

Q1. Aim: To create a trigger program for a row level trigger for table customers that would fire for INSERT or DELETE or UPDATE operations on table and display salary difference between old & new values.

Theory: Triggers are stored programs which are automatically fired or executed when some specific defined events occur in the database.

The following events can set off a trigger:

- DML commands such as UPDATE, INSERT, DELETE.
- DDL commands such as create, ALTER & DROP.
- Database operations: server on, login, logout, etc.

Code:

```
CREATE OR REPLACE TRIGGER sal - chg
BEFORE DELETE OR INSERT OR UPDATE
ON CUSTOMERS
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    differences number;
```

BEGIN

* differences =: NEW. salary -: OLD. salary

```
dbms_output.put_line ('Old Salary : ' || OLD.salary);  
dbms_output.put_line ('New Salary : ' || NEW.salary);  
dbms_output.put_line ('Difference : ' || difference);  
END;
```

{ UPDATE QUERY: }

```
UPDATE CUSTOMERS SET Salary = 25000  
WHERE id = 1;
```

Output:

Trigger created.

Old salary: 20000
New salary: 25000
Salary difference: 5000

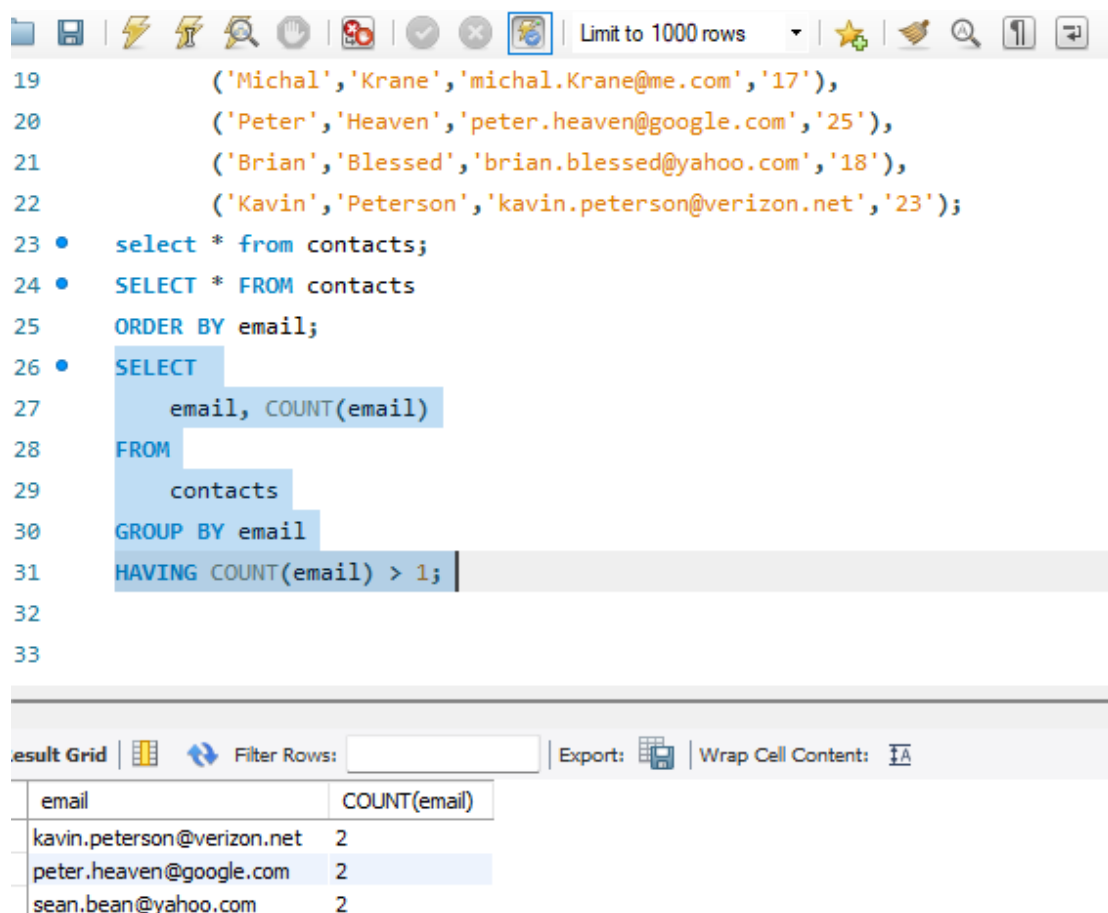
Q2 i) Aim: To write SQL queries to return ~~the~~ duplicate emails from the contact table.

Theory: To select duplicate emails, we shall group the tuples by email & print those whose count of email is greater than 1.

Code:

```
SELECT  
    email, COUNT(email)  
FROM  
    contacts  
GROUP BY email  
Having COUNT(email) > 1;
```

Output:



The screenshot shows a SQL IDE interface. At the top, there's a toolbar with various icons and a dropdown menu set to 'Limit to 1000 rows'. Below the toolbar, a SQL query is written in a text editor. The query is as follows:

```
19      ('Michal', 'Krane', 'michal.krane@me.com', '17'),
20      ('Peter', 'Heaven', 'peter.heaven@google.com', '25'),
21      ('Brian', 'Blessed', 'brian.blessed@yahoo.com', '18'),
22      ('Kavin', 'Peterson', 'kavin.peterson@verizon.net', '23');
23 •   select * from contacts;
24 •   SELECT * FROM contacts
25     ORDER BY email;
26 •   SELECT
27     email, COUNT(email)
28   FROM
29     contacts
30   GROUP BY email
31   HAVING COUNT(email) > 1;
```

Below the query editor, there's a 'result Grid' section. It has a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are displayed in a table with two columns: 'email' and 'COUNT(email)'.

email	COUNT(email)
kavin.peterson@verizon.net	2
peter.heaven@google.com	2
sean.beam@yahoo.com	2

ii) Aim: Write SQL Query to delete duplicate rows from contact table

Theory: If ~~dupl~~ duplicates exist, then we must delete the row which is older. So, we will check for duplicates & if the row is not the last row of duplicacy occurrence, then it is deleted. Here, we must use a JOIN.

2/3
19115045

Code:

DELETE t1 from contacts t1

JOIN contacts t2

WHERE

t1.id < t2.id AND

t1.email = t2.email;

3/3
19115045

Output:

```
23 • select * from contacts;
24 • SELECT * FROM contacts
25 ORDER BY email;
26 • SELECT
27     email, COUNT(email)
28 FROM
29     contacts
30 GROUP BY email
31 HAVING COUNT(email) > 1;
32 • DELETE t1 FROM contacts t1
33 JOIN contacts t2
34 WHERE
35     t1.id < t2.id AND
36     t1.email = t2.email;
37 • select * from contacts;
```

Result Grid					
		Filter Rows:		Edit:	
				Export/Import:	
				Wrap Cell Content:	
	id	first_name	last_name	email	age
▶	2	Nick	Jonas	nick.jonas@me.com	16
	4	Michal	Jackson	michal.jackson@aol.com	18
	6	Tom	Baker	tom.baker@aol.com	30
	7	Ben	Barnes	ben.barnes@comcast.net	21
	8	Mischa	Barton	mischa.barton@att.net	20
	9	Sean	Bean	sean.bean@yahoo.com	16
	10	Eliza	Bennett	eliza.bennett@yahoo.com	23
	11	Michal	Krane	michal.Krane@me.com	17
	12	Peter	Heaven	peter.heaven@google.com	25
	13	Brian	Blessed	brian.blessed@yahoo.com	18
	14	Kavin	Peterson	kavin.peterson@verizon.net	23
*	NULL	NULL	NULL	NULL	NULL