800 1222·2022 ANNI
UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Foundations of Databases A.Y. 2022-2023

Homework 3 – Physical Design

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

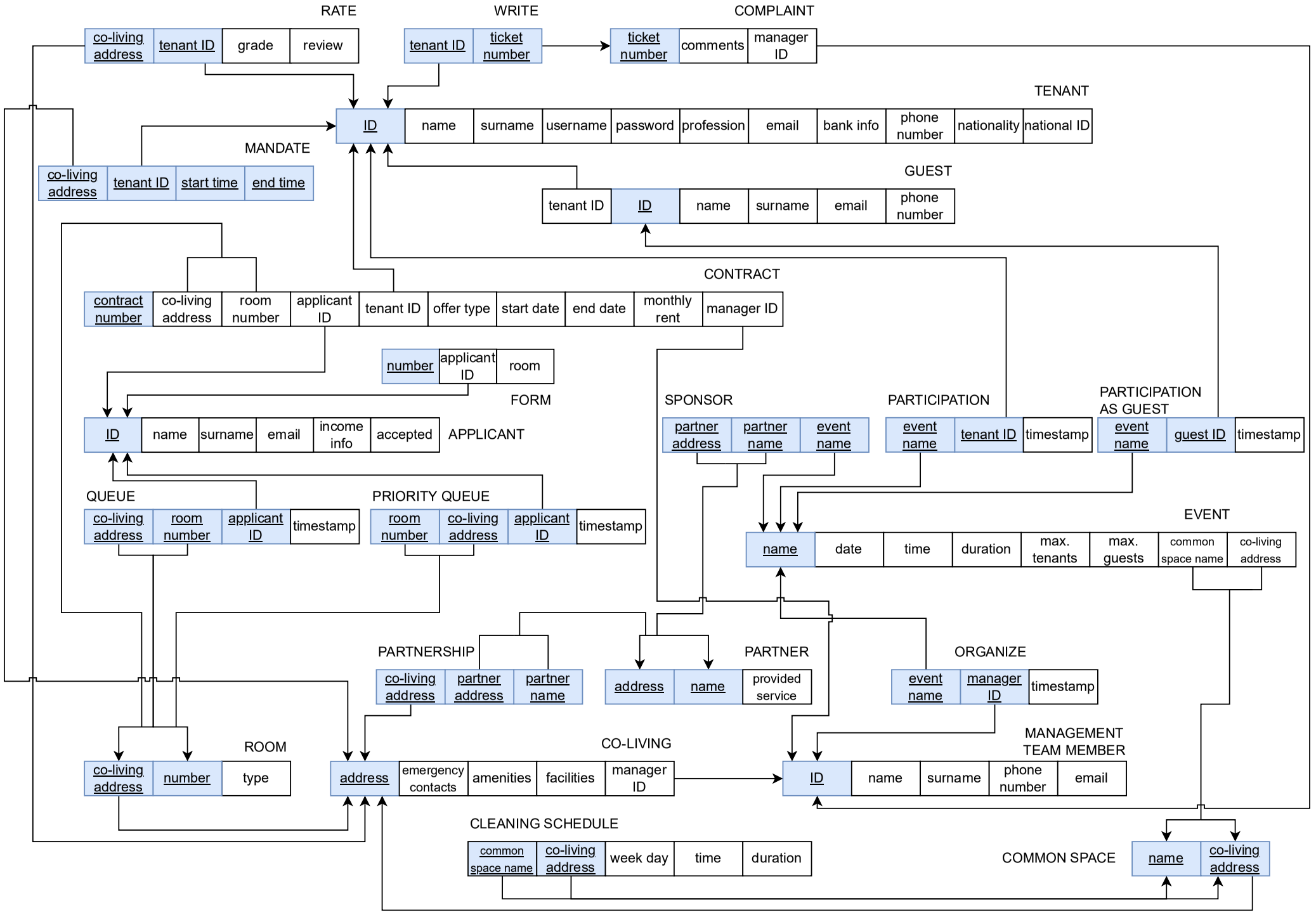
Deadline: December 17, 2022

Team acronym	CLDB	
Last Name	**First Name**	**Student Number**
Zattarin	Nicole	2057138
Battut	Anaïs	2065759
Fragne	Anaïs	2073077
Zanin	Daria	2089062
Broetto	Riccardo	2088250
Gasparini	Marco	2087927
Gobbin	Alberto	1232223
D'Este	Luca	2082123
Sulku	Euxhenio	2080611
Ortiz de Zárate	Nicolás	2044778

Variations to the Relational Schema

Timestamp attributes have been added in the *participation*, *participation as guest* and *organize* relations for sake of completeness, in order to better track events and people participating.

Moreover we got rid of *co-living name* attribute in the *form* relation, since it turned out to be redundant and we also deleted the *co-living addess* from the *complaint relation* for the same reason.


RATE
co-living address
tenant ID
grade
review
WRITE
tenant ID
ticket number
COMPLAINT
ticket number
comments
manager ID
TENANT
ID
name
surname
username
password
profession
email
bank info
phone number
nationality
national ID
MANDATE
co-living address
tenant ID
start time
end time
GUEST
tenant ID
ID
name
surname
email
phone number
CONTRACT
contract number
co-living address
room number
applicant ID
tenant ID
offer type
start date
end date
monthly rent
manager ID
number
applicant ID
room
FORM
SPONSOR
partner address
partner name
event name
PARTICIPATION
event name
tenant ID
timestamp
PARTICIPATION AS GUEST
event name
guest ID
timestamp
ID
name
surname
email
income info
accepted
APPLICANT
QUEUE
co-living address
room number
applicant ID
timestamp
PRIORITY QUEUE
room number
co-living address
applicant ID
timestamp
EVENT
name
date
time
duration
max. tenants
max. guests
common space name
co-living address
PARTNERSHIP
co-living address
partner address
partner name
address
name
provided service
PARTNER
ORGANIZE
event name
manager ID
timestamp
ROOM
co-living address
number
type
address
emergency contacts
amenities
facilities
manager ID
CO-LIVING
ID
name
surname
phone number
email
MANAGEMENT TEAM MEMBER
CLEANING SCHEDULE
common space name
co-living address
week day
time
duration
COMMON SPACE
name
co-living address

Physical Schema

```
-- Delete any pre-existing instance of the co-living database
DROP DATABASE IF EXISTS cldb;

--Create the database
CREATE DATABASE cldb ENCODING 'UTF-8';
COMMENT ON DATABASE cldb IS 'Database for managing a co-living business';

--Connect to the database
\connect cldb;

-- Name: co-living; Type: SCHEMA; Schema: -; Owner: postgres
-- this code is meant to be run once, so we drop the schema if it already exists

DROP SCHEMA IF EXISTS co_living CASCADE;

CREATE SCHEMA co_living;
ALTER SCHEMA co_living OWNER TO postgres;

/*************************************************************************
                        DOMAIN DEFINITIONS
*************************************************************************/

-- Name: address; Type: DOMAIN; Schema: co-living; Owner: postgres
-- addesses can be very different according to where they are located,
-- so we can decide if we want to keep this domain or not

CREATE DOMAIN co_living.address AS character varying(256) NOT NULL
    --street name, number, city, zip code, province, country
    -- check again
    CONSTRAINT address_check CHECK ((VALUE)::text ~* '^[^,]*,[^,]*,[^,]*,[^,]*,[^,]*,[^,]*$'
        ::text)
    CONSTRAINT street_check CHECK (split_part((VALUE)::text, ',', 1) ~* '^[A-Za-z0-9 ]{1,50}$
        '::text)
    CONSTRAINT number_check CHECK (split_part((VALUE)::text, ',', 2) ~* '^[0-9]{1,5}$'::text)
    CONSTRAINT city_check CHECK (split_part((VALUE)::text, ',', 3) ~* '^[A-Za-z0-9 ]{1,50}$'
        ::text)
    CONSTRAINT zip_check CHECK (split_part((VALUE)::text, ',', 4) ~* '^[0-9]{5}$'::text)
    CONSTRAINT province_check CHECK (split_part((VALUE)::text, ',', 5) ~* '^[A-Za-z0-9
        ]{1,50}$'::text)
    CONSTRAINT country_check CHECK (split_part((VALUE)::text, ',', 6) ~* '^[A-Za-z0-9 ]{1,50}
        $'::text);

-- Name: email; Type: DOMAIN; Schema: co-living; Owner: postgres
CREATE DOMAIN co_living.email AS character varying(256)
        CONSTRAINT properemail CHECK (((VALUE)::text ~* '^[A-Za-z0-9._%]+@[A-Za-z0-9.]+[.][A-
            Za-z]+$'::text));

-- Name: password; Type: DOMAIN; Schema: co-living; Owner: postgres
CREATE DOMAIN co_living.password AS character varying(256)
        CONSTRAINT properpassword CHECK (((VALUE)::text ~* '[A-Za-z0-9._%!]{8,}'::text));

-- Name: phone_number; Type: DOMAIN; Schema: co-living; Owner: postgres
```

```
CREATE DOMAIN co_living.phone_number AS character varying(15) NOT NULL
        CONSTRAINT proper_phone CHECK (((VALUE)::text ~* '[+][0-9]{2}[0-9]{9}'::text));

-- Name: serialID; Type: DOMAIN; Schema: co-living; Owner: postgres
CREATE DOMAIN co_living.ID AS character varying (10) NOT NULL
        CONSTRAINT proper_serial_id CHECK (((VALUE)::text ~* '[A-Z]{3}[0-9]{7}'::text));

-- Name: personalID; Type: DOMAIN; Schema: co-living; Owner: postgres
-- It must be a string of two words separated by a comma
-- The first word must be one of the following: passport, id card, driving license, health
    card
-- The second word must be the number of the corresponding ID
CREATE DOMAIN co_living.personalID AS character varying(256) NOT NULL
    -- must have a comma in between
    CONSTRAINT comma_check CHECK (((VALUE)::text ~* '^[^,]*,[^,]*$'::text))
    CONSTRAINT id_type_check CHECK (split_part((VALUE)::text, ',', 1) in ('passport', 'id
        card', 'driving license', 'health card'))
    CONSTRAINT id_number_check CHECK (split_part((VALUE)::text, ',', 2) ~* '^[A-Za-z0
        -9]{1,50}$'::text);

-- Name: income_info; Type: DOMAIN; Schema: co-living; Owner: postgres
CREATE DOMAIN co_living.income_info AS character varying(256)
    -- annual net income
    CONSTRAINT annual_income_check CHECK (((VALUE)::text ~* '^[0-9]*$'::text));

-- Name: bank information; Type: DOMAIN; Schema: co-living; Owner: postgres
CREATE DOMAIN co_living.bank_info AS character varying(256)
    -- IBAN
    CONSTRAINT iban_check CHECK (((VALUE)::text ~* '^[A-Z]{2}[0-9]{2}[A-Z0-9]{4}[0-9]{7}([A-
        Z0-9]?){0,16}$'::text));


/*****************************************************************************
                        TYPES DEFINITION
*****************************************************************************/

-- Name: amenities; Type: TYPE; Schema: co-living; Owner: postgres
CREATE TYPE co_living.amenities AS ENUM (
    'dishwasher',
    'washing machine',
    'iron',
    'microwave',
    'oven',
    'dryer',
    'drying rack',
    'wifi',
    'tv',
    'air conditioning',
    'heating',
    'balcony',
    'parking',
    'elevator'
);
```

```
-- Name: facilities; Type: TYPE; Schema: co-living; Owner: postgres
CREATE TYPE co_living.facilities AS ENUM (
    'gym',
    'pool',
    'sauna',
    'jacuzzi',
    'barbecue',
    'garden',
    'terrace',
    'library',
    'cinema',
    'games room',
    'playground',
    'parking',
    'elevator'
);

-- Name: common space type; Type: TYPE; Schema: co-living; Owner: postgres
CREATE TYPE co_living.common_space_type AS ENUM (
    'kitchen',
    'living room',
    'dining room',
    'garden',
    'hallway'
);

-- Name: type of room; Type: TYPE; Schema: co-living; Owner: postgres
CREATE TYPE co_living.offer_type AS ENUM (
    'standard',
    'premium'
);

-- Name: provided service; Type: TYPE; Schema: co-living; Owner: postgres
CREATE TYPE co_living.provided_service AS ENUM (
    'groceries',
    'transport',
    'babysitting',
    'petsitting',
    'car rental',
    'bike rental',
    'airport shuttle',
    'party planning',
    'catering'
);

/*************************************************************************
                            TABLES DEFINITION
*************************************************************************/

-- Name: Manager; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.manager (
    manager_id co_living.personalID NOT NULL,                          -- id of the manager
    manager_name character varying(50) NOT NULL,                -- name of the manager
    manager_surname character varying(50) NOT NULL,             -- surname of the manager
```

```
    manager_phone co_living.phone_number NOT NULL,              -- phone number of the
        manager
    manager_email co_living.email NOT NULL,                     -- email of the manager

    PRIMARY KEY (manager_id)
);

-- Name: applicant; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.applicant (
    applicant_id  co_living.personalID NOT NULL,                        -- personalID of the
        applicant
    applicant_name character varying(50) NOT NULL,              -- name of the applicant
    surname character varying(50) NOT NULL,                     -- surname of the applicant
    email co_living.email NOT NULL,                             -- email of the applicant
    income_info co_living.income_info NOT NULL,                 -- income info of the
        applicant
    accepted boolean NOT NULL,                                  -- if the applicant has been
        accepted

    PRIMARY KEY (applicant_id)
);

-- Name: tenant; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.tenant (
    tenant_id co_living.ID NOT NULL,                    -- id of the tenant
    tenant_name character varying(50) NOT NULL,                 -- name of the tenant
    surname character varying(50) NOT NULL,                     -- surname of the tenant
    username character varying(50) NOT NULL,                    -- username of the tenant
    user_password  co_living.password NOT NULL,                 -- password of the tenant
    profession character varying(50),                           -- profession of the tenant
    email co_living.email NOT NULL,                             -- email of the tenant
    bank_info co_living.bank_info NOT NULL,                     -- bank info of the tenant
    phone_number co_living.phone_number,                        -- phone number of the tenant
    nationality character varying(50),                          -- nationality
    national_ID co_living.personalID,                           -- personalID of the tenant

    PRIMARY KEY (tenant_id)
);

-- Name: guest; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.guest (
    guest_id co_living.personalID NOT NULL,                             -- personalID of the
        guest
    tenant_id co_living.ID NOT NULL,                    -- id of the tenant
    guest_name character varying(50) NOT NULL,
    guest_surname character varying(50) NOT NULL,
    email co_living.email,
    phone_number co_living.phone_number,

    PRIMARY KEY (guest_id),
    FOREIGN KEY (tenant_id) REFERENCES co_living.tenant(tenant_id)
);

-- Name: co-living; Type: TABLE; Schema: co-living; Owner: postgres
```

```
CREATE TABLE co_living.co_living (
    co_living_address co_living.address NOT NULL,              -- address of the co-living
    emergency_contacts co_living.phone_number[],               -- emergency contacts
    amenities co_living.amenities[],                           -- amenities
    facilities co_living.facilities[],                         -- facilities (list)
    manager_id co_living.personalID NOT NULL,                          -- personalID of the
        manager

    PRIMARY KEY (co_living_address),
    FOREIGN KEY (manager_id) REFERENCES co_living.manager(manager_id)
);


-- Name: common_space; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.common_space (
    common_space_name co_living.common_space_type NOT NULL,    -- name of the common space
    co_living_address co_living.address NOT NULL,              -- address of the co-living (
        external id)

    PRIMARY KEY (common_space_name, co_living_address),
    FOREIGN KEY (co_living_address) REFERENCES co_living.co_living(co_living_address)
);

-- Name: event; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.event (
    event_name character varying(256) NOT NULL,                -- name of the event
    event_date date NOT NULL,                                  -- date of the event
    starting_time time with time zone,                         -- time of the event (time
        with time zone)
    duration integer,                                          -- duration of the event (in
        minutes)
    max_tenants integer,                                       -- max number of tenants
    max_guests integer,                                        -- max number of guests
    common_space_name co_living.common_space_type,             -- name of the common space
    co_living_address co_living.address,                       -- address of the co-living (
        external id)

    PRIMARY KEY (event_name),
    FOREIGN KEY (common_space_name, co_living_address) REFERENCES co_living.common_space(
        common_space_name, co_living_address)
);

-- Name: participation as guest ; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.guest_participation (
    event_name character varying(256) NOT NULL,                -- name of the event
    guest_id co_living.personalID NOT NULL,                            -- personalID of the
        guest
    arrival_time time with time zone,                            -- time of arrival of the
        guest

    PRIMARY KEY (event_name, guest_id),
    FOREIGN KEY (event_name) REFERENCES co_living.event(event_name),
    FOREIGN KEY (guest_id) REFERENCES co_living.guest(guest_id)
);
```

```
-- Name: participation ; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.participation (
    event_name character varying(256) NOT NULL,                 -- name of the event
    tenant_id co_living.ID NOT NULL,                    -- id of the tenant
    arrival_time time with time zone,                            -- time of arrival of the
        tenant

    PRIMARY KEY (event_name, tenant_id),
    FOREIGN KEY (event_name) REFERENCES co_living.event(event_name),
    FOREIGN KEY (tenant_id) REFERENCES co_living.tenant(tenant_id)
);


-- Name: partner; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.partner (
    partner_address co_living.address NOT NULL,                 -- address of the partner
    partner_name character varying(50) NOT NULL,                -- name of the partner
    provided_services co_living.provided_service,               -- services provided by the
        partner

    PRIMARY KEY (partner_address, partner_name)
);

-- Name: sponsor; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.sponsor (
    partner_addess co_living.address NOT NULL,                  -- address of the partner (
        external id)
    partner_name character varying(50) NOT NULL,                -- name of the partner
    event_name character varying(50) NOT NULL,                  -- name of the event (
        external id)

    PRIMARY KEY (partner_addess, partner_name, event_name),
    FOREIGN KEY (partner_addess, partner_name) REFERENCES co_living.partner(partner_address,
        partner_name),
    FOREIGN KEY (event_name) REFERENCES co_living.event(event_name)
);

-- Name: partnership; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.partnership (
    partner_address co_living.address NOT NULL,                 -- address of the partner (
        external id)
    partner_name character varying(50) NOT NULL,                -- name of the partner (
        external id)
    co_living_address co_living.address NOT NULL,               -- address of the co-living (
        external id)

    PRIMARY KEY (partner_address, partner_name, co_living_address),
    FOREIGN KEY (partner_address, partner_name) REFERENCES co_living.partner(partner_address,
         partner_name),
    FOREIGN KEY (co_living_address) REFERENCES co_living.co_living(co_living_address)
);
```

```
-- Name: organize; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.organize (
    event_name character varying(256) NOT NULL,                 -- name of the event (
        external id)
    manager_id co_living.personalID NOT NULL,                   -- personalID of the manager
        (external id)
    event_time time with time zone,                             -- time of the event (time
        with time zone)

    PRIMARY KEY (event_name, manager_id),
    FOREIGN KEY (event_name) REFERENCES co_living.event(event_name),
    FOREIGN KEY (manager_id) REFERENCES co_living.manager(manager_id)
);


-- Name: cleaning schedule; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.cleaning_schedule (
    common_space_name co_living.common_space_type NOT NULL,     -- name of the common space
    co_living_address co_living.address NOT NULL,               -- address of the co-living (
        external id)
    week_day character varying(10),                             -- day of the week
    start_time time with time zone,                             -- time of the cleaning (time
         with time zone)
    duration integer,                                           -- duration of the cleaning (
        in minutes)

    PRIMARY KEY (common_space_name, co_living_address),
    FOREIGN KEY (common_space_name, co_living_address) REFERENCES co_living.common_space(
        common_space_name, co_living_address)
);

-- Name: room; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.room (
    co_living_address co_living.address NOT NULL,               -- address of the co-living (
        external id)
    room_number integer NOT NULL,                               -- number of the room
    room_type character varying(50),                            -- type of the room

    PRIMARY KEY (co_living_address, room_number),
    FOREIGN KEY (co_living_address) REFERENCES co_living.co_living(co_living_address)
);

-- Name: standard_queue; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.standard_queue (
    co_living_address co_living.address NOT NULL,               -- address of the co-living (
        external id)
    room_number integer NOT NULL,                               -- number of the room
    applicant_id  co_living.personalID NOT NULL,                -- personalID of the tenant
    entry_time timestamp with time zone NOT NULL,               -- time of the entry in the
        queue

    PRIMARY KEY (co_living_address, room_number, applicant_id),
    FOREIGN KEY (co_living_address, room_number) REFERENCES co_living.room(co_living_address,
         room_number),
```

```
    FOREIGN KEY (applicant_id) REFERENCES co_living.applicant(applicant_id)
);

-- Name: priority-queue; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.priority_queue (
    co_living_address co_living.address NOT NULL,              -- address of the co-living (
        external id)
    room_number integer NOT NULL,                              -- number of the room
    applicant_id  co_living.personalID NOT NULL,               -- personalID of the tenant
    entry_time timestamp with time zone NOT NULL,              -- time of the entry in the
        queue

    PRIMARY KEY (co_living_address, room_number, applicant_id),
    FOREIGN KEY (co_living_address, room_number) REFERENCES co_living.room(co_living_address,
         room_number),
    FOREIGN KEY (applicant_id) REFERENCES co_living.applicant(applicant_id)
);


-- Name: form; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.form (
    applicant_id  co_living.personalID NOT NULL,               -- personalID of the
        applicant
    room_number integer,                                       -- number of the room
    form_number integer NOT NULL,                              -- number of the form

    PRIMARY KEY (form_number),
    FOREIGN KEY (applicant_id) REFERENCES co_living.applicant(applicant_id)
);


-- Name: mandate; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.mandate (
    co_living_address co_living.address NOT NULL,              -- address of the co-living (
        external id)
    tenant_id co_living.ID NOT NULL,                       -- id of the tenant
    start_date date NOT NULL,                                  -- start date of the mandate
    end_date date NOT NULL,                                    -- end date of the mandate

    PRIMARY KEY (co_living_address, tenant_id, start_date, end_date),
    FOREIGN KEY (tenant_id) REFERENCES co_living.tenant(tenant_id),
    FOREIGN KEY (co_living_address) REFERENCES co_living.co_living(co_living_address)
);

-- Name: rate; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.rate (
    co_living_address co_living.address NOT NULL,              -- address of the co-living (
        external id)
    tenant_id co_living.ID NOT NULL,                       -- id of the tenant
    grade integer NOT NULL,                                    -- rate of the co-living
    review character varying(100),                             -- comment of the rate

    PRIMARY KEY (tenant_id, co_living_address),
    FOREIGN KEY (tenant_id) REFERENCES co_living.tenant(tenant_id),
```

```
    FOREIGN KEY (co_living_address) REFERENCES co_living.co_living(co_living_address)
);


-- Name: complaint; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.complaint (
    ticket_number integer NOT NULL,                              -- number of the ticket
    comments character varying(100),                             -- comments of the complaint
    manager_id co_living.personalID NOT NULL,                    -- personalID of the manager

    PRIMARY KEY (ticket_number),
    FOREIGN KEY (manager_id) REFERENCES co_living.manager(manager_id)
);

-- Name: write; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.write (
    tenant_id co_living.ID NOT NULL,                      -- id of the tenant
    ticket_number integer NOT NULL,                              -- number of the ticket

    PRIMARY KEY (tenant_id, ticket_number),
    FOREIGN KEY (tenant_id) REFERENCES co_living.tenant(tenant_id),
    FOREIGN KEY (ticket_number) REFERENCES co_living.complaint(ticket_number)
);

-- Name: contract; Type: TABLE; Schema: co-living; Owner: postgres
CREATE TABLE co_living.contract (
    contract_number integer NOT NULL,                            -- number of the contract
    applicant_id  co_living.personalID NOT NULL,                 -- personalID of the
        applicant
    co_living_address co_living.address NOT NULL,                -- address of the co-living (
        external id)
    room_number integer,                                         -- number of the room
    tenant_id co_living.ID NOT NULL,                      -- id of the tenant
    offer_type co_living.offer_type NOT NULL,                    -- type of the offer
    start_date date NOT NULL,                                    -- start date of the contract
    end_date date NOT NULL,                                      -- end date of the contract
    monthly_rent integer NOT NULL,                               -- monthly rent of the
        contract
    manager_id co_living.personalID NOT NULL,                    -- personalID of the manager

    PRIMARY KEY (contract_number),
    FOREIGN KEY (applicant_id) REFERENCES co_living.applicant(applicant_id),
    FOREIGN KEY (co_living_address, room_number) REFERENCES co_living.room(co_living_address,
         room_number),
    FOREIGN KEY (tenant_id) REFERENCES co_living.tenant(tenant_id),
    FOREIGN KEY (manager_id) REFERENCES co_living.manager(manager_id)
);
```

Populate the Database: Example

```
--MANAGEMENT TEAM MEMBER
```

```
INSERT INTO co_living.manager (manager_id, manager_name, manager_surname, manager_phone,
    manager_email) VALUES
('id card,18672692', 'Jones', 'Benjamin', '+39316098433','jones.benjamin@gmail.com'),
('id card,28391821', 'Kingsley', 'Marina', '+39396081418', 'kingsley.marina@gmail.com'),
('passport,FE83726153', 'Alan', 'Peter', '+39372033508', 'alan.peter@gmail.com');

--COLIVING
INSERT INTO co_living.co_living (co_living_address, emergency_contacts, amenities, facilities
    , manager_id) VALUES
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', '{"+39316098433", "+39396081418"}','{"
    dishwasher", "washing machine","microwave" ,"oven", "dryer", "wifi", "air conditioning",
    "heating", "elevator"}', '{"library", "games room", "sauna", "jacuzzi"}', 'id card
    ,18672692'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', '{"+39372033508"}','{"washing machine", "
    iron", "microwave", "oven", "drying rack", "wifi", "tv", "heating", "parking", "balcony"}
    ', '{"gym", "games room", "terrace", "parking"}', 'passport,FE83726153');

--ROOM
INSERT INTO co_living.room (co_living_address, room_number, room_type) VALUES
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 1,  'premium'),
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 2, 'premium'),
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 3, 'standard'),
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 4, 'standard'),
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 5, 'standard'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 1, 'premium'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 2, 'premium'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 3, 'premium'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 4, 'standard'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 5, 'standard'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 6, 'standard'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 7, 'standard');

--APPLICANT
INSERT INTO co_living.applicant(applicant_id, applicant_name, surname, email, income_info,
    accepted) VALUES
('passport,GA2539654', 'George', 'Adler', 'georgeadler@gmail.com', '77300', true),
('id card,934621753', 'Filippo', 'Filippi', 'filippo_filippi@gmail.com', '53000', false),
('id card,436577811', 'Sara', 'Carter', 'saracarter99@gmail.com', '31234', true),
('driving license,JS336722', 'Julie', 'Smith', 'smithjulie@live.com', '23000', true),
('driving license,MR996633', 'Marco', 'Rossi', 'marcom@live.com', '77600', true),
('id card,550033291', 'Mattew', 'Veen', 'mattveen96@gmail.com', '52160', true),
('passport,SR4459881', 'Silvia', 'Rossi', 'silviarossi@hotmail.it', '80000', true),
('health card,9876543210', 'Sophia', 'Victoria', 'vict_soph@gmail.com', '82190', true),
('passport,HS1012023', 'Harry', 'Smith', 'harrysmith@gmail.com', '58879', true),
('id card,550030317', 'Mark', 'Wolf', 'mwolf@gmail.com', '28440', true) ;

--FORM
INSERT INTO co_living.form(applicant_id, room_number, form_number) VALUES
('passport,HS1012023', 6, 5),
('driving license,MR996633', 4, 1),
('driving license,JS336722', 3, 6),
('id card,550030317', 7, 3),
('health card,9876543210', 2, 4),
('id card,550033291', 1, 9),
```

```
('passport,GA2539654', 1, 10),
('id card,934621753', 2, 8),
('id card,436577811', 6, 7),
('passport,SR4459881', 4, 2);

--STANNDARD QUEUE
INSERT INTO co_living.standard_queue(co_living_address, room_number, applicant_id, entry_time
    ) VALUES
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 6, 'passport,HS1012023', '2020-06-09
    19:15:00 CET'),
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 4, 'driving license,MR996633', '2020-06-10
    7:35:03 CET'),
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 3, 'driving license,JS336722', '2021-03-01
    19:15:00 CET'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 7, 'id card,550030317', '2021-08-24
    23:15:00 CET'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 2, 'health card,9876543210', '2021-11-24
    15:15:00 CET'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 1, 'id card,550033291', '2022-03-12
    20:20:00 CET'),
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 1, 'passport,GA2539654', '2022-09-21
    10:00:00 CET');

--TENANT
INSERT INTO co_living.tenant(tenant_id, tenant_name, surname, username, user_password,
    profession, email, bank_info,
phone_number, nationality, national_ID) VALUES
('BZZ2233440', 'Sara', 'Carter', 'SaraCarter','sar99htlL', 'taxi driver', 'saracarter99@gmail
    .com', 'CS45GG001111111TV3E','+45355897101', 'English', 'id card,436577811'),
('CFF3344551', 'Julie', 'Smith', 'JulieSmith', 'Julie88_E', 'teacher', 'smithjulie@live.com',
     'SJ8651PL2222222EX34I', '+39887321590', 'American', 'driving license,JS336722'),
('DYY4455662', 'Marco', 'Rossi', 'MarcoRossi', 'mr!Forza8', 'nurse', 'marcom@live.com', '
    RM53K0K17777777S3R','+33564778901', 'Italian', 'driving license,MR996633'),
('ERR5566773', 'Mattew', 'Veen', 'MattewVeen', 'Mv!!_3450', 'chef', 'mattveen96@gmail.com', '
    VM881U229685472','+74556879214', 'Dutch','id card,550033291'),
('FTT6677884', 'Silvia', 'Rossi', 'SilviaRossi', 'sil90_edW', 'software developer','
    silviarossi@hotmail.it', 'RS12TUM94444444P','+99764326198', 'Italian', 'passport,
    SR4459881'),
('GPP7788995', 'Sophia', 'Victoria', 'SophiaVict', 'hap_34svR', 'lawyer', 'vict_soph@gmail.
    com', 'VS00MM339999999','+45920348649', 'Russian','health card,9876543210'),
('HWW8899006', 'Harry', 'Smith', 'HarrySmith', 'pop!Dance04', 'dj', 'harrysmith@gmail.com', '
    SH6623RY8888888W2P','+88675599831', 'American', 'passport,HS1012023'),
('ILL9764351', 'Mark', 'Wolf', 'MarkWolf', 'root33!E', 'carpenter', 'mwolf@gmail.com', '
    WM82Q0004444444', '+51890752341', 'English', 'id card,550030317') ;

--PRIORITY QUEUE
INSERT INTO co_living.priority_queue(co_living_address, room_number, applicant_id, entry_time
    ) VALUES
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 6, 'id card,436577811', '2022-12-08
    19:15:00 CET'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 7, 'passport,SR4459881', '2023-04-08
    19:13:08 CET') ;
```

```
--CONTRACT
INSERT INTO co_living.contract (contract_number, applicant_id, co_living_address, room_number
    , tenant_id, offer_type, start_date, end_date, monthly_rent, manager_id) VALUES
(1, 'passport,SR4459881', 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 4, 'FTT6677884',
     'standard', '2020-02-01','2022-04-01','520','passport,FE83726153'),
(2, 'passport,HS1012023', 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 6, 'HWW8899006',
     'standard', '2020-06-30','2023-06-30','510','passport,FE83726153'),
(3, 'driving license,MR996633', 'Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 4, '
    DYY4455662', 'standard','2020-09-08','2023-01-01','530','id card,18672692'),
(4, 'driving license,JS336722', 'Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 3, '
    CFF3344551', 'standard','2021-03-17','2023-03-17','510','id card,28391821'),
(5, 'id card,550030317', 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 7, 'ILL9764351',
    'standard', '2021-10-01', '2023-02-25','530', 'passport,FE83726153'),
(6, 'id card,436577811', 'Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 2, 'BZZ2233440', '
    premium', '2022-01-01', '2022-10-31', '610', 'id card,18672692'),
(7, 'health card,9876543210', 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 2, '
    GPP7788995','premium','2022-01-01', '2024-01-01','630','passport,FE83726153'),
(8, 'id card,550033291', 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 1, 'ERR5566773',
    'premium','2022-04-10','2023-04-10','620','passport,FE83726153');

--MANDATE
INSERT INTO co_living.mandate(co_living_address, tenant_id, start_date, end_date) VALUES
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 'DYY4455662', '2020-09-09', '2022-04-23'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 'ILL9764351', '2022-08-25', '2022-12-09')
    ,
('Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 'BZZ2233440', '2022-04-24', '2022-10-31') ;

--COMMON SPACES
INSERT INTO co_living.common_space (common_space_name, co_living_address) VALUES
('kitchen','Via Luigi Cadorna,9,Padova,35123,Padova,Italy'),
('kitchen','Via Gaetano Trezza,18,Verona,37129,Verona,Italy'),
('dining room','Via Luigi Cadorna,9,Padova,35123,Padova,Italy'),
('dining room','Via Gaetano Trezza,18,Verona,37129,Verona,Italy'),
('hallway','Via Luigi Cadorna,9,Padova,35123,Padova,Italy'),
('hallway','Via Gaetano Trezza,18,Verona,37129,Verona,Italy'),
('living room','Via Gaetano Trezza,18,Verona,37129,Verona,Italy');

--CLEANING SCHEDULE
INSERT INTO co_living.cleaning_schedule (common_space_name, co_living_address, week_day,
    start_time, duration) VALUES
('kitchen','Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 'monday', '09:00:00 CET', '90'),
('kitchen','Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 'wednesday', '14:00:00 CET', '
    120'),
('dining room','Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 'monday', '10:30:00 CET','30'
    ),
('dining room','Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 'tuesday', '8:00:00 CET', '
    45'),
('hallway','Via Luigi Cadorna,9,Padova,35123,Padova,Italy', 'thursday', '16:00:00 CET', '30')
    ,
('hallway','Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 'tuesday', '8:45:00 CET', '30')
    ,
('living room','Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 'friday', '9:00:00 CET','60
    ');
```

```
--EVENT
INSERT INTO co_living.event(event_name, event_date, starting_time, duration, max_tenants,
    max_guests, common_space_name, co_living_address) VALUES
('Cooking class', '2022-10-12', '10:30:00', 120, 8, 2, 'kitchen', 'Via Gaetano Trezza,18,
    Verona,37129,Verona,Italy'),
('NYE Party', '2022-12-31', '20:00:00', 270, 30, 10, 'dining room', 'Via Luigi Cadorna,9,
    Padova,35123,Padova,Italy'),
('Karaoke night', '2023-01-03', '21:15:00', 180, 15, 8, 'living room', 'Via Gaetano Trezza
    ,18,Verona,37129,Verona,Italy') ;

--GUEST
INSERT INTO co_living.guest(guest_id, tenant_id, guest_name, guest_surname, email,
    phone_number) VALUES
('passport,AD89xd32', 'DYY4455662', 'Bianca', 'Rizzi', 'bianca_rizze@hotmail.it', '
    +99454567312'),
('id card,Bb923ss', 'HWW8899006', 'Mike', 'Miles', 'mikemiles@gmail.com', '+65457384721') ;

--PARTICIPATION
INSERT INTO co_living.participation(event_name, tenant_id, arrival_time) VALUES
('Cooking class', 'GPP7788995', '09:00:00 CET'),
('Cooking class', 'ILL9764351', '09:00:00 CET'),
('Cooking class','HWW8899006', '09:00:00 CET'),
('NYE Party', 'CFF3344551', '22:30:00 CET'),
('NYE Party', 'DYY4455662', '22:30:00 CET'),
('Karaoke night','HWW8899006', '20:30:00 CET'),
('Karaoke night', 'GPP7788995', '20:30:00 CET'),
('Karaoke night','ERR5566773', '20:30:00 CET'),
('Karaoke night', 'ILL9764351', '20:30:00 CET');

--GUEST PARTICIPATION
INSERT INTO co_living.guest_participation(event_name, guest_id, arrival_time) VALUES
('Cooking class', 'id card,Bb923ss', '09:00:00 CET'),
('NYE Party', 'passport,AD89xd32', '22:30:00 CET');

--RATE
INSERT INTO co_living.rate(co_living_address, tenant_id, grade, review) VALUES
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 'GPP7788995', 9, 'wonderful experience'),
('Via Gaetano Trezza,18,Verona,37129,Verona,Italy', 'ILL9764351', 8, 'it is good') ;

--PARTNER
INSERT INTO co_living.partner(partner_address, partner_name, provided_services) VALUES
('Via Enrico Toti,22,Padova,35135,Padova,Italy','Event lab', 'party planning'),
('Via Marco,18,Padova,35129,Padova,Italy','Galileo Ristorazione','catering'),
('Via Silvio Pellico,2,Verona,37123,Verona,Italy','Autovia','transport'),
('Via Giovanni della Casa,7,Verona,37122,Verona,Italy','AI Moschetto','groceries');

--PARTNERSHIP
INSERT INTO co_living.partnership (partner_address, partner_name, co_living_address) VALUES
('Via Giovanni della Casa,7,Verona,37122,Verona,Italy','AI Moschetto', 'Via Gaetano Trezza
    ,18,Verona,37129,Verona,Italy'),
('Via Silvio Pellico,2,Verona,37123,Verona,Italy','Autovia','Via Luigi Cadorna,9,Padova
    ,35123,Padova,Italy');

--SPONSOR
```

```
INSERT INTO co_living.sponsor (partner_addess, partner_name, event_name) VALUES
('Via Enrico Toti,22,Padova,35135,Padova,Italy', 'Event lab', 'NYE Party'),
('Via Marco,18,Padova,35129,Padova,Italy','Galileo Ristorazione', 'NYE Party'),
('Via Giovanni della Casa,7,Verona,37122,Verona,Italy','AI Moschetto','Karaoke night');

--COMPLAINTS
INSERT INTO co_living.complaint(ticket_number, comments, manager_id) VALUES
(1, 'the sink is broken in the kitchen, we need a plumber a.s.a.p', 'id card,18672692'),
(2, 'the heat in my room only work 30minutes a day and I dont know why...', 'id card,18672692
    ');

--WRITE
INSERT INTO co_living.write(tenant_id, ticket_number) VALUES
('DYY4455662',1),
('CFF3344551',2);

--ORGANIZE
INSERT INTO co_living.organize(event_name, manager_id, event_time) VALUES
('Cooking class','passport,FE83726153', '09:00:00 CET'),
('NYE Party','id card,28391821', '22:30:00 CET'),
('Karaoke night','passport,FE83726153', '20:30:00 CET');
```

Principal Queries

- Retrieve a list of the tenants staying in the co-living in 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy', together with their name, surname, national ID;

```
SELECT t.tenant_name, t.surname, t.national_id
FROM co_living.tenant as t
INNER JOIN co_living.contract as c ON t.tenant_id = c.tenant_ID
WHERE c.co_living_address = 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy';
```

The output is shown below.

```
cldb=# SELECT t.tenant_name, t.surname, t.national_id
cldb-# FROM co_living.tenant as t
cldb-# INNER JOIN co_living.contract as c ON t.tenant_id = c.tenant_ID
cldb-# WHERE c.co_living_address = 'Via Gaetano Trezza,18,Verona,37129,Verona,Italy';
 tenant_name | surname  |      national_id
-------------+----------+------------------------
 Mattew      | Veen     | id card,550033291
 Silvia      | Rossi    | passport,SR4459881
 Sophia      | Victoria | health card,9876543210
 Harry       | Smith    | passport,HS1012023
 Mark        | Wolf     | id card,550030317
(5 rows)
```

- Retrieve the list of the applicants with their related information that are accepted for the next renting of the co-living 'Via Luigi Cadorna,9,Padova,35123,Padova,Italy';

```
SELECT A.applicant_id, A.applicant_name, A.surname, A.email, A.income_info
FROM co_living.applicant AS A
INNER JOIN co_living.standard_queue AS S
ON A.applicant_id=S.applicant_id
WHERE A.accepted=true
AND S.co_living_address='Via Luigi Cadorna,9,Padova,35123,Padova,Italy';
```

The output is shown below.

```
cldb=# SELECT A.applicant_id, A.applicant_name, A.surname, A.email, A.income_info
cldb-# FROM co_living.applicant AS A
cldb-# INNER JOIN co_living.standard_queue AS S
cldb-# ON A.applicant_id=S.applicant_id
cldb-# WHERE A.accepted=true
cldb-# AND S.co_living_address='Via Luigi Cadorna,9,Padova,35123,Padova,Italy';
       applicant_id       | applicant_name | surname |         email         | income_info
--------------------------+----------------+---------+-----------------------+-------------
 passport,GA2539654       | George         | Adler   | georgeadler@gmail.com | 77300
 driving license,JS336722 | Julie          | Smith   | smithjulie@live.com   | 23000
 driving license,MR996633 | Marco          | Rossi   | marcom@live.com       | 77600
(3 rows)
```

- List the managers of the tenants who have rated a co-living higher than 8, alongside with the grade and the review;

```
Select manager_name, manager_surname, grade, review
From co_living.Manager AS M
Inner Join (
        (Select * From co_living.rate Where grade>8) AS R
        Inner Join co_living.Contract AS C ON R.tenant_id=C.tenant_id)
As L On M.manager_id=L.manager_id;
```

The output is shown below.

```
cldb=# Select manager_name, manager_surname, grade, review
cldb-# From co_living.Manager AS M
cldb-# Inner Join (
cldb(# (Select * From co_living.rate Where grade>8) AS R
cldb(# Inner Join co_living.Contract AS C ON R.tenant_id=C.tenant_id)
cldb-# As L On M.manager_id=L.manager_id;
 manager_name | manager_surname | grade |        review
--------------+-----------------+-------+----------------------
 Alan         | Peter           |     9 | wonderful experience
(1 row)
```

- Select which tenant invited a guest in the 'Cooking class' event.

```
SELECT Q.tenant_name
FROM
(SELECT * FROM co_living.tenant AS T
```

```
 INNER JOIN (Select * FROM co_living.guest AS G
  INNER JOIN (select * from co_living.guest_participation  where event_name = 'Cooking
      class' ) AS P
  ON G.guest_id = P.guest_id) AS S
ON T.tenant_id = S.tenant_id) AS Q;
```

The output is shown below.

```
cldb=# SELECT Q.tenant_name
cldb-# FROM
cldb-# (SELECT * FROM co_living.tenant AS T
cldb(#  INNER JOIN (Select * FROM co_living.guest AS G
cldb(#   INNER JOIN (select * from co_living.guest_participation  where event_name = 'Cooking class' ) AS P
cldb(#   ON G.guest_id = P.guest_id) AS S
cldb(# ON T.tenant_id = S.tenant_id) AS Q;
 tenant_name
-------------
 Harry
(1 row)
```

JDBC Implementations of the Principal Queries and Visualization

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.PreparedStatement;

/**
 * Prints all the co-livings tenants
 */

public class coLivingTenants {
    /**
     * The JDBC driver to be used
     */
    private static final String DRIVER = "org.postgresql.Driver";

    /**
     * The URL of the database to be accessed
     */
    private static final String DATABASE = "jdbc:postgresql://localhost/cldb";

    /**
     * The username for accessing the database
     */
    private static final String USER = "postgres";  // insert username here

    /**
     * The password for accessing the database
```

```
     */
    private static final String PASSWORD = "postgres";  // insert password here

    /**
     * List all the co-livings tenants
     * <p>
     * command-line arguments (not used).
     */

    public static void main(String[] args) {

        // the connection to the DBMS
        Connection con = null;

        // the statements to be executed
        PreparedStatement ps0 = null;
        Statement stmt1 = null;
        Statement stmt2 = null;

        // the results of the statement execution
        ResultSet rs0 = null;
        ResultSet rs1 = null;
        ResultSet rs2 = null;

        // start time of a statement
        long start;

        // end time of a statement
        long end;

        try {
            // register the JDBC driver
            Class.forName(DRIVER);

            System.out.printf("Driver %s successfully registered.%n", DRIVER);

        } catch (ClassNotFoundException e) {
            System.out.printf(
                    "Driver %s not found: %s.%n", DRIVER, e.getMessage());
            // terminate with a generic error code
            System.exit(-1);
        }

        try {

            // connect to the database
            start = System.currentTimeMillis();
            con = DriverManager.getConnection(DATABASE, USER, PASSWORD);
            end = System.currentTimeMillis();
            System.out.printf(
                    "Connection to database %s successfully established in %,d milliseconds.%
                        n",
                    DATABASE, end - start);
```

```
// first query using PREPAREDSTATEMENT
String sql0 = "SELECT Q.tenant_name " +
        "FROM " +
        "(SELECT * FROM co_living.tenant AS T" +
        " INNER JOIN (Select * FROM co_living.guest AS G" +
        " INNER JOIN (select * from co_living.guest_participation  where
            event_name = ? ) AS P" +
        " ON G.guest_id = P.guest_id) AS S" +
        " ON T.tenant_id = S.tenant_id) AS Q;";

start = System.currentTimeMillis();
ps0 = con.prepareStatement(sql0);
ps0.setString(1, "Cooking class");
end = System.currentTimeMillis();

System.out.printf("Statement successfully created in %,d milliseconds.%n",end -
    start);

start = System.currentTimeMillis();
rs0 = ps0.executeQuery();
end = System.currentTimeMillis();

System.out.printf("Query 1: successfully executed %,d milliseconds.%n", end -
    start);
System.out.printf("Query 1 results:%n");

String tenant_name;

while (rs0.next()) {
    tenant_name = rs0.getString("tenant_name");

    System.out.printf("%nTenant name: %s%n", tenant_name);
}
rs0.close();
ps0.close();

// execute the query
String sql = "Select * FROM co_living.co_living;";
start = System.currentTimeMillis();
stmt1 = con.createStatement();
end = System.currentTimeMillis();

// create the statement to execute the query
System.out.printf(
        "%nStatement successfully created in %,d milliseconds.%n",
        end - start);

start = System.currentTimeMillis();
rs1 = stmt1.executeQuery(sql);
end = System.currentTimeMillis();

String address;

// cycle on the query results and print them
```

```
            while (rs1.next()) {
                address = rs1.getString("co_living_address");

                System.out.printf("%nCo-living - %s%n", address);
                System.out.printf("%nNAME\t\tSURNAME\t\tNATIONAL ID%n");

                sql = "SELECT t.tenant_name as Name, t.surname as Surname, t.national_id as
                    ID " +
                        "FROM co_living.tenant as t " +
                        "INNER JOIN co_living.contract as c ON t.tenant_id = c.tenant_ID " +
                        "WHERE c.co_living_address ='" + address + "';";

                stmt2 = con.createStatement();
                rs2 = stmt2.executeQuery(sql);
                String name;
                String surname;
                String id;

                while (rs2.next()) {
                    name = rs2.getString("Name");
                    surname = rs2.getString("Surname");
                    id = rs2.getString("ID");
                    System.out.printf("%-8s\s\s\s\s\s\s\s\s%-8s\s\s\s\s\s\s\s\s\s%s%n", name,
                        surname, id);
                }
                rs2.close();
                stmt2.close();
            }
            rs1.close();
            stmt1.close();
            con.close();
            end = System.currentTimeMillis();
            System.out.printf("%nData correctly exctracted and visualized in %d milliseconds
                %n", end - start);

        } catch (SQLException e) {
            System.out.printf("Database access error :%n");
            // cycle in the exception chain
            while (e != null) {
                //e.printStackTrace();
                System.out.printf("- Message : %s%n", e.getMessage());
                System.out.printf("- SQL status code : %s%n", e.getSQLState());
                System.out.printf("- SQL error code : %s%n", e.getErrorCode());
                System.out.printf("%n");
                e = e.getNextException();
            }

        } finally {
            try {
                // close the used resources

                if (!rs0.isClosed() || !rs1.isClosed() || !rs2.isClosed()){
                    start = System.currentTimeMillis();
                    rs0.close();
```

```
            rs1.close();
            rs2.close();
            end = System.currentTimeMillis();
            System.out
                    .printf("Result set successfully closed in finally block +
                         in  %, d milliseconds. % n",
            end - start);
        }

        if (!ps0.isClosed()) {
            start = System.currentTimeMillis();
            ps0.close();
            end = System.currentTimeMillis();
            System.out
                    .printf("PREPAREDSTATEMENT successfully closed in finally
                        block +  in  %, d milliseconds. % n",
                            end - start);
        }

        if (!stmt1.isClosed() || !stmt2.isClosed()) {
            start = System.currentTimeMillis();
            stmt1.close();
            stmt2.close();
            end = System.currentTimeMillis();
            System.out
                    .printf("Statement successfully closed in finally block +  in
                         %, d milliseconds. % n",
            end - start);
        }

        if (!con.isClosed()) {
            start = System.currentTimeMillis();
            con.close();
            end = System.currentTimeMillis();
            System.out
                    .printf("Connection successfully closed in finally block +
                         in  %, d milliseconds. % n",
                            end - start);
        }
    } catch (SQLException e) {

        System.out.printf("Error while releasing resources in finally block:%n");
        // cycle in the exception chain
        while (e != null) {
        System.out.printf(" - Message: %s % n",e.getMessage());
        System.out.printf(" - SQL status code: %s % n",e.getSQLState());
        System.out.printf(" - SQL error code: %s % n",e.getErrorCode());
        System.out.printf(" % n");
        e = e.getNextException();
        }
    } finally {

        rs0 = null;
        rs1 = null;
```

```
                rs2 = null;
                ps0 = null;
                stmt1 = null;
                stmt2 = null;
                con = null;
                System.out.printf("Resources released to the garbage collector.%n");
            }

        }
        System.out.printf("Program end.%n");
    }
}
```

The image below shows the output of the queries:

```
Driver org.postgresql.Driver successfully registered.
Connection to database jdbc:postgresql://localhost/cldb successfully established in 362 milliseconds.
Statement successfully created in 9 milliseconds.
Query 1: successfully executed 13 milliseconds.
Query 1 results:

Tenant name: Harry

Statement successfully created in 1 milliseconds.

Co-living - Via Luigi Cadorna,9,Padova,35123,Padova,Italy

NAME            SURNAME         NATIONAL ID
Sara            Carter          id card,436577811
Julie           Smith           driving license,JS336722
Marco           Rossi           driving license,MR996633

Co-living - Via Gaetano Trezza,18,Verona,37129,Verona,Italy

NAME            SURNAME         NATIONAL ID
Mattew          Veen            id card,550033291
Silvia          Rossi           passport,SR4459881
Sophia          Victoria        health card,9876543210
Harry           Smith           passport,HS1012023
Mark            Wolf            id card,550030317

Data correctly exctracted and visualized in 12 milliseconds
Resources released to the garbage collector.
Program end.
```

Group Members Contribution

All members of the group contributed to the design and update of the ER schema. Each person took care of a specific section, in particular:

- Variations to the relational schema: we did some minimal modifications according to the suggestions of the professor, discussed all together. Alberto took care of modifying the schema itself.
- Physical schema: Nicole Zattarin, Nicolas Ortiz e Zarate

- Populate the Database: Daria Zanin, Anaïs Battut
- Principal Queries: Euxhenio Sulku, Riccardo Broetto, Anaïs Fragne, Marco Gasparini
- JDBC: Alberto Gobbin, Luca D'Este

Of course, since each part of the homework requires all the previous ones to be done properly, the debugging and refinements have been done with the contribution of everybody during the development cycle.