Database I
IoT Solutions Inc.

-> Link to GitHub Database Project <-

Matrikelnr.:
Dozent: apl. Prof. Dr. Nuo Li
Kurs: Database
Application: DB Design

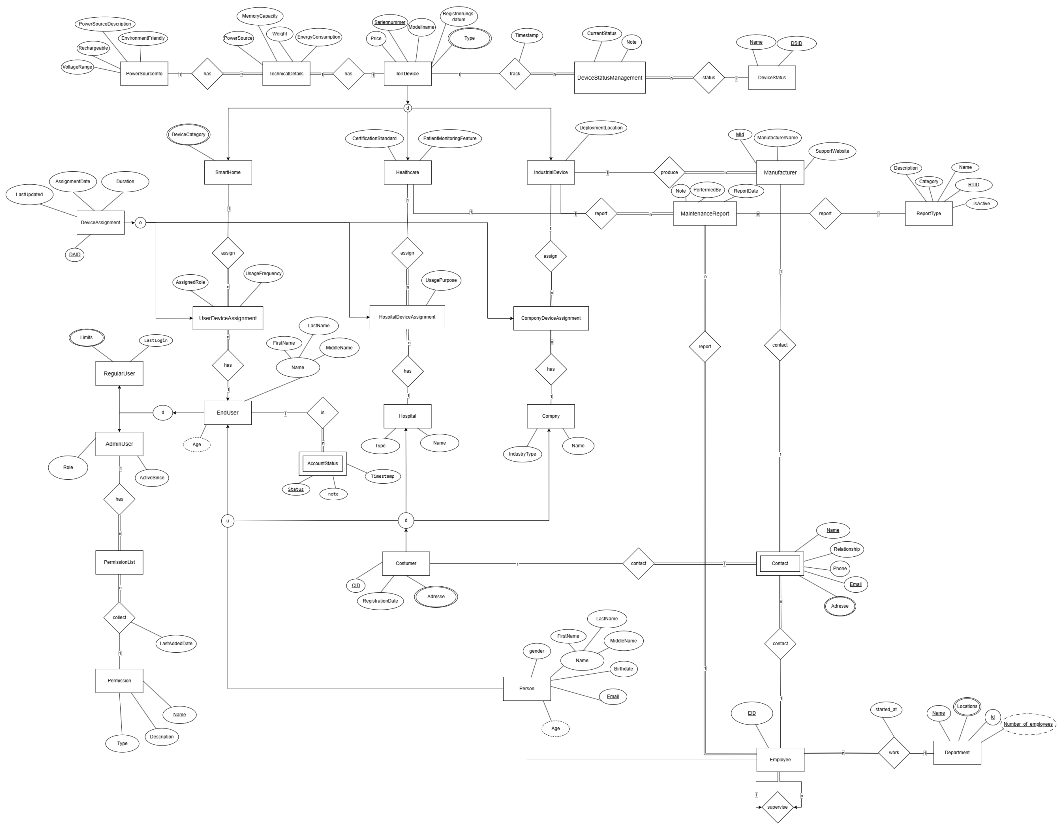
26. April 2025

1 Intruduction

We as **IoT Solutions Inc.** are a company specializing in the management of IoT (Internet of Things) devices across various sectors. We provide innovative solutions for industries such as healthcare, industrial operations, and smart home automation, whereas smart home devices are provided to endusers or small businesses. Our Productline consists of a wide range of IoT devices, including security cameras, health monitors and also industrial sensors.

1.1 ER/EER Model

The ER/EER Model is designed to provide a comprehensive overview of the relationships between different entities in our system. The model includes the following entities:



Link to EER Image

IoTDevice : This entity is the foundation of the system, representing the physical IoT devices being managed. Each device needs unique identifiers, such as serial numbers and model names, as well as attributes to track its registration and operational state.

- **IndustrialDevice (Inherited from IoTDevice):** This entity represents IoT devices used in industrial settings.
- **SmartHomeDevice (Inherited from IoTDevice):** This entity is Specialization for IoT devices used in smart home environments.
- **HealthcareIoTDevice (Inherited from IoTDevice):** This entity is specialized for healthcare-related IoT devices.

Customer: Represents the companies or individuals who purchase or use IoT devices managed by the system. Customers are often linked to specific device assignments.

- **Enduser (Inherited from Customer):** customers who are private individuals using IoT devices for personal purposes, such as smart home devices.
  - **RegularUser** Represents typical users with standard permissions and limited access to device configurations.
  - **AdminUser:** Represents users with elevated privileges for managing the their own devices, or other users.
- **Hospital (Inherited from Customer):** Represents organizations or Hospitals purchasing or managing IoT devices.
- **Compny (Inherited from Customer):** Represents organizations or companies purchasing or managing IoT devices for business purposes.

DeviceAssignment: This Entity tracks the assignment of devices to entities like users, hospitals, or companies.

- **HospitalDeviceAssignment:** Represents devices assigned to hospitals for healthcare-specific use cases.
- **CompanyDeviceAssignment:** Represents devices assigned to companies for business or industrial use.
- **UserDeviceAssignment:** Represents devices assigned to individual users (e.g., for personal or smart home use).

Manufacturer: Represents the companies that produce part of IoT devices, specifically industrial devices. The **IoT Solutions Inc.** then assemble it together and sell it as its own product.

2 Normalization

1NF: The data is organized into tables, and each table has a primary key. Each attribute in the table contains atomic values, and there are no repeating groups or arrays. As an example, **Person** table has a composed value for the attribute **name** which consists of **first name**, **last name** and **middle name**. This is not atomic and should be separated into three different attributes.

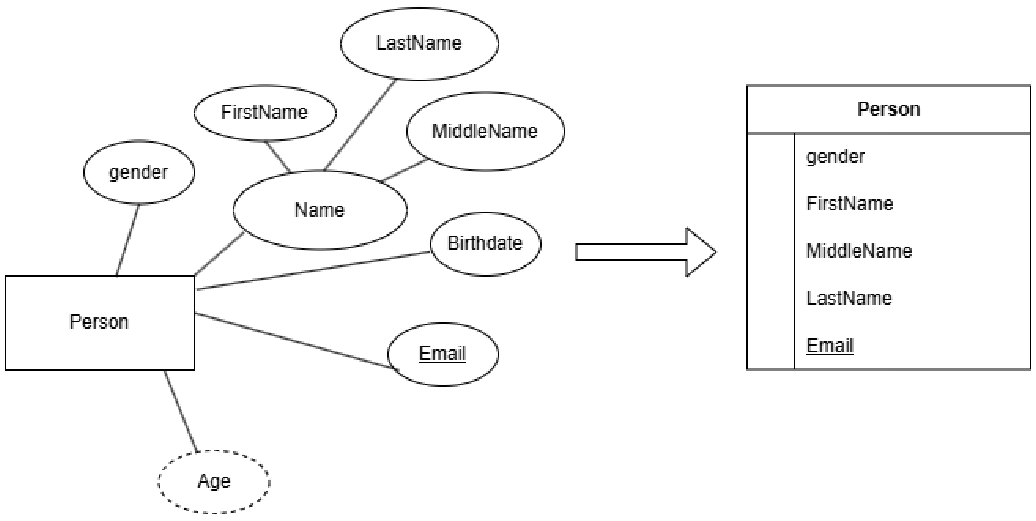


Abbildung 1: 1NF Normalization based on the Person table.

2NF: All non-key attributes are fully functionally dependent on the primary key. as an example the **TechnicalDetails** table has two primary keys, **TDID** and **PowerSource**. All non-key attributes are not fully functionally dependent on the primary key, as for example powerSourceDescription, environmentfriendly or rechargeable attributes depends on the attribute PowerSource. This means that for selecting the powerSourceDescription, environmentfriendly or rechargeable attributes, we need to combine the two primary keys. This is not 2NF, so we could decompose the table into two tables, one for the power source and one for the technical details. The power source table will have the primary key PowerSource and the technical details table will have the primary key TDID. The power source table will have the attributes PowerSourceDescription, environmentfriendly and rechargeable. The technical details table will have all remaining attributes. This way, we can select the power source description, environment friendly or rechargeable attributes without having to combine the two primary keys.

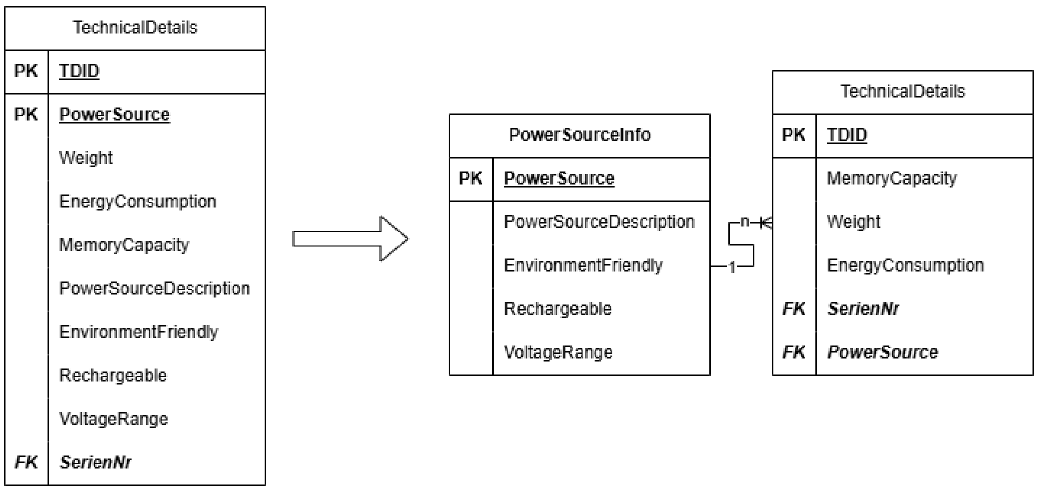


Abbildung 2: 2NF Normalization based on the TechnicalDetails table.

3NF: All transitive dependencies are removed. This means that all non-key attributes are not dependent on other non-key attributes. As an example, the **Adress** table has a transitive dependency on the **city**, **postcode** and **country** attributes. This means that city is dependent on the postcode attribute and the postcode is dependent on the country attribute. This is not 3NF, so we could decompose the table into three tables, one for the postcodeInfo, one for the Address.

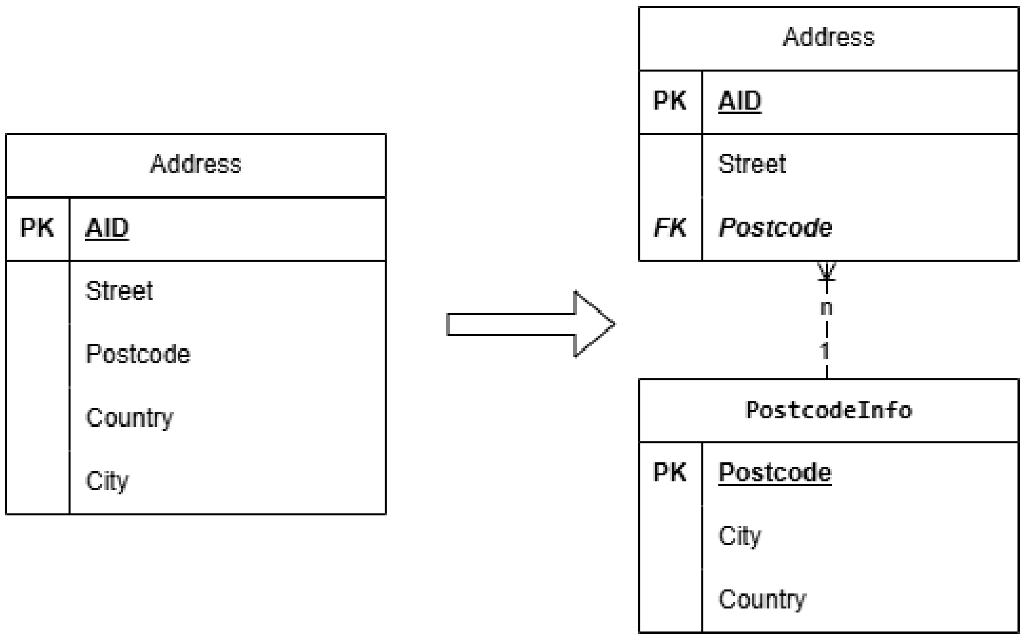
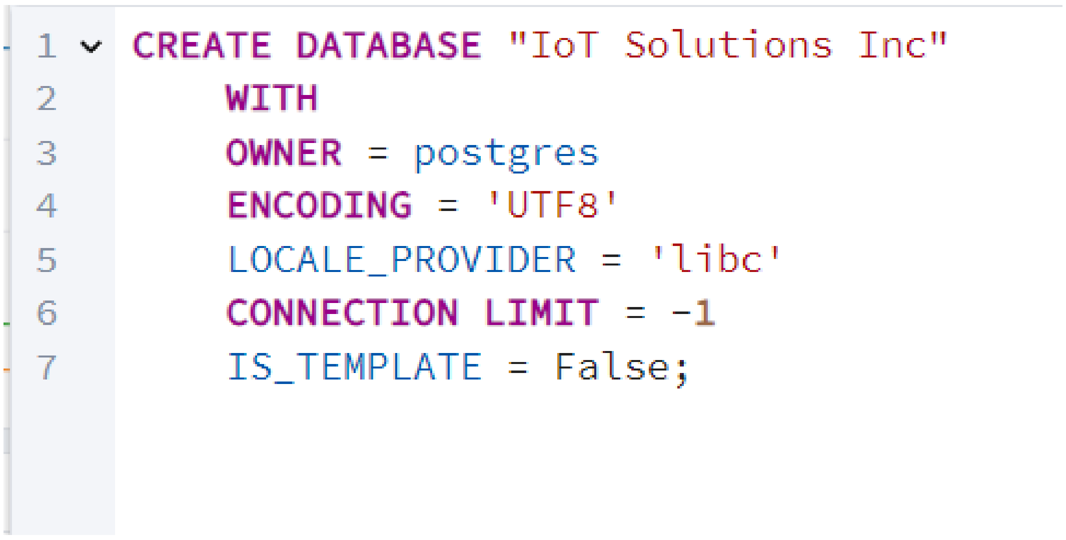


Abbildung 3: 3NF Normalization based on the Address table.

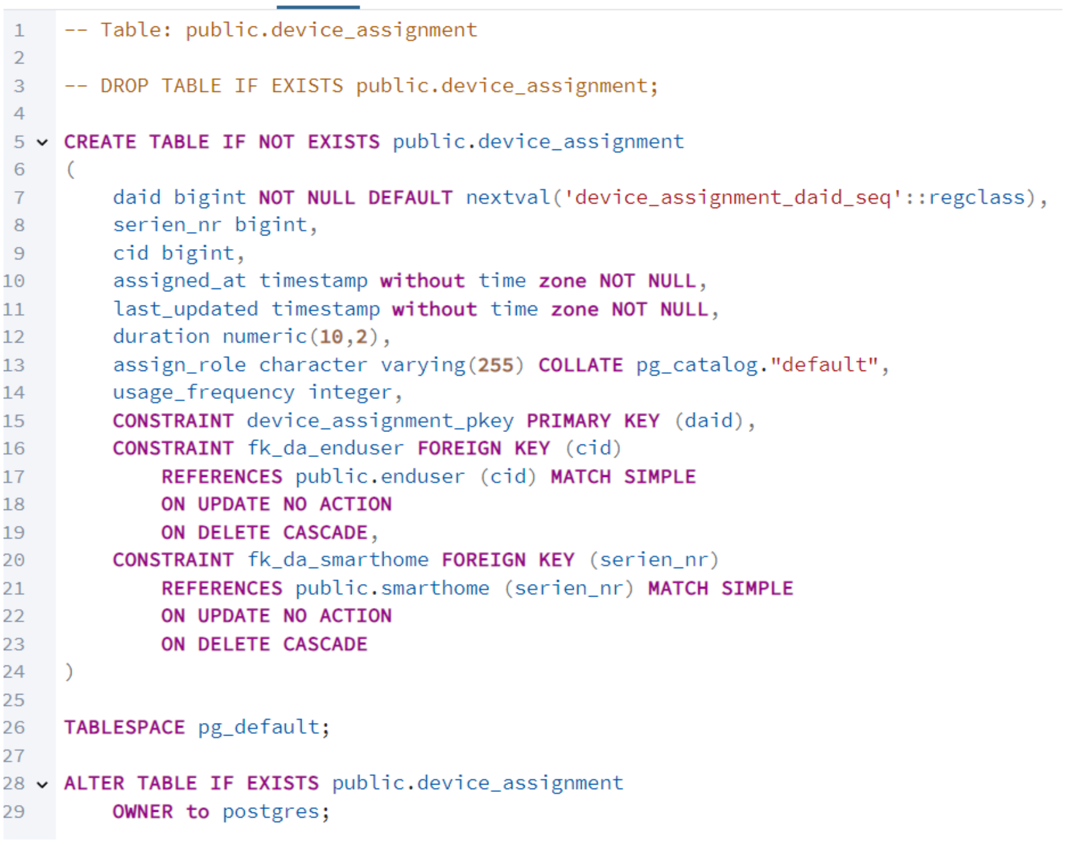
3 Implementation screenshots

3.1 Creating tables



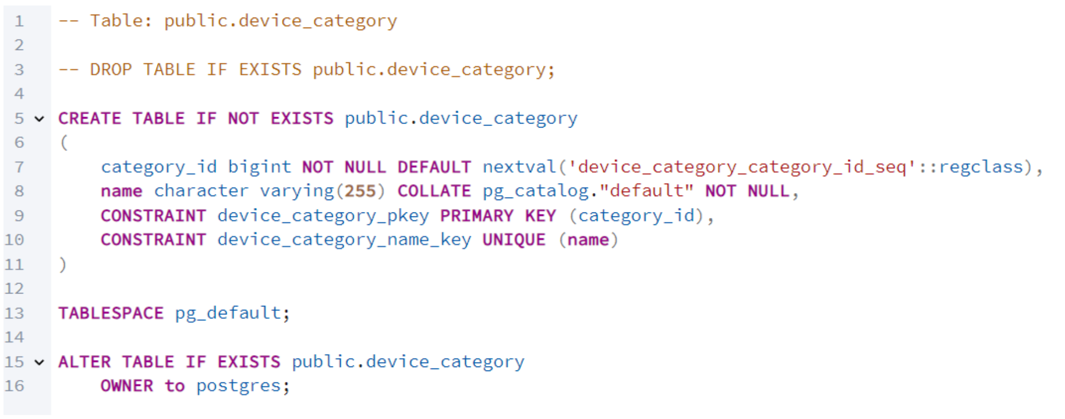
```sql
CREATE DATABASE "IoT Solutions Inc"
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LOCALE_PROVIDER = 'libc'
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;
```

Abbildung 4: Iot Solution Inc Databse



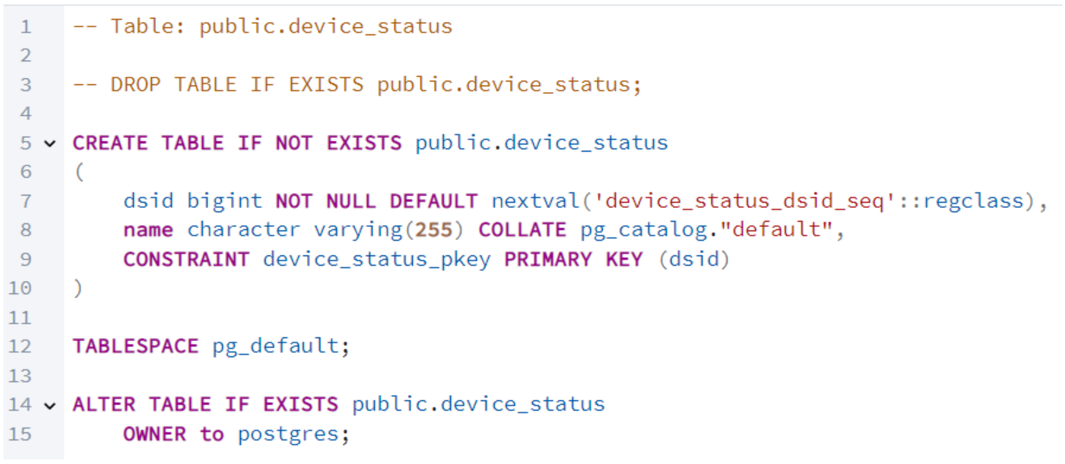
```sql
-- Table: public.device_assignment

-- DROP TABLE IF EXISTS public.device_assignment;

CREATE TABLE IF NOT EXISTS public.device_assignment
(
    daid bigint NOT NULL DEFAULT nextval('device_assignment_daid_seq'::regclass),
    serien_nr bigint,
    cid bigint,
    assigned_at timestamp without time zone NOT NULL,
    last_updated timestamp without time zone NOT NULL,
    duration numeric(10,2),
    assign_role character varying(255) COLLATE pg_catalog."default",
    usage_frequency integer,
    CONSTRAINT device_assignment_pkey PRIMARY KEY (daid),
    CONSTRAINT fk_da_enduser FOREIGN KEY (cid)
        REFERENCES public.enduser (cid) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT fk_da_smarthome FOREIGN KEY (serien_nr)
        REFERENCES public.smarthome (serien_nr) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.device_assignment
    OWNER to postgres;
```

Abbildung 5: device assignment table



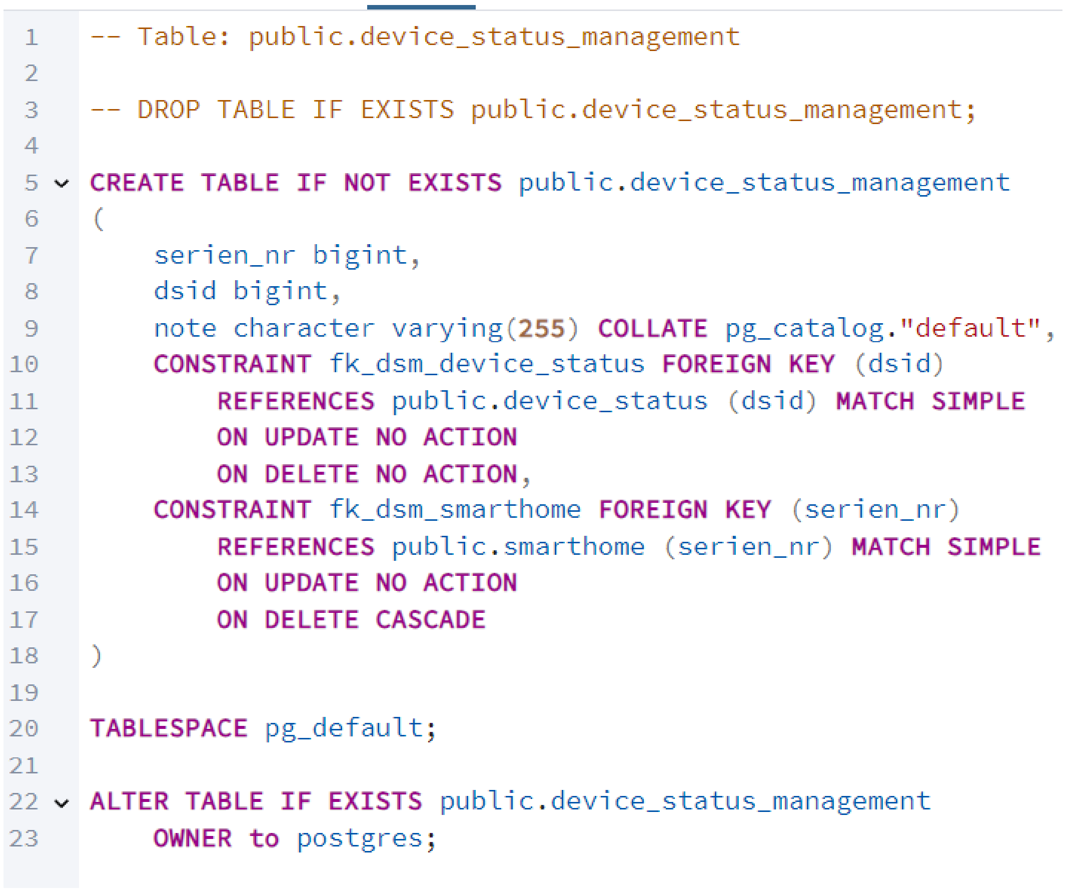
```sql
-- Table: public.device_category

-- DROP TABLE IF EXISTS public.device_category;

CREATE TABLE IF NOT EXISTS public.device_category
(
    category_id bigint NOT NULL DEFAULT nextval('device_category_category_id_seq'::regclass),
    name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT device_category_pkey PRIMARY KEY (category_id),
    CONSTRAINT device_category_name_key UNIQUE (name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.device_category
    OWNER to postgres;
```

Abbildung 6: device category table



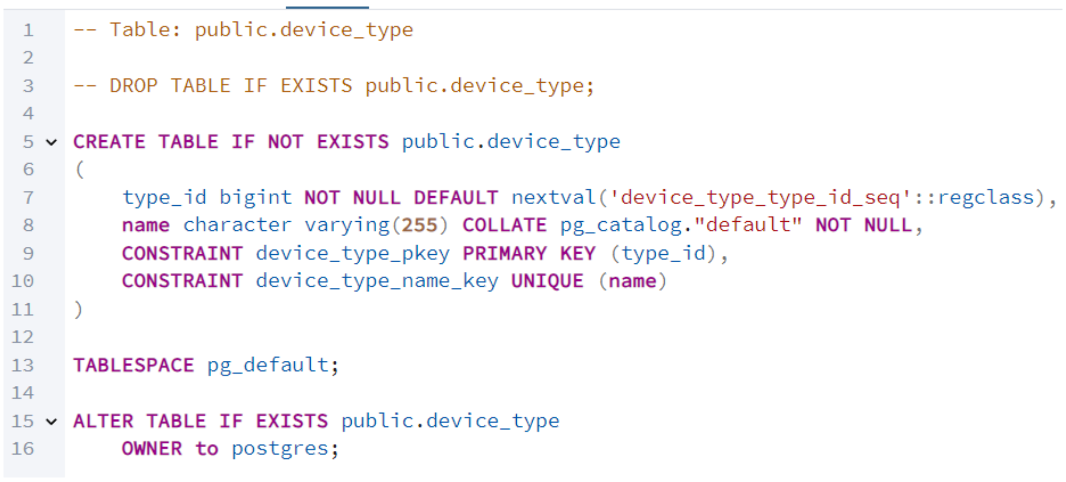
```sql
-- Table: public.device_status

-- DROP TABLE IF EXISTS public.device_status;

CREATE TABLE IF NOT EXISTS public.device_status
(
    dsid bigint NOT NULL DEFAULT nextval('device_status_dsid_seq'::regclass),
    name character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT device_status_pkey PRIMARY KEY (dsid)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.device_status
    OWNER to postgres;
```

Abbildung 7: device status table



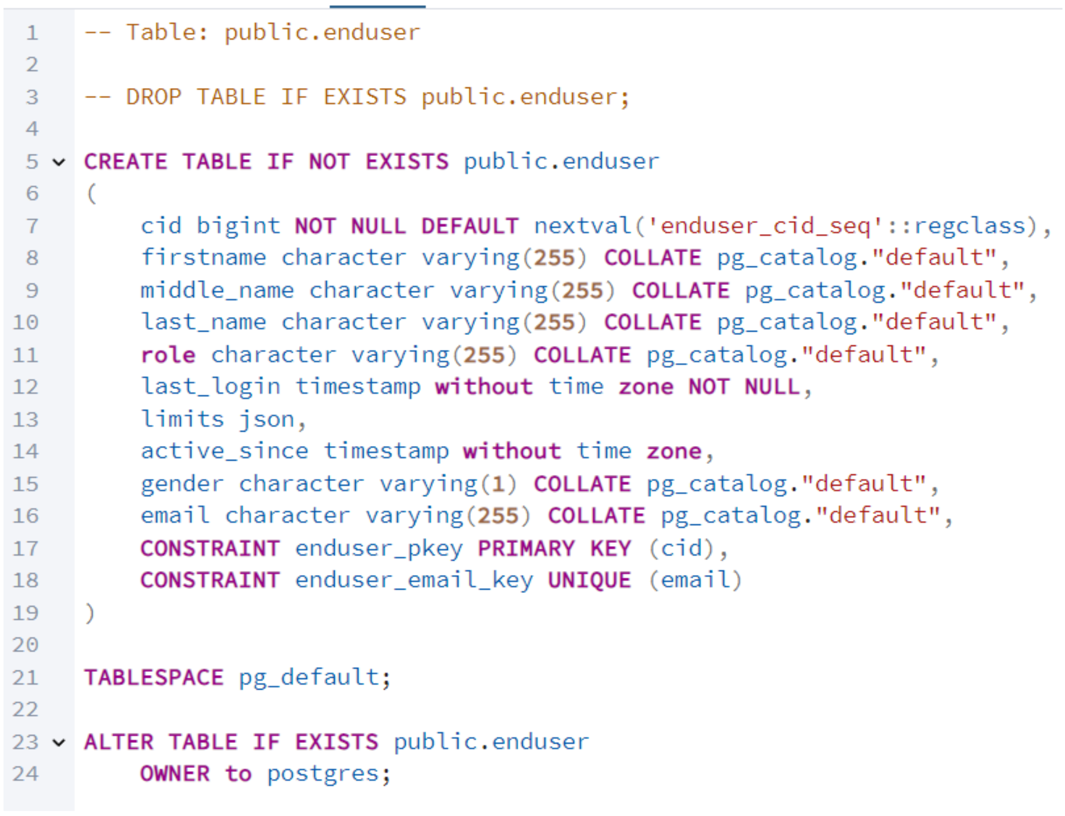
```sql
-- Table: public.device_status_management

-- DROP TABLE IF EXISTS public.device_status_management;

CREATE TABLE IF NOT EXISTS public.device_status_management
(
    serien_nr bigint,
    dsid bigint,
    note character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT fk_dsm_device_status FOREIGN KEY (dsid)
        REFERENCES public.device_status (dsid) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_dsm_smarthome FOREIGN KEY (serien_nr)
        REFERENCES public.smarthome (serien_nr) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.device_status_management
    OWNER to postgres;
```

Abbildung 8: device status managment table



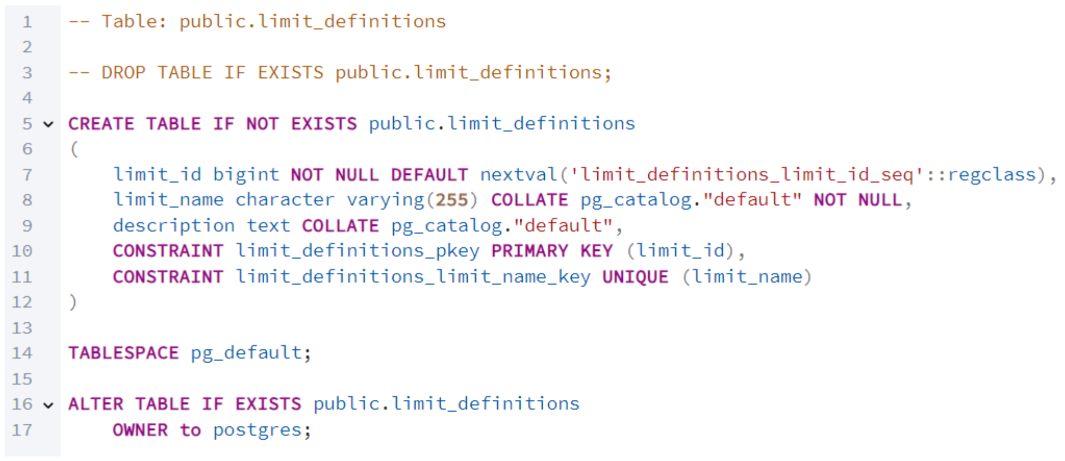
```sql
-- Table: public.device_type

-- DROP TABLE IF EXISTS public.device_type;

CREATE TABLE IF NOT EXISTS public.device_type
(
    type_id bigint NOT NULL DEFAULT nextval('device_type_type_id_seq'::regclass),
    name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT device_type_pkey PRIMARY KEY (type_id),
    CONSTRAINT device_type_name_key UNIQUE (name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.device_type
    OWNER to postgres;
```

Abbildung 9: device type table



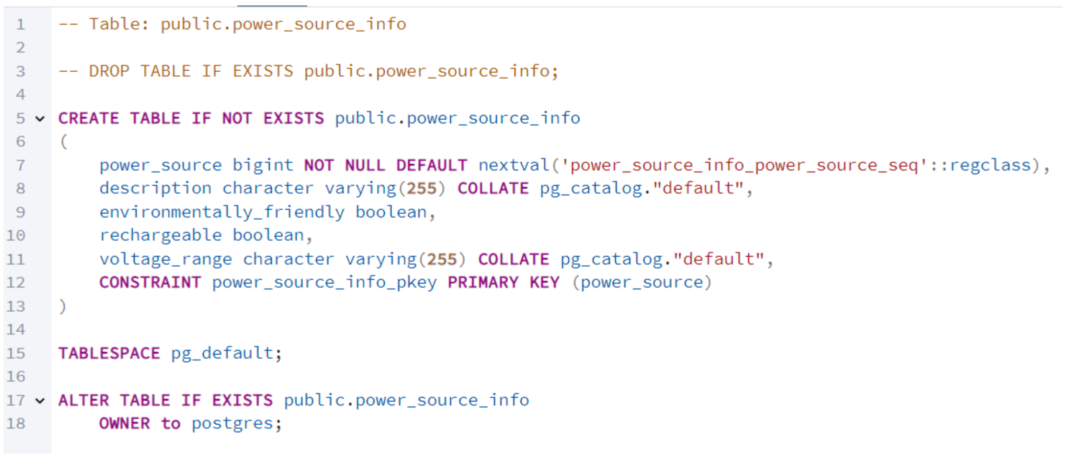
```sql
-- Table: public.enduser

-- DROP TABLE IF EXISTS public.enduser;

CREATE TABLE IF NOT EXISTS public.enduser
(
    cid bigint NOT NULL DEFAULT nextval('enduser_cid_seq'::regclass),
    firstname character varying(255) COLLATE pg_catalog."default",
    middle_name character varying(255) COLLATE pg_catalog."default",
    last_name character varying(255) COLLATE pg_catalog."default",
    role character varying(255) COLLATE pg_catalog."default",
    last_login timestamp without time zone NOT NULL,
    limits json,
    active_since timestamp without time zone,
    gender character varying(1) COLLATE pg_catalog."default",
    email character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT enduser_pkey PRIMARY KEY (cid),
    CONSTRAINT enduser_email_key UNIQUE (email)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.enduser
    OWNER to postgres;
```

Abbildung 10: enduser table



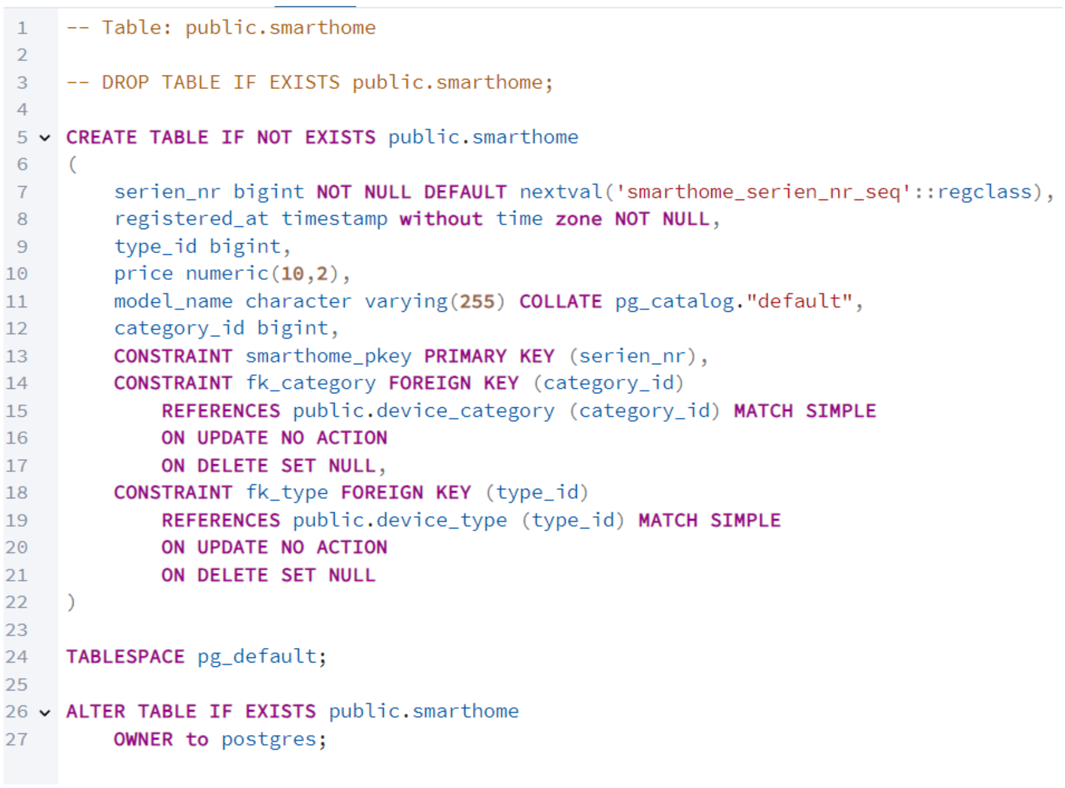
```sql
-- Table: public.limit_definitions

-- DROP TABLE IF EXISTS public.limit_definitions;

CREATE TABLE IF NOT EXISTS public.limit_definitions
(
    limit_id bigint NOT NULL DEFAULT nextval('limit_definitions_limit_id_seq'::regclass),
    limit_name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    description text COLLATE pg_catalog."default",
    CONSTRAINT limit_definitions_pkey PRIMARY KEY (limit_id),
    CONSTRAINT limit_definitions_limit_name_key UNIQUE (limit_name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.limit_definitions
    OWNER to postgres;
```

Abbildung 11: limit definition table



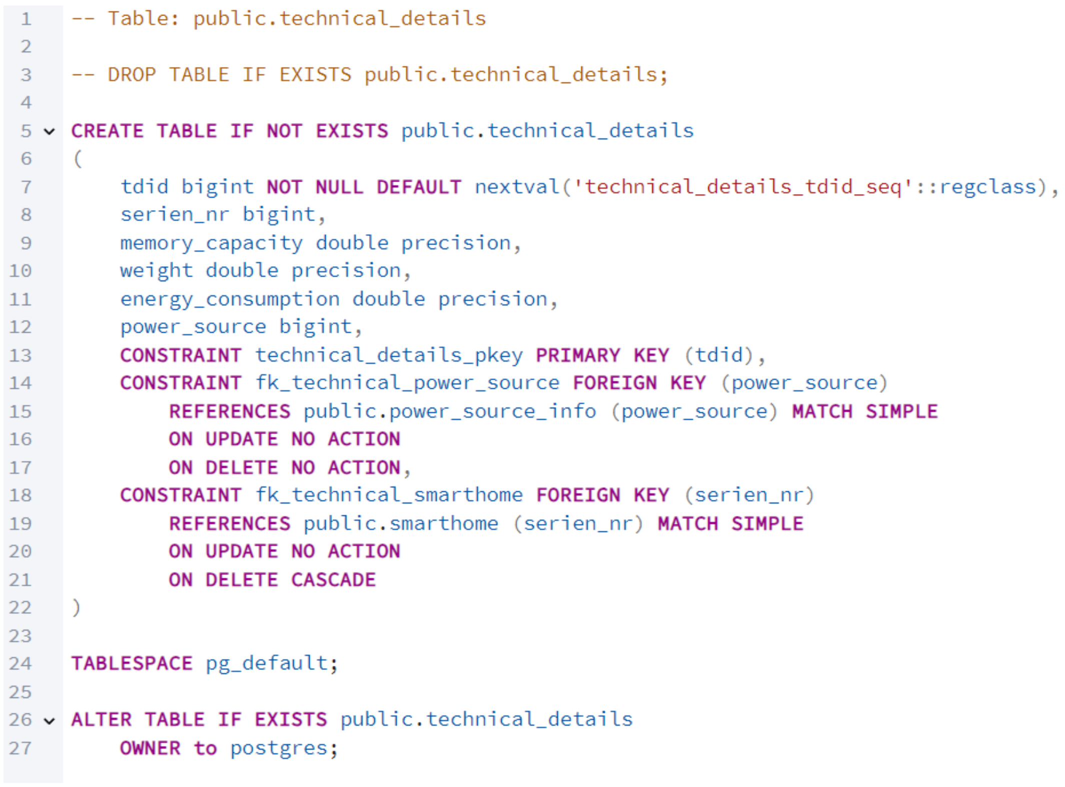
```sql
-- Table: public.power_source_info

-- DROP TABLE IF EXISTS public.power_source_info;

CREATE TABLE IF NOT EXISTS public.power_source_info
(
    power_source bigint NOT NULL DEFAULT nextval('power_source_info_power_source_seq'::regclass),
    description character varying(255) COLLATE pg_catalog."default",
    environmentally_friendly boolean,
    rechargeable boolean,
    voltage_range character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT power_source_info_pkey PRIMARY KEY (power_source)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.power_source_info
    OWNER to postgres;
```

Abbildung 12: power source Info table



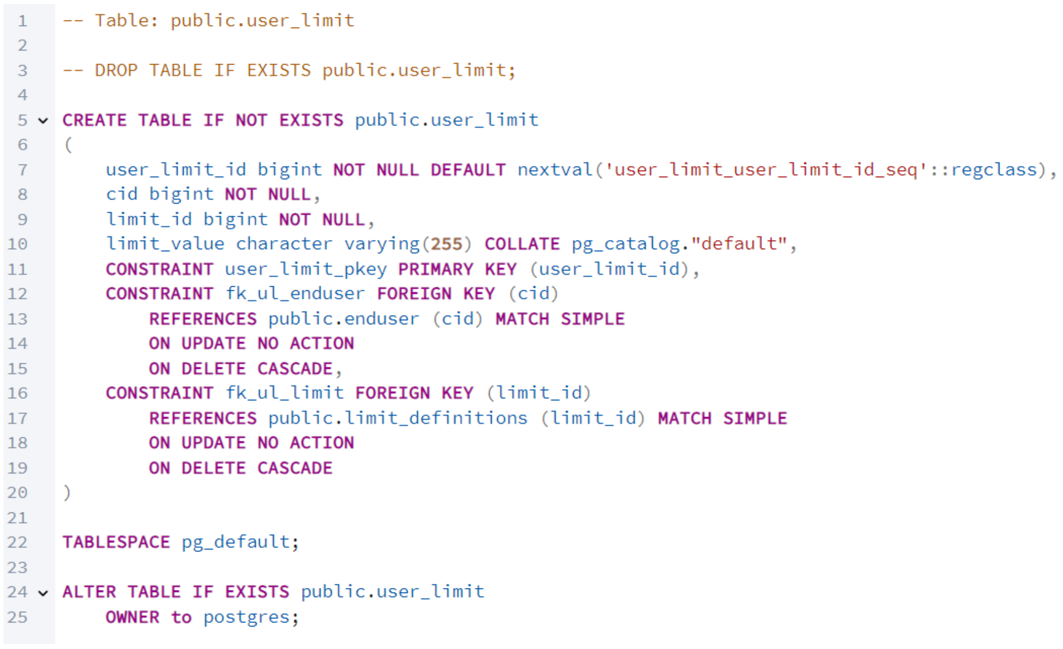
```sql
-- Table: public.smarthome

-- DROP TABLE IF EXISTS public.smarthome;

CREATE TABLE IF NOT EXISTS public.smarthome
(
    serien_nr bigint NOT NULL DEFAULT nextval('smarthome_serien_nr_seq'::regclass),
    registered_at timestamp without time zone NOT NULL,
    type_id bigint,
    price numeric(10,2),
    model_name character varying(255) COLLATE pg_catalog."default",
    category_id bigint,
    CONSTRAINT smarthome_pkey PRIMARY KEY (serien_nr),
    CONSTRAINT fk_category FOREIGN KEY (category_id)
        REFERENCES public.device_category (category_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE SET NULL,
    CONSTRAINT fk_type FOREIGN KEY (type_id)
        REFERENCES public.device_type (type_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE SET NULL
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.smarthome
    OWNER to postgres;
```

Abbildung 13: smarthome table



```sql
-- Table: public.technical_details

-- DROP TABLE IF EXISTS public.technical_details;

CREATE TABLE IF NOT EXISTS public.technical_details
(
    tdid bigint NOT NULL DEFAULT nextval('technical_details_tdid_seq'::regclass),
    serien_nr bigint,
    memory_capacity double precision,
    weight double precision,
    energy_consumption double precision,
    power_source bigint,
    CONSTRAINT technical_details_pkey PRIMARY KEY (tdid),
    CONSTRAINT fk_technical_power_source FOREIGN KEY (power_source)
        REFERENCES public.power_source_info (power_source) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_technical_smarthome FOREIGN KEY (serien_nr)
        REFERENCES public.smarthome (serien_nr) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.technical_details
    OWNER to postgres;
```

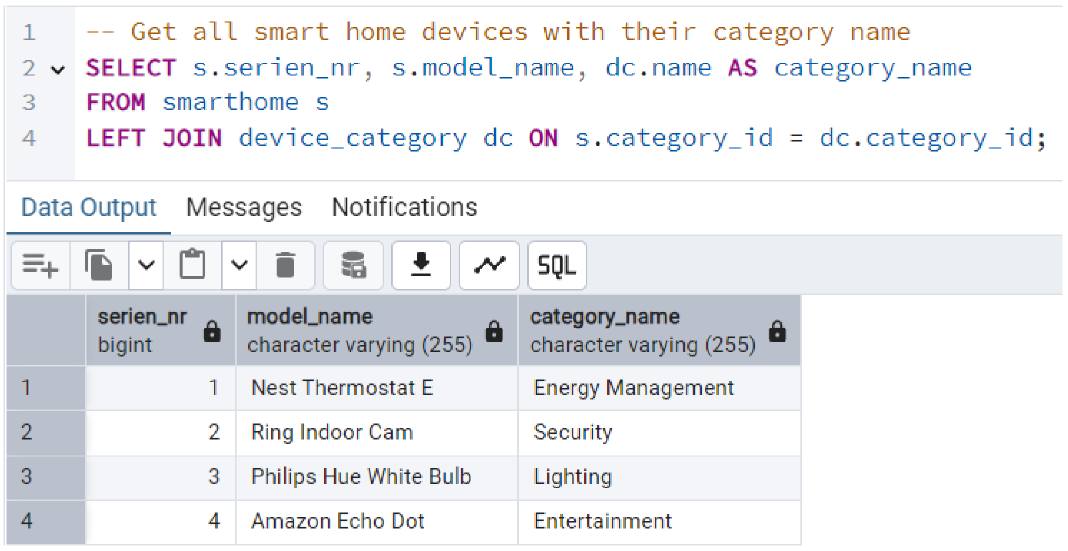
Abbildung 14: technical details table



```sql
-- Table: public.user_limit

-- DROP TABLE IF EXISTS public.user_limit;

CREATE TABLE IF NOT EXISTS public.user_limit
(
    user_limit_id bigint NOT NULL DEFAULT nextval('user_limit_user_limit_id_seq'::regclass),
    cid bigint NOT NULL,
    limit_id bigint NOT NULL,
    limit_value character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT user_limit_pkey PRIMARY KEY (user_limit_id),
    CONSTRAINT fk_ul_enduser FOREIGN KEY (cid)
        REFERENCES public.enduser (cid) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT fk_ul_limit FOREIGN KEY (limit_id)
        REFERENCES public.limit_definitions (limit_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.user_limit
    OWNER to postgres;
```

Abbildung 15: user limit table

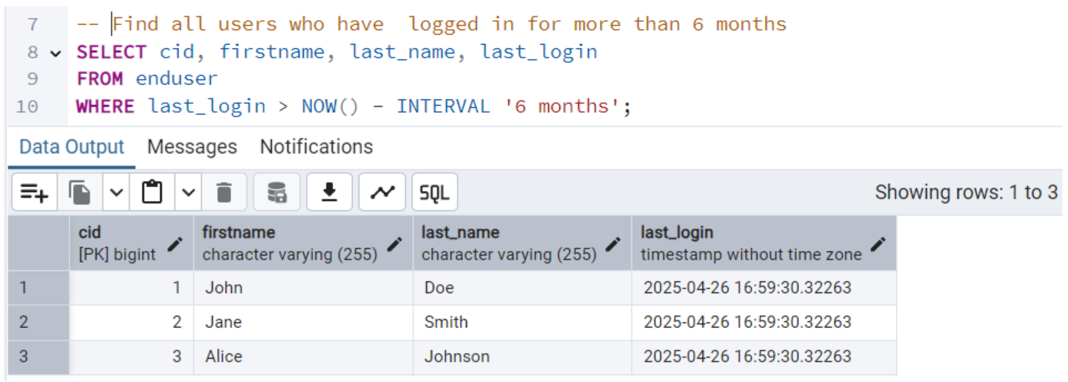
3.2 Queries



```sql
-- Get all smart home devices with their category name
SELECT s.serien_nr, s.model_name, dc.name AS category_name
FROM smarthome s
LEFT JOIN device_category dc ON s.category_id = dc.category_id;
```

	serien_nr bigint	model_name character varying (255)	category_name character varying (255)
1	1	Nest Thermostat E	Energy Management
2	2	Ring Indoor Cam	Security
3	3	Philips Hue White Bulb	Lighting
4	4	Amazon Echo Dot	Entertainment

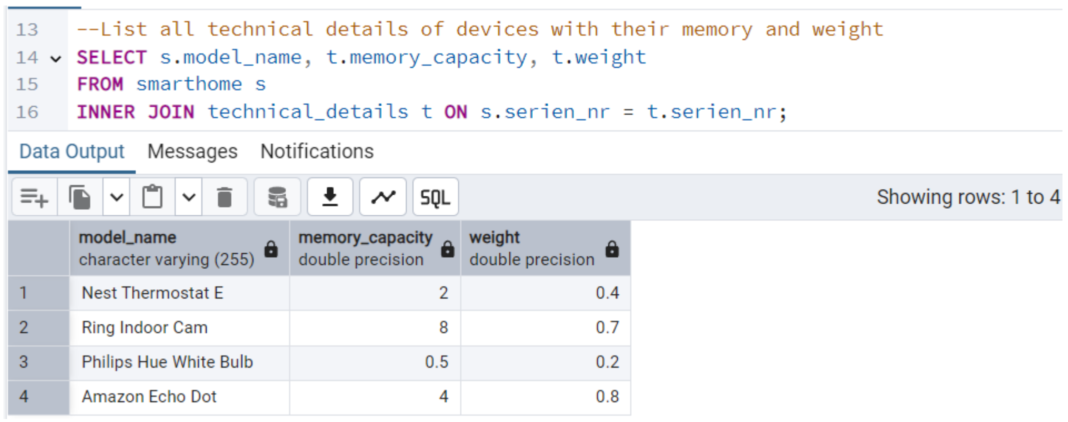
Abbildung 16: Get all smart home devices with their category name



```sql
-- Find all users who have  logged in for more than 6 months
SELECT cid, firstname, last_name, last_login
FROM enduser
WHERE last_login > NOW() - INTERVAL '6 months';
```

	cid [PK] bigint	firstname character varying (255)	last_name character varying (255)	last_login timestamp without time zone
1	1	John	Doe	2025-04-26 16:59:30.32263
2	2	Jane	Smith	2025-04-26 16:59:30.32263
3	3	Alice	Johnson	2025-04-26 16:59:30.32263

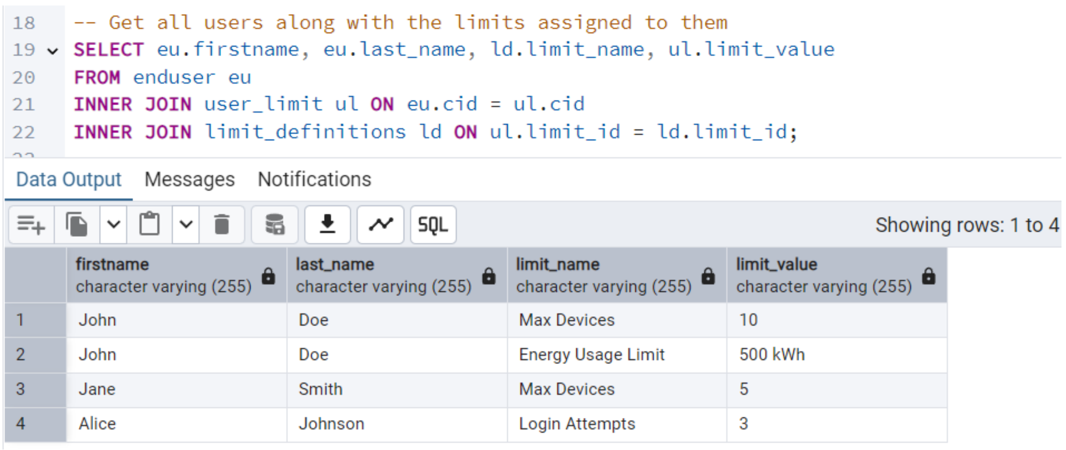
Abbildung 17: Find all users who have logged in for more than 6 months



```sql
--List all technical details of devices with their memory and weight
SELECT s.model_name, t.memory_capacity, t.weight
FROM smarthome s
INNER JOIN technical_details t ON s.serien_nr = t.serien_nr;
```

	model_name character varying (255)	memory_capacity double precision	weight double precision
1	Nest Thermostat E	2	0.4
2	Ring Indoor Cam	8	0.7
3	Philips Hue White Bulb	0.5	0.2
4	Amazon Echo Dot	4	0.8

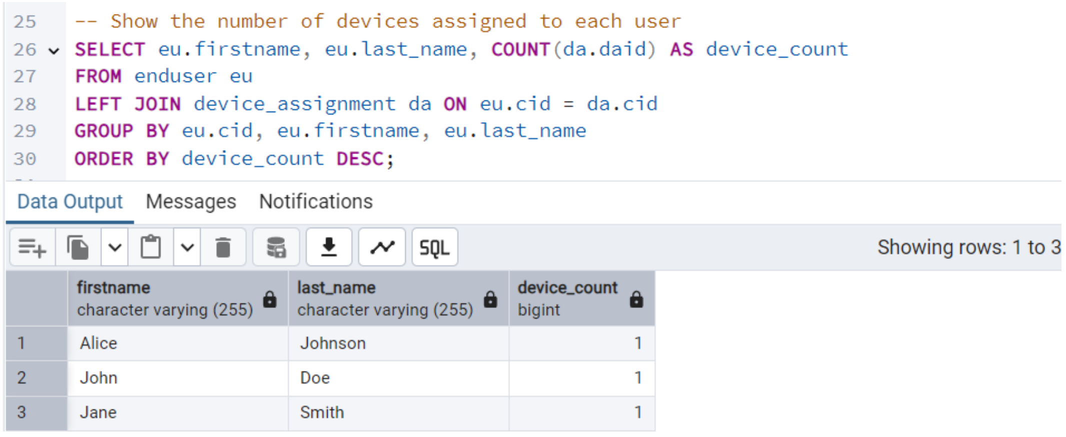
Abbildung 18: List all technical details of devices with their memory and weight



```sql
-- Get all users along with the limits assigned to them
SELECT eu.firstname, eu.last_name, ld.limit_name, ul.limit_value
FROM enduser eu
INNER JOIN user_limit ul ON eu.cid = ul.cid
INNER JOIN limit_definitions ld ON ul.limit_id = ld.limit_id;
```

	firstname character varying (255)	last_name character varying (255)	limit_name character varying (255)	limit_value character varying (255)
1	John	Doe	Max Devices	10
2	John	Doe	Energy Usage Limit	500 kWh
3	Jane	Smith	Max Devices	5
4	Alice	Johnson	Login Attempts	3

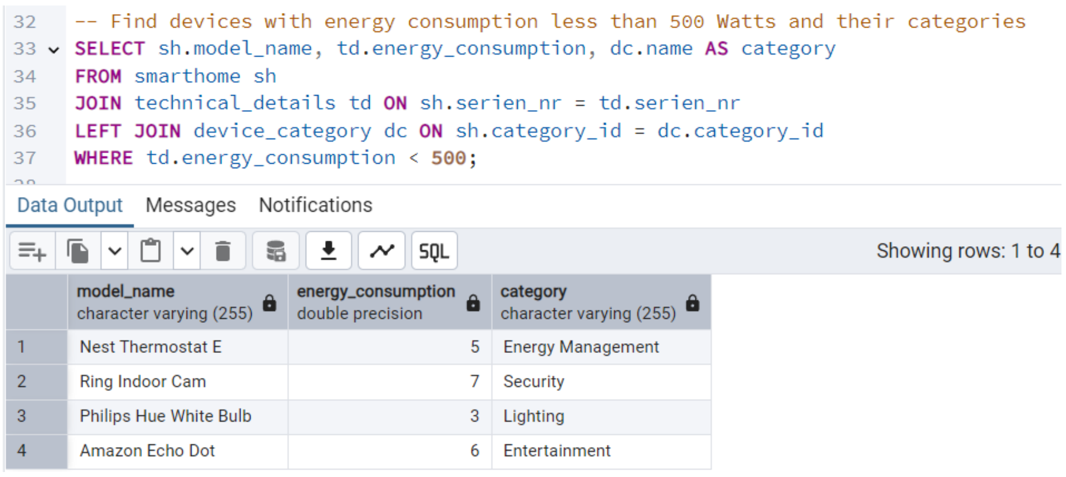
Abbildung 19: Get all users along with the limits assigned to them



```sql
-- Show the number of devices assigned to each user
SELECT eu.firstname, eu.last_name, COUNT(da.daid) AS device_count
FROM enduser eu
LEFT JOIN device_assignment da ON eu.cid = da.cid
GROUP BY eu.cid, eu.firstname, eu.last_name
ORDER BY device_count DESC;
```

	firstname character varying (255)	last_name character varying (255)	device_count bigint
1	Alice	Johnson	1
2	John	Doe	1
3	Jane	Smith	1

Abbildung 20: Show the number of devices assigned to each user



```sql
-- Find devices with energy consumption less than 500 Watts and their categories
SELECT sh.model_name, td.energy_consumption, dc.name AS category
FROM smarthome sh
JOIN technical_details td ON sh.serien_nr = td.serien_nr
LEFT JOIN device_category dc ON sh.category_id = dc.category_id
WHERE td.energy_consumption < 500;
```

	model_name character varying (255)	energy_consumption double precision	category character varying (255)
1	Nest Thermostat E	5	Energy Management
2	Ring Indoor Cam	7	Security
3	Philips Hue White Bulb	3	Lighting
4	Amazon Echo Dot	6	Entertainment

Abbildung 21: Find devices with energy consumption less than 500 Watts and their categories