**Тема №5**

**Проект по системи за управление на база от данни за сервиз за битова техника**

**Изготвил: Стелиана Димитрова – ИС, III курс , ф.н. 4MI0700044**

1. **Обхват на модела. Дефиниране на задачата.**

Базата от данни за сервиз за битова техника ще съхранява информация за техниците и ремонтите, извършени от тях.

В сервиза се пази следната информация за всеки ремонтиран уред: уникален идентификатор, категория уред, производител, модел, име на клиент и годината на производство, ако тя е известна (валидна е всяка година след 1900).

За техниците, работещи в сервиза за битова техника, също се пази информация. Задължително се записва техния уникален идентификатор, име, ЕГН и категории уреди, които може да ремонтира.

Един техник може да извършва много ремонти на уреди. Един уред може да бъде поправян много пъти от различен техник. За всяка поправка се пази дата и цена. За дадена категория уреди може да има няколко техници в сервиза.

1. **Множества от същности и техните атрибути**

Уред – уникален идентификатор, категория (пералня, телевизор…), производител, модел, име на клиент, година на производство;

Техник – уникален идентификатор, име, ЕГН, категория уред (която може да ремонтира);

Клиент – име, телефон

1. **Домейн на атрибутите**

Уред – уникален идентификатор: низ, категория (пералня, телевизор…): низ, производител: низ до 50 символа, модел: низ до 20 символа, име на клиент: низ, година на производство: цяло положително число по-голямо от 1900;

Техник – уникален идентификатор: низ, име: низ до 100 символа, ЕГН: низ точно 10 символа , категория уред (която може да ремонтира): низ;

Клиент – име: низ до 100 символа, телефон: низ точно 10 символа;

1. **Връзки**

Един техник може да извършва много ремонти на уреди. Един уред може да бъде поправян много пъти от различни техници.

За дадена категория уреди може да има няколко техници.

Един клиент може да има много уреди. Един уред е на един клиент.

1. **Ограничения по единствена стойност, референтна цялостност и друг тип ограничения**

Уред – уникален идентификатор: еднозначно определя уреда

Техник – уникален идентификатор: еднозначно определя техника

Клиент – телефон: еднозначно определя клиента (не може да има двама човека с един и същ номер)

1. **Правила и проверки**

За уреда – производител: проверка за дължината (до 50 символа)

За уреда – модел: проверка за дължината (до 20 символа)

За уреда – година на производство: проверка за числото ( по-голямо от 1900)

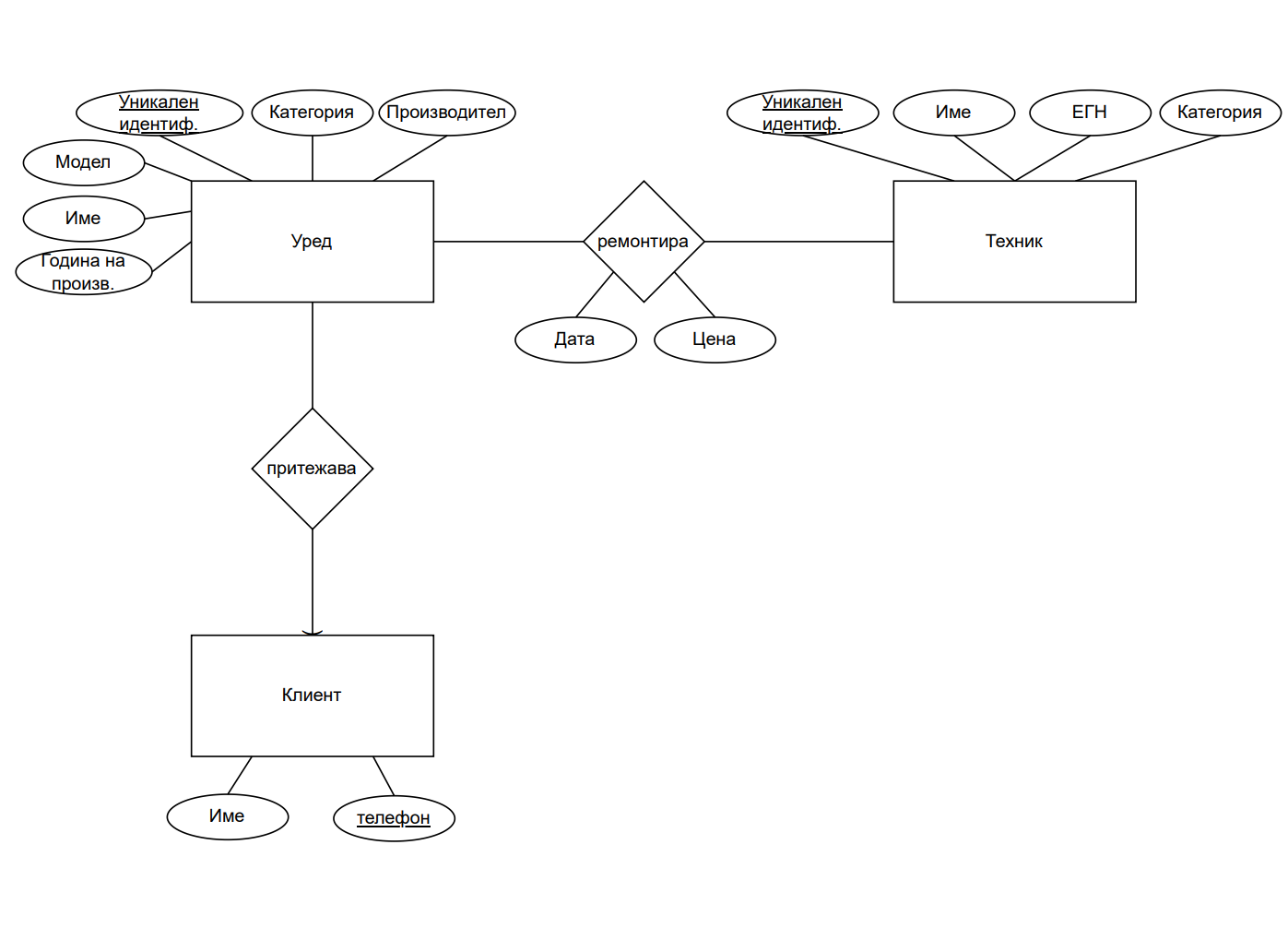
За техника – име: проверка за дължината (до 100 символа)

За техника – ЕГН: проверка за дължината (точно 10 символа)

За клиента – име: проверка за дължината (до 100 символа)

За клиента – телефон: проверка за дължината (точно 10 символа)

За всички положителни числа – проверка дали са по-големи от нула

1. **E/R модел на данни**

1. **Релационен модел на данни**

В E/R модела на данни няма isa йерархии, затова ще започнем с преобразуването на множествата от същности, след това връзката много – много и накрая връзката много – един. Завършваме с оптимизиране на връзката много – един.

* **Преобразуване на множествата от същности:**

Appliance (name, model, applianceId, applianceCategory, manufacturer, yearOfManufacture) <- Уреди

Technician (name, EGN, texnicianId, texnicianCategory) <- Техници

Customer (name, phoneNumber) <- Клиенти

* **Преобразуване на връзката много – много**

Repair (applianceId, texnicianId, date, price)

* **Преобразуване на връзката много – един**

Owns ( applianceId, phoneNumber)

* **Получаваме следния релационен модел:**

Appliance (name, model, applianceId, applianceCategory, manufacturer, yearOfManufacture)

Technician (name, EGN, texnicianId, texnicianCategory)

Customer (name, phoneNumber)

Repair (applianceId, texnicianId, date, price)

Owns ( applianceId, phoneNumber)

* **Оптимизираме връзката много – един и получаваме:**

Appliance (name, model, applianceId, applianceCategory, manufacturer, yearOfManufacture, customerPhoneNumber)

Technician (name, EGN, texnicianId, texnicianCategory)

Customer (name, phoneNumber)

Repair (applianceId, texnicianId, date, price)

Owns ( applianceId, customerPhoneNumber) – отпада - оптимизираме в Appliance

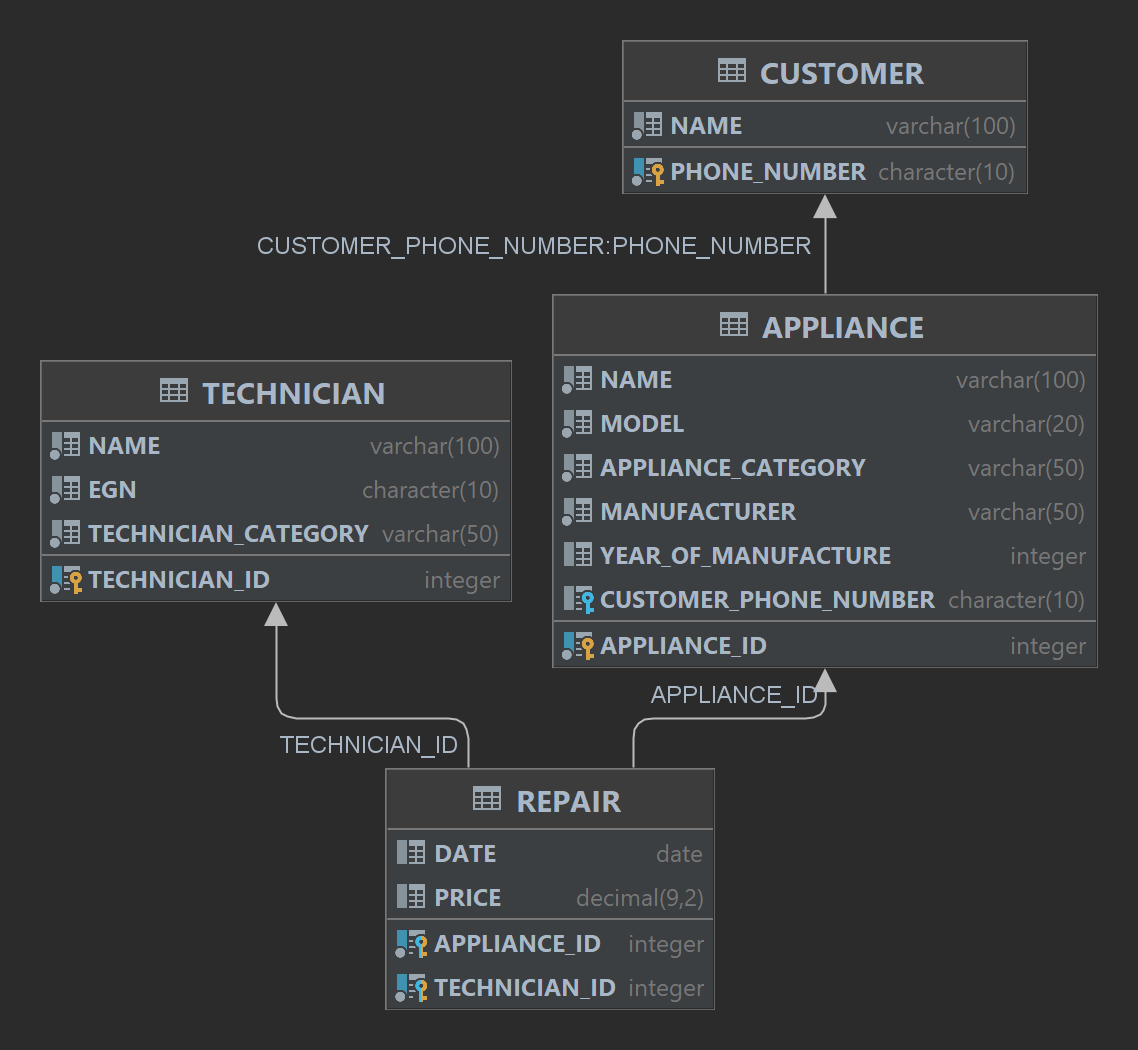
* **Окончателно за схемата на базата от данни получаваме:**

Appliance (name, model, applianceId, applianceCategory, manufacturer, yearOfManufacture, customerPhoneNumber)

Technician (name, EGN, texnicianId, texnicianCategory)

Customer (name, phoneNumber)

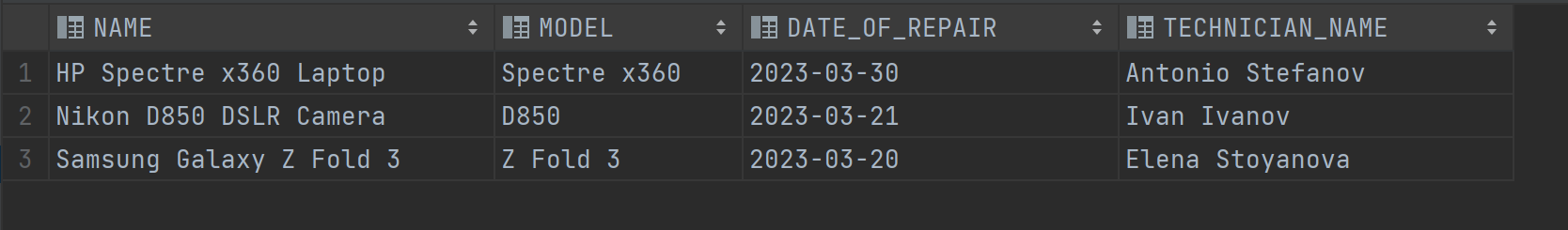
Repair (applianceId, texnicianId, date, price)

1. **Схема на базата от данни**
2. **Описание на функциите**

В проектът „Сервиз за битова техника“ са създадени две функции.

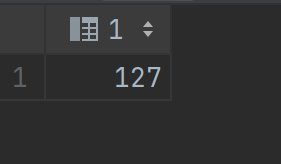
Първата е създадена с цел да извлече информация за уредите на един клиент. Тя е наречена F\_GET\_ALL\_APPLIANCES() и представя имената на уредите, техните модели, дата на ремонт и името на техника, който ги е ремонтирал. Функцията приема телефонен номер на клиента (P\_PHONE\_NUMBER) и връща таблица с четири колони: NAME (име на уреда), MODEL (модел на уреда), DATE\_OF\_REPAIR (дата на ремонт) и TECHNICIAN\_NAME (име на техниката). За да я извикаме използваме заявка към таблицата.

CREATE FUNCTION *F\_GET\_ALL\_APPLIANCES*(P\_PHONE\_NUMBER CHAR(10))  
RETURNS TABLE(NAME VARCHAR(100), MODEL VARCHAR(20), DATE\_OF\_REPAIR DATE, TECHNICIAN\_NAME VARCHAR(100))  
RETURN  
 SELECT A.NAME, A.MODEL, R.DATE, T.NAME  
 FROM APPLIANCE A, REPAIR R, TECHNICIAN T  
 WHERE A.CUSTOMER\_PHONE\_NUMBER = P\_PHONE\_NUMBER  
 AND A.APPLIANCE\_ID = R.APPLIANCE\_ID  
 AND R.TECHNICIAN\_ID = T.TECHNICIAN\_ID;  
  
SELECT \*  
FROM *TABLE*(FN4MI0700044.*F\_GET\_ALL\_APPLIANCES*('0899000112')) T;



Втората функция се казва F\_AVG\_PRICE\_FOR\_CATEGORY(). Тя приема параметър P\_CATEGORY от тип VARCHAR(50) и връща резултат от тип INT. Функцията е създадена с цел да изчисли средната цена на ремонтите за уреди от определена категория. Използва блок от BEGIN и END, в който се дефинира променлива P\_AVG\_PRICE от тип DECIMAL(9, 2), която ще съхранява средната цена. След това правим заявка SELECT AVG(PRICE) INTO P\_AVG\_PRICE, където се изчислява средната цена на ремонтите за уреди от зададената категория. Използва се подзаявка за извличане на APPLIANCE\_ID на уредите, които са от желаната категория. Накрая функцията връща средната цена P\_AVG\_PRICE като резултат.

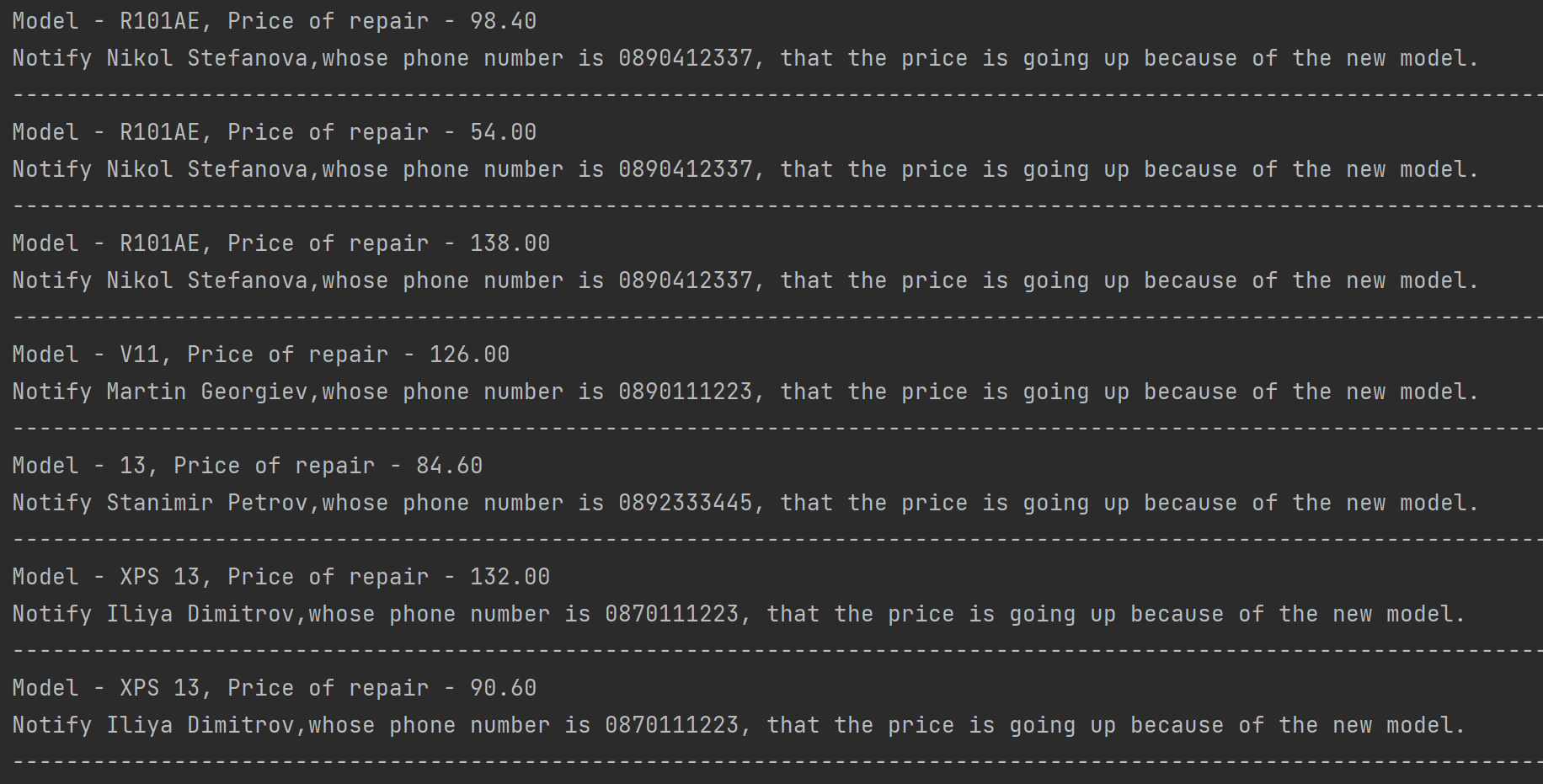
CREATE OR REPLACE FUNCTION *F\_AVG\_PRICE\_FOR\_CATEGORY* (P\_CATEGORY VARCHAR(50))  
RETURNS INT  
BEGIN  
 DECLARE P\_AVG\_PRICE DECIMAL(9, 2);  
  
 SELECT *AVG*(PRICE) INTO P\_AVG\_PRICE  
 FROM REPAIR  
 WHERE APPLIANCE\_ID IN ( SELECT APPLIANCE\_ID  
 FROM APPLIANCE  
 WHERE APPLIANCE\_CATEGORY = P\_CATEGORY);  
  
 RETURN P\_AVG\_PRICE;  
END;  
  
VALUES FN4MI0700044.*F\_AVG\_PRICE\_FOR\_CATEGORY*('TV');



1. **Описание на процедурите**

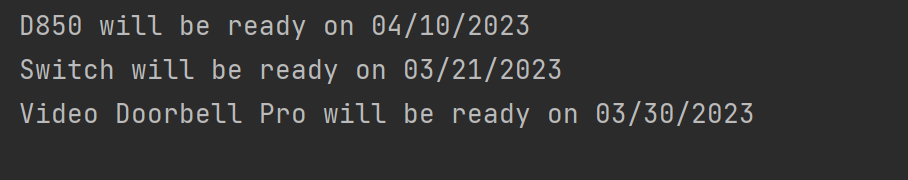
Първата ни процедура P\_NEW\_PRICE\_AFTER\_2020 е създадена за да увеличи цените на ремонтите за уреди, произведени след 2020 г., да изведе информация за тези ремонти, както и да покаже съобщение кои клиенти трябва да бъдат уведомени за увеличението на цените. Процедурата използва курсор за обхождане на резултатите от заявката и взима данните от три свързани таблици - APPLIANCE, REPAIR и CUSTOMER. Чрез FETCH извличаме първият ред от резултатите. След това се използва цикъл за обхождане на резултатите от курсора. Ако годината на производство на уреда е след 2020 г., цената на ремонта се увеличава с 20% и се извежда информация за ремонта, както и уведомление към клиента. Цикълът продължава с извличането на следващите редове, докато няма повече резултати. Накрая курсорът се затваря.

CREATE OR REPLACE PROCEDURE *P\_NEW\_PRICE\_AFTER\_2020*()  
BEGIN  
 DECLARE V\_MODEL VARCHAR(30);  
 DECLARE V\_YEAR INT;  
 DECLARE V\_PRICE DECIMAL(9,2);  
 DECLARE V\_NAME VARCHAR(50);  
 DECLARE V\_PHONE\_NUMBER CHAR(10);  
 DECLARE SQLCODE INT;  
  
 DECLARE C1 CURSOR WITH RETURN FOR SELECT A.MODEL, A.YEAR\_OF\_MANUFACTURE, R.PRICE, C.NAME, A.CUSTOMER\_PHONE\_NUMBER  
 FROM APPLIANCE A, REPAIR R, CUSTOMER C  
 WHERE A.CUSTOMER\_PHONE\_NUMBER = C.PHONE\_NUMBER  
 AND R.APPLIANCE\_ID = A.APPLIANCE\_ID;  
  
 OPEN C1;  
 FETCH C1 INTO V\_MODEL, V\_YEAR, V\_PRICE, V\_NAME, V\_PHONE\_NUMBER;  
  
 WHILE SQLCODE = 0 DO  
 IF V\_YEAR > 2020 THEN  
 SET V\_PRICE = V\_PRICE \* 1.2;  
  
 CALL DBMS\_OUTPUT.PUT\_LINE('Model - ' || V\_MODEL || ', Price of repair - ' || V\_PRICE);  
 CALL DBMS\_OUTPUT.PUT\_LINE('Notify ' || V\_NAME || ',whose phone number is ' || V\_PHONE\_NUMBER || ', that the price is going up because of the new model.');  
 CALL DBMS\_OUTPUT.PUT\_LINE('---------------------------------------------------------------------------------------------------------------------------');  
  
 END IF;  
  
 FETCH C1 INTO V\_MODEL, V\_YEAR, V\_PRICE, V\_NAME, V\_PHONE\_NUMBER;  
 END WHILE;  
  
CLOSE C1;  
END;  
  
CALL FN4MI0700044.*P\_NEW\_PRICE\_AFTER\_2020*();



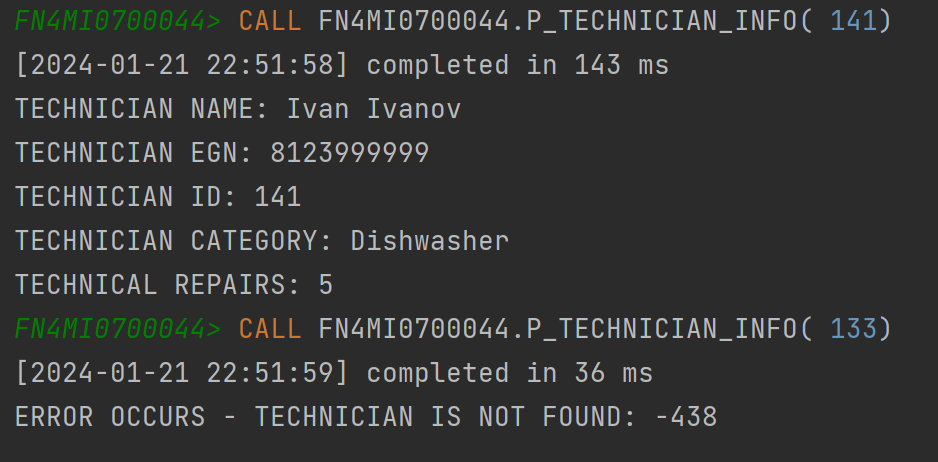
Процедурата с име P\_DAYS\_OFF приема параметри за брой дни, начална дата на отпуската, id на техник и връща новата дата, на която ще бъде готов уреда, чрез параметър с ключовата дума OUT. Процедурата използва курсор за извличане на резултатите от заявката за ремонти, които ще се извършат от определен техник след началната дата на отпуската. В цикъла се изчислява новата дата (P\_NEW\_DATE), като се добавят дните от параметъра P\_NUMBER\_OF\_DAYS. След което се извежда се информация за уреда и новата дата на вземането му.

CREATE OR REPLACE PROCEDURE *P\_DAYS\_OFF*(IN P\_NUMBER\_OF\_DAYS INT, IN P\_FROM\_DATE DATE, IN P\_TECHNICIAN\_ID INT, OUT P\_NEW\_DATE DATE)  
BEGIN  
 DECLARE V\_MODEL VARCHAR(30);  
 DECLARE V\_DATE DATE;  
  
 DECLARE SQLCODE INT;  
  
 DECLARE C1 CURSOR WITH RETURN FOR SELECT A.MODEL, R.DATE  
 FROM APPLIANCE A, REPAIR R  
 WHERE R.TECHNICIAN\_ID = P\_TECHNICIAN\_ID  
 AND R.DATE >= P\_FROM\_DATE  
 AND A.APPLIANCE\_ID = R.APPLIANCE\_ID;  
  
 OPEN C1;  
 FETCH C1 INTO V\_MODEL, V\_DATE;  
  
 WHILE SQLCODE = 0 DO  
 SET P\_NEW\_DATE = *DATE*(V\_DATE) + P\_NUMBER\_OF\_DAYS DAYS;  
  
 CALL DBMS\_OUTPUT.PUT\_LINE(V\_MODEL || ' will be ready on ' || P\_NEW\_DATE);  
  
 FETCH C1 INTO V\_MODEL, V\_DATE;  
 END WHILE;  
  
CLOSE C1;  
END;  
  
BEGIN  
 DECLARE NEW\_DATE DATE;  
 CALL FN4MI0700044.*P\_DAYS\_OFF*(20, '2023-03-01' , 141, NEW\_DATE);  
END;

****

Последната процедура с име P\_TECHNICIAN\_INFO приема id на техника и извежда информацията за него, включително име, ЕГН, id, категорията, която ремонтира, и брой на техническите ремонти, които е извършил. Процедурата включва обработка на грешка при ненамерен техник. Имаме 2 заявки – първата се използва за извличане на броя ремонти на дадения техник, а втората прави проверка дали техникът съществува. Използваме EXIT HANDLER FOR NOT\_FOUND, който се активира в случай, че техникът не е намерен (SQLSTATE '21000'). Процедурата извършва проверка дали техникът съществува и, ако не, генерира грешка чрез SIGNAL NOT\_FOUND. Ако техникът съществува, извлича се информацията за него и се извежда в конзолата. Примери за извикване на процедурата са в края на кода, където се извикват с идентификаторите 141 и 133. В първия случай техникът съществува, а във втория - не.

CREATE OR REPLACE PROCEDURE *P\_TECHNICIAN\_INFO*(IN P\_TECHNICIAN\_ID INT)  
BEGIN  
 DECLARE V\_NAME VARCHAR(100);  
 DECLARE V\_EGN CHAR(10);  
 DECLARE V\_CATEGORY VARCHAR(50);  
  
 DECLARE V\_ID\_COLLECTION INT;  
 DECLARE V\_APP\_COUNT INT;  
  
 DECLARE r\_error INT DEFAULT 0;  
 DECLARE SQLCODE INT DEFAULT 0;  
 DECLARE NOT\_FOUND CONDITION FOR SQLSTATE '21000';  
  
 DECLARE EXIT HANDLER FOR NOT\_FOUND  
 BEGIN  
 SET r\_error = SQLCODE;  
 CALL DBMS\_OUTPUT.PUT\_LINE('ERROR OCCURS - TECHNICIAN IS NOT FOUND: ' || r\_error);  
 END;  
  
 SELECT *COUNT*(\*) INTO V\_APP\_COUNT  
 FROM REPAIR R  
 WHERE R.TECHNICIAN\_ID = P\_TECHNICIAN\_ID;  
  
 SELECT *COUNT*(\*) INTO V\_ID\_COLLECTION  
 FROM TECHNICIAN  
 WHERE TECHNICIAN\_ID = P\_TECHNICIAN\_ID;  
  
 IF(V\_ID\_COLLECTION = 0) THEN  
 SIGNAL NOT\_FOUND;  
 ELSE  
 SELECT NAME, EGN, TECHNICIAN\_CATEGORY INTO V\_NAME, V\_EGN, V\_CATEGORY  
 FROM TECHNICIAN  
 WHERE TECHNICIAN\_ID = P\_TECHNICIAN\_ID;  
  
 CALL DBMS\_OUTPUT.PUT\_LINE('TECHNICIAN NAME: ' || V\_NAME);  
 CALL DBMS\_OUTPUT.PUT\_LINE('TECHNICIAN EGN: ' || V\_EGN);  
 CALL DBMS\_OUTPUT.PUT\_LINE('TECHNICIAN ID: ' || P\_TECHNICIAN\_ID);  
 CALL DBMS\_OUTPUT.PUT\_LINE('TECHNICIAN CATEGORY: ' || V\_CATEGORY);  
 CALL DBMS\_OUTPUT.PUT\_LINE('TECHNICAL REPAIRS: ' || V\_APP\_COUNT);  
 END IF;  
  
END;  
  
CALL FN4MI0700044.*P\_TECHNICIAN\_INFO*( 141);  
CALL FN4MI0700044.*P\_TECHNICIAN\_INFO*( 133);

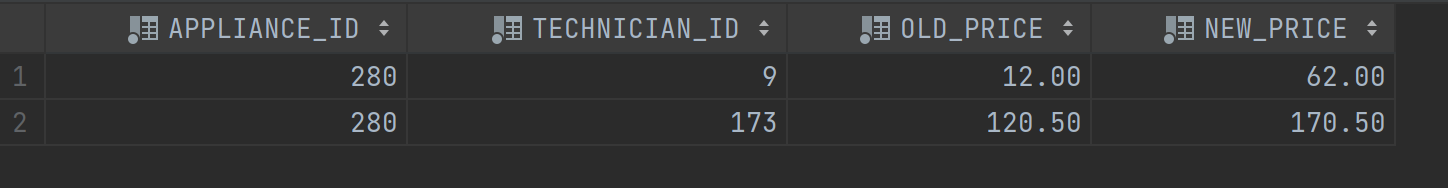
****

1. **Описание на тригерите**

В проекта са създадени 2 тригера.

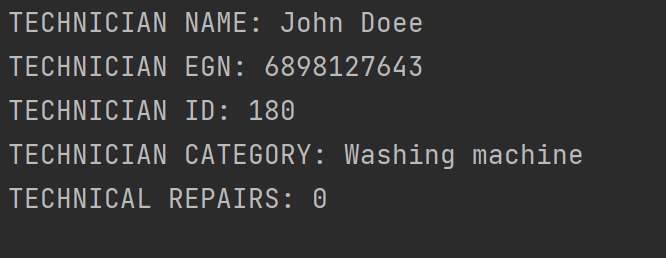
Първият е наречен TRIG\_AFTER\_UPDATE\_REPAIR\_PRICE и се изпълнява след актуализация на PRICE в таблицата REPAIR. Този тригер следи за увеличение на цената на ремонт и записва информацията в таблицата REPAIR\_PRICE\_INCREASE\_LOG (тя е създадена преди създаването на тригера). Предназначена е за съхранение на информацията за увеличенията на цените на ремонтите. Когато новата цена (N.PRICE) е по-голяма от старата цена (O.PRICE), тригерът извършва вмъкване на запис в таблицата REPAIR\_PRICE\_INCREASE\_LOG. Имаме 2 примера, в които се увеличава цената на ремонта за уред с APPLIANCE\_ID = 280 с 50 и след това се намалява цената за уред с APPLIANCE\_ID = 290 с 50. В резултат от тях виждаме, че се добавя само увеличението.

CREATE TABLE REPAIR\_PRICE\_INCREASE\_LOG (  
 APPLIANCE\_ID INT NOT NULL,  
 TECHNICIAN\_ID INT NOT NULL,  
 OLD\_PRICE DECIMAL(9, 2) NOT NULL,  
 NEW\_PRICE DECIMAL(9, 2) NOT NULL  
);  
  
CREATE OR REPLACE TRIGGER TRIG\_AFTER\_UPDATE\_REPAIR\_PRICE  
AFTER UPDATE OF PRICE ON REPAIR  
REFERENCING NEW AS N OLD AS O  
FOR EACH ROW  
WHEN (N.PRICE > O.PRICE)  
 INSERT INTO REPAIR\_PRICE\_INCREASE\_LOG(APPLIANCE\_ID, TECHNICIAN\_ID, OLD\_PRICE, NEW\_PRICE)  
 VALUES (N.APPLIANCE\_ID, N.TECHNICIAN\_ID, O.PRICE, N.PRICE);  
  
UPDATE REPAIR  
SET PRICE = PRICE + 50  
WHERE APPLIANCE\_ID = 280;  
  
UPDATE REPAIR  
SET PRICE = PRICE - 50  
WHERE APPLIANCE\_ID = 290;  
  
SELECT \* FROM REPAIR\_PRICE\_INCREASE\_LOG;



Вторият е TRIG\_AFTER\_INSERT\_TECHNICIAN и се изпълнява след вмъкване на нов запис в таблицата TECHNICIAN. За всеки нов ред (текущия) тригерът извиква процедурата P\_TECHNICIAN\_INFO, като предоставя TECHNICIAN\_ID на въведения техник като параметър. За него се извежда информация като име, ЕГН, id, категория и брой на техническите ремонти.

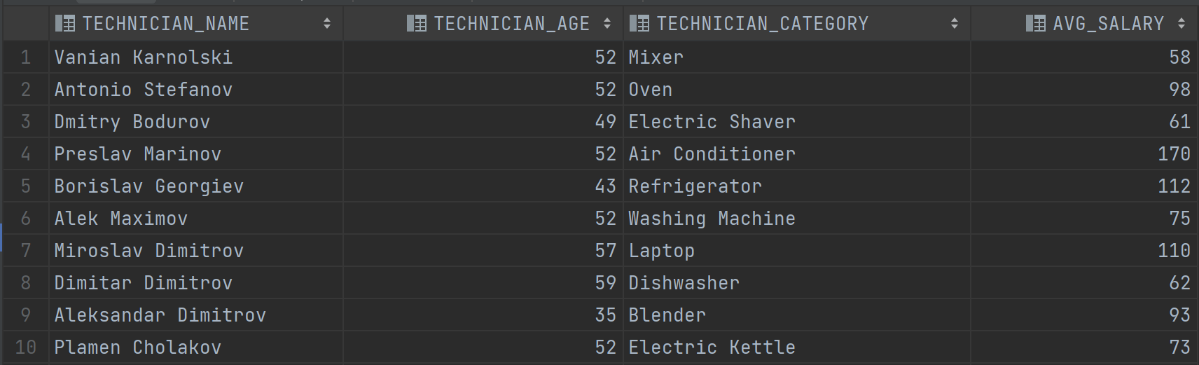
CREATE OR REPLACE TRIGGER TRIG\_AFTER\_INSERT\_TECHNICIAN  
AFTER INSERT ON TECHNICIAN  
REFERENCING NEW AS N  
FOR EACH ROW  
BEGIN  
 DECLARE V\_TECHNICIAN\_ID INT;  
  
 SET V\_TECHNICIAN\_ID = N.TECHNICIAN\_ID;  
 CALL FN4MI0700044.*P\_TECHNICIAN\_INFO*(V\_TECHNICIAN\_ID);  
END;  
  
INSERT INTO TECHNICIAN(NAME, EGN, TECHNICIAN\_CATEGORY)  
VALUES ('John Doеe', 6898127643, 'Washing machine');



1. **Описание на изгледите**

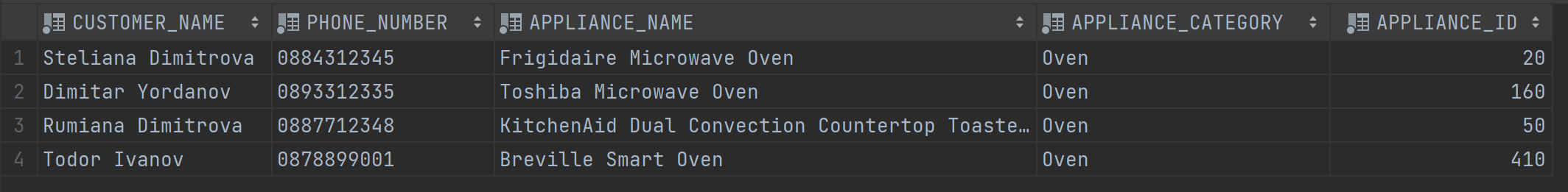
Изгледът с име TECHNICIAN\_INFO представлява виртуална таблица, в която се представя информация за техниците. Той използва функции и операции за обработка на данни, за да предостави нужната информация. След името на изгледа в скобите се задават имената на колоните, а след това чрез SELECT извличаме информацията. Възрастта на техниците се изчислява като се използва EXTRACT и CAST за извличане на годината от текущата дата и конвертиране на двуцифрената година от ЕГН към цяло число. След това двете се изваждат и получаваме годините. За колоната AVG\_SALARY се използва функцията FN4MI0700044.F\_AVG\_PRICE\_FOR\_CATEGORY за изчисляване на средната цена на ремонтите за категорията на техника. Накрая чрез заявка извличаме редовете и колоните от изгледа.

CREATE VIEW TECHNICIAN\_INFO (TECHNICIAN\_NAME, TECHNICIAN\_AGE, CATEGORY, AVG\_SALARY)  
AS  
SELECT T.NAME, *EXTRACT*(YEAR FROM CURRENT\_DATE) - *CAST*(*CONCAT*('19', *SUBSTR*(T.EGN, 1, 2)) AS INT) AS , T.TECHNICIAN\_CATEGORY,  
 FN4MI0700044.*F\_AVG\_PRICE\_FOR\_CATEGORY*(T.TECHNICIAN\_CATEGORY)  
FROM TECHNICIAN T;  
  
SELECT \* FROM TECHNICIAN\_INFO;

****

Изгледът с име V\_CUSTOMER\_APPLIANCE\_INFO представлява виртуална таблица, която комбинира информацията от две базови таблиците - CUSTOMER и APPLIANCE. Той предоставя информация за връзката между клиенти и техните уреди. Чрез SELECT избираме необходимите колони от двете таблици. Създаваме връзка между CUSTOMER и APPLIANCE чрез телефонния номер на клиента и клиентския телефонен номер на уреда. Накрая чрез заявка извличаме всички редове и колони от изгледа V\_CUSTOMER\_APPLIANCE\_INFO, където категорията на уреда е 'Oven'.

CREATE VIEW V\_CUSTOMER\_APPLIANCE\_INFO (CUSTOMER\_NAME, PHONE\_NUMBER, APPLIANCE\_NAME, APPLIANCE\_CATEGORY, APPLIANCE\_ID)  
AS  
SELECT C.NAME as CUSTOMER\_NAME, C.PHONE\_NUMBER, A.NAME as APPLIANCE\_NAME, A.APPLIANCE\_CATEGORY, A.APPLIANCE\_ID  
FROM CUSTOMER C, APPLIANCE A  
WHERE C.PHONE\_NUMBER = A.CUSTOMER\_PHONE\_NUMBER;  
  
SELECT \*  
FROM V\_CUSTOMER\_APPLIANCE\_INFO  
WHERE APPLIANCE\_CATEGORY = 'Oven';

****