_set_has_ep USING (ep_set_id))
             JOIN entry_permission ep USING (entry_permission_id))
          WHERE (in_out.person_id = person.person_id))));
-- 17)View structure for outdated_documents
```

```
CREATE OR REPLACE VIEW "public"."outdated_documents" AS
 SELECT document.person_id,
    document.document_id,
    document.document_type_id,
    document_type.type_name,
    document_type.identity_lvl,
    (now() - (document.update_to_date)::timestamp with time zone) AS overdue_for,
    person.first_name,
    person.family_name
   FROM ((document
     JOIN document_type USING (document_type_id))
     JOIN person USING (person_id))
  WHERE (document.update_to_date < now());
-- 18)View structure for outdated_documents_of_students_and_employee
CREATE OR REPLACE VIEW "public"."outdated_documents_of_students_and_employee" AS
 SELECT document.person_id,
    document.document_id,
    document.document_type_id,
    document_type.type_name,
    document_type.identity_lvl,
    (now() - (document.update_to_date)::timestamp with time zone) AS overdue_for,
    person.first_name,
    person.family_name
   FROM ((document
     JOIN document_type USING (document_type_id))
     JOIN person USING (person_id))
  WHERE ((document.update_to_date < now()) AND (document.person_id IN ( SELECT student.person_id
           FROM student
        UNION
         SELECT employee.person_id
           FROM employee)));
-- 19)View structure for personnel_attendance_control
CREATE OR REPLACE VIEW "public"."personnel_attendance_control" AS
 SELECT person.person_id,
    person.first_name,
    person.family_name,
    (now() - (t.last_accepted_entry)::timestamp with time zone) AS no_action_for
   FROM (( SELECT in_out.person_id,
            max(in_out.date_time) AS last_accepted_entry
           FROM in_out
          WHERE (((now() - (in_out.date_time)::timestamp with time zone) > '168:00:00'::interval) AND
(in_out.accepted = true))
          GROUP BY in_out.person_id
          ORDER BY (max(in_out.date_time))) t
     JOIN person ON ((t.person_id = person.person_id)));
-- 20)View structure for persons_inside_campus_now
CREATE OR REPLACE VIEW "public"."persons_inside_campus_now" AS
 SELECT in_out.person_id,
    in_out.ecm_id,
    in_out.direction,
    in_out.date_time,
    in_out.accepted,
    entry_check_machine.building_id,
    entry_check_machine.entry_permission_id,
    entry_check_machine.constraint_group_id,
    person.first_name,
    person.middle_name,
    person.family_name,
    person.date_of_birth,
    person.gender,
    person.main_document_id
   FROM ((in_out
     JOIN entry_check_machine USING (ecm_id))
     JOIN person USING (person_id))
  WHERE ((in_out.person_id IN ( SELECT last_time_person_accepted_in_out.person_id
           FROM last_time_person_accepted_in_out)) AND (in_out.date_time = ( SELECT
```

```
last_time_person_accepted_in_out.last_accepted_entry
           FROM last_time_person_accepted_in_out
          WHERE (last_time_person_accepted_in_out.person_id = in_out.person_id))) AND
((entry_check_machine.constraint_group_id <> 1) OR (in_out.direction <> 'o'::bpchar)));
-- next two views used for data generation and in future will be helpful for queries like how long person
was inside the campus (total or in some time interval)
-- 21)View structure for persons_with_first_accepted_campus_entry_equals_out
CREATE OR REPLACE VIEW "public"."persons_with_first_accepted_campus_entry_equals_out" AS
 SELECT in_out.person_id,
    in_out.date_time,
    in_out.ecm_id,
    in_out.direction,
    in_out.accepted
   FROM (in_out
     JOIN entry_check_machine USING (ecm_id))
  WHERE ((in_out.accepted = true) AND (entry_check_machine.constraint_group_id = 1) AND (in_out.direction =
'o'::bpchar) AND (in_out.date_time <= ALL ( SELECT io.date_time
           FROM (in_out io
             JOIN entry_check_machine ecm USING (ecm_id))
          WHERE ((io.accepted = true) AND (ecm.constraint_group_id = 1) AND (io.person_id =
in_out.person_id)))))
  GROUP BY in_out.person_id, in_out.date_time, in_out.ecm_id, in_out.direction, in_out.accepted;
-- 22)View structure for persons_with_last_accepted_campus_entry_equals_in
CREATE OR REPLACE VIEW "public"."persons_with_last_accepted_campus_entry_equals_in" AS
 SELECT in_out.person_id,
    in_out.date_time,
    in_out.ecm_id,
    in_out.direction,
    in_out.accepted,
    now() AS now
   FROM (in_out
     JOIN entry_check_machine USING (ecm_id))
  WHERE ((in_out.accepted = true) AND (entry_check_machine.constraint_group_id = 1) AND (in_out.direction =
'i'::bpchar) AND (in_out.date_time >= ALL ( SELECT io.date_time
           FROM (in_out io
             JOIN entry_check_machine ecm USING (ecm_id))
          WHERE ((io.accepted = true) AND (ecm.constraint_group_id = 1) AND (io.person_id =
in_out.person_id)))))
  GROUP BY in_out.person_id, in_out.date_time, in_out.ecm_id, in_out.direction, in_out.accepted;
-- 23)View structure for rental_fee_balance
CREATE OR REPLACE VIEW "public"."rental_fee_balance" AS
 SELECT account.person_id,
    person.first_name,
    person.family_name,
    account.balance AS rental_fee_balance
   FROM ((account
     JOIN person USING (person_id))
     JOIN account_type ON ((account_type.account_type_id = account.account_type_id)))
  WHERE ((account_type.type_name)::text = 'Rental Fee'::text);
-- 24)View structure for rental_fee_balance_negative
CREATE OR REPLACE VIEW "public"."rental_fee_balance_negative" AS
 SELECT account.person_id,
    person.first_name,
    person.family_name,
    account.balance AS rental_fee_balance
   FROM ((account
     JOIN person USING (person_id))
     JOIN account_type ON ((account_type.account_type_id = account.account_type_id)))
  WHERE (((account_type.type_name)::text = 'Rental Fee'::text) AND (account.balance < (0)::numeric));
-- 25)View structure for students_without_apartment
CREATE OR REPLACE VIEW "public"."students_without_apartment" AS
 SELECT p.person_id,
    p.first_name,
    p.middle_name,
    p.family_name,
    p.date_of_birth,
```

```
        p.gender
       FROM ((person p
         JOIN student s ON ((p.person_id = s.person_id)))
         LEFT JOIN lives_in li ON ((p.person_id = li.person_id)))
      WHERE (li.apartment_number IS NULL);
    -- 26)View structure for tuition_fee_balance
    CREATE OR REPLACE VIEW "public"."tuition_fee_balance" AS
     SELECT account.person_id,
        person.first_name,
        person.family_name,
        account.balance AS rental_fee_balance
       FROM ((account
         JOIN person USING (person_id))
         JOIN account_type ON ((account_type.account_type_id = account.account_type_id)))
      WHERE ((account_type.type_name)::text = 'Tuition Fee'::text);
    -- 27)View structure for tuition_fee_balance_negative
    CREATE OR REPLACE VIEW "public"."tuition_fee_balance_negative" AS
     SELECT account.person_id,
        person.first_name,
        person.family_name,
        account.balance AS rental_fee_balance
       FROM ((account
         JOIN person USING (person_id))
         JOIN account_type ON ((account_type.account_type_id = account.account_type_id)))
      WHERE (((account_type.type_name)::text = 'Tuition Fee'::text) AND (account.balance < (0)::numeric));

    -- 28…)Functions (sorry for text highlighting - MS Word have some problems with it)
CREATE OR REPLACE FUNCTION insert_into_in_out(_person_id INT4, _ecm_id INT4, _direction CHAR(1)) RETURNS BOOL AS
$$
  DECLARE const_g_id INT4;
  DECLARE last_direction CHAR(1);
  BEGIN
    IF (exists(SELECT ep.entry_permission_id, ecm.ecm_id  FROM person
                  NATURAL JOIN person_has_ep_set
                  NATURAL JOIN entry_permission_set
                  NATURAL JOIN ep_set_has_ep
                  NATURAL JOIN entry_permission ep
                  JOIN entry_check_machine ecm ON ep.entry_permission_id = ecm.entry_permission_id
                  WHERE person.person_id = _person_id AND ecm.ecm_id = _ecm_id))
    THEN
      SELECT constraint_group_id FROM entry_check_machine
   WHERE ecm_id = _ecm_id
   INTO const_g_id;
      IF ((const_g_id) ISNULL )
      THEN
        INSERT INTO in_out (person_id, ecm_id, direction, date_time, accepted) VALUES
(_person_id,_ecm_id,_direction,now(),TRUE);
        RETURN TRUE;
      ELSE
        SELECT direction FROM in_out
                            NATURAL JOIN entry_check_machine
                      WHERE in_out.person_id = _person_id
                            AND  entry_check_machine.constraint_group_id = const_g_id
                            AND in_out.accepted = TRUE
                      ORDER BY date_time DESC
                      LIMIT 1
        INTO last_direction;
        IF (last_direction = _direction)
          THEN
            RAISE NOTICE 'several entry in one direction';
            INSERT INTO in_out (person_id, ecm_id, direction, date_time, accepted) VALUES
(_person_id,_ecm_id,_direction,now(),FALSE );
            RETURN FALSE;
          ELSE
            INSERT INTO in_out (person_id, ecm_id, direction, date_time, accepted) VALUES
(_person_id,_ecm_id,_direction,now(),TRUE);
```

```
            RETURN TRUE;
        END IF;
      END IF;
    ELSE
      RAISE NOTICE 'no permission';
      INSERT INTO in_out (person_id, ecm_id, direction, date_time, accepted) VALUES
(_person_id,_ecm_id,_direction,now(),FALSE );
      RETURN FALSE;
    END IF;
  END;
$$ LANGUAGE plpgsql;
CREATE OR REPLACE FUNCTION add_person_into_appartment(person_id INTEGER,
                                                      apartment_number INTEGER, building_id_param INTEGER,
date_time_start_param TIMESTAMP) RETURNS VOID AS $$
BEGIN
  IF (apartment_number IN ( SELECT a.apartment_number FROM apartments_with_free_beds AS a WHERE a.building_id =
building_id_param))
  THEN
    INSERT INTO lives_in VALUES (person_id, apartment_number, building_id_param, date_time_start_param);
  ELSE
    RAISE EXCEPTION 'No free places'
    USING HINT = 'Check apt number and building id';
  END IF ;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION add_new_student (first_name VARCHAR(50),middle_name VARCHAR(50),family_name
VARCHAR(200),
                                            gender CHAR, dob TIMESTAMP, image_path VARCHAR(200),
                                            scholarship INTEGER) RETURNS VOID AS $$
DECLARE
  new_person_id INTEGER;
  new_doc_id INTEGER;
BEGIN
  SELECT nextval('person_person_id_seq'::REGCLASS) INTO new_person_id;
  BEGIN
    SET CONSTRAINTS ALL DEFERRED;
    INSERT INTO document VALUES (nextval('document_document_id_seq'::REGCLASS), new_person_id, current_timestamp
+ INTERVAL '1 year', 1, image_path)
    RETURNING document_id INTO new_doc_id;
    INSERT INTO person VALUES (new_person_id, first_name, middle_name, family_name, dob, gender,  new_doc_id);
  END;
    INSERT INTO student VALUES (new_person_id, scholarship);
    INSERT INTO account VALUES (nextval('account_account_id_seq'::REGCLASS), new_person_id,1); -- rental_fee
    INSERT INTO account VALUES (nextval('account_account_id_seq'::REGCLASS), new_person_id,2); -- tuition_fee
    INSERT INTO account VALUES (nextval('account_account_id_seq'::REGCLASS), new_person_id,4); -- Scholarship
    INSERT INTO person_has_ep_set VALUES (new_person_id, 4);  -- base for persons
    INSERT INTO person_has_ep_set VALUES (new_person_id, 1);  -- base for students
END;
$$ LANGUAGE plpgsql;
CREATE OR REPLACE FUNCTION add_new_employee (first_name VARCHAR(50),middle_name VARCHAR(50),family_name
VARCHAR(200),
                                             gender CHAR, dob TIMESTAMP, salary INTEGER,
                                             role VARCHAR(50), image_path VARCHAR(200)) RETURNS VOID AS $$
DECLARE
  new_person_id INTEGER;
  new_doc_id INTEGER;
  role_id INTEGER;
BEGIN
  SELECT employee_role.employee_role_id INTO role_id FROM employee_role WHERE role_name LIKE role;
  SELECT nextval('person_person_id_seq'::REGCLASS) INTO new_person_id;
  BEGIN
    SET CONSTRAINTS ALL DEFERRED;
    INSERT INTO document VALUES (nextval('document_document_id_seq'::REGCLASS), new_person_id, current_timestamp
+ INTERVAL '1 year', 1, image_path)
    RETURNING document_id INTO new_doc_id;
```

```
    INSERT INTO person VALUES (new_person_id, first_name, middle_name, family_name, dob, gender,  new_doc_id);
  END;
    INSERT INTO employee VALUES (new_person_id, salary, role_id);
    INSERT INTO account VALUES (nextval('account_account_id_seq'::REGCLASS), new_person_id, 1); -- rental_fee
    INSERT INTO account VALUES (nextval('account_account_id_seq'::REGCLASS), new_person_id, 3); -- Salary
    INSERT INTO person_has_ep_set VALUES (new_person_id, 4);  -- base for persons
    IF (role_id = 4 OR role_id = 5)
    THEN
      INSERT INTO person_has_ep_set VALUES (new_person_id, 6);  -- canteen staff
    ELSEIF (role_id = 3 OR role_id = 6)
      THEN
        INSERT INTO person_has_ep_set VALUES (new_person_id, 2);  -- administrator
    ELSEIF (role_id = 2)
      THEN
        INSERT INTO person_has_ep_set VALUES (new_person_id, 7);  -- cleaning
    END IF;
END;
$$ LANGUAGE plpgsql;
-- -----------------------
-- add guest to person
-- -----------------------
DROP FUNCTION IF EXISTS add_guest_to_person(integer,integer);
CREATE OR REPLACE FUNCTION public.add_guest_to_person(_guest_person_id integer, _person_id integer)
 RETURNS BOOL
 LANGUAGE plpgsql
AS $function$
BEGIN
 IF (NOT exists(SELECT * FROM guest WHERE guest.person_id = _guest_person_id))
  THEN
    RAISE NOTICE 'No guest person';
   RETURN FALSE;
 END IF;
  IF (not exists(SELECT person_id from student WHERE student.person_id = _person_id
                 UNION
               SELECT person_id FROM  employee WHERE employee.person_id = _person_id))
    THEN
    RAISE NOTICE 'Host should be students or employee';
    RETURN FALSE;
  END IF;
  IF (exists(SELECT * FROM guest_to_person
                                      WHERE guest_to_person.guest_person_id = _guest_person_id
                                        AND  guest_to_person.date_time_end IS NULL))
    THEN
    RAISE NOTICE 'Guest should leave before come in again';
    RETURN FALSE;
  END IF;
  INSERT INTO guest_to_person (guest_person_id, person_id, date_time_start, date_time_end )  VALUES
(_guest_person_id,_person_id,now(),NULL);
  RETURN TRUE;
END;
$function$

CREATE OR REPLACE FUNCTION public.create_guest_person(_first_name character varying, _middle_name character
varying, _family_name character varying, _gender character, _date_of_birth timestamp without time zone,
_update_to_date timestamp without time zone, _document_type_id integer, _image_path character varying,
_host_person_id integer)
 RETURNS boolean
 LANGUAGE plpgsql
AS $function$
DECLARE
  new_person_id INTEGER;
  new_guest_id INTEGER;
  new_doc_id INTEGER;
BEGIN
  SELECT person.person_id FROM person WHERE person.date_of_birth = _date_of_birth
                                      AND person.family_name = _family_name
```

```
                                        AND person.first_name = _first_name
  INTO new_person_id;
  IF (new_person_id IS NOT NULL )
    THEN
      SELECT person_id FROM guest WHERE person_id = new_person_id INTO new_guest_id;
      IF (new_guest_id IS NULL )
        THEN
          RAISE EXCEPTION 'Person is exist but not as guest'
          USING HINT = 'You can not create a guest who already working here';
        ELSE
          IF (SELECT add_guest_to_person(new_guest_id,_host_person_id) = TRUE)
            THEN
            RETURN TRUE;
            ELSE
            RETURN FALSE;
          END IF;
      END IF;
    ELSE
      SELECT nextval('person_person_id_seq'::REGCLASS) INTO new_person_id;
      SELECT nextval('document_document_id_seq'::REGCLASS) INTO new_doc_id;
      BEGIN
        SET CONSTRAINTS ALL DEFERRED;
        INSERT INTO document VALUES (new_doc_id, new_person_id, _update_to_date, _document_type_id,
_image_path);
        INSERT INTO person VALUES (new_person_id, _first_name, _middle_name, _family_name, _date_of_birth,
_gender,  new_doc_id);
        INSERT INTO guest VALUES (new_person_id);
      END;
      IF (SELECT add_guest_to_person(new_person_id,_host_person_id) = TRUE)
        THEN
          RETURN TRUE;
        ELSE
          BEGIN
            SET CONSTRAINTS ALL DEFERRED;
            DELETE FROM person WHERE person_id = new_person_id;
            DELETE FROM document WHERE document_id = new_doc_id;
            DELETE FROM guest WHERE person_id = new_person_id;
          END;
          RETURN FALSE;
      END IF;
  END IF;
END;
$function$
DROP FUNCTION IF EXISTS guest_left_from_person(integer,integer);
CREATE OR REPLACE FUNCTION public.guest_left_from_person(_guest_person_id integer, _person_id integer)
 RETURNS boolean
 LANGUAGE plpgsql
AS $function$
BEGIN
 IF (NOT exists(SELECT * FROM guest WHERE guest.person_id = _guest_person_id))
  THEN
    RAISE NOTICE 'No guest person';
    RETURN FALSE;
  END IF;
  IF (not exists(SELECT person_id from student WHERE student.person_id = _person_id
                 UNION
               SELECT person_id FROM  employee WHERE employee.person_id = _person_id))
    THEN
    RAISE NOTICE 'Host should be students or employee';
   RETURN FALSE;
  END IF;
  IF (NOT exists(SELECT * FROM guest_to_person
                                      WHERE guest_to_person.guest_person_id = _guest_person_id
                                        AND  guest_to_person.person_id = _person_id
                                        AND  guest_to_person.date_time_end IS NULL))
    THEN
```

```
      RAISE NOTICE 'Not registered';
      RETURN FALSE;
    ELSE
      UPDATE guest_to_person SET date_time_end = now()
      WHERE guest_person_id = _guest_person_id
                                        AND  person_id = _person_id
                                        AND  date_time_end IS NULL;
      RETURN TRUE;
  END IF;
END;
$function$
```

8. Database Creation queries

```
-- Sequence structure for account_account_id_seq
DROP SEQUENCE IF EXISTS "public"."account_account_id_seq" CASCADE;
CREATE SEQUENCE "public"."account_account_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 64
 CACHE 1;
SELECT setval('"public"."account_account_id_seq"', 64, true);
-- Sequence structure for apartment_type_apartment_type_id_seq
DROP SEQUENCE IF EXISTS "public"."apartment_type_apartment_type_id_seq" CASCADE;
CREATE SEQUENCE "public"."apartment_type_apartment_type_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 4
 CACHE 1;
-- Sequence structure for building_building_id_seq
DROP SEQUENCE IF EXISTS "public"."building_building_id_seq" CASCADE;
CREATE SEQUENCE "public"."building_building_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 6
 CACHE 1;
-- Sequence structure for document_document_id_seq
DROP SEQUENCE IF EXISTS "public"."document_document_id_seq" CASCADE;
CREATE SEQUENCE "public"."document_document_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 27
 CACHE 1;
-- Sequence structure for document_type_document_type_id_seq
DROP SEQUENCE IF EXISTS "public"."document_type_document_type_id_seq" CASCADE;
CREATE SEQUENCE "public"."document_type_document_type_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 10
 CACHE 1;
SELECT setval('"public"."document_type_document_type_id_seq"', 10, true);
-- Sequence structure for employee_role_employee_role_id_seq
DROP SEQUENCE IF EXISTS "public"."employee_role_employee_role_id_seq" CASCADE;
CREATE SEQUENCE "public"."employee_role_employee_role_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 7
 CACHE 1;
-- Sequence structure for entry_check_machine_ecm_id_seq
DROP SEQUENCE IF EXISTS "public"."entry_check_machine_ecm_id_seq" CASCADE;
CREATE SEQUENCE "public"."entry_check_machine_ecm_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 84
 CACHE 1;
SELECT setval('"public"."entry_check_machine_ecm_id_seq"', 84, true);
-- Sequence structure for entry_permission_entry_permission_id_seq
DROP SEQUENCE IF EXISTS "public"."entry_permission_entry_permission_id_seq" CASCADE;
CREATE SEQUENCE "public"."entry_permission_entry_permission_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 23
 CACHE 1;
SELECT setval('"public"."entry_permission_entry_permission_id_seq"', 23, true);
-- Sequence structure for entry_permission_set_ep_set_id_seq
DROP SEQUENCE IF EXISTS "public"."entry_permission_set_ep_set_id_seq" CASCADE;
```

```
CREATE SEQUENCE "public"."entry_permission_set_ep_set_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 8
 CACHE 1;
-- Sequence structure for person_person_id_seq
DROP SEQUENCE IF EXISTS "public"."person_person_id_seq" CASCADE;
CREATE SEQUENCE "public"."person_person_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 26
 CACHE 1;
-- Sequence structure for transaction_transaction_id_seq
DROP SEQUENCE IF EXISTS "public"."transaction_transaction_id_seq" CASCADE;
CREATE SEQUENCE "public"."transaction_transaction_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 1175
 CACHE 1;
-- Sequence structure for transaction_type_transaction_type_id_seq
DROP SEQUENCE IF EXISTS "public"."transaction_type_transaction_type_id_seq" CASCADE;
CREATE SEQUENCE "public"."transaction_type_transaction_type_id_seq"
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 5
 CACHE 1;
-- Table structure for account
DROP TABLE IF EXISTS "public"."account" CASCADE;
CREATE TABLE "public"."account" (
"account_id" int4 DEFAULT nextval('account_account_id_seq'::regclass) NOT NULL,
"person_id" int4 NOT NULL,
"account_type_id" int4 NOT NULL,
"balance" numeric(255));
-- Table structure for account_type
DROP TABLE IF EXISTS "public"."account_type" CASCADE;
CREATE TABLE "public"."account_type" (
"account_type_id" int4 DEFAULT nextval('transaction_type_transaction_type_id_seq'::regclass) NOT NULL,
"type_name" varchar(50) COLLATE "default" NOT NULL);
-- Table structure for apartment
DROP TABLE IF EXISTS "public"."apartment" CASCADE;
CREATE TABLE "public"."apartment" (
"building_id" int4 NOT NULL,
"apartment_number" int4 NOT NULL,
"apartment_type_id" int4 NOT NULL);
-- Table structure for apartment_type
DROP TABLE IF EXISTS "public"."apartment_type" CASCADE;
CREATE TABLE "public"."apartment_type" (
"apartment_type_id" int4 DEFAULT nextval('apartment_type_apartment_type_id_seq'::regclass) NOT NULL,
"beds" int4,
"space" numeric(4,2),
"rooms" int4,
"monthly_cost" numeric(8,2),
"type_name" varchar(50) COLLATE "default");
-- Table structure for building
DROP TABLE IF EXISTS "public"."building" CASCADE;
CREATE TABLE "public"."building" (
"building_id" int4 DEFAULT nextval('building_building_id_seq'::regclass) NOT NULL,
"city" varchar(50) COLLATE "default" NOT NULL,
"street" varchar(100) COLLATE "default" NOT NULL,
"number_in_street" varchar(6) COLLATE "default" NOT NULL,
"zip_code" varchar(30) COLLATE "default",
"type_name" varchar(50) COLLATE "default");
-- Table structure for document
DROP TABLE IF EXISTS "public"."document" CASCADE;
CREATE TABLE "public"."document" (
"document_id" int4 DEFAULT nextval('document_document_id_seq'::regclass) NOT NULL,
```

```
"person_id" int4 NOT NULL,
"update_to_date" timestamp(6) NOT NULL,
"document_type_id" int4 NOT NULL,
"image_path" varchar(255) COLLATE "default" NOT NULL);
-- Table structure for document_type
DROP TABLE IF EXISTS "public"."document_type" CASCADE;
CREATE TABLE "public"."document_type" (
"document_type_id" int4 DEFAULT nextval('document_type_document_type_id_seq'::regclass) NOT NULL,
"type_name" varchar(50) COLLATE "default" NOT NULL,
"identity_lvl" varchar(255) COLLATE "default" NOT NULL);
-- Table structure for employee
DROP TABLE IF EXISTS "public"."employee" CASCADE;
CREATE TABLE "public"."employee" (
"person_id" int4 NOT NULL,
"salary" numeric(8,2) NOT NULL,
"employee_role_id" int4 NOT NULL);
-- Table structure for employee_role
DROP TABLE IF EXISTS "public"."employee_role" CASCADE;
CREATE TABLE "public"."employee_role" (
"employee_role_id" int4 DEFAULT nextval('employee_role_employee_role_id_seq'::regclass) NOT NULL,
"role_name" varchar(50) COLLATE "default" NOT NULL,
"description" varchar(255) COLLATE "default");
-- Table structure for entry_check_machine
DROP TABLE IF EXISTS "public"."entry_check_machine" CASCADE;
CREATE TABLE "public"."entry_check_machine" (
"ecm_id" int4 DEFAULT nextval('entry_check_machine_ecm_id_seq'::regclass) NOT NULL,
"building_id" int4 NOT NULL,
"entry_permission_id" int4 NOT NULL,
"constraint_group_id" int4);
-- Table structure for entry_permission
DROP TABLE IF EXISTS "public"."entry_permission" CASCADE;
CREATE TABLE "public"."entry_permission" (
"entry_permission_id" int4 DEFAULT nextval('entry_permission_entry_permission_id_seq'::regclass) NOT NULL,
"permission_name" varchar(100) COLLATE "default" NOT NULL);
-- Table structure for entry_permission_set
DROP TABLE IF EXISTS "public"."entry_permission_set" CASCADE;
CREATE TABLE "public"."entry_permission_set" (
"ep_set_id" int4 DEFAULT nextval('entry_permission_set_ep_set_id_seq'::regclass) NOT NULL,
"set_name" varchar(50) COLLATE "default" NOT NULL);
-- Table structure for ep_set_has_ep
DROP TABLE IF EXISTS "public"."ep_set_has_ep" CASCADE;
CREATE TABLE "public"."ep_set_has_ep" (
"entry_permission_id" int4 NOT NULL,
"ep_set_id" int4 NOT NULL);


-- Table structure for guest
DROP TABLE IF EXISTS "public"."guest" CASCADE;
CREATE TABLE "public"."guest" (
"person_id" int4 NOT NULL);
-- Table structure for guest_to_person
DROP TABLE IF EXISTS "public"."guest_to_person" CASCADE;
CREATE TABLE "public"."guest_to_person" (
"person_id" int4 NOT NULL,
"date_time_start" timestamp(6) NOT NULL,
"date_time_end" timestamp(6),
"guest_person_id" int4 NOT NULL);
-- Table structure for in_out
DROP TABLE IF EXISTS "public"."in_out" CASCADE;
CREATE TABLE "public"."in_out" (
"person_id" int4 NOT NULL,
"ecm_id" int4 NOT NULL,
"direction" char(1) COLLATE "default" NOT NULL,
"date_time" timestamp(6) NOT NULL,
"accepted" bool NOT NULL);
-- Table structure for lives_in
DROP TABLE IF EXISTS "public"."lives_in" CASCADE;
CREATE TABLE "public"."lives_in" (
"person_id" int4 NOT NULL,
```

```
"apartment_number" int4 NOT NULL,
"building_id" int4 NOT NULL,
"date_time_start" timestamp(6) NOT NULL);
-- Table structure for person
DROP TABLE IF EXISTS "public"."person" CASCADE;
CREATE TABLE "public"."person" (
"person_id" int4 DEFAULT nextval('person_person_id_seq'::regclass) NOT NULL,
"first_name" varchar(50) COLLATE "default" NOT NULL,
"middle_name" varchar(50) COLLATE "default",
"family_name" varchar(50) COLLATE "default" NOT NULL,
"date_of_birth" timestamp(6) NOT NULL,
"gender" char(1) COLLATE "default" NOT NULL,
"main_document_id" int4 NOT NULL);
-- Table structure for person_has_ep_set
DROP TABLE IF EXISTS "public"."person_has_ep_set" CASCADE;
CREATE TABLE "public"."person_has_ep_set" (
"person_id" int4 NOT NULL,
"ep_set_id" int4 NOT NULL);
-- Table structure for student
DROP TABLE IF EXISTS "public"."student" CASCADE;
CREATE TABLE "public"."student" (
"person_id" int4 NOT NULL,
"scholarship" numeric(8,2) NOT NULL);
-- Table structure for transaction
DROP TABLE IF EXISTS "public"."transaction" CASCADE;
CREATE TABLE "public"."transaction" (
"transaction_id" int4 DEFAULT nextval('transaction_transaction_id_seq'::regclass) NOT NULL,
"amount" numeric(8,2) NOT NULL,
"account_id" int4 NOT NULL,
"date_time" timestamp(6) NOT NULL,
"description" varchar(255) COLLATE "default");
ALTER SEQUENCE "public"."account_account_id_seq" OWNED BY "account"."account_id";
ALTER SEQUENCE "public"."apartment_type_apartment_type_id_seq" OWNED BY "apartment_type"."apartment_type_id";
ALTER SEQUENCE "public"."building_building_id_seq" OWNED BY "building"."building_id";
ALTER SEQUENCE "public"."document_document_id_seq" OWNED BY "document"."document_id";
ALTER SEQUENCE "public"."document_type_document_type_id_seq" OWNED BY "document_type"."document_type_id";
ALTER SEQUENCE "public"."employee_role_employee_role_id_seq" OWNED BY "employee_role"."employee_role_id";
ALTER SEQUENCE "public"."entry_check_machine_ecm_id_seq" OWNED BY "entry_check_machine"."ecm_id";
ALTER SEQUENCE "public"."entry_permission_entry_permission_id_seq" OWNED BY
"entry_permission"."entry_permission_id";
ALTER SEQUENCE "public"."entry_permission_set_ep_set_id_seq" OWNED BY "entry_permission_set"."ep_set_id";
ALTER SEQUENCE "public"."person_person_id_seq" OWNED BY "person"."person_id";
ALTER SEQUENCE "public"."transaction_transaction_id_seq" OWNED BY "transaction"."transaction_id";
ALTER SEQUENCE "public"."transaction_type_transaction_type_id_seq" OWNED BY "account_type"."account_type_id";
-- Indexes structure for table account
CREATE UNIQUE INDEX "account_person_id_account_type_id_idx" ON "public"."account" USING btree ("person_id",
"account_type_id");
-- Uniques structure for table account
ALTER TABLE "public"."account" ADD UNIQUE ("person_id", "account_type_id");
-- Primary Key structure for table account
ALTER TABLE "public"."account" ADD PRIMARY KEY ("account_id");
-- Uniques structure for table account_type
ALTER TABLE "public"."account_type" ADD UNIQUE ("type_name") DEFERRABLE;
-- Primary Key structure for table account_type
ALTER TABLE "public"."account_type" ADD PRIMARY KEY ("account_type_id");
-- Indexes structure for table apartment
CREATE UNIQUE INDEX "apartment_building_id_apartment_number_idx" ON "public"."apartment" USING btree
("building_id", "apartment_number");
-- Uniques structure for table apartment
ALTER TABLE "public"."apartment" ADD UNIQUE ("building_id", "apartment_number");
-- Primary Key structure for table apartment
ALTER TABLE "public"."apartment" ADD PRIMARY KEY ("building_id", "apartment_number");
-- Uniques structure for table apartment_type
ALTER TABLE "public"."apartment_type" ADD UNIQUE ("type_name") DEFERRABLE;
-- Primary Key structure for table apartment_type
ALTER TABLE "public"."apartment_type" ADD PRIMARY KEY ("apartment_type_id");
```

```
-- Uniques structure for table building
ALTER TABLE "public"."building" ADD UNIQUE ("city", "street", "number_in_street");
-- Primary Key structure for table building
ALTER TABLE "public"."building" ADD PRIMARY KEY ("building_id");
-- Indexes structure for table document
CREATE UNIQUE INDEX "document_person_id_document_type_id_idx" ON "public"."document" USING btree ("person_id",
"document_type_id");
CREATE INDEX "document_person_id_clust_index" ON "public"."document" USING btree ("person_id");
ALTER TABLE "public"."document" CLUSTER ON "document_person_id_clust_index";
-- Primary Key structure for table document
ALTER TABLE "public"."document" ADD PRIMARY KEY ("document_id");
-- Uniques structure for table document_type
ALTER TABLE "public"."document_type" ADD UNIQUE ("type_name") DEFERRABLE;
-- Primary Key structure for table document_type
ALTER TABLE "public"."document_type" ADD PRIMARY KEY ("document_type_id");
-- Primary Key structure for table employee
ALTER TABLE "public"."employee" ADD PRIMARY KEY ("person_id");
-- Uniques structure for table employee_role
ALTER TABLE "public"."employee_role" ADD UNIQUE ("role_name") DEFERRABLE;
-- Primary Key structure for table employee_role
ALTER TABLE "public"."employee_role" ADD PRIMARY KEY ("employee_role_id");
-- Primary Key structure for table entry_check_machine
ALTER TABLE "public"."entry_check_machine" ADD PRIMARY KEY ("ecm_id");
-- Uniques structure for table entry_permission
ALTER TABLE "public"."entry_permission" ADD UNIQUE ("permission_name") DEFERRABLE;
-- Primary Key structure for table entry_permission
ALTER TABLE "public"."entry_permission" ADD PRIMARY KEY ("entry_permission_id");
-- Uniques structure for table entry_permission_set
ALTER TABLE "public"."entry_permission_set" ADD UNIQUE ("set_name") DEFERRABLE;
-- Primary Key structure for table entry_permission_set
ALTER TABLE "public"."entry_permission_set" ADD PRIMARY KEY ("ep_set_id");
-- Primary Key structure for table ep_set_has_ep
ALTER TABLE "public"."ep_set_has_ep" ADD PRIMARY KEY ("entry_permission_id", "ep_set_id");
-- Primary Key structure for table guest
ALTER TABLE "public"."guest" ADD PRIMARY KEY ("person_id");
-- Primary Key structure for table guest_to_person
ALTER TABLE "public"."guest_to_person" ADD PRIMARY KEY ("person_id", "date_time_start", "guest_person_id");
-- Primary Key structure for table in_out
ALTER TABLE "public"."in_out" ADD PRIMARY KEY ("date_time", "person_id", "ecm_id");
-- Indexes structure for table lives_in
CREATE INDEX "apartment_clustered_index" ON "public"."lives_in" USING btree ("apartment_number", "building_id");
ALTER TABLE "public"."lives_in" CLUSTER ON "apartment_clustered_index";
-- Primary Key structure for table lives_in
ALTER TABLE "public"."lives_in" ADD PRIMARY KEY ("person_id");
-- Uniques structure for table person
ALTER TABLE "public"."person" ADD UNIQUE ("main_document_id") DEFERRABLE;
ALTER TABLE "public"."person" ADD UNIQUE ("first_name", "family_name", "date_of_birth", "middle_name");
-- Primary Key structure for table person
ALTER TABLE "public"."person" ADD PRIMARY KEY ("person_id");
-- Primary Key structure for table person_has_ep_set
ALTER TABLE "public"."person_has_ep_set" ADD PRIMARY KEY ("person_id", "ep_set_id");
-- Primary Key structure for table student
ALTER TABLE "public"."student" ADD PRIMARY KEY ("person_id");
-- Indexes structure for table transaction
CREATE INDEX "transaction_account_id_index" ON "public"."transaction" USING btree ("account_id");
ALTER TABLE "public"."transaction" CLUSTER ON "transaction_account_id_index";
-- Triggers structure for table transaction
CREATE OR REPLACE FUNCTION update_account_balance_on_insert() RETURNS TRIGGER AS $$
DECLARE
BEGIN
   UPDATE account
    SET balance = (SELECT sum(amount) FROM transaction WHERE transaction.account_id = new.account_id)
     WHERE account.account_id = new.account_id;
  RETURN new;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION update_account_balance_on_delete() RETURNS TRIGGER AS $$
DECLARE
BEGIN
   UPDATE account
    SET balance = (SELECT sum(amount) FROM transaction WHERE transaction.account_id = old.account_id)
     WHERE account.account_id = old.account_id;
  RETURN old;
END;
$$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS account_balance_update_on_insert ON "public"."transaction" CASCADE;
CREATE TRIGGER "account_balance_update_on_insert" AFTER INSERT ON "public"."transaction"
FOR EACH ROW
EXECUTE PROCEDURE "update_account_balance_on_insert"();
DROP TRIGGER IF EXISTS account_balance_on_delete ON "public"."transaction" CASCADE;
CREATE TRIGGER "account_balance_on_delete" AFTER DELETE ON "public"."transaction"
FOR EACH ROW
EXECUTE PROCEDURE "update_account_balance_on_delete"();
-- Uniques structure for table transaction
ALTER TABLE "public"."transaction" ADD UNIQUE ("account_id", "date_time");
-- Primary Key structure for table transaction
ALTER TABLE "public"."transaction" ADD PRIMARY KEY ("transaction_id");
-- Foreign Key structure for table "public"."account"
ALTER TABLE "public"."account" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."account" ADD FOREIGN KEY ("account_type_id") REFERENCES "public"."account_type"
("account_type_id") ON DELETE RESTRICT ON UPDATE CASCADE;
-- Foreign Key structure for table "public"."apartment"
ALTER TABLE "public"."apartment" ADD FOREIGN KEY ("building_id") REFERENCES "public"."building" ("building_id")
ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."apartment" ADD FOREIGN KEY ("apartment_type_id") REFERENCES "public"."apartment_type"
("apartment_type_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."document"
ALTER TABLE "public"."document" ADD FOREIGN KEY ("document_type_id") REFERENCES "public"."document_type"
("document_type_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."document" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE CASCADE ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."employee"
ALTER TABLE "public"."employee" ADD FOREIGN KEY ("employee_role_id") REFERENCES "public"."employee_role"
("employee_role_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."employee" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."entry_check_machine"
ALTER TABLE "public"."entry_check_machine" ADD FOREIGN KEY ("entry_permission_id") REFERENCES
"public"."entry_permission" ("entry_permission_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."entry_check_machine" ADD FOREIGN KEY ("building_id") REFERENCES "public"."building"
("building_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."ep_set_has_ep"
ALTER TABLE "public"."ep_set_has_ep" ADD FOREIGN KEY ("ep_set_id") REFERENCES "public"."entry_permission_set"
("ep_set_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."ep_set_has_ep" ADD FOREIGN KEY ("entry_permission_id") REFERENCES
"public"."entry_permission" ("entry_permission_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."guest"
ALTER TABLE "public"."guest" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON DELETE
RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."guest_to_person"
ALTER TABLE "public"."guest_to_person" ADD FOREIGN KEY ("guest_person_id") REFERENCES "public"."guest"
("person_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."guest_to_person" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id")
ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."in_out"
ALTER TABLE "public"."in_out" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON DELETE
RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."in_out" ADD FOREIGN KEY ("ecm_id") REFERENCES "public"."entry_check_machine" ("ecm_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."lives_in"
```

```
ALTER TABLE "public"."lives_in" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."lives_in" ADD FOREIGN KEY ("building_id", "apartment_number") REFERENCES
"public"."apartment" ("building_id", "apartment_number") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."person"
ALTER TABLE "public"."person" ADD FOREIGN KEY ("main_document_id") REFERENCES "public"."document"
("document_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."person_has_ep_set"
ALTER TABLE "public"."person_has_ep_set" ADD FOREIGN KEY ("ep_set_id") REFERENCES
"public"."entry_permission_set" ("ep_set_id") ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
ALTER TABLE "public"."person_has_ep_set" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person"
("person_id") ON DELETE CASCADE ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."student"
ALTER TABLE "public"."student" ADD FOREIGN KEY ("person_id") REFERENCES "public"."person" ("person_id") ON
DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
-- Foreign Key structure for table "public"."transaction"
ALTER TABLE "public"."transaction" ADD FOREIGN KEY ("account_id") REFERENCES "public"."account" ("account_id")
ON DELETE RESTRICT ON UPDATE CASCADE DEFERRABLE;
```