**Question NO 01:**

**CUSTOMER TABLE**

**1NF:** Given table has already 1NF because there is only atomic values.

**2NF:** In 2NF we will remove functional dependency

All non\_key attributes must be fully dependent on key. In this postal code is depending on city and address ,contact also. So,

Customer(Customer\_ID,customer\_name,country)

Contacts(Customer\_id,contact\_name,contact\_email,address,city,postal\_code)

**3NF:** No transitive dependency

Customer(Customer\_ID,customer\_name,country)

Contacts(Customer\_id,contact\_name,contact\_email,address\_ID)

Address(Address\_id,address,city,postal\_code)

**ORDER TABLE**

**1NF:** There is already 1NF in this table

**2NF:** functional dependency in order id, ship via

Orders ( id,customer,id,order date)

Ships (id,shipped date,ship via)

**3NF:** There is No transitive dependency

Orders ( id,customer,id,order date)

Ships (id,shipped date,ship via)

**Order details**

**1NF:** Order (id, price)

Details (id,product\_id,quantity)

Product(id,name,unit price)

**2NF:** Price can be predict by unit price and quantity so

Order\_Details (id,product\_id,quantity)

Product(id,name,unit price)

**3NF:** Already in 3NF.

**Question NO 02:**

**Staff branch table**

**1NF:**

Already in 1NF.

**2NF:**

Staff(satffno,name,pos,salary,branchno)

Branch(branchno,branchAddress,telNO)

**3NF:**

Already in 3NF.

**Staff**

**1NF:**

Already in 1NF

**2NF**

Staff (staffno,name,position,salary)

Branch (branchNo,satffNo,position)

**3NF**

Already in 3NF

**Branch**

**1NF**

Already in 1NF

As there is atomic values

**2NF**

No partial dependencies

**Question NO 03:**

To apply 5NF to the given tables, we need to analyze the functional dependencies and remove the redundancies.

For table 1 CUSTOMER\_ORDERS, we can identify the following functional dependencies:

Order ID -> Customer ID, Order Date

Order ID, Product ID -> Quantity

To normalize to 5NF, we can create the following tables:

1. ORDERS

Order ID (PK)

Customer ID (FK)

Order Date

1. ORDER\_PRODUCTS

Order ID (PK, FK)

Product ID (PK, FK)

Quantity

For table 2 STUDENT\_COURSES, we can identify the following functional dependencies:

student\_id, course -> instructor\_name, grade

To normalize to 5NF, we can create the following tables:

1. STUDENTS

student\_id (PK)

1. COURSES

course\_id (PK)

instructor\_name

1. ENROLLMENT

student\_id (PK, FK)

course\_id (PK, FK)

grade

The final normalized tables are:

1. ORDERS

Order ID (PK)

Customer ID (FK)

Order Date

1. ORDER\_PRODUCTS

Order ID (PK, FK)

Product ID (PK, FK)

Quantity

1. STUDENTS

student\_id (PK)

1. COURSES

course\_id (PK)

instructor\_name

1. ENROLLMENT

student\_id (PK, FK)

course\_id (PK, FK)

grade

**Question No 04: Procedures, Functions, PL/SQL (60)**

**1.** Write a procedure to transfer an employee from one manager to another and update the hire date

accordingly.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE TRANSFER\_EMPLOYEE(

new\_employee\_id IN employees.employee\_id%TYPE,

new\_manager\_id IN employees.manager\_id%TYPE,

new\_hire\_date IN employees.hire\_date%TYPE

) IS

BEGIN

UPDATE EMPLOYEES

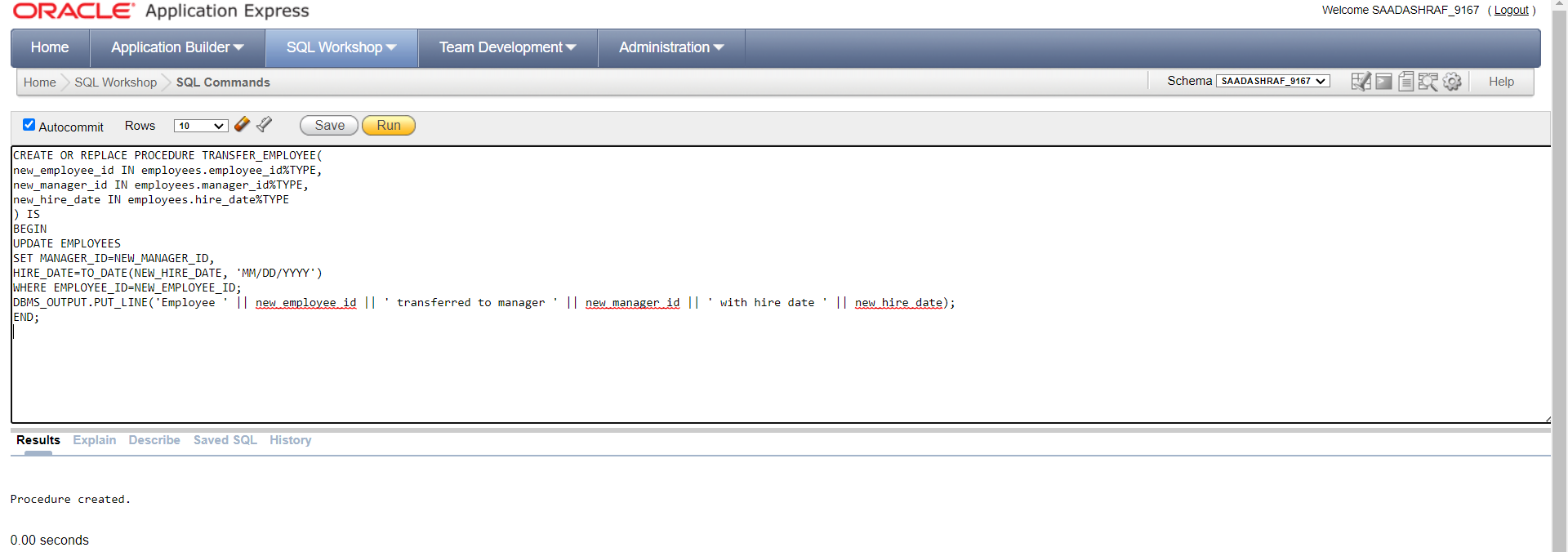
SET MANAGER\_ID=NEW\_MANAGER\_ID,

HIRE\_DATE=TO\_DATE(NEW\_HIRE\_DATE, 'MM/DD/YYYY')

WHERE EMPLOYEE\_ID=NEW\_EMPLOYEE\_ID;

DBMS\_OUTPUT.PUT\_LINE('Employee ' || new\_employee\_id || ' transferred to manager ' || new\_manager\_id || ' with hire date ' || new\_hire\_date);

END;

****

BEGIN

TRANSFER\_EMPLOYEE(new\_employee\_id => 100, new\_manager\_id => 200, new\_hire\_date => '02-01-2022');

END;



**2.** Write a procedure to generate a report of products that have low inventory level and their

categories’ names using CURSOR.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE inventory\_report

IS

CURSOR prod\_cursor

IS

SELECT MIN(p.product\_name) as p\_name, MIN(c.category\_name) as c\_name, MIN(i.quantity) AS low\_level

FROM products p

JOIN product\_categories c ON p.category\_id = c.category\_id

JOIN inventories i ON p.product\_id=i.product\_id;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('PRODUCT\_NAME' || CHR(9) || 'CATEGORY\_NAME' || CHR(9) || 'QUANTITY');

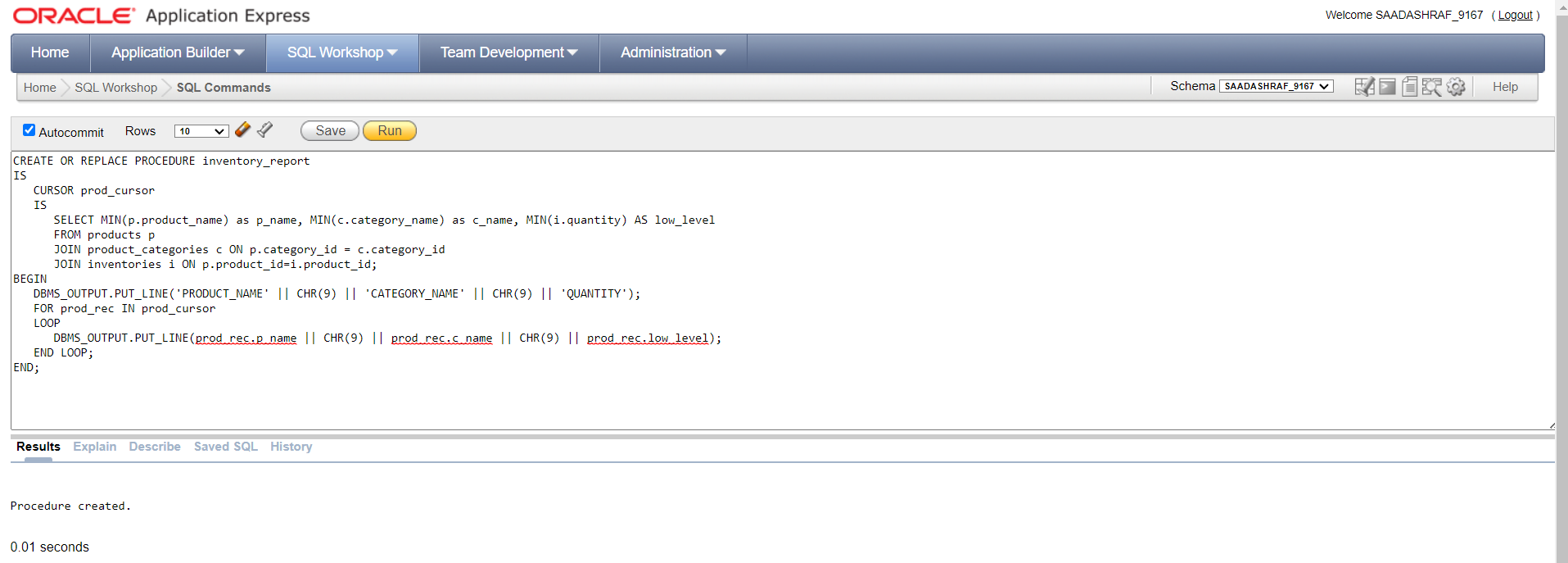
FOR prod\_rec IN prod\_cursor

LOOP

DBMS\_OUTPUT.PUT\_LINE(prod\_rec.p\_name || CHR(9) || prod\_rec.c\_name || CHR(9) || prod\_rec.low\_level);

END LOOP;

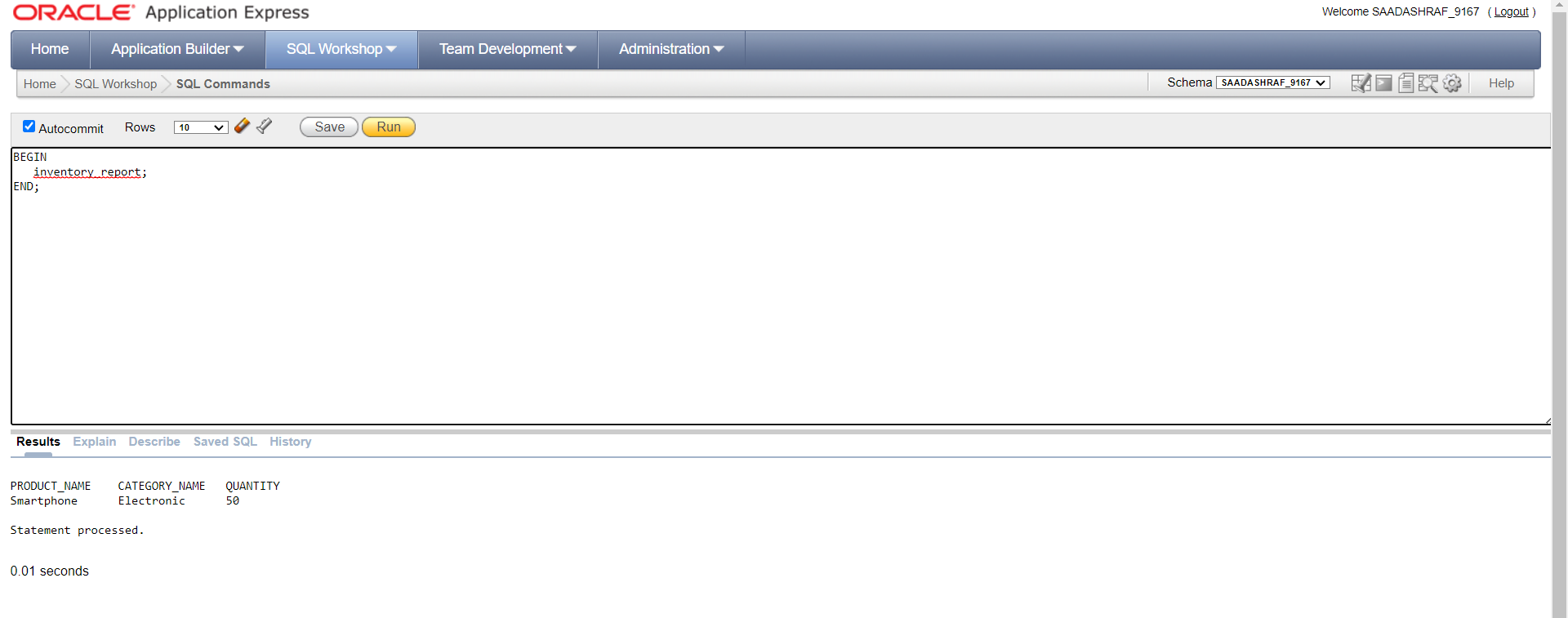
END;



BEGIN

inventory\_report;

END;



**3.** Create a procedure to display complete information of table employees having employee first

name contain “A” in their name using CURSOR.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE employee\_report

IS

CURSOR emp\_cursor

IS

SELECT \*

FROM employees

WHERE first\_name LIKE '%A%';

BEGIN

DBMS\_OUTPUT.PUT\_LINE('EMPLOYEE\_ID' || CHR(9) || 'First name:' || CHR(9) || 'last name:' || CHR(9) || 'email id: ' || CHR(9) || 'contact:' || CHR(9) || 'hire date:' || CHR(9) || 'manager:' || CHR(9) || 'job:');

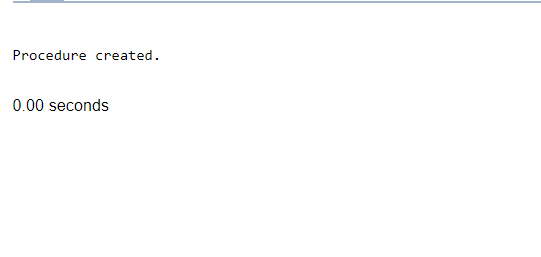
FOR emp\_rec IN emp\_cursor

LOOP

DBMS\_OUTPUT.PUT\_LINE(emp\_rec.employee\_id || CHR(9) || emp\_rec.first\_name || CHR(9) || emp\_rec.last\_name || CHR(9) || emp\_rec.email || CHR(9) || emp\_rec.phone || CHR(9) || emp\_rec.hire\_date || CHR(9) || emp\_rec.manager\_id || CHR(9) || emp\_rec.JOB\_TITLE);

END LOOP;

END;

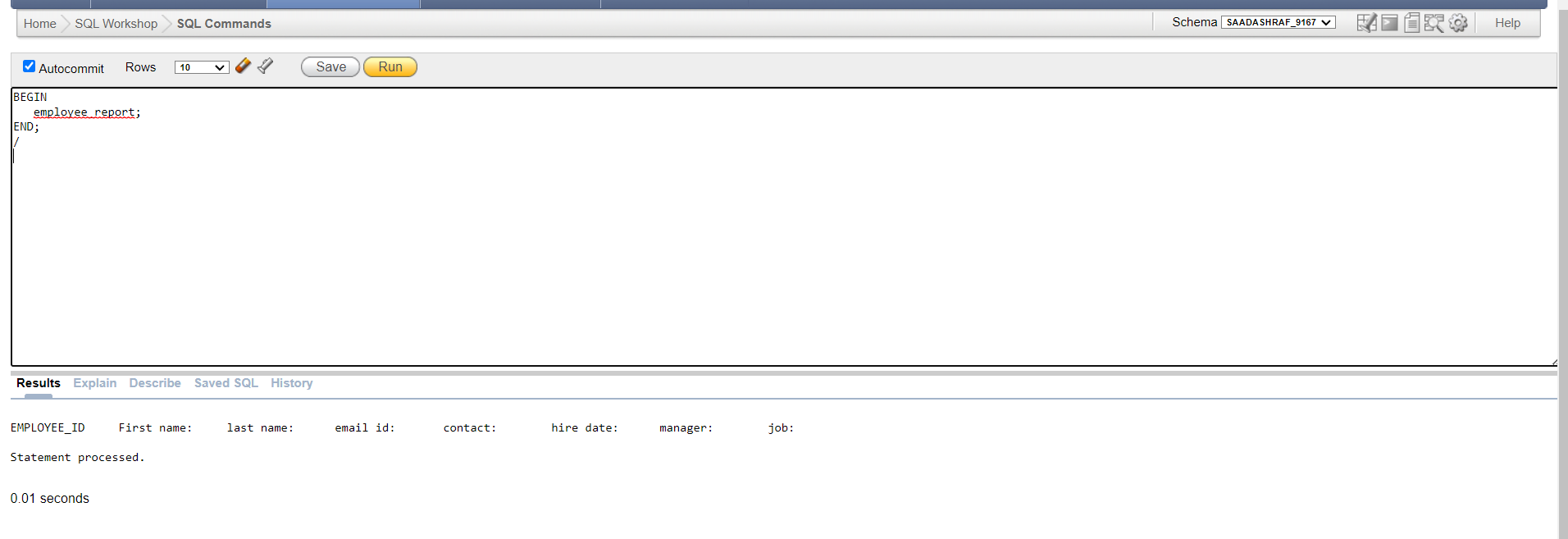


BEGIN

employee\_report;

END;

/



NO DATA DISPLAYED BECAUSE NO EMPLOYEE HAS A NAME THAT STARTS WITH **“A”.**

**4.** Create a Procedure that retrieves the number of employees managed by a given manager.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE get\_employee\_count (manager\_id IN NUMBER, emp\_count OUT NUMBER)

IS

BEGIN

SELECT COUNT(\*) INTO emp\_count

FROM employees

WHERE manager\_id = manager\_id;

END;



DECLARE

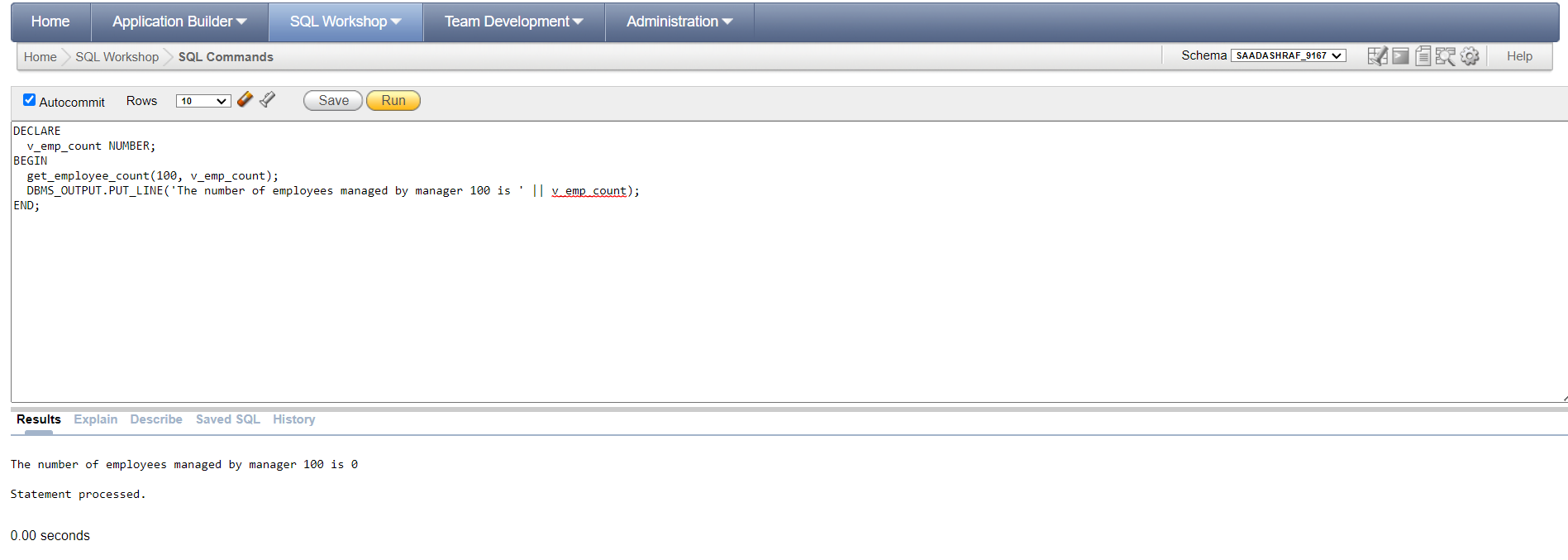
v\_emp\_count NUMBER;

BEGIN

get\_employee\_count(100, v\_emp\_count);

DBMS\_OUTPUT.PUT\_LINE('The number of employees managed by manager 100 is ' || v\_emp\_count);

END;



**5.** Create a procedure to decrease the quantity in inventory. Make sure the quantity in inventory do

not become negative.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE decrease\_inventory\_quantity(

p\_product\_id IN NUMBER,

p\_quantity IN NUMBER

) AS

v\_current\_quantity NUMBER;

BEGIN

-- Get the current quantity of the product in inventory

SELECT quantity INTO v\_current\_quantity

FROM inventories

WHERE product\_id = p\_product\_id;

-- Check if there is enough quantity to decrease

IF v\_current\_quantity >= p\_quantity THEN

-- Decrease the quantity

UPDATE inventories

SET quantity = quantity - p\_quantity

WHERE product\_id = p\_product\_id;

ELSE

-- Raise an exception if there is not enough quantity

RAISE\_APPLICATION\_ERROR(-20001, 'Not enough quantity in inventory');

END IF;

END;



BEGIN

decrease\_inventory\_quantity(289, 5);

END;

If you check in sql table data the amount of product is decreased.

**6.** Write a function to create a new product category with a given name and return its ID.

**SOLUTION:**

CREATE OR REPLACE FUNCTION create\_category(new\_category\_name IN VARCHAR2)

RETURN NUMBER

IS

v\_category\_id NUMBER;

BEGIN

SELECT MAX(category\_id) + 1 INTO v\_category\_id FROM product\_categories;

IF v\_category\_id IS NULL THEN

v\_category\_id := 1;

END IF;

INSERT INTO product\_categories (category\_id, category\_name)

VALUES (v\_category\_id, new\_category\_name);

RETURN v\_category\_id;

END;



DECLARE

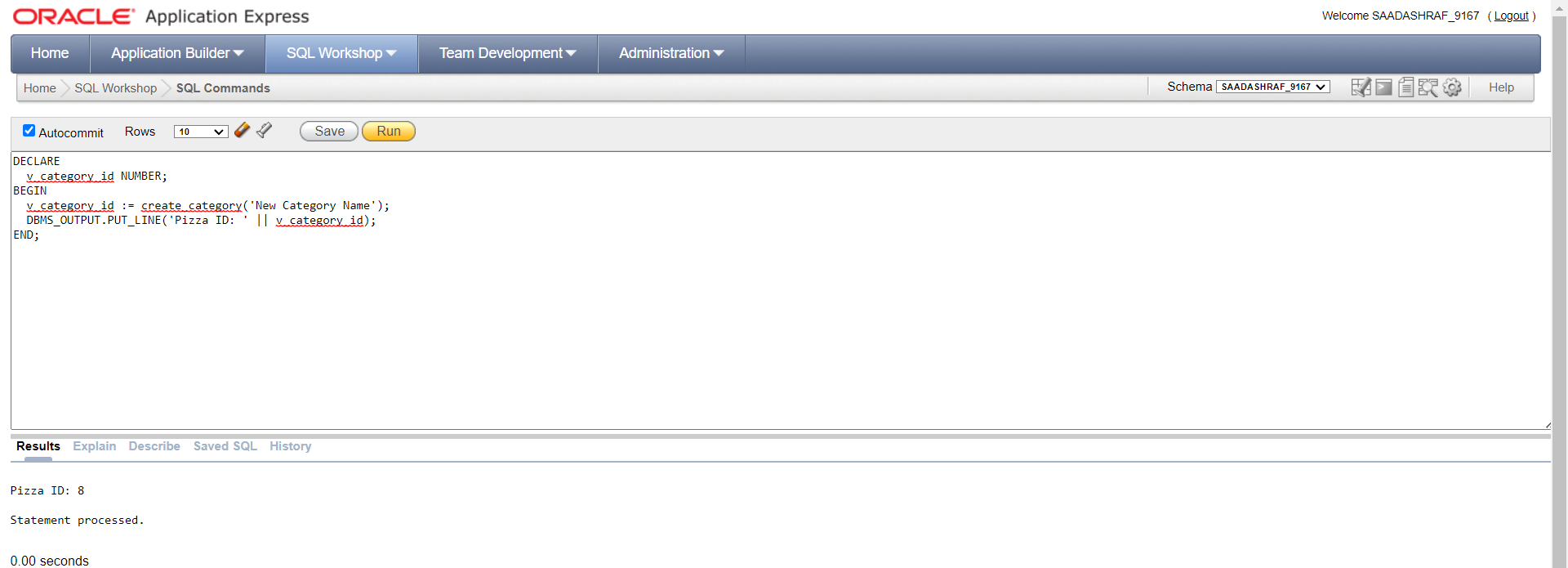
v\_category\_id NUMBER;

BEGIN

v\_category\_id := create\_category('New Category Name');

DBMS\_OUTPUT.PUT\_LINE('Pizza ID: ' || v\_category\_id);

END;



**7.** Create a pl/sql program that will use the function in (7) to create a product category and then use

that id to store a product data.

**SOLUTION:**

DECLARE

v\_category\_id NUMBER;

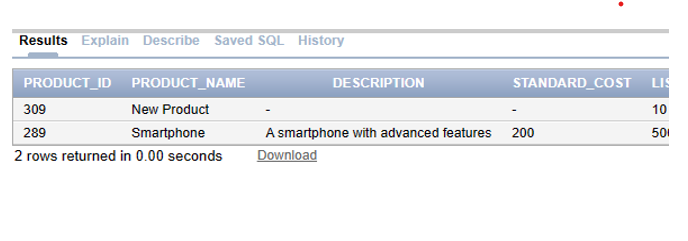
BEGIN

v\_category\_id := create\_category('New Category');

INSERT INTO products (product\_name, category\_id, list\_price)

VALUES ('New Product', v\_category\_id, 10.00);

END;



**8.** Write a function that retrieves the total number of products in a given category.

**SOLUTION:**

CREATE OR REPLACE FUNCTION TOTAL\_PRODUCT(category\_input IN NUMBER) RETURN NUMBER

IS

total\_count NUMBER;

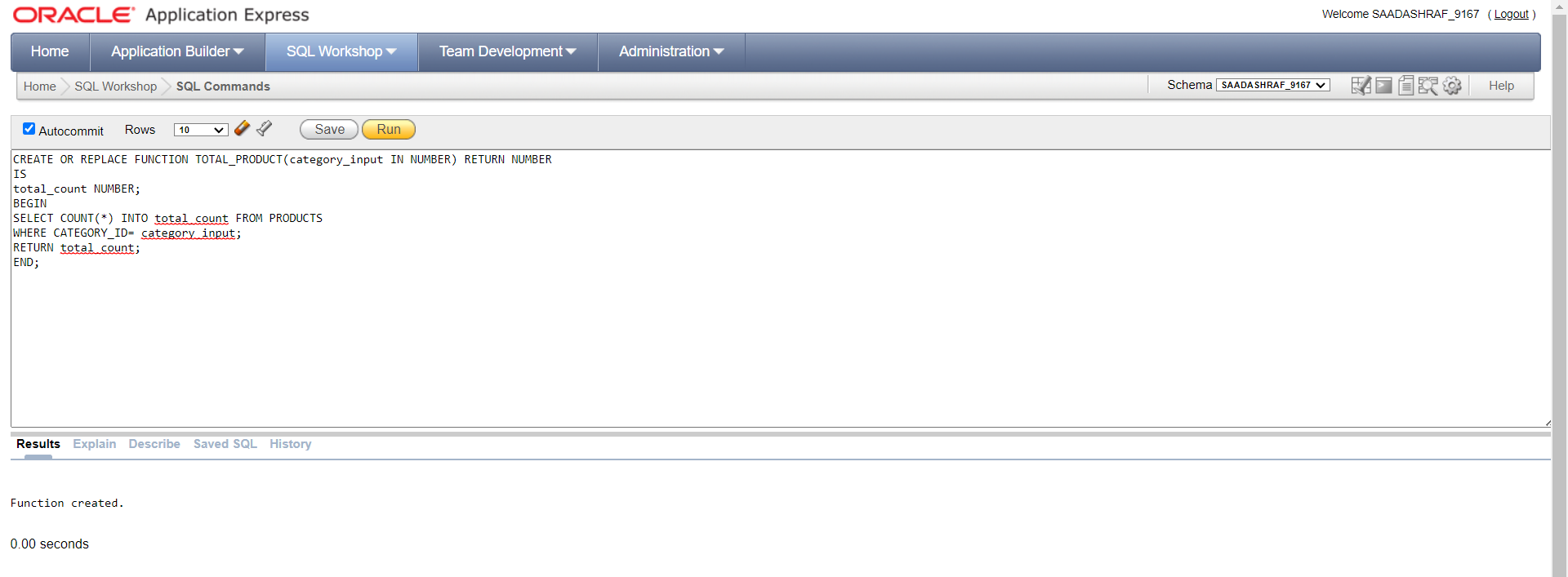
BEGIN

SELECT COUNT(\*) INTO total\_count FROM PRODUCTS

WHERE CATEGORY\_ID= category\_input;

RETURN total\_count;

END;



SELECT TOTAL\_PRODUCT(1) FROM dual;



**9.** Create a function that returns the sales total for a given customer.

**SOLUTION:**

CREATE OR REPLACE FUNCTION GET\_SALES(cust\_id IN NUMBER) RETURN NUMBER

IS

total\_sales number(8,2);

BEGIN

SELECT SUM(od.quantity \* od.unit\_price) INTO total\_sales

FROM ORDERS O JOIN ORDER\_ITEMS OD ON OD.ORDER\_ID=O.ORDER\_ID

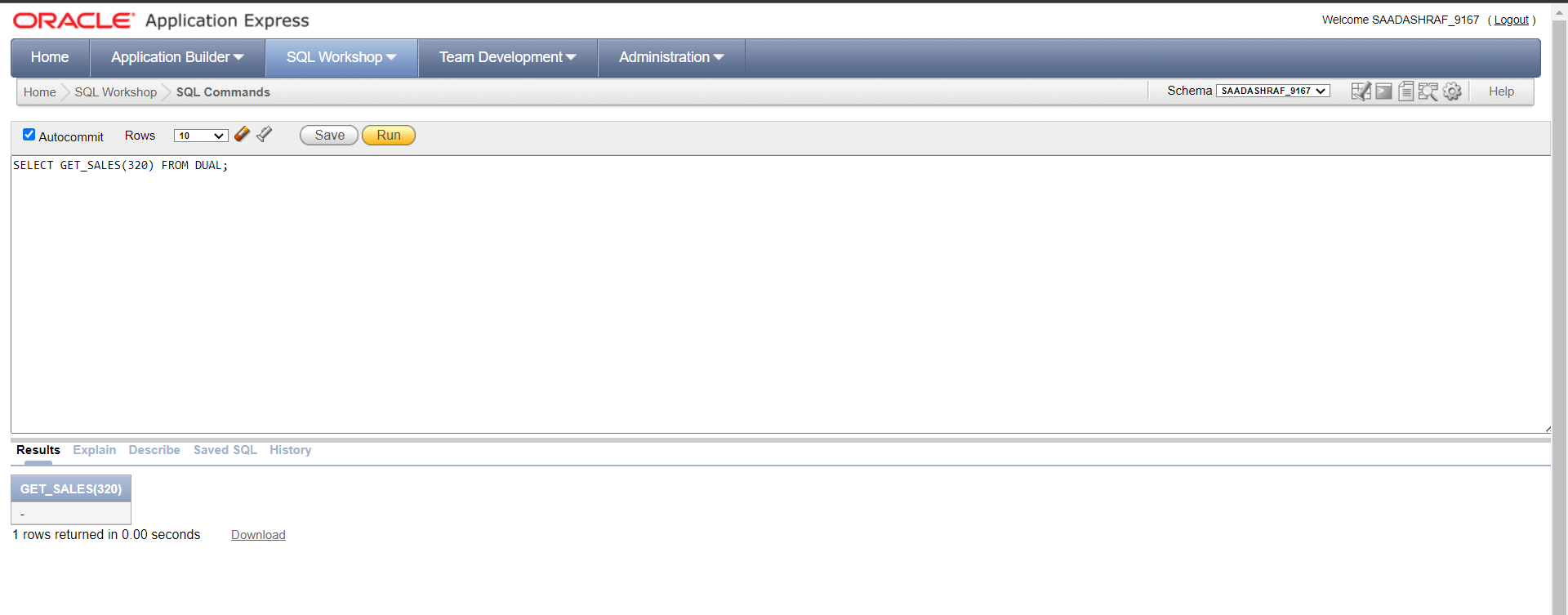
WHERE O.CUSTOMER\_ID =cust\_id;

RETURN total\_sales;

END;



SELECT GET\_SALES(320) FROM DUAL;



**10.** Call all functions in a pl/sql program and show output.

**SOLUTION:**

DECLARE

CHOICE NUMBER;

RETURNING NUMBER;

EMP\_COUNT NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('1- TRANSFER EMPLOYEES ');

DBMS\_OUTPUT.PUT\_LINE('2- GENERATE REPORTS ');

DBMS\_OUTPUT.PUT\_LINE('3- DISPLAY EMPLUEES CONTAIN A ');

DBMS\_OUTPUT.PUT\_LINE('4- NO OF EMP MANAGED BY GIVEN MANAGER ');

DBMS\_OUTPUT.PUT\_LINE('5- DECREASE QUANTITY ');

DBMS\_OUTPUT.PUT\_LINE('6- CREATE PRODUCT CATEGORY ');

DBMS\_OUTPUT.PUT\_LINE('7- STORE PRODUCT ');

DBMS\_OUTPUT.PUT\_LINE('8- TOTAL PRODUCTS IN GIVEN CATEGORY ');

DBMS\_OUTPUT.PUT\_LINE('9- RETURN SALES OF CUSTOMER');

DBMS\_OUTPUT.PUT\_LINE('ENTER CHOICE : ');

CHOICE:=:TEMP;

CASE

WHEN CHOICE=1

THEN

TRANSFER\_EMPLOYEE(7499,7486,TO\_DATE('2022-05-06', 'YYYY-MM-DD'));

WHEN CHOICE=2

THEN

inventory\_report;

WHEN CHOICE=3

THEN

employee\_report;

WHEN CHOICE=4

THEN

get\_employee\_count(108, emp\_count);

DBMS\_OUTPUT.PUT\_LINE('Total number of employees managed by given manager: ' || emp\_count);

WHEN CHOICE=5

THEN

decrease\_inventory\_quantity(289,20);

WHEN CHOICE=6

THEN

RETURNING:=create\_category('NEW CATEGORY');

DBMS\_OUTPUT.PUT\_LINE(RETURNING);

WHEN CHOICE=7

THEN

DBMS\_OUTPUT.PUT\_LINE('NEW PRODUCT INSERTED');

WHEN CHOICE=8

THEN

RETURNING:=TOTAL\_PRODUCT(27);

DBMS\_OUTPUT.PUT\_LINE(RETURNING);

WHEN CHOICE=9

THEN

RETURNING:=GET\_SALES(320);

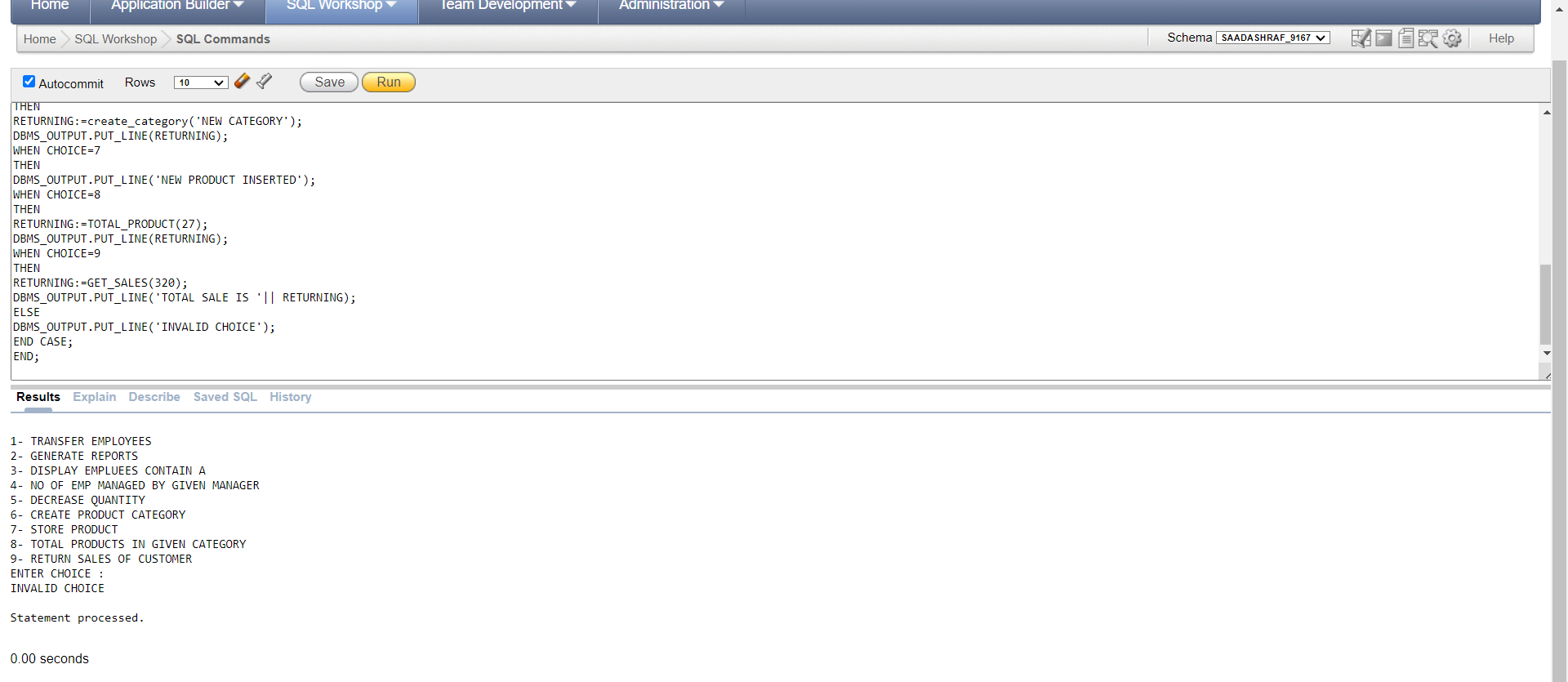
DBMS\_OUTPUT.PUT\_LINE('TOTAL SALE IS '|| RETURNING);

ELSE

DBMS\_OUTPUT.PUT\_LINE('INVALID CHOICE');

END CASE;

END;



**Question No 05: Triggers (70)**

**1.** Create an Oracle trigger to automatically insert a new region into the regions table when a new

country is added to the countries table?

**SOLUTION:**

CREATE OR REPLACE TRIGGER insert\_region

BEFORE INSERT ON COUNTRIES

FOR EACH ROW

DECLARE

v\_region\_id regions.region\_id%type;

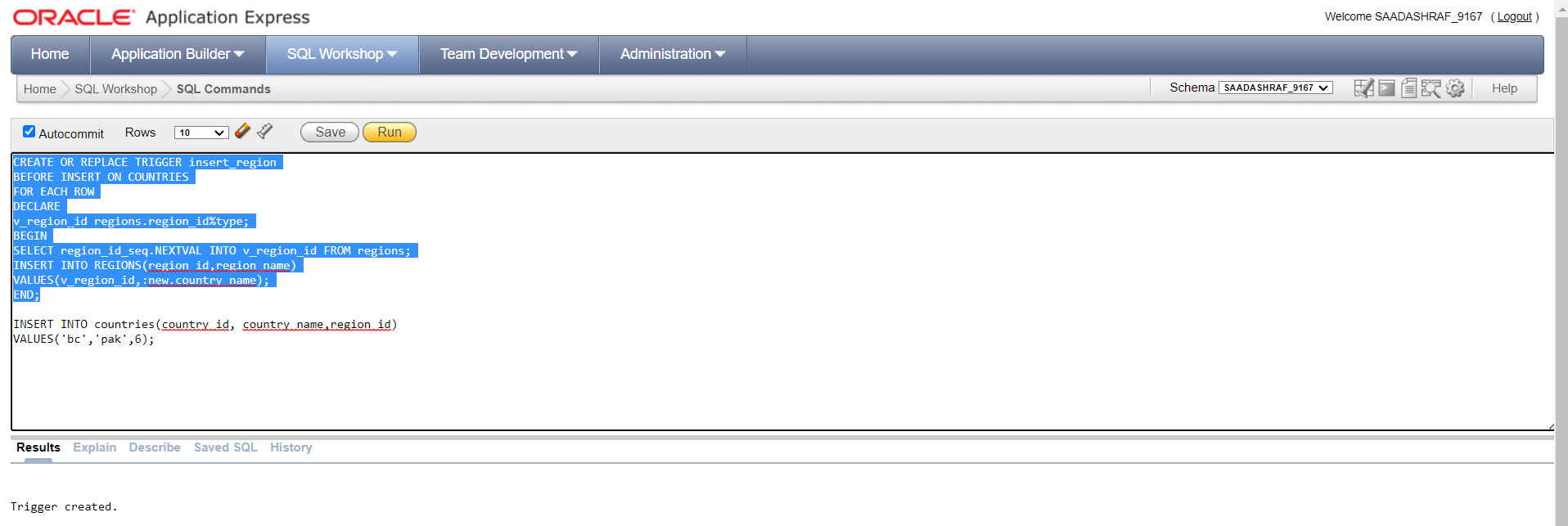
BEGIN

SELECT region\_id\_seq.NEXTVAL INTO v\_region\_id FROM regions;

INSERT INTO REGIONS(region\_id,region\_name)

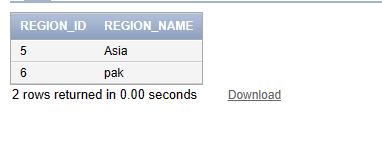
VALUES(v\_region\_id,:new.country\_name);

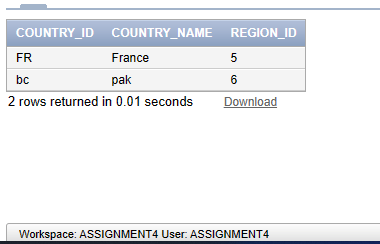
END;



INSERT INTO countries(country\_id, country\_name,region\_id)

VALUES('bc','pak',6);





**2.** Create an Oracle trigger to update the quantity of a product in the inventories table after an order

is placed in the order\_items table?

**SOLUTION:**

CREATE OR REPLACE TRIGGER update\_inventory\_quantity

AFTER INSERT ON order\_items

FOR EACH ROW

BEGIN

UPDATE inventories

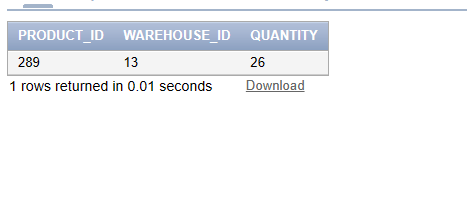
SET quantity = quantity - :NEW.quantity

WHERE product\_id = :NEW.product\_id;

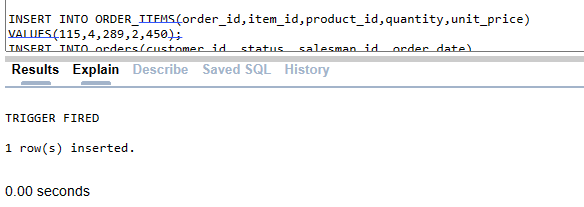
END;

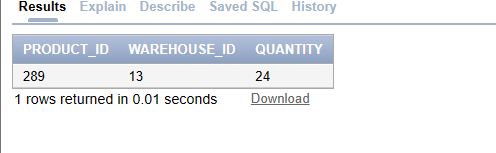
INSERT INTO ORDER\_ITEMS(order\_id,item\_id,product\_id,quantity,unit\_price)

VALUES(115,4,289,2,450);



AFTER INSERTING IN ORDER ITEMS and 2 quantities are utlized





**3.** Create an Oracle trigger to automatically delete all contacts associated with a customer when that

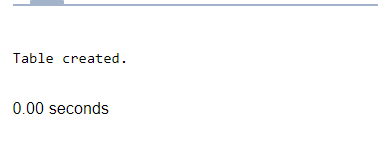
customer is deleted from the customers table?

**SOLUTION:**

CREATE GLOBAL TEMPORARY TABLE temp\_customer\_ids (

customer\_id NUMBER

) ON COMMIT PRESERVE ROWS;



CREATE OR REPLACE TRIGGER delete\_contacts

AFTER DELETE ON customers

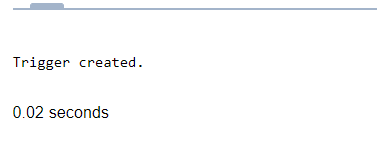
FOR EACH ROW

BEGIN

INSERT INTO temp\_customer\_ids (customer\_id)

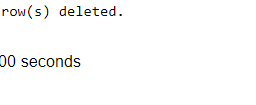
VALUES (:OLD.customer\_id);

END;



DELETE FROM contacts

WHERE customer\_id IN (SELECT customer\_id FROM temp\_customer\_ids);

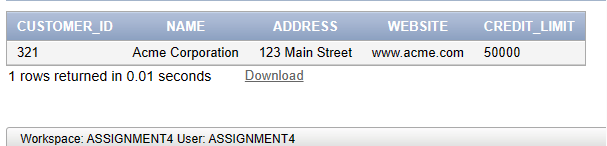


DELETE FROM customers

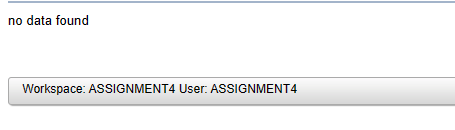
WHERE customer\_id =321;

**Customer table**

before

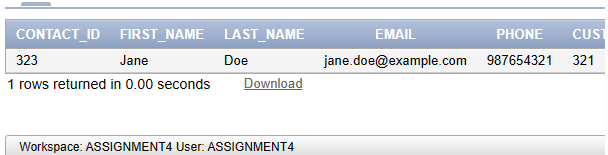


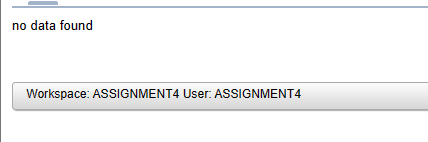
After

****

**Contacts table**

**Before**

****

****

**4.** Create an Oracle trigger to update the credit limit of a customer in the customers table when a

new order is placed in the orders table and the order total exceeds the current credit limit of the

customer?

**SOLUTION:**

CREATE OR REPLACE TRIGGER CREDIT\_LIMIT

BEFORE INSERT ON ORDER\_ITEMS

FOR EACH ROW

DECLARE

v\_customer\_id orders.customer\_id%TYPE;

total\_cost NUMBER;

v\_credit\_limit customers.credit\_limit%type;

BEGIN

SELECT customer\_id INTO v\_customer\_id FROM ORDERS

WHERE order\_id=:NEW.order\_id;

SELECT credit\_limit INTO v\_credit\_limit

FROM CUSTOMERS where customer\_id = v\_customer\_id;

SELECT SUM(quantity\*unit\_price) INTO total\_cost FROM ORDER\_ITEMS

where order\_id=:NEW.order\_id;

IF total\_cost>v\_credit\_limit

THEN

DBMS\_OUTPUT.PUT\_LINE('TRGGER FIRED TOTAL COST EXCEEDS');

ELSE

UPDATE CUSTOMERS

SET credit\_limit = v\_credit\_limit - total\_cost

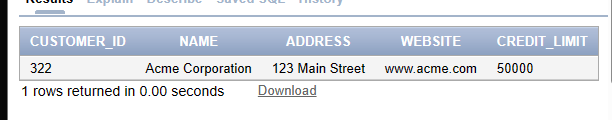
WHERE customer\_id = v\_customer\_id;

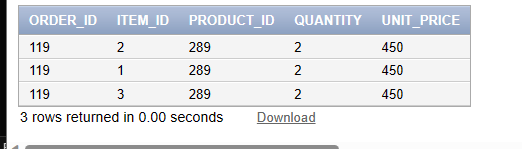
END IF;

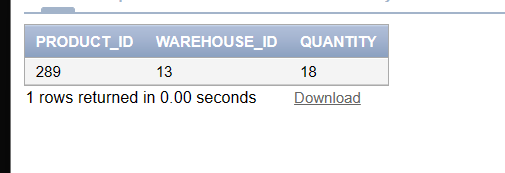
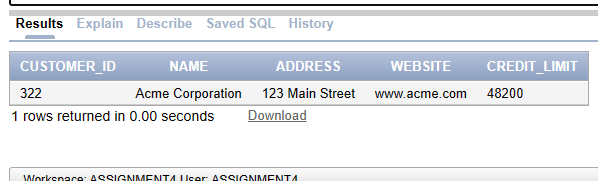
END;

INSERT INTO ORDER\_ITEMS(order\_id,item\_id,product\_id,quantity,unit\_price)

VALUES(119,3,289,2,450);







**5.** Create an Oracle trigger to automatically update the product\_category column in the products

table when a product is added to the table and the product name contains a name from category?

e.g Cheese Pizza contains category name(Pizza) in its name. (You can use Cursor or arrays for

this purpose).

**SOLUTION:**

CREATE OR REPLACE TRIGGER update\_product\_category

BEFORE INSERT ON products

FOR EACH ROW

DECLARE

v\_category\_id number;

v\_category\_name varchar(50);

TYPE category\_list IS TABLE OF VARCHAR2(100);

categories category\_list := category\_list('Pizza', 'Pasta', 'Wings');

CURSOR c\_product\_name IS

SELECT product\_name FROM products WHERE product\_id = :new.product\_id;

BEGIN

IF (INSTR(LOWER(:new.product\_name), 'pizza') > 0) THEN

v\_category\_name := 'Pizza';

ELSIF (INSTR(LOWER(:new.product\_name), 'Pasta') > 0) THEN

v\_category\_name := 'Pasta';

ELSIF (INSTR(LOWER(:new.product\_name), 'Wings') > 0) THEN

v\_category\_name := 'Wings';

ELSE

v\_category\_name := 'Other';

END IF;

OPEN c\_product\_name;

FETCH c\_product\_name INTO v\_category\_name;

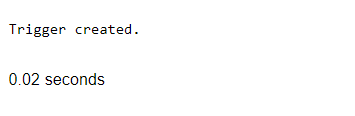
CLOSE c\_product\_name;

SELECT prod\_seq.NEXTVAL INTO v\_category\_id FROM product\_categories;

INSERT INTO PRODUCT\_CATEGORIES (category\_id,category\_name)

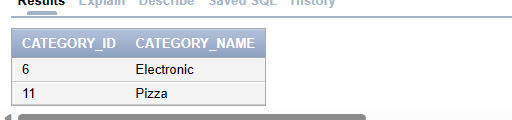
VALUES (v\_category\_id,v\_category\_name);

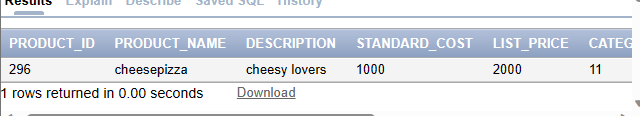
END;



INSERT INTO PRODUCTS

VALUES(296,'cheesepizza','cheesy lovers', 1000,2000,8);



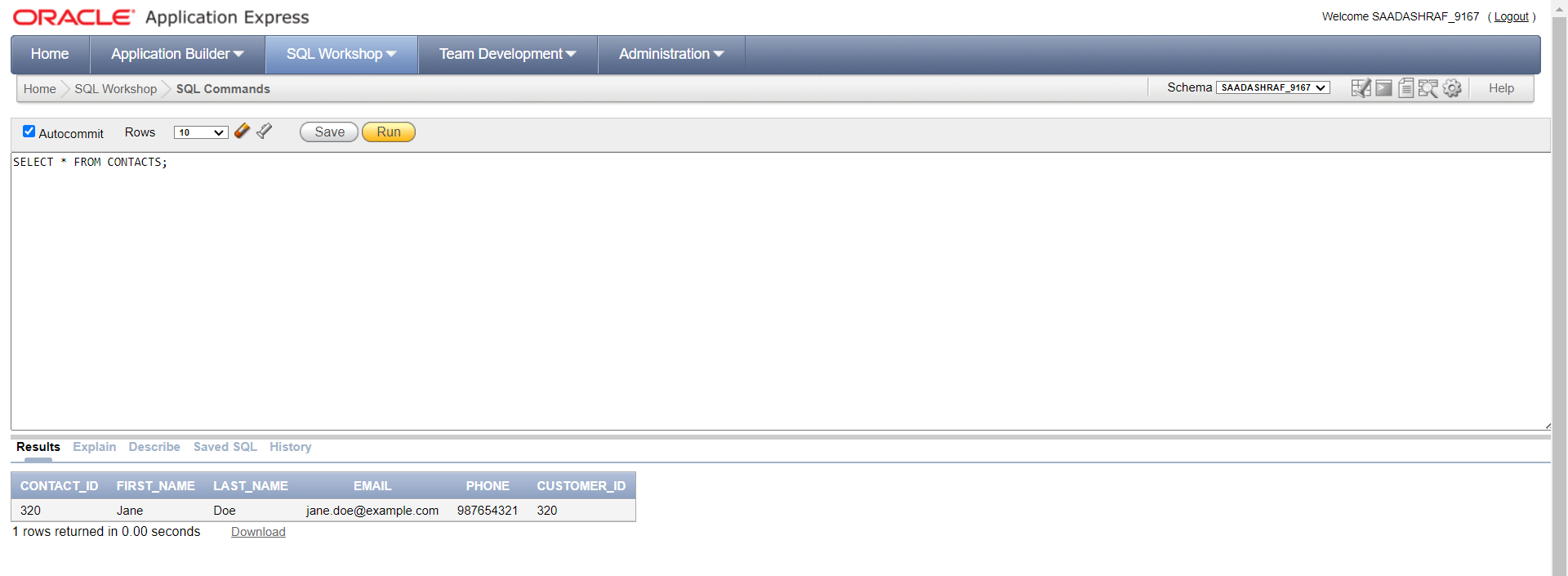


**6.** Create a table contacts\_backup using the create query of contacts but do not use any

constraint(make those fields simple). Now Create a Trigger that will run before any contact is deleted. It will save the data of the contact in contacts\_backup table.

**SOLUTION:**

SELECT \* FROM CONTACTS;



CREATE TABLE contacts\_backup (

contact\_id NUMBER PRIMARY KEY,

first\_name VARCHAR2(50),

last\_name VARCHAR2(50),

email VARCHAR2(100),

phone VARCHAR2(20),

customer\_id NUMBER,

backup\_date DATE DEFAULT SYSDATE

);

CREATE OR REPLACE TRIGGER backup\_contact

BEFORE DELETE ON contacts

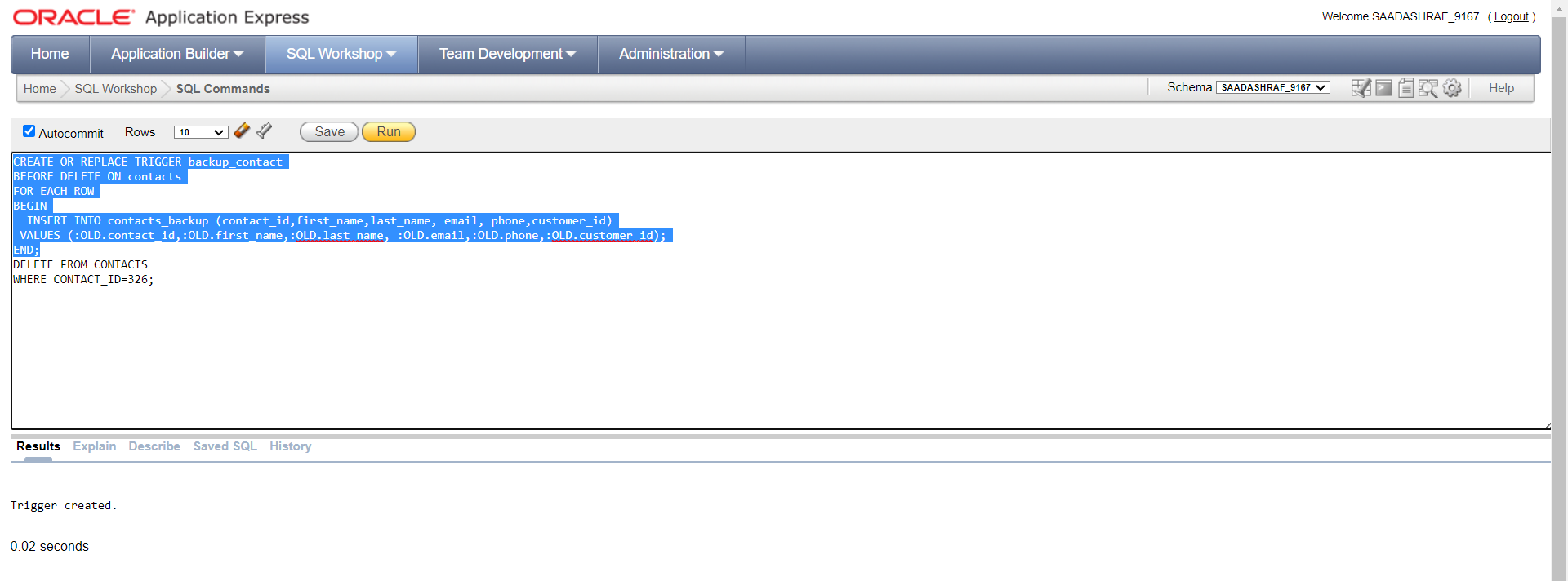
FOR EACH ROW

BEGIN

INSERT INTO contacts\_backup (contact\_id,first\_name,last\_name, email, phone,customer\_id)

VALUES (:OLD.contact\_id,:OLD.first\_name,:OLD.last\_name, :OLD.email,:OLD.phone,:OLD.customer\_id);

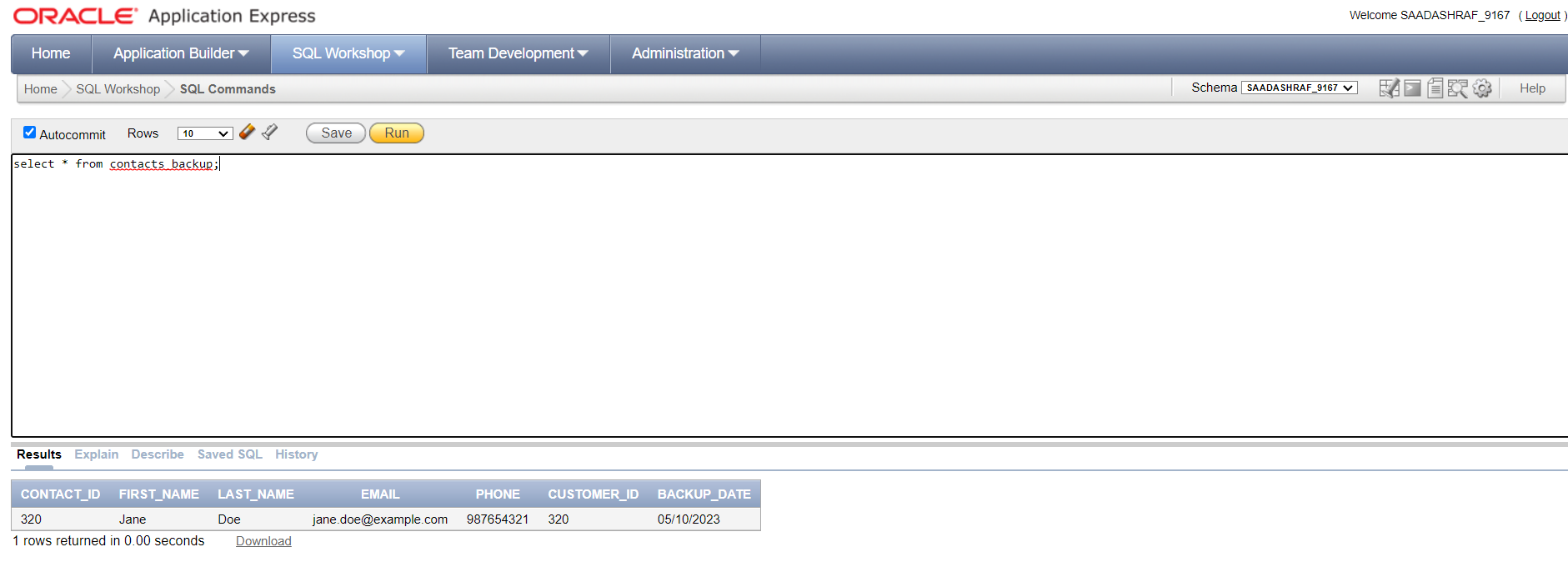
END;



DELETE FROM CONTACTS

WHERE CONTACT\_ID=320;

Select \* from contacts\_backup;



select \* from contacts;

