Advanced DataBases C4 Group
Bank Management System

Samuel Lúcio Vicente, 251720 Daniel Silva, 251702
Jorge Marrero Camiruaga, 251438

Politechnika Wrocławska December 19, 2019

Active Rules

Rule 1: Trigger

Name	Add amount to Account when inserting Deposit row
Triggering Events	When inserting row in deposit
Conditions	When inserting row in deposit
Description	When a deposit is registerd in the database the Account where the deposit happened must be updated to reflect its new amount
Complexity	$\mathcal{O}(1)$

SQL

```
1 CREATE OR REPLACE TRIGGER deposit_money
2 AFTER INSERT ON deposit
3 REFERENCING NEW AS NEW
4 FOR EACH ROW
5 BEGIN
6     UPDATE ACCOUNT
7     SET ACCOUNT.amount = (ACCOUNT.amount + :NEW.amount)
8     WHERE accountid = :NEW.accountaccountid;
9 END;
```

Times

	Max	Min	Avg	#
Off	1.415	1.163	1,265	5
On	2.911	2.345	2,5262	5

Rule 2: Trigger

Name	Close previous Role when assigning new one
Triggering Events	When you add a new row in RoleHistory table
Conditions	Add a new row in RoleHistory table, and there is a row already assossiated with employee
Description	When you add a new row in RoleHistory table, we need check if the employee already has a Role, and if so, change de Null Value of the EndDate column to the current date value
Complexity	$\mathcal{O}(1)$

SQL

```
1  CREATE OR REPLACE TRIGGER change_job
2  BEFORE INSERT ON rolehistory
3  REFERENCING NEW AS NEW
4  FOR EACH ROW
5  BEGIN
6  DBMS_OUTPUT.ENABLE;
7  DBMS_OUTPUT.ENABLE;
8  DBMS_OUTPUT.PUT_LINE( 'change_job fired' );
9      UPDATE rolehistory
10     SET enddate = current_date
11     WHERE employeeemployeeid = :NEW.employeeemployeeid AND enddate IS NULL;
12 END;
```

Times

	Max	Min	Avg	#
Off	4.143	3.675	3.9066	5
On	16.106	14.073	15,0156	5

Rule 3: Trigger (Constraint)

Name	Verify Job
Triggering Events	When you add a new row in RoleHistory table
Conditions	Add a new row in RoleHistory table
Description	When you add a new row in RoleHistory table we check if the job ID is valid or not
Complexity	$\mathcal{O}(1)$

SQL

```
1  CREATE OR REPLACE TRIGGER VERIFY_JOB
2  BEFORE INSERT ON ROLEHISTORY
3  REFERENCING NEW AS NEW OLD AS OLD
4  FOR EACH ROW
5  BEGIN
6         DBMS_OUTPUT.ENABLE;
7    DBMS_OUTPUT.PUT_LINE( 'verify_job fired' );
8         IF(:NEW.ROLEROLEID IN (1,2,3,4,5,6,7,8)) THEN
9         DBMS_OUTPUT.PUT_LINE( 'Job is valid' );
10     ELSE
11        RAISE_APPLICATION_ERROR( -20001, 'Job is invalid' );
12     END IF;
13 END;
```

Times

	Max	Min	Avg	#
Off	2.341	1.925	2,073	5
On	3.187	2.458	2,8072	5

Execution order of triggers

Rule 2 and 3 have the same trigger condition.
To test which one was triggered first we added a DBMS_OUTPUT to each of them.
When we tested we saw that there is no way to predict which one fired first.
We tried droping them and add them in the reverse order, this changed nothing we saw the same order.

Rule 4: Check Constraint

Name	Trying insert a birthday date greater than current date
Triggering Events	When you try to insert a new row in the table of Persons, where the row represents a person.
Conditions	The current date must be greater than the Birthday date
Description	When we try to insert the row, this is not inserted in the table because there are no negative ages
Complexity	$\mathcal{O}(1)$

SQL

```
1   CREATE OR REPLACE TRIGGER check_birthday
2  BEFORE INSERT ON person
3  REFERENCING NEW AS NEW
4  FOR EACH ROW
5  BEGIN
6         dbms_output.ENABLE;
7         IF(:NEW.birthday > current_date) THEN
8         raise_application_error( -20001, 'Birthday is bigger than Current Date' );
9    END IF;
10 END;
```

Times

	Max	Min	Avg	#
Off	3.240	2.652	2,8136	5
On	3.631	2.875	3,3794	5

Rule 5: Trigger by clock (Job)

Name	Every 10 seconds first 2000 accounts get added 1 amount
Triggering Events	Every 10 seconds
Description	Every 10 seconds first 2000 accounts get added 1 amount
Complexity	$\mathcal{O}(2000)$

SQL

This is a time event so we used Jobs to create it.

```
1  BEGIN
2      DBMS_SCHEDULER.CREATE_JOB (
3              job_name => '"SYSTEM"."CLOCK_EVENT"',
4              job_type => 'PLSQL_BLOCK',
5              job_action => 'DECLARE
6  ␣␣␣␣accid␣NUMBER␣:=␣1;
7  BEGIN
8  WHILE␣accid<2000
9  LOOP
10 UPDATE␣account␣set␣amount␣=␣amount␣+␣1␣where␣accountID␣=␣accid;
11 accid:=accid+1;
12 END␣LOOP;
13 END;',
14             number_of_arguments => 0,
15             start_date => NULL,
16             repeat_interval => 'FREQ=SECONDLY;INTERVAL=10',
17             end_date => NULL,
18             enabled => FALSE,
19             auto_drop => FALSE,
20             comments => 'Every␣10␣seconds␣the␣first␣2000␣accounts␣get␣1␣amount');
21
22     DBMS_SCHEDULER.SET_ATTRIBUTE(
23              name => '"SYSTEM"."CLOCK_EVENT"',
24              attribute => 'logging_level', value => DBMS_SCHEDULER.LOGGING_OFF);
25
26     DBMS_SCHEDULER.enable(
27              name => '"SYSTEM"."CLOCK_EVENT"');
28 END;
```

To test the preformance we let it run while also updating 40000 accounts. The times displayed are measured running the following PLSQL while the job is enabled and disabled:

```
1  DECLARE
2      accid NUMBER := 1;
3  BEGIN
4  WHILE accid<40000
5  LOOP
6
7  UPDATE account set amount = amount + 1 where accountID = accid;
8
9  accid:=accid+1;
10 END LOOP;
11 END;
```

Times

	Max	Min	Avg	#
Off	15.075	10.932	13,3352	5
On	16.783	12.204	14,1958	5