# Робота з Базою даних

## Тексти генераторів

В розробленій базі даних, генератори використовуються для всіх табличок. Вони використовуються для створення унікального ідентифікатора, який є первинним ключом таблички. Далі наведений текст скрипта, що задає поле з генератором:

id int GENERATED ALWAYS AS IDENTITY

У всіх табличках поле для ідентифікатора називається “id”. Тому для інших табличок, генератор створюється аналогічно.

## Збережені процедури та функцій

Відповідно до поставлених вимог та аналізу предметного середовища було розроблено наступні 11 функцій.

### Функція ActualCarKilometrage

Данна функція приймає як параметр ідентифікатор автомобіля та повертає поточний пробіг автомобіля. Результат виконання наведений на рисунку 7.1.

CREATE OR REPLACE FUNCTION ActualCarKilometrage(carID IN integer)

RETURNS int

AS $$

DECLARE

actualKilometrage int;

BEGIN

SELECT kilometrage

FROM inspections WHERE car\_id = carID

ORDER BY inspection\_date DESC LIMIT 1

INTO actualKilometrage;

RETURN actualKilometrage;

END;

$$ LANGUAGE plpgsql;

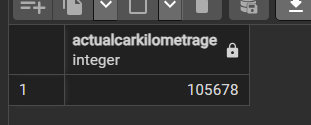


Рисунок 7.1 – Результат виконання ActualCarKilometrage для авто з ідентифікатором 15

### Функція GetLastCarInspectionDate

Функція GetLastCarInspectionDate повертає дату останнього технічного огляду автомобіля, ідентифікатор автомобіля передається як параметр. Результат виконання наведений на рисунку 7.2.

CREATE OR REPLACE FUNCTION GetLastCarInspectionDate(carID IN integer)

RETURNS date

AS $$

DECLARE

lastInspectionDate date;

BEGIN

SELECT inspection\_date FROM inspections

WHERE car\_id = carID ORDER BY inspection\_date DESC

LIMIT 1 INTO lastInspectionDate;

RETURN lastInspectionDate;

END;

$$ LANGUAGE plpgsql;

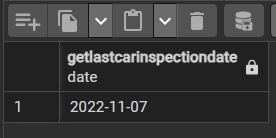


Рисунок 7.2 - Результат виконання GetLastCarInspectionDate для авто з ідентифікатором 12

### Функція GetCarsThatNeedInspection

Розроблена функція повертає набір автомобілів, які не проходили технічний огляд протягом заданого проміжку часу. Функція може бути корисна менеджеру підрозділу, щоб отримати інформацію про автомобілі, які потребують огляду. Результат виконання наведений на рисунку 7.3.

CREATE OR REPLACE FUNCTION GetCarsThatNeedInspection(timeSpan IN interval)

RETURNS TABLE(

car\_name varchar(50),

license\_plate varchar(8),

time\_since\_last\_inspection interval

)

AS $$

DECLARE

carRecord record;

last\_inspection\_date timestamp;

BEGIN

for carRecord in (SELECT id,cars.car\_name, cars.license\_plate FROM cars) LOOP

last\_inspection\_date := GetLastCarInspectionDate(carRecord.id);

time\_since\_last\_inspection := age(CURRENT\_DATE, last\_inspection\_date);

if time\_since\_last\_inspection >= timeSpan THEN

car\_name := carRecord.car\_name;

license\_plate := carRecord.license\_plate;

RETURN NEXT;

END IF;

END LOOP;

END;

$$ LANGUAGE plpgsql;

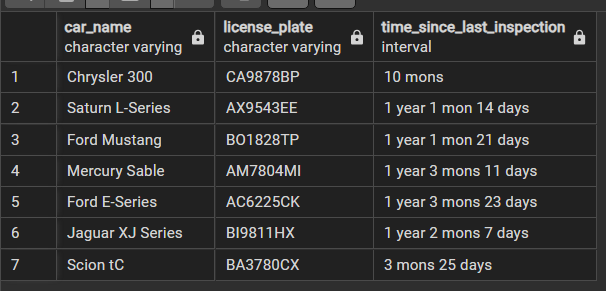


Рисунок 7.3 – Результат виконання ActualCarKilometrage

На рисунку 7.3 виведені всі автомобілі, що не мали технічного огляду більше ніж трьох місяців.

### Функція GetTimeForEquipmentReservation

Дана функція повертає вільні інтервали часу в які обладнання не використовується. Вона приймає як параметри дату, для якої обраховується час, час початку робочого дня, час кінця робочого дня та ідентифікатор обладнання. Результат виконання наведений на рисунку 7.4.

CREATE OR REPLACE FUNCTION GetTimeForEquipmentReservation

(

workDayStart time,

workDayEnd time,

reservationDate date,

equipmentID int

)

RETURNS TABLE(

startTime time,

endTime time

)

AS $$

DECLARE

reservation record;

previousTime time;

BEGIN

previousTime := workDayStart;

for reservation in (

SELECT CAST(equipment\_schedule.startTime AS time), CAST(equipment\_schedule.endTime AS time)

FROM equipment\_schedule WHERE CAST(equipment\_schedule.startTime AS date) = reservationDate

AND equipment\_id = equipmentID) LOOP

if reservation.startTime::time != previousTime THEN

startTime := previousTime;

endTime := reservation.startTime;

RETURN NEXT;

END IF;

previousTime := reservation.endTime;

END LOOP;

if workDayEnd != previousTime THEN

startTime := previousTime;

endTime := workDayEnd;

RETURN NEXT;

END IF;

END;

$$ LANGUAGE plpgsql;

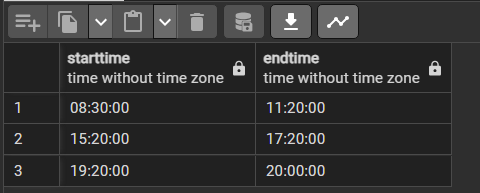


Рисунок 7.4 - Результат виконання GetTimeForEquipmentReservation

На рисунку 7.4 зображені час, протягом якого обладнання з ідентифікатором 5 вільне на 3 вересня 2023 року, якщо робочий день від 8:30 до 20:00.

### Функція GetEmployeeWorkHours

Дана функція обчислює скільки годин пропрацював працівник в заданий термін. В якості параметрів вона приймає ідентифікатор працівника, та період за який рахуються години. Результат виконання наведений на рисунку 7.6.

CREATE OR REPLACE FUNCTION GetEmployeeWorkHours

(

fromDate date,

toDate date,

employeeID int

)

RETURNS int

AS $$

DECLARE

workhours int;

BEGIN

SELECT EXTRACT(hour FROM SUM(endTime-startTime)) as hours FROM employees

JOIN work\_schedule ON work\_schedule.employee\_id = employees.id

WHERE CAST(startTime AS date) BETWEEN fromDate AND toDate

AND employee\_id = employeeID

INTO workhours;

RETURN workhours;

END;

$$ LANGUAGE plpgsql;

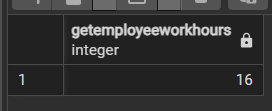


Рисунок 7.6 – Результат виконання GetEmployeeWorkHours

На рисунку 7.6 виведено скільки годин пропрацював працівник з ідентифікатором 15 з 1 по 22 грудня.

### Функція CalculateEmpolyeesWorkLoad

Функція CalculateEmpolyeesWorkLoad обчислює завантаженість робітників у відсотках. Як параметри вона приймає час початку та кінця робочого дня та часовий інтервал для якого робиться обрахунок. Результат виконання наведений на рисунку 7.7.

CREATE OR REPLACE FUNCTION CalculateEmpolyeesWorkLoad

(

workDayStart time,

workDayEnd time,

fromDate date,

toDate date

)

RETURNS TABLE(

fullname varchar(100),

workload varchar(5)

)

AS $$

DECLARE

scheduleRecord record;

maxHours decimal;

BEGIN

maxHours:= EXTRACT(hour FROM (toDate - fromDate) \* (workDayEnd - workDayStart))/2;

for scheduleRecord in (SELECT employees.fullname, SUM(endTime-startTime) as hours FROM employees JOIN work\_schedule ON work\_schedule.employee\_id = employees.id

WHERE CAST(startTime AS date) BETWEEN fromDate AND toDate

GROUP BY employees.fullname

) LOOP

fullname := scheduleRecord.fullname;

workload := CAST(ROUND((EXTRACT(hour FROM scheduleRecord.hours) / maxHours)\*100, 1) AS varchar(5)) || '%';

RETURN NEXT;

END LOOP;

END;

$$ LANGUAGE plpgsql;

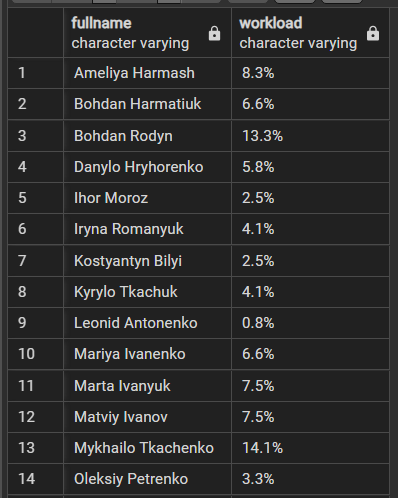


Рисунок 7.7 – Результат виконання CalculateEmpolyeesWorkLoad

На рисунку 7.7 зображена завантаженість робітників з 1 по 22 грудня, якщо робочий день з 8:30 до 20:00.

### Функція CountCarsByMake

Дана функція рахує всі автомобілі вказаної марки. Результат виконання наведений на рисунку 7.8.

CREATE OR REPLACE FUNCTION CountCarsByMake(brand IN varchar(25))

RETURNS int

AS $$

DECLARE

resultCount int;

BEGIN

SELECT COUNT(\*) FROM (SELECT car\_name FROM cars

WHERE car\_name like brand||'%') INTO resultCount;

RETURN resultCount;

END;

$$ LANGUAGE plpgsql;

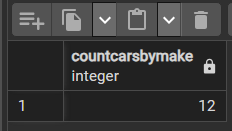


Рисунок 7.8 – Результат виконання CountCarsByMake

На рисунку 7.8 наведена кількість автомобілів марки “ Фольксваген”

### Процедура AddSpecificDetailToRepair

Розроблена функція додає до списку використаних деталей певного ремонту деталь, якої немає в базі. Як аргументи функція приймає назву деталі, її ціну, кількість деталей та ідентифікатор ремонту.

Щоб переконатись, що процедура працює, виконаєм виконаємо наступну команду:

CALL AddSpecificDetailToRepair(8,'Dynamic TurboBoost Module', 50000,1);

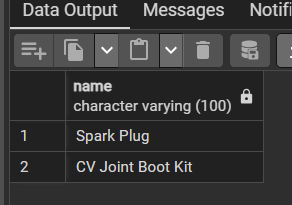


Рисунок 7.9 – Список замінених деталей до виконання процедури

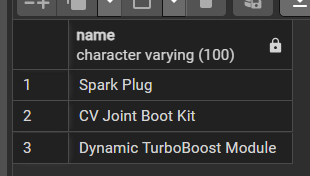


Рисунок 7.10 – Список замінених деталей після виконання процедури

### Функція GetTimeForEmployeeReservation

Розроблена функція повертає вільні інтервали часу в які працівник не працює над ремонтом. Вона приймає як параметри дату, для якої обраховується час, час початку робочого дня, час кінця робочого дня та ідентифікатор працівника. Результат виконання наведений на рисунку 7.5.

CREATE OR REPLACE FUNCTION GetTimeForEmployeeReservation

(

workDayStart time,

workDayEnd time,

reservationDate date,

employeeID int

)

RETURNS TABLE(

startTime time,

endTime time

)

AS $$

DECLARE

reservation record;

previousTime time;

BEGIN

previousTime := workDayStart;

for reservation in (

SELECT CAST(work\_schedule.startTime AS time), CAST(work\_schedule.endTime AS time)

FROM work\_schedule WHERE CAST(work\_schedule.startTime AS date) = reservationDate

AND employee\_id = employeeID) LOOP

if reservation.startTime::time != previousTime THEN

startTime := previousTime;

endTime := reservation.startTime;

RETURN NEXT;

END IF;

previousTime := reservation.endTime;

END LOOP;

if workDayEnd != previousTime THEN

startTime := previousTime;

endTime := workDayEnd;

RETURN NEXT;

END IF;

END;

$$ LANGUAGE plpgsql;

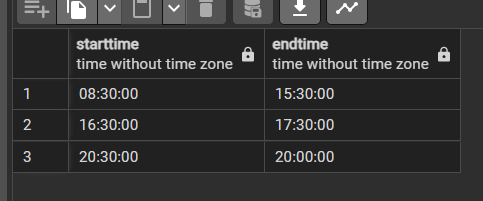


Рисунок 7.5 - Результат виконання GetTimeForEmployeeReservation

На рисунку 7.5 зображені час, протягом якого працівник з ідентифікатором 5 вільний на 13 листопада 2023 року, якщо робочий день від 8:30 до 20:00.

### Функція MakeReportAboutCar

Дана функція виводить всю історію ремонтів, обслуговування та замінених деталей вказаного автомобіля. Результат виконання наведений на рисунку 7.11.

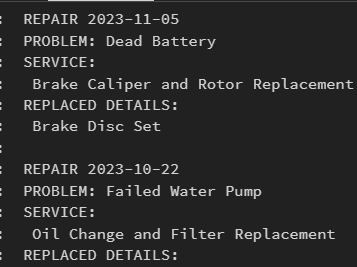


Рисунок 7.11 – Результат виконання MakeReportAboutCar

На рисунку 7.11 наведена повна історія ремонтів, замінених деталей та проведеного обслуговування для автомобіля з ідентифікатором 28.

### Функція CanReservateEquipment

Дана функція перевіряє чи можна забронювати обладнання на вказаний час. Вона перевіряє чи не пересікається вказаний часовий проміжок з іншим бронюванням. Як параметри приймає ідентифікатор обладнання, час початку бронювання та час кінця. Функція використовується в роботі тригерів reservateEquipment та UpdateReservationEquipment, результат виконання функції наведений в пункті …

CREATE OR REPLACE FUNCTION CanReservateEquipment(newStartTime IN timestamp, newEndTime IN timestamp, equipmentID in integer)

RETURNS bool

AS $$

DECLARE

reservationDate date;

reservation record;

canReservate bool := true;

BEGIN

reservationDate := CAST(newStartTime AS date);

for reservation in (

SELECT equipment\_schedule.startTime, equipment\_schedule.endTime FROM equipment\_schedule

WHERE CAST(startTime AS date) = reservationDate AND equipment\_id = equipmentID

) LOOP

if (reservation.startTime, reservation.endTime) OVERLAPS (newStartTime, newEndTime) THEN

canReservate := false;

END IF;

END LOOP;

RETURN canReservate;

END;

$$ LANGUAGE plpgsql;

### Функція CanReservateEmployee

Дана функція перевіряє чи можна додати запис в розклад працівника на вказаний час. Вона перевіряє чи не пересікається вказаний часовий проміжок з іншими записами в його розкладі. Як параметри приймає ідентифікатор працівника, час початку роботи над ремонтом та час кінця. Функція використовується в роботі тригерів reservateEmployee та UpdateReservationEmployee , результат виконання функції наведений в пункті …

## Тригери

### Тригер checkRepair

Тригер спрацьовує перед доданням до таблиці “repairs” та відповідає за збереження цілісності даних в ній. Його призначення перевіряти виконуваність бізнес-правила “В одного автомобіля не може відбуватись два ремонти одночасно”. Результати спрацювання тригера наведені на рисунку 7.12.

CREATE OR REPLACE FUNCTION checkRepair()

RETURNS trigger

AS $$

DECLARE

carID integer;

BEGIN

carID := NEW.car\_id;

IF EXISTS( SELECT \* FROM repairs WHERE car\_id = carID AND endDate IS NULL) THEN

RETURN NULL;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER checkRepair

BEFORE INSERT ON repairs

FOR EACH ROW EXECUTE FUNCTION checkRepair();

Щоб перевірити роботу тригера, спробуємо зареєструвати ремонт для машини, яка вже ремонтується. Для цього виконаємо наступну команду:

INSERT INTO repairs(problem, startdate, car\_id) VALUES('Engine overheating', '2023-12-05', 113);

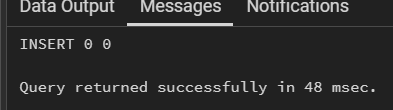


Рисунок 7.12 – Результат спрацювання тригера checkRepair

### Тригер deleteCar

Розроблений тригер спрацьовує перед видаленням з таблиці “cars”. Він виконує функцію каскадного видалення, при видалення автомобіля з бази спочатку видаляється вся інформація про його ремонти та огляди, а потім вже сама інформація про автомобіль.

CREATE OR REPLACE FUNCTION deleteCar()

RETURNS trigger

AS $$

BEGIN

DELETE FROM repairs WHERE car\_id = OLD.id;

DELETE FROM inspections WHERE car\_id = OLD.id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER deleteCar

BEFORE DELETE On cars

FOR EACH ROW EXECUTE FUNCTION deleteCar();

Щоб перевірити, чи спрацьовує тригер, видалимо автомобіль з ідентифікатором 15.

DELETE FROM cars WHERE id = 15;

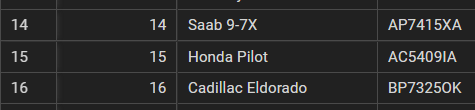


Рисунок 7.13 – таблиця “cars” до видалення

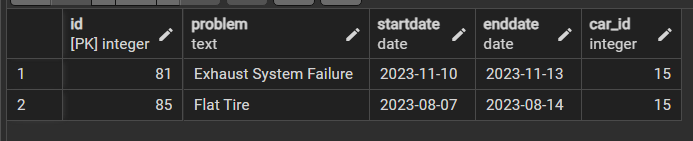


Рисунок 7.14 – Ремонти автомобіля до видалення

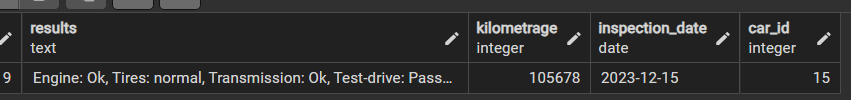


Рисунок 7.15 – Огляди автомобіля до видалення

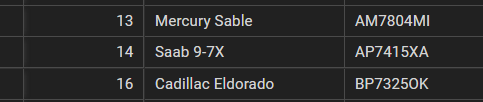


Рисунок 7.16 – таблиця “cars” після видалення

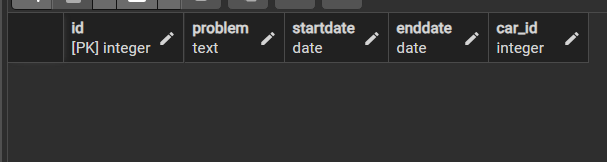


Рисунок 7.17 – Ремонти автомобіля після видалення

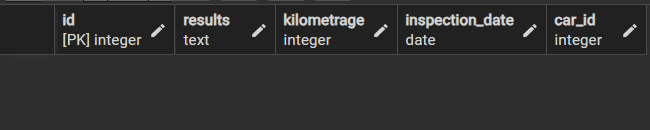


Рисунок 7.18 – Технічні огляди автомобіля після видалення

Як бачимо із рисунків 7.13 – 7.18 після видалення автомобіля інформація про його огляди та ремонти теж видалилась.

### Тригер checkLicensePlateBeforeAddition

Даний тригер виконуєтся перед додаванням в таблицю “cars” та контролює, щоб номерні знаки були коректного формату. Результат спрацювання тригера наведений на рисунку 7.19.

CREATE OR REPLACE FUNCTION checkLicensePlate() RETURNS trigger

AS $$

DECLARE

regionCode char(2);

series char(2);

numbers int;

BEGIN

regionCode := substring(new.license\_plate, 1,2);

series := substring(new.license\_plate, 7, 2);

BEGIN

numbers := substring(new.license\_plate, 3,4)::int;

EXCEPTION WHEN invalid\_text\_representation THEN RETURN NULL;

END;

IF regionCode IN ('AA', 'AB', 'AC', 'AE', 'AH', 'AI' ,'AM', 'AO', 'AP', 'АT', 'AX', 'BA', 'BB', 'BC', 'BE', 'BH', 'BI', 'BM', 'BO', 'BP', 'BT', 'BX',

'CA', 'CB', 'CE') THEN

IF series ~ '[A-Z]{2}' THEN RETURN NEW;

END IF;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER checkLicensePlateBeforeAddition

BEFORE INSERT ON cars

FOR EACH ROW EXECUTE FUNCTION checkLicensePlate();

Спробуємо додати автомобіль з неправильними номерними знаками:

INSERT INTO cars(car\_name, license\_plate, year\_of\_manufacture)

VALUES ('Volkswagen Passat', 'GF54d5GD', 2009);

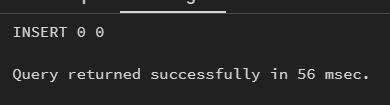


Рисунок 7.19 – Результат спрацювання тригера checkLicensePlateBeforeAddition

Як бачимо з рисунку 7.19 не вдалося додати автомобіль з некоректними номерними знаками.

### Тригер checkLicensePlateBeforeUpdating

Даний тригер аналогічний попередньому, але перевіряє умову коректності знаків перед оновленням запису. Результат спрацювання тригера наведений на рисунку 7.20.

CREATE OR REPLACE TRIGGER checkLicensePlateBeforeUpdating

BEFORE UPDATE ON cars

FOR EACH ROW EXECUTE FUNCTION checkLicensePlate();

Для того, щоб перевірити його роботу спробуємо оновити номерні знаки у автомобіля з ідентифікатором 15 на неправильні.

UPDATE cars SET license\_plate = '34АААА45' WHERE id = 15;

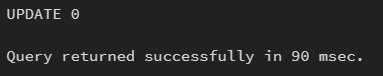


Рисунок 7.20 – Результат спрацювання тригера checkLicensePlateBeforeAddition

Як бачимо з рисунку 7.20 не вдалося оновити номерні знаки на некоректні.

### Тригер checkKilometrageBeforeUpdating

Тригер checkKilometrageBeforeUpdating спрацьовує перед оновленням пробігу автомобіля в таблиці “inspections”. Його основне призначення перевіряти, чи виконується бізнес-правило “Пробіг зафіксований на одному із оглядів авто повинен бути більший або рівний пробігу з попереднього огляду”

CREATE OR REPLACE FUNCTION checkKilometrage()

RETURNS trigger

AS $$

DECLARE

lowerBound int;

upperBound int;

BEGIN

SELECT kilometrage FROM inspections

WHERE car\_id = new.car\_id AND inspection\_date < new.inspection\_date

ORDER BY inspection\_date DESC LIMIT 1 INTO lowerBound;

SELECT kilometrage FROM inspections

WHERE car\_id = new.car\_id AND inspection\_date > new.inspection\_date

ORDER BY inspection\_date DESC LIMIT 1 INTO upperBound;

IF lowerBound IS NULL THEN

IF new.kilometrage <= upperBound THEN

RETURN new;

END IF;

ELSEIF upperBound IS NULL THEN

IF new.kilometrage >= lowerBound THEN

RETURN new;

END IF;

ELSE

IF new.kilometrage >= lowerBound AND new.kilometrage <= upperBound THEN

RETURN new;

END IF;

END IF;

RETURN null;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER checkKilometrageBeforeUpdating

BEFORE UPDATE ON inspections

FOR EACH ROW EXECUTE FUNCTION checkKilometrage();

Для перевірки тригер спробуємо оновити пробіг для одного із оглядів автомобіля з id 75. На рисунку 7.21 наведені всі огляди даного автомобіля перед оновленням.

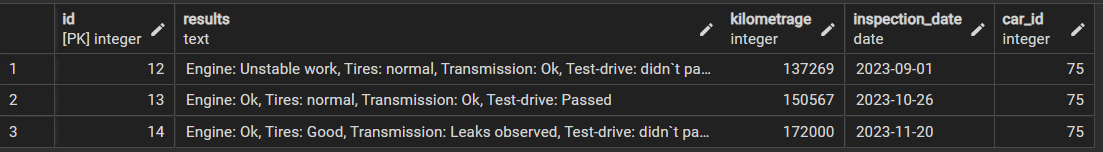


Рисунок 7.21 – Технічні огляди автомобіля з id 75

Для оновлення пробігу виконаємо наступну команду:

UPDATE inspections SET kilometrage = 200000 WHERE id = 13;

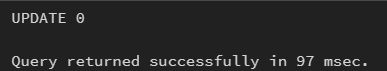


Рисунок 7.22 – Результат спрацювання тригеру checkKilometrageBeforeUpdating

Як бачимо із рисунку 7.22 тригер спрацював і не дав оновити пробіг на більший ніж у наступному огляді.

### Тригер checkKilometrageBeforeInserting

Даний тригер аналогічний попередньому, але спрацьовує перед додаванням інформації про новий технічний огляд.

CREATE OR REPLACE TRIGGER checkKilometrageBeforeInserting

BEFORE INSERT ON inspections

FOR EACH ROW EXECUTE FUNCTION checkKilometrage();

Спробуємо додати новий технічний огляд з пробігом 100000 для автомобіля з id 75.

INSERT INTO inspections(results, kilometrage, inspection\_date, car\_id)

VALUES('Test', 100000, '2023-12-15', 75);

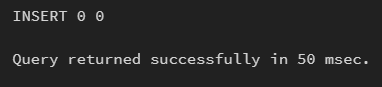


Рисунок 7.23 – Результат спрацювання тригера checkKilometrageBeforeInserting

Як бачимо із рисунку 7.23 тригер спрацював і не дав додати огляд з пробігом, що менший за попередній.

### Тригер TryReservateEquipment

Розроблений тригер спрацьовує перед додаванням нового запису до таблиці “Equipment\_schedule”. Його основне призначення – перевіряти, чи допустимий проміжок часу намагається додати користувач. Якщо час, на який користувач хоче забронювати обладнання, перетинається із іншими записами в розкладі, то такий запис не може бути доданий.

CREATE OR REPLACE FUNCTION СheckReservationTimeForEquipment ()

RETURNS trigger

AS $$

BEGIN

IF (new.startTime < new.endTime) AND CanReservateEquipment(new.startTime, new.endTime, new.equipment\_id) THEN

RETURN new;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER TryReservateEquipment

BEFORE INSERT ON equipment\_schedule

FOR EACH ROW EXECUTE FUNCTION СheckReservateTimeForEquipment ();

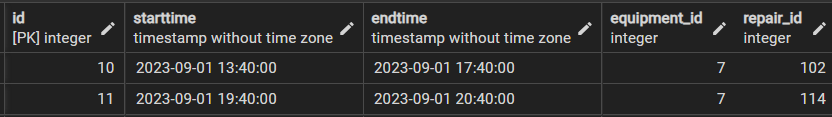


Рисунок 7.24 – Наявні бронювання для обладнання з id 7 на 1 вересня

Спробуємо додати бронювання для обладнання з id 7, що перетинається з раніше доданими бронюванням:

INSERT INTO equipment\_schedule(startTime, endTime, equipment\_id, repair\_id)

VALUES('2023-09-01 12:00', '2023-09-01 14:30', 7, 23);

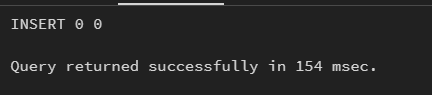


Рисунок 7.25 – Результат спрацювання тригера TryReservateEquipment

Як бачимо із рисунка 7.25, не вдалося додати бронювання з некоректним часом.

### Тригер UpdateReservationTimeForEquipment

Даний тригер аналогічний попередньому, але спрацьовує при оновлені часу бронювання обладнання.

CREATE OR REPLACE TRIGGER UpdateReservationTimeForEquipment

BEFORE UPDATE ON equipment\_schedule

FOR EACH ROW EXECUTE FUNCTION СheckReservationTimeForEquipment();

Для перевірки тригера спробуємо оновити одне з бронювань, таким чином, щоб воно перетнулося в часі з іншим.

UPDATE equipment\_schedule SET endTime = '2023-09-01 20:00' WHERE id = 10;

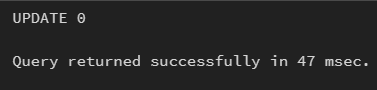


Рисунок 7.26 – Результат спрацювання тригера UpdateReservationTimeForEquipment

Як бачимо із рисунка 7.26, не вдалося оновити бронювання з некоректним часом.

### Тригер TryReservateEmployee

Даний тригер спрацьовує перед додаванням нового запису до таблиці “Work\_schedule”. Його основне призначення – перевіряти, чи допустимий проміжок часу намагається додати користувач. Якщо час, на який користувач хоче доручити ремонт працівнику, перетинається з іншими записами в його розкладу, то такий запис не може бути доданий.

CREATE OR REPLACE FUNCTION СheckReservationTimeForEmployee()

RETURNS trigger

AS $$

BEGIN

IF (new.startTime < new.endTime) AND CanReservateEmployee(new.startTime, new.endTime, new.employee\_id) THEN

RETURN new;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER TryReservateEmployee

BEFORE INSERT ON work\_schedule

FOR EACH ROW EXECUTE FUNCTION СheckReservationTimeForEmployee();

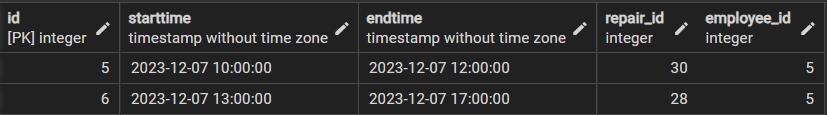


Рисунок 7.27 – Розклад роботи працівника з id 5 на 7 грудня

Щоб перевірити, чи спрацює тригер, додаймо для цього працівника запис в розклад, що перетинається в часі з іншими записами:

INSERT INTO work\_schedule(startTime, endTime, repair\_id, employee\_id)

VALUES('2023-12-07 15:00', '2023-12-07 18:00', 25, 5);

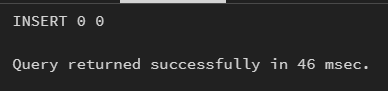


Рисунок 7.28 – Результат спрацювання тригера TryReservateEmployee

Як бачимо із рисунка 7.28, не вдалося додати запис із неправильним часом.

### Тригер UpdateReservationTimeForEmployee

Даний тригер аналогічний попередньому, але спрацьовує при оновленні запису в таблиці “work\_schedule”.

CREATE OR REPLACE TRIGGER UpdateReservationTimeForEmployee

BEFORE UPDATE ON work\_schedule

FOR EACH ROW EXECUTE FUNCTION СheckReservationTimeForEmployee();

Для перевірки спробуємо оновити час на такий, що перетинається з часом в інших записах:

UPDATE work\_schedule SET endTime = '2023-12-07 15:00' WHERE id = 5;

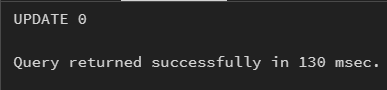


Рисунок 7.29 – Результат спрацювання тригера UpdateReservationTimeForEmployee

Як бачимо із рисунка 7.29, не вдалося оновити час запису на некоректний.