Database systems

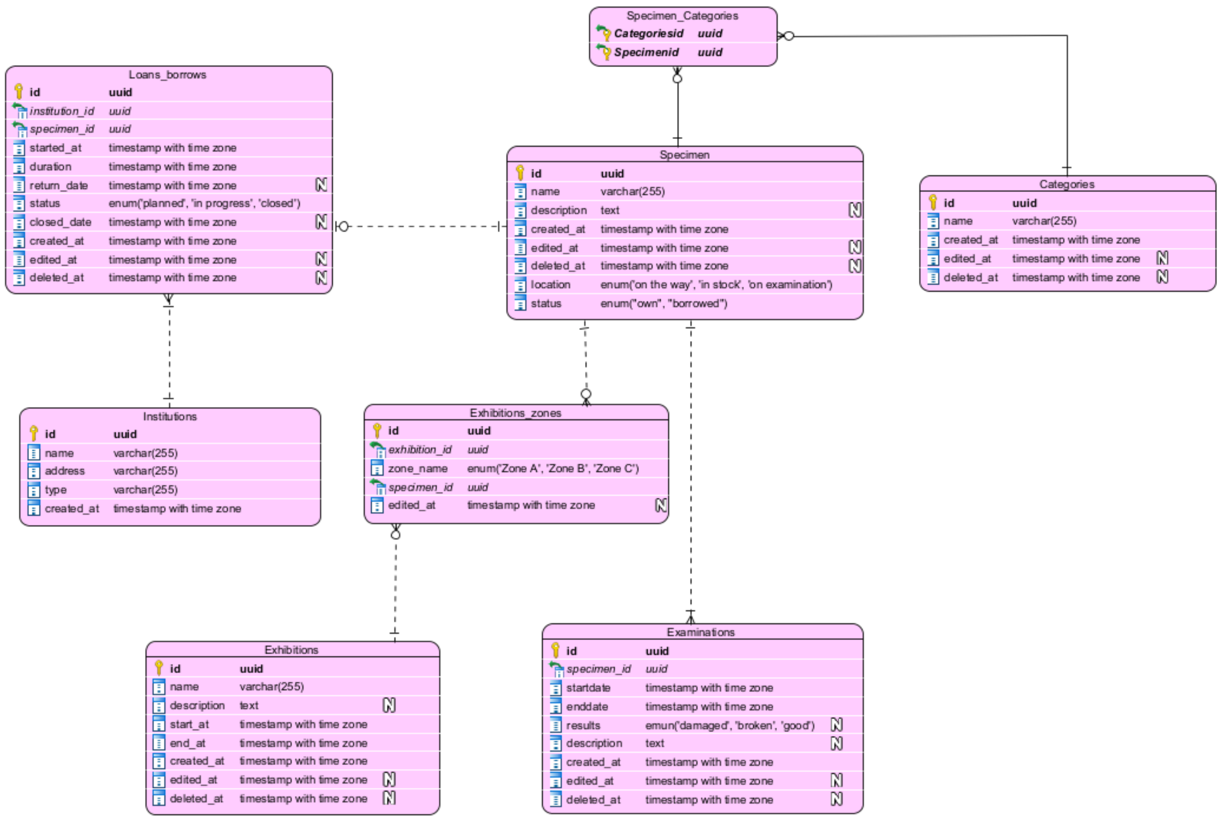
ASSIGNMENT 5
MUSEUM DATABASE

Maryna Kolesnykova
ID: 122475

TABLE OF CONTEXT

PHYSICAL MODEL

The physical model of the database represents the detailed diagram for my implementation of fourth assignment. The diagram was implemented in Visual Paradigm and is shown below:


Specimen_Categories
Categoriesid uuid
Specimenid uuid
Loans_borrows
id uuid
institution_id uuid
specimen_id uuid
started_at timestamp with time zone
duration timestamp with time zone
return_date timestamp with time zone
status enum('planned', 'in progress', 'closed')
closed_date timestamp with time zone
created_at timestamp with time zone
edited_at timestamp with time zone
deleted_at timestamp with time zone
Specimen
id uuid
name varchar(255)
description text
created_at timestamp with time zone
edited_at timestamp with time zone
deleted_at timestamp with time zone
location enum('on the way', 'in stock', 'on examination')
status enum("own", "borrowed")
Categories
id uuid
name varchar(255)
created_at timestamp with time zone
edited_at timestamp with time zone
deleted_at timestamp with time zone
Institutions
id uuid
name varchar(255)
address varchar(255)
type varchar(255)
created_at timestamp with time zone
Exhibitions_zones
id uuid
exhibition_id uuid
zone_name enum('Zone A', 'Zone B', 'Zone C')
specimen_id uuid
edited_at timestamp with time zone
Exhibitions
id uuid
name varchar(255)
description text
start_at timestamp with time zone
end_at timestamp with time zone
created_at timestamp with time zone
edited_at timestamp with time zone
deleted_at timestamp with time zone
Examinations
id uuid
specimen_id uuid
startdate timestamp with time zone
enddate timestamp with time zone
results emun('damaged', 'broken', 'good')
description text
created_at timestamp with time zone
edited_at timestamp with time zone
deleted_at timestamp with time zone

The pdf file is also attached to the zip folder of assignment for better clarity and well seeing of the model.

PERFORMED CHANGES TO THE PHYSICAL MODEL

In comparison to the previous version, the new database is simpler. The "Exhibitions_changes" and "Loans_Borrows_changes" tables from the prior version are no longer present in this model, which also does not contain any self-looping code. Additionally, the table specimen has been updated with new fields such as "status" and "location," both of which are represented as enumerative values (the potential enumerations are displayed in the above physical model photo). The museum's zones were previously stored in the table "zones," which is now represented as an enum type in the "exhibitions_zones" table. It was created with the intention of making the process of organizing an exhibition easier because, for large institutions, we always know the zones (rooms) that the museum has at the stage of database creation. The table 'Exhibitions_zones' is the only one that wasn't represented in the previous version of physical model and besides of its own id and edited_at (used to see when the changes to the table were made. For example, change the zone the specimen is placed in within the exhibition) this table is now has foreign keys to exhibition and specimen and is likely to be used as a storage for the historical records of exhibitions, specimens that were exposed exhibition and zones the specimen was located it. 'Loans_borrows' table also has less columns and for storing data about deals that were made with specified specimen now uses only 'started_date' – the date the loan/borrow is about to start, 'duration' – duration of the deal, 'return_date' – the estimate date when my museum get the artefact(in both cases either loan: 'return_date' – the date the specimen returns to my institution, or borrow: 'rerturn_date' – the date I get the borrowed specimen) and status and 'closed_date' that remain the same meaning and goal from the previous implementation.

CONSTRAINS

The implemented database has is regulated with implemented constrains. The most basic ones are ‘nullable marks’, that are represented with ‘N’ letter in the physical model in previous section. This constrain helps me avoid situation of being in luck of data for performing any operation or just helps logically understand and visualize the model. For example, not nullable name in specimen table is logical constrain because I cannot record nameless artifact in the museum even though the name of a specimen as barely used in the application logic. On the other hand, ‘zone_name’ column in ‘exhibitions_zones’ table is widely used while implementing scenarios and cannot be null under any circumstances. Other important constrains are associated with creating the records and are represented as default values. These constrains are not seen on the model photo. Nevertheless, in this implementation all created_at dates are by default are set to current time and are defined in the database script:

```
created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
```

Another default value is uuid. The user of the database do not have to come up with a new uuid every time inserting new record, the database will do it instead using ‘uuid-ossp ’ extension and default calling function uuid_generate_v4().

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
 id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
```

The last default constrain is location ‘in stock’ for every inserted element in specimen table.

```
location location_enum DEFAULT 'in stock' NOT NULL,
```

The time appropriacy (the start date of an event cannot be after the end date of the event) is also handled by constrains. For example, during the exhibition it is very important to keep an eye on correct dates recording as it could have unwanted consequences.

```
CREATE TABLE Exhibitions (
    id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    start_at TIMESTAMPTZ NOT NULL,
    end_at TIMESTAMPTZ NOT NULL,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    edited_at TIMESTAMPTZ,
    deleted_at TIMESTAMPTZ,
    CHECK (start_at < end_at)
);
```

Similar constrain is implemented on ‘Loans_Borrows’ and ‘Examinations’ tables.

On delete constrains are also present in this implementation of the database and implemented using either ‘restrict’ or ‘cascade’. Restrict constrain is used on categories id, specimen id, institution id and examination id and do not allow me to delete these values if they are used in other tables as foreign keys not to lose any important historical, active or planes records. Cascade delete is used for exhibition id, so that if the exhibition is being canceled and the recors is deleted it also deletes the records in ‘exhibition_zones’ table.

```
FOREIGN KEY (exhibition_id) REFERENCES Exhibitions (id) ON DELETE CASCADE,
FOREIGN KEY (specimen_id) REFERENCES Specimen (id) ON DELETE RESTRICT
```

PROCESSES

This section shows possible successful implementation of required processes for the museum that owns the system with current database connected to it. All triggers and functions with the examples of error messages they generate are describes and shown in another section. The scenarios are implemented on a prefilled database.

In these section following cases are implemented:

- Organization of the exhibition (Naplanovanie expozicie)
- Insertion of a new specimen (Vkladanie noveho exemplaru)
- Specimens zone changes (Presun exemplaru do inej zony)
- Taking over specimen from another institution (Prevziate exemplaru z inej institucie)
- Borrowing process from another institution (Zapozicanie exemplaru z inej institucie)

ORGANIZATION OF THE EXHIBITION

To organize an exhibition the database must already have records about the specimen that the museum owns. Notice that there are existing records in specimen table at the moment of creation the exhibition. To perform current procedure the user should use two queries:

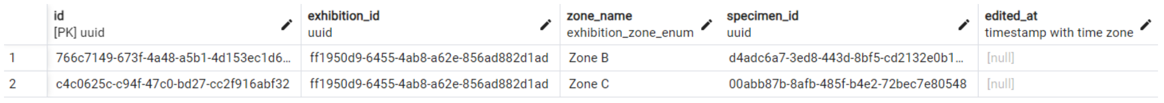
Firstly, the exhibition itself is being created with its name, start and end dates.

```
INSERT INTO Exhibitions (name, start_at, end_at)
VALUES
    ('History of art', '2024-05-01', '2024-05-30');
```

After that the exhibition is filles with specimens and arranges to different zones in 'Exhibition_Zones' table.

```
INSERT INTO Exhibitions_zones (exhibition_id, zone_name, specimen_id)
VALUES
    ('ff1950d9-6455-4ab8-a62e-856ad882d1ad', 'Zone B', 'd4adc6a7-3ed8-443d-8bf5-cd2132e0b1c9'),
    ('ff1950d9-6455-4ab8-a62e-856ad882d1ad', 'Zone C', '00abb87b-8afb-485f-b4e2-72bec7e80548');
```

The resulting exhibitions_zones table lokks like this:



	id [PK] uuid	exhibition_id uuid	zone_name exhibition_zone_enum	specimen_id uuid	edited_at timestamp with time zone
1	766c7149-673f-4a48-a5b1-4d153ec1d6...	ff1950d9-6455-4ab8-a62e-856ad882d1ad	Zone B	d4adc6a7-3ed8-443d-8bf5-cd2132e0b1...	[null]
2	c4c0625c-c94f-47c0-bd27-cc2f916abf32	ff1950d9-6455-4ab8-a62e-856ad882d1ad	Zone C	00abb87b-8afb-485f-b4e2-72bec7e80548	[null]

After that the exhibition is successfully created!

INSERTION OF A NEW SPECIMEN

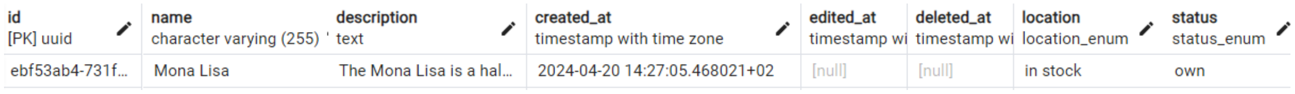
The insertion of a new specimen requires two insert queries by the person who performs the operation. First of all, it is the insertion of the specimen itself using following query:

```
INSERT INTO Specimen (name, description, status)
VALUES
    ('Mona Lisa', 'The Mona Lisa is a half-length portrait painting by Italian artist Leonardo da Vinci.', 'own');
```

According to the fact that every item must be assigned at least into one category, the administrator must choose the appropriate one and add the record into 'Specimen_Ctegories' table using ids of specimen and category.(notice that I have already created a demonstrative category for this process)

```
INSERT INTO Specimen_Categories (Categoriesid, Specimenid)
VALUES
    ('39998ac3-2ddf-4d72-8605-4be07096647c', 'ebf53ab4-731f-4e3c-a103-97add6b6d5aa');
```

After these operations the specimen is successfully inserted into the database.



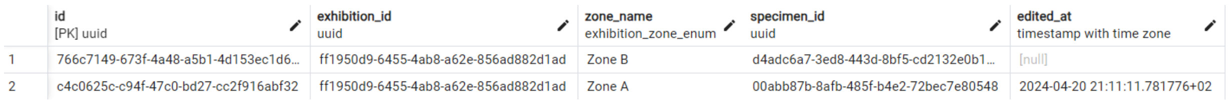
id [PK] uuid	name character varying (255)	description text	created_at timestamp with time zone	edited_at timestamp wi	deleted_at timestamp wi	location location_enum	status status_enum
ebf53ab4-731f...	Mona Lisa	The Mona Lisa is a hal...	2024-04-20 14:27:05.468021+02	[null]	[null]	in stock	own

SPECIMEN ZONE CHANGES

In this process I use update function:

```
update exhibitions_zones
set zone_name = 'Zone A'
where id = 'c4c0625c-c94f-47c0-bd27-cc2f916abf32'
```

After performing this operation, the edited_at date is automatically set to current date and time and the record is successfully updated.



	id [PK] uuid	exhibition_id uuid	zone_name exhibition_zone_enum	specimen_id uuid	edited_at timestamp with time zone
1	766c7149-673f-4a48-a5b1-4d153ec1d6...	ff1950d9-6455-4ab8-a62e-856ad882d1ad	Zone B	d4adc6a7-3ed8-443d-8bf5-cd2132e0b1...	[null]
2	c4c0625c-c94f-47c0-bd27-cc2f916abf32	ff1950d9-6455-4ab8-a62e-856ad882d1ad	Zone A	00abb87b-8afb-485f-b4e2-72bec7e80548	2024-04-20 21:11:11.781776+02

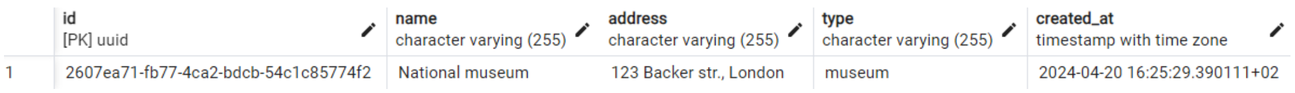
TAKING OVER SPECIMEN FROM ANOTHER INSTITUTION

This process requires created active deal in loans_borrows table
This process consists of three steps:

1. Set the closed date to current date in appropriate record
   ```
   update loans_borrows
   set closed_date = '2025-01-01'
   where id = '3933fcdb-032a-4d81-9073-4023fdeb867a'
   ```
2. Update the specimen location from 'on the way' to 'on examination'
   ```
   update specimen
   set location = 'on examination'
   where id = 'ef70b801-a240-4474-bcd6-c982c8744a87'
   ```
3. Create new record in examination table using borrowed specimen id
   ```
   insert into examinations (specimen_id, startdate, enddate)
   values('ef70b801-a240-4474-bcd6-c982c8744a87', '2025-01-01', '2025-01-05')
   ```

BORROWING PROCESS FROM ANOTHER INSTITUTION

To complete the process of borrowing item from another institution the record of the deal must be created in the loans_borrows table. Notice that for this scenario I have already created institution:



	id [PK] uuid	name character varying (255)	address character varying (255)	type character varying (255)	created_at timestamp with time zone
1	2607ea71-fb77-4ca2-bdcb-54c1c85774f2	National museum	123 Backer str., London	museum	2024-04-20 16:25:29.390111+02

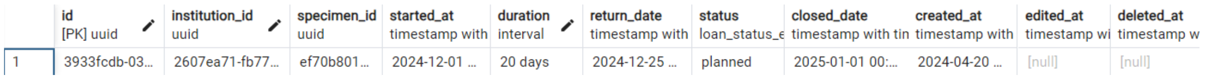
1. First of all, to create a deal, the record about desired specimen must be inserted into specimen table. For example, I want to borrow ''Girl with a Pearl Earring'' and I need to create a record about this painting. Notice that in planes exhibitions I always set location to 'on the way' even if borrow is planned for the next year. The location is changed only after receiving the specimen.

```
INSERT INTO Specimen (name, description, status, location)
VALUES
    ('Girl with a Pearl Earring',
     'Girl with a Pearl Earring is an oil painting by Dutch Golden Age painter Johannes Vermeer, dated c. 1665.',
     'borrowed', 'on the way');
```

2. Create a record in loans_borrows table:

```
insert into loans_borrows (institution_id, specimen_id, started_at, return_date, duration, status)
values
    ('2607ea71-fb77-4ca2-bdcb-54c1c85774f2',
     'ef70b801-a240-4474-bcd6-c982c8744a87',
     '2024-12-01', '2024-12-25', '20 days', 'planned');
```

After this the borrow is successfully recorded!



	id [PK] uuid	institution_id uuid	specimen_id uuid	started_at timestamp with	duration interval	return_date timestamp with	status loan_status_e	closed_date timestamp with tin	created_at timestamp with	edited_at timestamp wi	deleted_at timestamp w
1	3933fcdb-03...	2607ea71-fb77...	ef70b801...	2024-12-01 ...	20 days	2024-12-25 ...	planned	2025-01-01 00:...	2024-04-20 ...	[null]	[null]

FUNCTIONS AND TRIGGERS

In this database several mismatches can occur and were solved by implemented functions and triggers. Let's take a closer look.

OVERLAPPING EXHIBITIONS

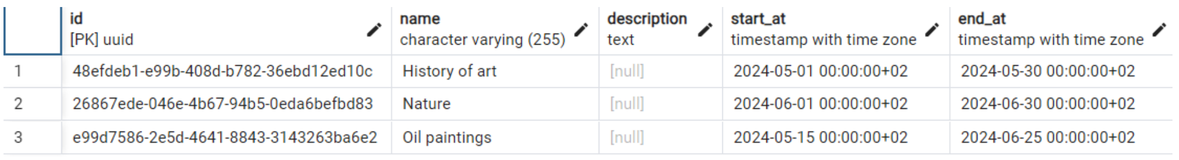
In the term of overlapping exhibitions the user can face two problems:

1. It is impossible to plan an exhibition in the zone that has already assigned overlapping exhibition. For example, there is a planned exhibition 'History of art' that starts 01.05.2024 and have end date 30.05.2024. imagine that this exhibition will be performed in Zone A and Zone B. I also plan exhibition 'Oil paintings' that will be set from 15.05.2024 to 25.06.2024. As you can see these two exhibitions are overlapping

```
BEGIN

    IF EXISTS (
        SELECT 1
        FROM Exhibitions_zones
        WHERE exhibition_id = NEW.exhibition_id
          AND zone_name = NEW.zone_name
          AND specimen_id != NEW.specimen_id

    ) THEN
        RETURN NEW;
    end if;

    SELECT e.start_at, e.end_at, z.zone_name, z.specimen_id INTO existing_exhibition_start, existing_exhibition_end, zone_name_u
    FROM Exhibitions_zones z
    JOIN Exhibitions e ON z.exhibition_id = e.id
    WHERE z.zone_name = NEW.zone_name;

    SELECT start_at, end_at INTO inserted_exhibition_start, inserted_exhibition_end
    FROM Exhibitions
    WHERE id = NEW.exhibition_id;

    IF (existing_exhibition_start, existing_exhibition_end) OVERLAPS (inserted_exhibition_start, inserted_exhibition_end)
    and zone_name_u = new.zone_name THEN
        RAISE EXCEPTION 'Cannot add a zone with overlapping exhibitions';
    END IF;

    RETURN NEW;
```

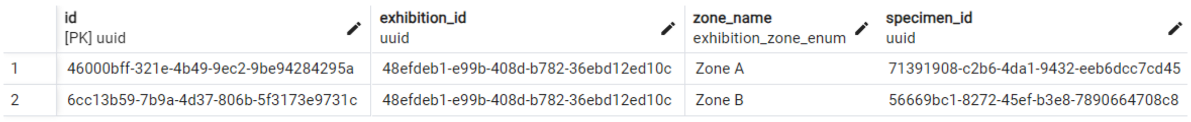
This trigger stores start and end dates from existing records into variables existing_exhibition_strat and existing_exhibition_end and stores the start and end date based on the exhibition the user inserts into inserted_... variables. Then it checks zones for overlapping and raise error if their zone_names are equal.

Example of usage:



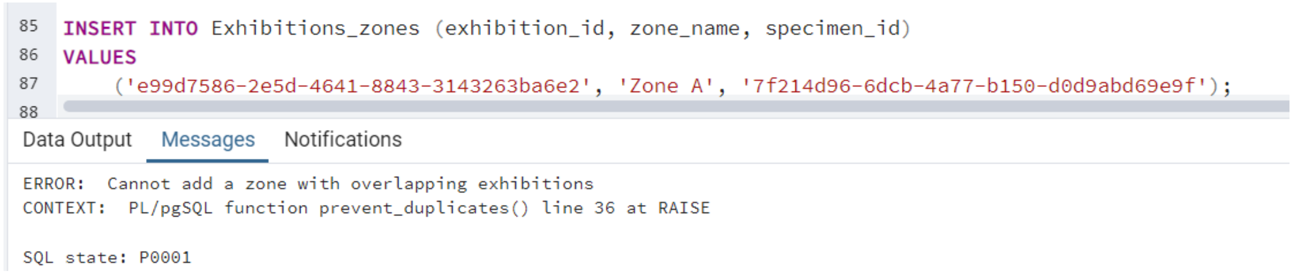
	id [PK] uuid	name character varying (255)	description text	start_at timestamp with time zone	end_at timestamp with time zone
1	48efdeb1-e99b-408d-b782-36ebd12ed10c	History of art	[null]	2024-05-01 00:00:00+02	2024-05-30 00:00:00+02
2	26867ede-046e-4b67-94b5-0eda6befbd83	Nature	[null]	2024-06-01 00:00:00+02	2024-06-30 00:00:00+02
3	e99d7586-2e5d-4641-8843-3143263ba6e2	Oil paintings	[null]	2024-05-15 00:00:00+02	2024-06-25 00:00:00+02

Existing exhibition with assigned zones:



	id [PK] uuid	exhibition_id uuid	zone_name exhibition_zone_enum	specimen_id uuid
1	46000bff-321e-4b49-9ec2-9be94284295a	48efdeb1-e99b-408d-b782-36ebd12ed10c	Zone A	71391908-c2b6-4da1-9432-eeb6dcc7cd45
2	6cc13b59-7b9a-4d37-806b-5f3173e9731c	48efdeb1-e99b-408d-b782-36ebd12ed10c	Zone B	56669bc1-8272-45ef-b3e8-7890664708c8

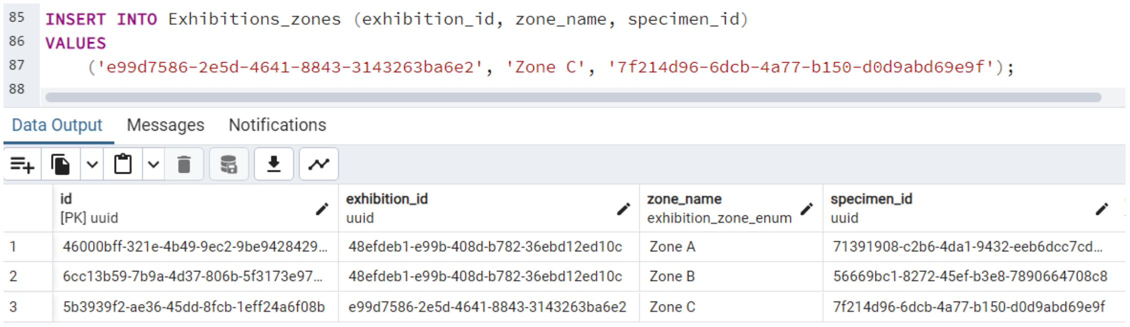
Lets try to insert another specimen into Zone A but for 'Oil paintings' exhibition



```
INSERT INTO Exhibitions_zones (exhibition_id, zone_name, specimen_id)
VALUES
    ('e99d7586-2e5d-4641-8843-3143263ba6e2', 'Zone A', '7f214d96-6dcb-4a77-b150-d0d9abd69e9f');
```

```
ERROR:  Cannot add a zone with overlapping exhibitions
CONTEXT:  PL/pgSQL function prevent_duplicates() line 36 at RAISE

SQL state: P0001
```

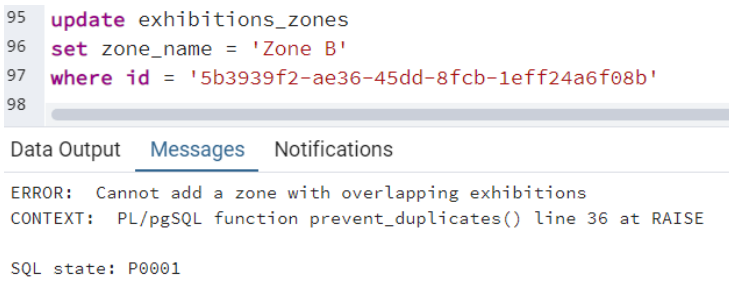
It shows the trigger in use. It didn't allow pasting the overlapping exhibition into the same zone. However, if I try to insert this specimen into 'Oil paintings' exhibition in free zone, the insertion is performed with no problems.



```
INSERT INTO Exhibitions_zones (exhibition_id, zone_name, specimen_id)
VALUES
    ('e99d7586-2e5d-4641-8843-3143263ba6e2', 'Zone C', '7f214d96-6dcb-4a77-b150-d0d9abd69e9f');
```

	id [PK] uuid	exhibition_id uuid	zone_name exhibition_zone_enum	specimen_id uuid
1	46000bff-321e-4b49-9ec2-9be9428429…	48efdeb1-e99b-408d-b782-36ebd12ed10c	Zone A	71391908-c2b6-4da1-9432-eeb6dcc7cd…
2	6cc13b59-7b9a-4d37-806b-5f3173e97…	48efdeb1-e99b-408d-b782-36ebd12ed10c	Zone B	56669bc1-8272-45ef-b3e8-7890664708c8
3	5b3939f2-ae36-45dd-8fcb-1eff24a6f08b	e99d7586-2e5d-4641-8843-3143263ba6e2	Zone C	7f214d96-6dcb-4a77-b150-d0d9abd69e9f

The same works for updating. If I try to update 'oil paintings' and paste used specimen into taken zone, the error message will appear:



```
update exhibitions_zones
set zone_name = 'Zone B'
where id = '5b3939f2-ae36-45dd-8fcb-1eff24a6f08b'
```

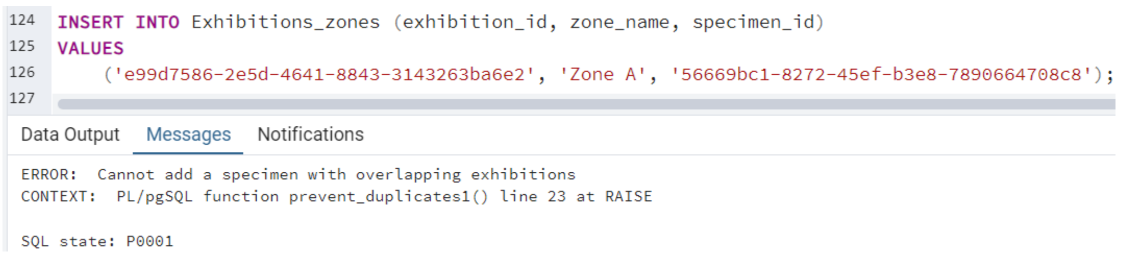
```
ERROR:  Cannot add a zone with overlapping exhibitions
CONTEXT:  PL/pgSQL function prevent_duplicates() line 36 at RAISE

SQL state: P0001
```

2. Error message will appear if user try to insert the same specimen in two different overlapping exhibitions. Imagine I try to insert 'Mona Lisa' into 'Oil paintings' exhibition, but this piece of art is already on 'History of Art' exhibition that overlaps with the new one. Before the insertion I freed the Zone A to show the error message.

Very similar to the previous trigger this also stores start and end dates but then raises error based on the similarity of specimen_ids

```
BEGIN

    IF EXISTS (
        SELECT 1
        FROM Exhibitions_zones
        WHERE exhibition_id = NEW.exhibition_id
          AND zone_name = NEW.zone_name
          AND specimen_id != NEW.specimen_id
    ) THEN
        RETURN NEW;
    end if;

    SELECT e.start_at, e.end_at, z.zone_name, z.specimen_id, z.id INTO existing_exhibition_start, existing_exhibition_end, zone_name_
    FROM Exhibitions_zones z
    JOIN Exhibitions e ON z.exhibition_id = e.id
    WHERE z.specimen_id = NEW.specimen_id;

    SELECT start_at, end_at INTO inserted_exhibition_start, inserted_exhibition_end
    FROM Exhibitions
    WHERE id = NEW.exhibition_id;

    IF (existing_exhibition_start, existing_exhibition_end) OVERLAPS (inserted_exhibition_start, inserted_exhibition_end)
    AND specimen_id_u = NEW.specimen_id and new.id != idd THEN
        RAISE EXCEPTION 'Cannot add a specimen with overlapping exhibitions';
    END IF;

    RETURN NEW;
END;
```

Example of use:



```
INSERT INTO Exhibitions_zones (exhibition_id, zone_name, specimen_id)
VALUES
    ('e99d7586-2e5d-4641-8843-3143263ba6e2', 'Zone A', '56669bc1-8272-45ef-b3e8-7890664708c8');
```

```
ERROR:  Cannot add a specimen with overlapping exhibitions
CONTEXT:  PL/pgSQL function prevent_duplicates1() line 23 at RAISE

SQL state: P0001
```

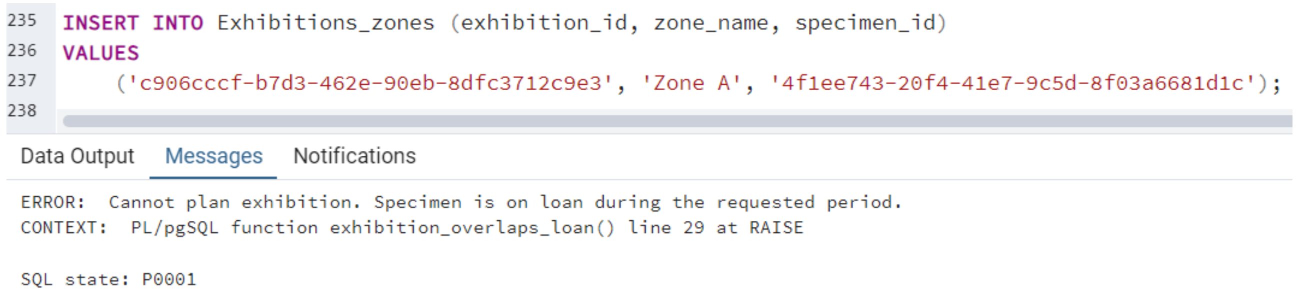
ALREADY IN USE

Another problem that can appear is that the specimen will be promised to someone or will be on examination during the requested exhibition period. Lets split it into two parts:

1. The specimen in on loan

```
CREATE OR REPLACE FUNCTION exhibition_overlaps_loan()
RETURNS TRIGGER AS $$
DECLARE
    conflict_exists BOOLEAN;
    loan_start TIMESTAMP;
    exhibition_end TIMESTAMP;
    specimen_id_use uuid;
    loan_close_date TIMESTAMP;
BEGIN
    conflict_exists := FALSE;

    SELECT lb.started_at, lb.specimen_id
    INTO loan_start, specimen_id_use
    FROM loans_borrows lb
    WHERE lb.specimen_id = NEW.specimen_id
    AND lb.closed_date IS NULL;

    select e.end_at from exhibitions e
    into exhibition_end
    where e.id = new.exhibition_id
    and loan_close_date is null;

    IF loan_start IS NOT NULL THEN
        IF exhibition_end >= loan_start THEN
            RAISE EXCEPTION 'Cannot plan exhibition. Specimen is on loan during the requested period.';
        END IF;
    END IF;

    RETURN NEW;
```

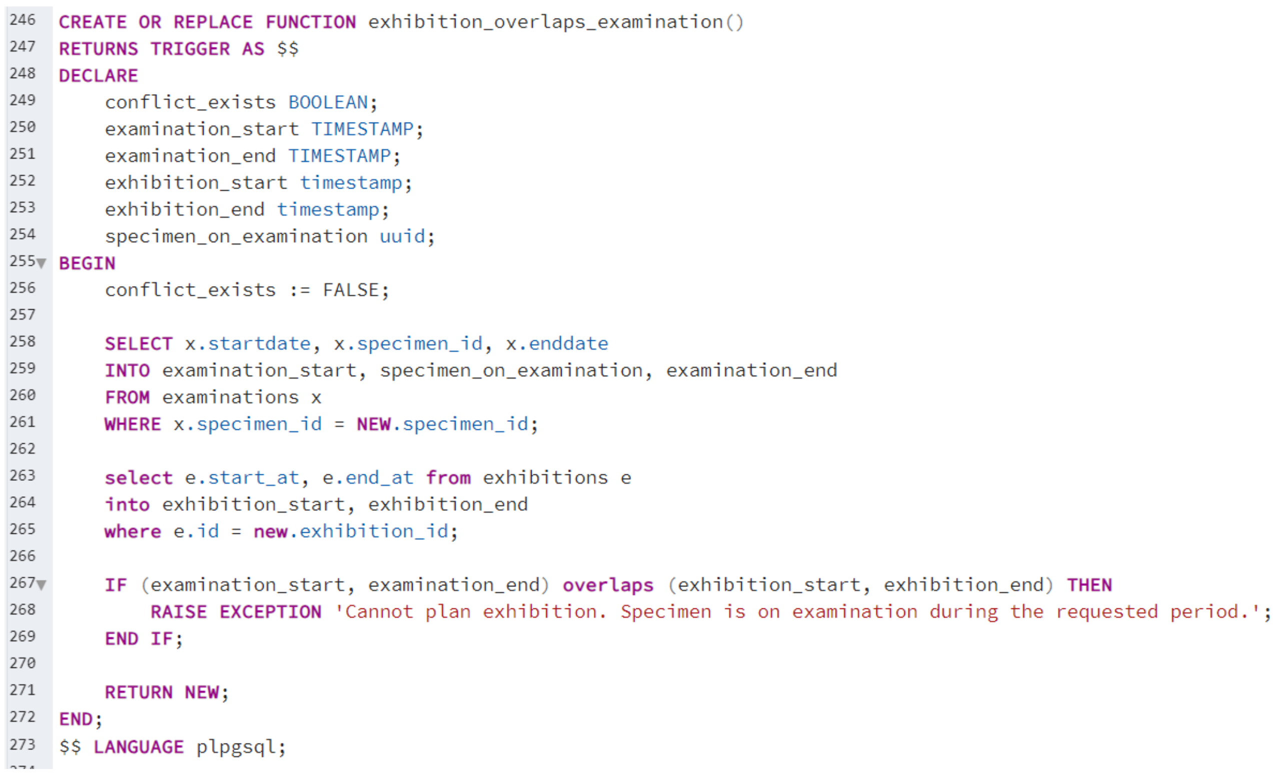
This trigger compares specimen_id that you try to paste and the one that is used in the loan. If the closed_date is null (it means that the loan is currently on progressed) it collects the start date of the loan and end date of the exhibition and raises error if I try to insert specimen into an exhibition that ends after the start of the loan. For example I have a painting 'Girl with a Pearl Earring' that is promised to be loaned to another institution from the 1st of December. If I try to insert it to the exhibition that is for example from 31st of November and end on the 12th of December, the system will raise an error.



```
INSERT INTO Exhibitions_zones (exhibition_id, zone_name, specimen_id)
VALUES
    ('c906cccf-b7d3-462e-90eb-8dfc3712c9e3', 'Zone A', '4f1ee743-20f4-41e7-9c5d-8f03a6681d1c');
```

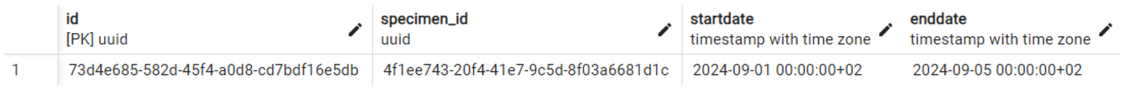
```
ERROR:  Cannot plan exhibition. Specimen is on loan during the requested period.
CONTEXT:  PL/pgSQL function exhibition_overlaps_loan() line 29 at RAISE

SQL state: P0001
```

2. The specimen is on examination



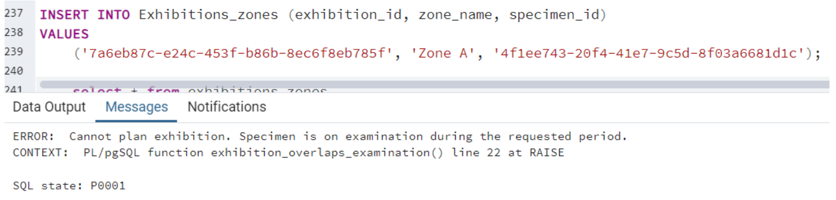
```
CREATE OR REPLACE FUNCTION exhibition_overlaps_examination()
RETURNS TRIGGER AS $$
DECLARE
    conflict_exists BOOLEAN;
    examination_start TIMESTAMP;
    examination_end TIMESTAMP;
    exhibition_start timestamp;
    exhibition_end timestamp;
    specimen_on_examination uuid;
BEGIN
    conflict_exists := FALSE;

    SELECT x.startdate, x.specimen_id, x.enddate
    INTO examination_start, specimen_on_examination, examination_end
    FROM examinations x
    WHERE x.specimen_id = NEW.specimen_id;

    select e.start_at, e.end_at from exhibitions e
    into exhibition_start, exhibition_end
    where e.id = new.exhibition_id;

    IF (examination_start, examination_end) overlaps (exhibition_start, exhibition_end) THEN
        RAISE EXCEPTION 'Cannot plan exhibition. Specimen is on examination during the requested period.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

This function checks for overlapses between the dates of examinations and exhibition I plan.if such cases appear, it raises an error. For example, there is planes examination from the f1st of September to 5th of September and the user tries to insert the specimen into the exhibition that starts on the 2nd of September. The system will not allow it.



	id [PK] uuid	specimen_id uuid	startdate timestamp with time zone	enddate timestamp with time zone
1	73d4e685-582d-45f4-a0d8-cd7bdf16e5db	4f1ee743-20f4-41e7-9c5d-8f03a6681d1c	2024-09-01 00:00:00+02	2024-09-05 00:00:00+02

Results:



```
INSERT INTO Exhibitions_zones (exhibition_id, zone_name, specimen_id)
VALUES
    ('7a6eb87c-e24c-453f-b86b-8ec6f8eb785f', 'Zone A', '4f1ee743-20f4-41e7-9c5d-8f03a6681d1c');

    select * from exhibitions_zones
```

Data Output   Messages   Notifications

```
ERROR:  Cannot plan exhibition. Specimen is on examination during the requested period.
CONTEXT:  PL/pgSQL function exhibition_overlaps_examination() line 22 at RAISE

SQL state: P0001
```

ADDITIONAL TRIGGER (EDITED_AT)

```
CREATE OR REPLACE FUNCTION update_edited_at()
RETURNS TRIGGER AS $$
BEGIN
    NEW.edited_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_edited_at
BEFORE UPDATE ON exhibitions_zones
FOR EACH ROW
EXECUTE FUNCTION update_edited_at();
```

This mini trigger is used to automatically set the edit date in exhibition_zones table when updating the data(for example swapping the specimens between zones)

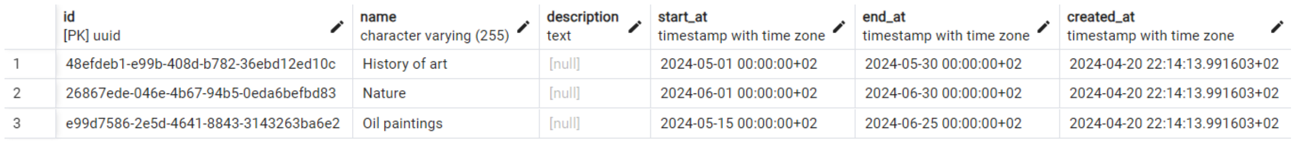
LOANS OVERLAPPING WITH THE EXHIBITION/EXAMINATION/LOAN

In this section the opposite situation is describe. Imagine you have planned an exhibition where you expose a specimen and now you try to plan a deal.

```
BEGIN
    conflict_exists := FALSE;

    SELECT EXISTS (
        SELECT 1
        FROM Specimen s
        JOIN Exhibitions_Zones z ON s.id = z.specimen_id
        JOIN Exhibitions e ON z.exhibition_id = e.id
        WHERE z.specimen_id = NEW.specimen_id
        AND NEW.started_at <= e.end_at
        LIMIT 1
    ) INTO conflict_exists;

    IF conflict_exists THEN
        RAISE EXCEPTION 'Cannot plan loan. Specimen is part of an exhibition during the requested loan period.';
    END IF;
```

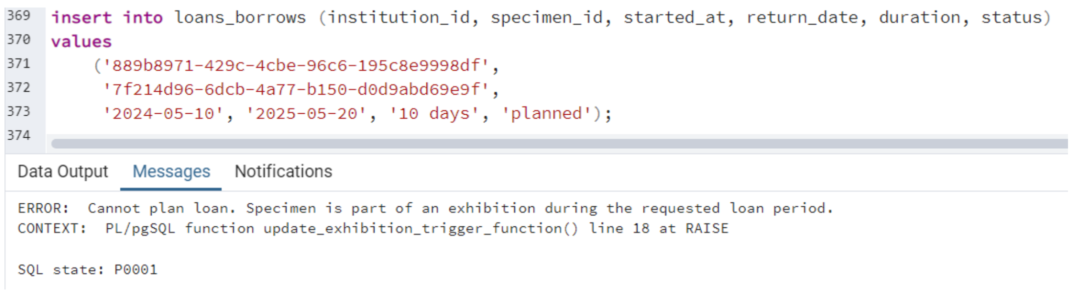
1. The specimen is on exhibition

   This function checks that the deal starts after the end of the exhibition the inserted specimen is a part of. Let's take a closer look on an example of 3rd exhibition.



	id [PK] uuid	name character varying (255)	description text	start_at timestamp with time zone	end_at timestamp with time zone	created_at timestamp with time zone
1	48efdeb1-e99b-408d-b782-36ebd12ed10c	History of art	[null]	2024-05-01 00:00:00+02	2024-05-30 00:00:00+02	2024-04-20 22:14:13.991603+02
2	26867ede-046e-4b67-94b5-0eda6befbd83	Nature	[null]	2024-06-01 00:00:00+02	2024-06-30 00:00:00+02	2024-04-20 22:14:13.991603+02
3	e99d7586-2e5d-4641-8843-3143263ba6e2	Oil paintings	[null]	2024-05-15 00:00:00+02	2024-06-25 00:00:00+02	2024-04-20 22:14:13.991603+02

   The exhibition is already planed and starts on the 15th of May and ends on the 25th of June. If I will try to plan a deal for example from the 10th of May to the 20th of May the system will call exception.

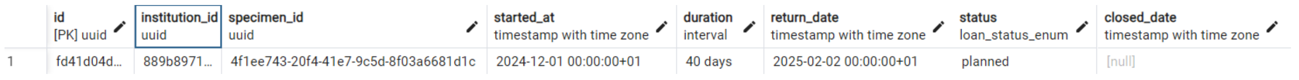


```
insert into loans_borrows (institution_id, specimen_id, started_at, return_date, duration, status)
values
    ('889b8971-429c-4cbe-96c6-195c8e9998df',
     '7f214d96-6dcb-4a77-b150-d0d9abd69e9f',
     '2024-05-10', '2025-05-20', '10 days', 'planned');
```

```
ERROR:  Cannot plan loan. Specimen is part of an exhibition during the requested loan period.
CONTEXT:  PL/pgSQL function update_exhibition_trigger_function() line 18 at RAISE

SQL state: P0001
```

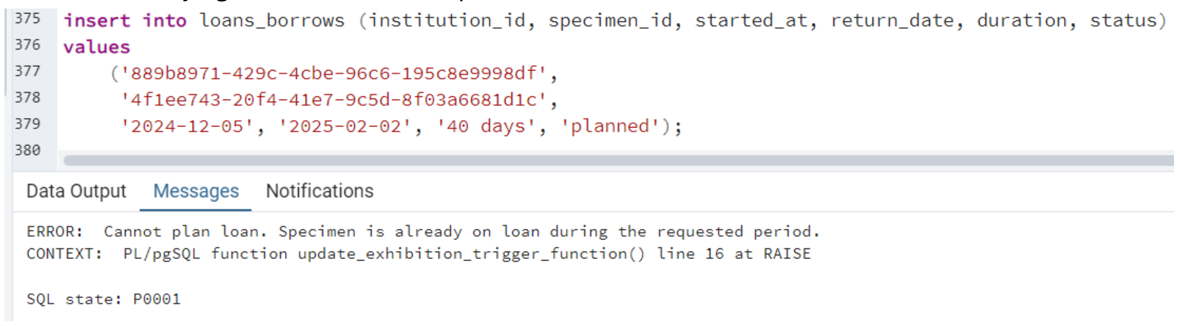
2. The specimen is a part of another deal

   Moreover, the user cannot plan a deal with the specimen if this specimen is already assigned to the open loans or borrow. The following trigger goes through the list of existing loans and compares the specimen_id to the one being inserted. If the deal is still opened(closed date = null), user is not allowed to plan another loan.



	id [PK] uuid	institution_id uuid	specimen_id uuid	started_at timestamp with time zone	duration interval	return_date timestamp with time zone	status loan_status_enum	closed_date timestamp with time zone
1	fd41d04d...	889b8971...	4f1ee743-20f4-41e7-9c5d-8f03a6681d1c	2024-12-01 00:00:00+01	40 days	2025-02-02 00:00:00+01	planned	[null]

The results of trying to insert the same specimen:



```
insert into loans_borrows (institution_id, specimen_id, started_at, return_date, duration, status)
values
    ('889b8971-429c-4cbe-96c6-195c8e9998df',
     '4f1ee743-20f4-41e7-9c5d-8f03a6681d1c',
     '2024-12-05', '2025-02-02', '40 days', 'planned');
```

```
ERROR:  Cannot plan loan. Specimen is already on loan during the requested period.
CONTEXT:  PL/pgSQL function update_exhibition_trigger_function() line 16 at RAISE

SQL state: P0001
```

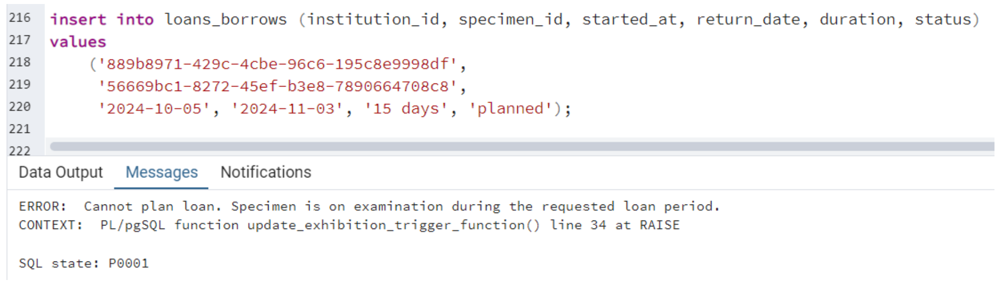
3. The specimen is on examination
   Just like the firs case described in this subsection, there is an implemented trigger to check is the specimen is on examination during the requested loans period.

```
 ---------------------------------------------------------------------
  SELECT EXISTS (
     SELECT 1
     FROM Specimen s
     JOIN Examinations x ON x.specimen_id = s.id
     WHERE x.specimen_id = NEW.specimen_id
     AND NEW.started_at <= x.enddate
     LIMIT 1
 ) INTO conflict_exists;

 -- If conflict exists, raise an error
 IF conflict_exists THEN
     RAISE EXCEPTION 'Cannot plan loan. Specimen is on examination during the requested loan period.';
 END IF;
```

This trigger is used to compare the dates of start of the deal and the end date od examination. If the last one is less that the start of the loan, the insertion as successful. Otherwise, an exception is being raised.
For example I have a specimen that has assignes examination on the period from the 4th of October to the 20th of October

```
insert into examinations (specimen_id, startdate, enddate)
values
    ('56669bc1-8272-45ef-b3e8-7890664708c8',
     '2024-10-04', '2024-10-20');
```

When I try to insert the same specimen to the loans_borrows table, the exception is raised:



```
insert into loans_borrows (institution_id, specimen_id, started_at, return_date, duration, status)
values
    ('889b8971-429c-4cbe-96c6-195c8e9998df',
     '56669bc1-8272-45ef-b3e8-7890664708c8',
     '2024-10-05', '2024-11-03', '15 days', 'planned');
```

```
ERROR:  Cannot plan loan. Specimen is on examination during the requested loan period.
CONTEXT:  PL/pgSQL function update_exhibition_trigger_function() line 34 at RAISE

SQL state: P0001
```