Database and Information Systems

Michal Krátký & Radim Bača

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VŠB – Technical University of Ostrava

2020/2021

Content

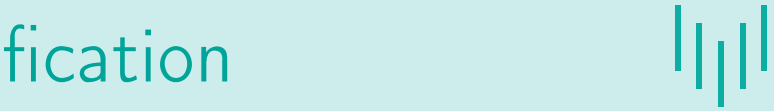
Trigger

- Trigger is a PL/SQL block which is automatically launched by an DML command such as Insert, Update or Delete.
- Generaly it is possible to launch a trigger also by some other operations (DML on views, DDL commands, system events).

Trigger, syntax

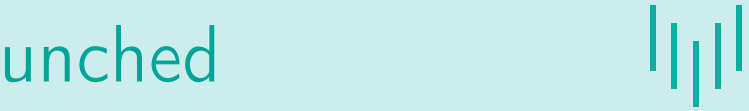
```
CREATE [OR REPLACE ] TRIGGER trigger_name
 {BEFORE | AFTER | INSTEAD OF }
 {INSERT [OR] | UPDATE [OR] | DELETE}
 [OF column_name]
ON table_name
 [REFERENCING OLD AS old_value NEW AS new_value]
 [FOR EACH ROW [WHEN (condition)]]
BEGIN
   commands
END;
```

Trigger syntax, SQL command specification



- Specification of the SQL operation launching the trigger: INSERT, UPDATE, DELETE.
- We can specify more operations.
- OF `column_ name` – the trigger is launched only during the attribute `column_name` update.
- ON `table_name` – we specify the table bounded with the trigger.

Trigger syntax, when the trigger is launched



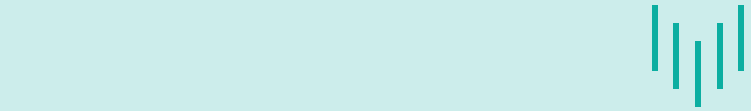
- A required part specifying when the trigger is launched:
- **BEFORE** - before the SQL command processing.
- **AFTER** - after the SQL command processing.
- **INSTEAD OF** - instead of the SQL command.

Trigger syntax, FOR EACH ROW

[**FOR EACH ROW** [**WHEN** (`condition`)]]

- Implicitly a trigger is launched only once for one command. However, this command can handle more records.
- This parameter specify that the trigger is launched for each row which is updated by the SQL command.
- We can specify a condition after **WHEN** saying when the trigger is launched.

Naming of the new and old variables



[**REFERENCING OLD AS** `old_value` **NEW AS** `new_value`]

- An optional parameter.
- Allows us to name old and new values of the record which the trigger manipulates.
- Implicitly they are named `:OLD` a `:NEW`.

Example

We store a record deleted from table `Student` in the table `Hist_stud`.

```
CREATE OR REPLACE TRIGGER del_student
BEFORE DELETE ON student
FOR EACH ROW
BEGIN
   INSERT INTO Hist_stud(login, name, surname)
     VALUES(:OLD.login, :OLD.name, :OLD.surname);
END;
```

Mutating table problem

- If we try to read or modify the same table we get the *mutating table error* (i.e. ORA-04091).
- We should avoid such a trigger but there are techniques which allows us this functionality.

Compound trigger

- In the case of a problematic trigger, e.g. a trigger working with the same table leading to the mutating table error, we can use a compound trigger[1]:

```
CREATE OR REPLACE TRIGGER compound_trigger
  FOR UPDATE OF salary ON employees COMPOUND TRIGGER
  -- Declarative part (optional)
  BEFORE STATEMENT IS ...
  BEFORE EACH ROW IS ...
  AFTER EACH ROW IS ...
  AFTER STATEMENT IS ...
END compound_trigger;
```

[1] https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#LNPLS2005

Automatic generation of primary key's value

In the previous versions of Oracle, we have used triggers and sequences for it.

```
CREATE TABLE "User" (
   idUser      INT GENERATED ALWAYS AS IDENTITY
     NOT NULL PRIMARY KEY,
   login       VARCHAR(10) NOT NULL UNIQUE,
   name        VARCHAR(20) NOT NULL,
   surname     VARCHAR(20) NOT NULL,
   address     VARCHAR(40));

INSERT INTO "User"(login, name, surname, address)
  VALUES('kra28', 'Michal', 'Sobota', 'Kopřivová 128,
Havířov');

SELECT * FROM "User";
-- 1  kra28  Michal  Sobota  Kopřivová 128, Havířov
```

Static PL/SQL

We can not call all available SQL commands in the PL/SQL block. Commands which we can call in the PL/SQL are called **static PL/SQL commands**:

- SELECT, INSERT, UPDATE, DELETE, MERGE
- LOCK TABLE, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION
- Evidently, commands which can not be directly called are all **DDL commands**.
- They have to be invoked (together with operations which they are not known in the time of compilation) as **dynamic PL/SQL**.

Dynamic PL/SQL

- Dynamic PL/SQL allows to compile and call any SQL command during the run-time.
- A **disadvantage** is that it is not possible to check the syntax of the command.
- We use the `EXECUTE IMMEDIATE` command to run the dynamic PL/SQL.
- **Warning: If it is possible to use the static PL/SQL use it instead of the dynamic PL/SQL!**
  - **Why?** They are some performance and security issues.

Dynamic PL/SQL, Example

In this example, we create and delete tables using the `EXECUTE IMMEDIATE` command.

```
DECLARE
   v_command VARCHAR2(50);
BEGIN
   EXECUTE IMMEDIATE 'Create table book ' ||
      '(id INT UNIQUE, name VARCHAR2(50), ' ||
      'author INT REFERENCES author(author_id))';
   v_command := 'DROP TABLE book';
   EXECUTE IMMEDIATE v_command;
END;
```

Dynamic PL/SQL, Issues

There are two issues of dynamic PL/SQL:

- Security issue – SQL injection (see a next lecture).
- Performance issue – it is related to query processing.

Query processing

Let us consider a situation when a database system sends thousands of SELECT commands such as:

```
SELECT fname, lname, address FROM Student
  WHERE login = 'kra228';
```

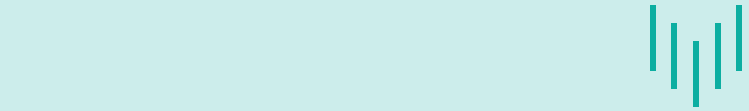
Query processing

The database system first checks whether the command was processed previously or not.

- *If it was send for the first time:*
  - Every query is parsed and the query plan is created[2].
  - The query can be processed using many different ways and system looks for the most efficient way.
  - This process takes a time and sometimes it can take longer than the query processing itself.
- *If the query was processed previously:* RDBMS uses the previously compiled query plan.

[2] See later lectures.

Query uniqueness



- When an DBMS checks whether the query was already processed or not, it compare the whole query string.
- Therefore, these two queries are not identical and the query evaluation plain is built again for the second query:

```
SELECT fname, lname, address FROM Student
  WHERE login = 'kra228';
SELECT fname, lname, address FROM Student
  WHERE login = 'fer452';
```

- This problem typically arise when the system is used by many users.
- We should be aware of that since the IS are not written for one user.

Bind variables

- Bind variables allows us to use previously created query plans for queries having only different values.
- Bind variables replace the original value of the query, for example:

```
SELECT fname, lname, address FROM Student
  WHERE login = :login;
```

- Queries have the same syntax even though we send different values.
- It reduces the query processing time and increases the throughput of the system.

Static PL/SQL

- Static PL/SQL automatically uses bind variables, for example:

```
CREATE OR REPLACE PROCEDURE
      loginIntoSystem(p_login IN VARCHAR)
AS
  num INT;
BEGIN
  SELECT COUNT(*) INTO num FROM Student
    WHERE login=p_login;
  ...
END;
/
```

- It means, each variable in static PL/SQL is automatically the bind variable.

Dynamic PL/SQL

- Unfortunately, it is not possible in the case of dynamic PL/SQL (operations are put in a string executed by `EXECUTE IMMEDIATE`).
- As result, a query created by a string concatenation means a low performance of query processing, for example:

```
CREATE OR REPLACE PROCEDURE
                updateClass(p_login IN VARCHAR)
AS
BEGIN
   EXECUTE IMMEDIATE
      'UPDATE Student class = class + 1
          WHERE login = ' || p_login;
   COMMIT;
END;
/
```

Dynamic PL/SQL

- In this case, the bind variables must be utilized using the keyword `USING`, for example:

```
CREATE OR REPLACE PROCEDURE
                    updateClass(p_login IN VARCHAR)
AS
BEGIN
   EXECUTE IMMEDIATE
      'UPDATE Student class = class + 1
               WHERE login = :x' USING p_login;
   COMMIT;
END;
/
```

- *Note:* the bind variable can be used only in the case of literals (like attribute values), in other cases, the attribute name, table name and so on, we must use the string concatenation.

Performance comparison

- *Task:* write an anonymous procedure, to print out the names of objects from table `ALL_OBJECTS` for objects with id 1 – 1000.

  Let us note that a common way is to write one SELECT instead of many SELECTs, but ...

- We will test the performance with and without bind variables.

Example, without bind variables 1/2

```
DECLARE
   TYPE rc IS REF CURSOR;
   v_rc rc;
   v_dummy ALL_OBJECTS.OBJECT_NAME%type;
   v_start NUMBER DEFAULT DBMS_UTILITY.GET_TIME;
BEGIN
   FOR i IN 1 .. 1000
   LOOP
      OPEN v_rc FOR
         'select object_name from all_objects
            where object_id = ' || i;
      FETCH v_rc INTO v_dummy;
      CLOSE v_rc;
      -- DBMS_OUTPUT.PUT_LINE(v_dummy);
   END LOOP;
```

Example, without bind variables 2/2

```
    DBMS_OUTPUT.PUT_LINE(round(
      (DBMS_UTILITY.GET_TIME-v_start)/100, 2) || ' s' );
END;
/
```

Remark: `DBMS_UTILITY.GET_TIME` returns a value of the counter which can be used to determine the time.

Example, with bind variables 1/2

```
DECLARE
   TYPE rc IS REF CURSOR;
   v_rc rc;
   v_dummy ALL_OBJECTS.OBJECT_NAME%type;
   v_start NUMBER DEFAULT DBMS_UTILITY.GET_TIME;
BEGIN
   FOR i IN 1 .. 1000
   LOOP
      OPEN v_rc FOR
         'select object_name from all_objects
            where object_id = :x' USING i;
      FETCH v_rc INTO v_dummy;
      CLOSE v_rc;
      -- DBMS_OUTPUT.PUT_LINE(v_dummy);
   END LOOP;
```

Example, with bind variables 2/2

```
    DBMS_OUTPUT.PUT_LINE(round(
      (DBMS_UTILITY.GET_TIME-v_start)/100, 2) || ' s' );
END;
/
```

Performance comparison

- Time without bind variables: 65.48s.
- Time with bind variables: 0.25s
- Clearly, if we do not use the bind variables we significantly reduce the performance! (And the probability of the security issue called SQL injection is higher.)

Bind variables in programming languages

- C# (ADO.NET) and Java (JDBC) also support the bind variables.
- They are called parametrized (or prepared) queries.
- These features can be valid also in other RDBMS.

Exceptions

- In PL/SQL, errors are handled by exceptions.
- The PL/SQL language has its own exception handling mechanism.
- An exception can occur in the Oracle server (an error of the SQL command processing) or it can be handled or invoked by PL/SQL code.

EXCEPTION part

- The `EXCEPTION` part in the PL/SQL block serves for the exception handling.

```
 ...
BEGIN
 ...
EXCEPTION
 WHEN exception_name THEN
 exception handling
END;
```

- In the case of an exception, the program automatically jumps to the exception part which handles it.

EXCEPTION part

```
 ...
BEGIN
 ...
EXCEPTION
 WHEN exception_name THEN
 exception handling
END;
```

- In the case of successful handling of an exception we do not propagate it to the methods which invoked this PL/SQL block.
- In the case that we want to handle any exception occurring in our code then we use the `OTHERS` keyword.

Exceptions of the STANDARD package

In this table we see some exceptions of the STANDARD package:

Exception name	Bug number	Description
ACCESS_INTO_NULL	ORA-06530	Attempt to assign a value into an uninitialized object
DUP_VAL_ON_INDEX	ORA-00001	Attempt to insert duplicity value
INVALID_CURSOR	ORA-01001	Invalid operation with cursor
INVALID_NUMBER	ORA-01722	Conversion from the number into string failed
NO_DATA_FOUND	ORA-01403	SELECT command did not return any data
TOO_MANY_ROWS	ORA-01422	SELECT command returned more then one row
VALUE_ERROR	ORA-06502	Invalid manipulation with value

Exception handling

The following procedure print out the message 'Value of the login must be unique!' in a case of an DUP_VAL_ON_INDEX exception. In the case of another exception it prints out its error message.

```
BEGIN
  INSERT INTO Student(login, fname, lname)
    VALUES('bon007', 'James', 'Bond');
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.put_line('Value of the login must be unique!');
  WHEN OTHERS THEN
    DBMS_OUTPUT.put_line(DBMS_UTILITY.FORMAT_ERROR_STACK);
END;
```

Exception raise

- PL/SQL allows to invoke an exception in the case of an error.
- The `RAISE` keyword can be used for that.
- It is possible to raise the standard or user defined exception.

User defined exception

- We can define an exception in the definition part of the PL/SQL block.
- Therefore, the exception is defined together with local variables or cursors as follows:
  `exception_name EXCEPTION;`

  Exception visibility:
  - The visibility of the exception is limited only on a procedure.
  - If we want to handle an exception defined outside of the procedure we have to use the `OTHERS` clause.
  - Or we should use **packages**.

Exception

We raise the `too_many_records` exception which is not handled in the procedure. Therefore, this exception is propagated into the method of the caller.

```
DECLARE
   too_many_records EXCEPTION;
   v_records INT;
BEGIN
   SELECT count(*) INTO v_records FROM Student;
   IF v_records > 20 THEN
      RAISE too_many_records
   ELSE
      INSERT INTO Student (login, fname, lname)
         VALUES('bon007', 'James', 'Bond');
   END IF;
END;
```

References

- Oracle Portal:
  `https://docs.oracle.com/en/database/oracle/oracle-database/21/books.html`:
  - PL/SQL Language Reference
  - PL/SQL Packages and Types Reference
- Bind variables - The key to application performance, `http://www.akadia.com/services/ora_bind_variables.html`, 2010.