UNIVERSITÉ LUMIÈRE LYON 2

Département d'Informatique et de Statistique
DIS
Institut de la Communication

Database programming

M1 Informatique
Year 2015-2016
Jérôme Darmont

http://eric.univ-lyon2.fr/~jdarmont/

DBprog news

http://eric.univ-lyon2.fr/~jdarmont/?page_id=451

http://eric.univ-lyon2.fr/~jdarmont/?feed=rss2

https://twitter.com/darmont_lyon2 *hashtag* #dbprog

Outline

- Introduction
- Variable and constant declaration
- Operators and control structures
- Abstract data types
- Cursors
- Exceptions
- Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Motivation for DB programming

- SQL: query language for relational databases
  - Standard
  - Query optimizers available
  - Low-level
- Programming is needed for:
  - Complex tasks
  - User interfaces

Using SQL queries in a program

- Embedded SQL: SQL queries are incorporated in the source code of a programming language (PL/SQL, T-SQL, PL/pgSQL, Pro*C…).
- Application programming interfaces: SQL queries are allowed by a set of functions of the programming language (Java Persistence API, PHP Data Objects…).
- Call level interfaces: middleware between programming language and DBMS (ODBC, JDBC, ADO…)
- Stored procedures: SQL functions stored in database and executed by the DBMS (written in PL/SQL, T-SQL, PL/pgSQL)

Using SQL queries in a program

- Embedded SQL
- APIs
- Call level interfaces
- Stored procedures

} Unique principle: cursors

- Our choice of programming language :

Oracle PL/SQL

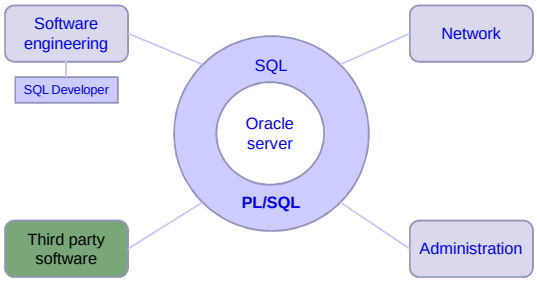
PL/SQL features

- 4th generation procedural language
- Extension of SQL
- Declaration of variables and constants
- Abstract data types (collections, records, objects)
- Modularity (subprograms, packages)
- Run-time error management (exceptions)
- Tight interaction with Oracle/SQL (same data types)

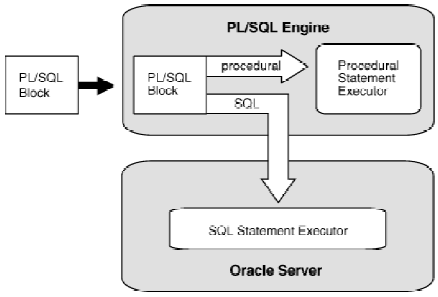
PL/SQL features

- Support of SQL, including dynamic SQL
- Support of object-oriented programming
- Performance (batch processes)
- Productivity (uniformity of all tools)
- Portability (on all Oracle systems)
- Security (stored procedures, triggers)

Oracle architecture



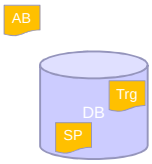
PL/SQL engine



Oracle Database PL/SQL User's Guide and Reference

Types of PL/SQL blocks

- Anonymous block
  - Stored out of the database (file)
  - Compiled and executed on-the-fly
- Stored procedure
  - Separately compiled
  - Permanently stored in the database
- Trigger
  - Stored procedure associated with a given table
  - Automatic execution when an event occurs



PL/SQL block structure

```
[DECLARE
     -- Types, constants and variables]
BEGIN
     -- PL/SQL statements
[EXCEPTION
     -- Error management]
END;
```

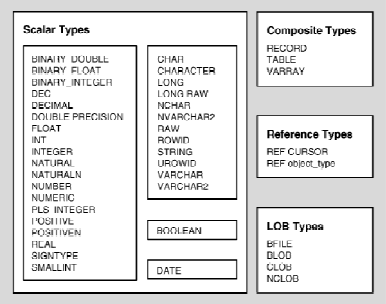
Outline

- ✓ Introduction
- Variable and constant declaration
- Operators and control structures
- Abstract data types
- Cursors
- Exceptions
- Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Variables & constants

- Declaration in PL/SQL block DECLARE section
- Variables
  ex. birth_date DATE;
  counter INTEGER := 0; -- Initialization
  counter2 INTEGER DEFAULT 0; -- Default value
  id CHAR(5) NOT NULL := 'AP001';
- Constants
  ex. vat CONSTANT REAL := 0.2;

Data types



Oracle Database PL/SQL User's Guide and Reference

Referencing an existing type

- Type of another variable
  ex. credit REAL;
  debit credit%TYPE;
- Type of a table attribute
  ex. emp_num EMP.EMPNO%TYPE;
- Type of a table tuple (row)
  ex. my_student STUDENT%ROWTYPE;

Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- Operators and control structures
- Abstract data types
- Cursors
- Exceptions
- Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Affectation operators

- Simple affectation

  ex. n := 0;
  n := n + 1;
- Value(s) from a database

  ex. SELECT custname INTO name
  FROM customer WHERE custnum = 10;

  SELECT ename, sal INTO name, salary
  FROM emp WHERE empno = 5000;

Expressions and comparisons

- Arithmetic operators: + - / * **
- Concatenation operator: ||
- Comparison operators
  - = < > <= >= <>
  - IS NULL, LIKE, BETWEEN, IN
- Logical operators: AND, OR, NOT

Control structures: Selection

IF-THEN, IF-THEN-ELSE or IF-THEN-ELSIF

```
IF condition1 THEN
   -- Statements
[ELSIF condition2 THEN
   -- Statements]
[ELSE
   -- Statements]
END IF;
```


Control structures: Iteration

- *For* loop
  FOR iterator IN [REVERSE] min..max LOOP
  -- Statements
  END LOOP;
- *While* loop
  WHILE condition LOOP
  -- Statements
  END LOOP;
- *Repeat until* loop
  LOOP
  -- Statements
  EXIT WHEN condition;
  END LOOP;

Screen output

DBMS_OUTPUT.PUT('string'); /* No CR */
DBMS_OUTPUT.PUT_LINE('string'); /* With CR */

```
DBMS_OUTPUT.PUT('Hello');
DBMS_OUTPUT.PUT_LINE('name='|| name);
DBMS_OUTPUT.PUT_LINE('num='|| TO_CHAR(num));
DBMS_OUTPUT.PUT_LINE('num='|| num);
```

NB: To make screen output work, environment variable SERVEROUTPUT must be set to ON.

`SET SERVEROUTPUT ON` under SQL Developer

In case of overflow, the size of the display buffer must be increased.

Ex. `DBMS_OUTPUT.ENABLE(10000);`

Sample anonymous block

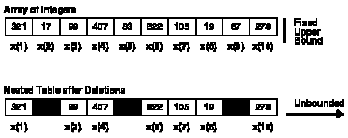
```
-- Computation of tax-included price
DECLARE
      vat CONSTANT REAL := 0.2;
      price product.rawprice%TYPE;
BEGIN
      -- Affectation
      SELECT rawprice INTO price FROM product
      WHERE prodcode = 'Pr345blue';
      -- Add in VAT
      price := price * (1 + vat);
      -- Screen output
      DBMS_OUTPUT.PUT_LINE(price || ' euros');
END;
```


Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and control structures
- Abstract data types
- Cursors
- Exceptions
- Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Collections

- Definition: Ordered set of elements of identical type. Each element bears an index that determines its position in the collection.
- Two types of collections:
  - Array (VARRAY): bounded size, dense
  - Table (TABLE): extensible, non-dense



Oracle Database PL/SQL User's Guide and Reference

Declaring and using collections

1. Declare a type collection

   ex. TYPE TableChar IS TABLE OF VARCHAR(20);
   TYPE ArrayInt IS VARRAY(10) OF INTEGER;

2. Declare a variable of that type & initialize it

   ex. my_table TableChar := TableChar('Aa', 'Bb', 'Cc');
   my_array ArrayInt := ArrayInt();

- NB: A collection may be initialized empty. It is not mandatory to initialize all elements of an array.

Collection affectation

- Whole collections

```
ex.  DECLARE    TYPE T1 IS TABLE OF INT;
                TYPE T2 IS TABLE OF INT;
                et11 T1 := T1(1, 2, 3, 4);
                et12 T1 := T1(5, 6);
                et2 T2 := T2();
     BEGIN      et12 := et11;    -- Legal
                et2 := et11;     -- Illegal
     ...
```

- Collection elements

  ex. et11(1) := 10;

Collection manipulation

- Set of predefined methods (≈ procedures)
  Usage: col_name.method_name[(parameters)]
- EXISTS(i) returns TRUE if the i^{th} element exists in the collection.
- COUNT returns the number of elements in the collection.
- LIMIT returns the maximum size of the collection (NULL for tables).
- EXTEND(n) increases collection size by n elements. EXTEND is equivalent to EXTEND(1).

Collection manipulation

- TRIM(n) removes n elements from the end of the collection (collection size decreases accordingly). TRIM is equivalent to TRIM(1).
- DELETE(i) and DELETE delete the i^{th} element and all elements from the collection, respectively (tables only).
- FIRST and LAST return the index of the first and last element in the collection, respectively.
  NB: FIRST = 1 et LAST = COUNT in an array.
- PRIOR(n) and NEXT(n) return the index of the previous and next element of element of index n, respectively.

Sample collection manipulation block

```
DECLARE
      TYPE IntList IS TABLE OF INTEGER;
      stack IntList := IntList();
      element INTEGER;
BEGIN
      -- Add on top of stack (push)
      stack.EXTEND;
      stack(stack.COUNT) := 1;
      stack.EXTEND;
      stack(stack.COUNT) := 11;
      -- Delete on top of stack (pop)
      element := stack(stack.COUNT); -- element = 11
      stack.TRIM;
```

Records

- Definition: Set of logically related data stored in fields.

1. Declare a type record

ex.
```
TYPE Student IS RECORD(
        student_num INTEGER,
        name VARCHAR(50),
        age INTEGER );
```

2. Declare a variable of that type

ex. a_student Student;

Record affectation

- Direct reference

ex.
```
a_student.student_num := 12212478;
a_student.name := 'Toto';
a_student.age := 6;

a_student := my_student; -- declared as STUDENT%ROWTYPE
```

- Query result

ex.
```
SELECT student_number, student_name, student_age
INTO a_student
FROM STUDENT
WHERE student_number = 12212478;
```


Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and control structures
- ✓ Abstract data types
- Cursors
- Exceptions
- Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Cursors

- Definition: Data structure that stores the result of a query returning several tuples (rows).
- Declaration: CURSOR cursor_name IS SQL_query;

ex.
```
CURSOR vat IS
    SELECT prod_num, price * 1.2 taxinc_price
    FROM product;
```

NB : A cursor tuple is of type vat%ROWTYPE.

Cursor usage

```
-- Simple (complete) traversal
DECLARE
      CURSOR vat IS
      SELECT prod_num, price * 1.2 taxinc_price
      FROM product;
      tuple vat%ROWTYPE;
BEGIN
      FOR tuple IN vat LOOP
            DBMS_OUTPUT.PUT_LINE(
                  tuple.prod_num
                  || ' : ' ||
                  tuple.taxinc_price);
      END LOOP;
```


Cursor usage

```
-- Ad-hoc traversal
DECLARE
      ...
BEGIN
      OPEN vat;
      FETCH vat INTO tuple;          -- 1st row
      WHILE vat%FOUND LOOP
            ...
            FETCH vat INTO tuple;    -- Next row
      END LOOP;
      CLOSE vat;
END;
```

Cursor attributes

- %NOTFOUND equals to FALSE if FETCH returns a result.
- %FOUND is the logical opposite of %NOTFOUND.
- %ROWCOUNT returns the number of rows read.
- %ISOPEN equals to TRUE if cursor is open.

Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and control structures
- ✓ Abstract data types
- ✓ Cursors
- Exceptions
- Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Exceptions

- When a run-time error occurs, an exception id raised (executed). Exceptions are handled in separate routines.
- Advantages
  - Systematic error management
  - Grouped management of similar errors
  - Code readability (error management is out of main code)
- PL/SQL functions for error management
  - SQLCODE: Code of the last raised exception
  - SQLERRM: Associate error message

System exceptions

Name	Error code	SQLCODE
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	-1403
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Personalized exceptions

- Declare exception (DECLARE section)
  exception_name EXCEPTION;
- Raise (fire) exception (BEGIN section)
  IF condition THEN
  RAISE exception_name;
  END IF;
- Handle exception (EXCEPTION section)
  WHEN exception_name THEN …;

Sample exception

```
DECLARE
      c INTEGER;
      nobody EXCEPTION;
BEGIN
      SELECT COUNT(*) INTO c FROM EMP;
      IF c = 0 THEN
            RAISE nobody;
      END IF;
EXCEPTION
      WHEN nobody THEN
         RAISE_APPLICATION_ERROR(-20501,'Error!');
END; -- Error code between –20999 and -20001
```

Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and control structures
- ✓ Abstract data types
- ✓ Cursors
- ✓ Exceptions
- Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Procedures

```
-- Procedure definition

PROCEDURE proc_name (param1, param2…) IS
    -- Local declarations
BEGIN
    -- Statements
[EXCEPTION
    -- Exception handling]
END;
```


Functions

```
-- Function definition

FUNCTION function_name (param1, param2…)
RETURN return_value_type IS
   -- Local declarations
BEGIN
   -- Statements
   RETURN return_value;
[EXCEPTION
   -- Exception handling]
END;
```


Declaration and parameterization

- Declaration: Any subprogram must be defined before being called.
  ⇒ definition in the DECLARE section if a PL/SQL block
- Parameter definition and mode

  param_name [IN | OUT | IN OUT] TYPE
  ex. result OUT REAL
  - IN: Input (read only / by value) parameter
  - OUT: Output (write only / by reference) parameter
  - IN OUT: Input-output (read-write / by reference) parameter

Sample procedure

```
PROCEDURE Conversion_FF_Euro (priceF IN REAL,
                              priceE OUT REAL) IS
      euro CONSTANT REAL := 6.55957;
      nullprice EXCEPTION;
BEGIN
      IF priceF IS NOT NULL THEN
            priceE := priceF / euro;
      ELSE
            RAISE nullprice;
      END IF;
EXCEPTION
      WHEN nullprice THEN
            DBMS_OUTPUT.PUT_LINE('Cannot compute!');
END;
```


Sample (recursive) function

```
-- Let's compute n!

FUNCTION facto (n INTEGER) RETURN INTEGER IS

BEGIN
      IF n = 1 THEN     -- Stop condition
            RETURN 1;
      ELSE              -- Recursive call
            RETURN n * facto(n - 1);
      END IF;
END;
```

Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and control structures
- ✓ Abstract data types
- ✓ Cursors
- ✓ Exceptions
- ✓ Subprograms
- Stored procedures and packages
- Triggers
- Dynamic SQL

Stored procedures

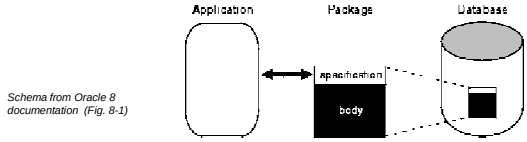
- Definition: Precompiled procedures permanently stored in the database
- Creation
  CREATE PROCEDURE ProcName (Parameters) AS ...
  ex.
```
CREATE PROCEDURE HelloWorld AS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World !');
END;
/
```
- Execution

	under SQL Developer	in PL/SQL
ex.	EXECUTE HelloWorld	HelloWorld;

- Suppression
  ex. DROP PROCEDURE HelloWorld;

Packages

- Definition: Set of types, cursors, variables et subprograms that are logically related and permanently stored together
- A package is subdivided into two parts:
  - Specification: interface (public declarations),
  - Package body: private declaration and code.



Schema from Oracle 8 documentation (Fig. 8-1)

Packages

```
-- Package specification definition
CREATE [OR REPLACE] PACKAGE package_name AS
      [-- Type definitions]
      [-- Cursor specifications]
      [-- Variable declarations]
      [-- Subprograms declarations]
END;

-- Package body definition (optional)
CREATE [OR REPLACE] PACKAGE BODY package_name AS
      [-- Type definitions]
      [-- Cursor specifications]
      [-- Variable declarations]
      [-- Subprograms declarations]
END;
```


Packages

```
-- Sample specifications
CREATE OR REPLACE PACKAGE Employees AS
   TYPE EmpTuple IS RECORD (ename emp.ename%TYPE,
                            salary emp.sal%TYPE);
   CURSOR desc_salary RETURN EmpTuple;
   PROCEDURE hire (
      empno  NUMBER,
      ename  VARCHAR,
      job    VARCHAR,
      mgr    NUMBER,
      sal    NUMBER,
      comm   NUMBER,
      deptno NUMBER);
   PROCEDURE fire (emp_id NUMBER);
END;
```


Packages

```
-- Sample body
CREATE OR REPLACE PACKAGE BODY Employees AS
   CURSOR desc_salary RETURN EmpTuple IS
     SELECT empno, sal FROM emp ORDER BY sal DESC;
   PROCEDURE hire (empno  NUMBER,
      ename  VARCHAR, job    VARCHAR,
      mgr    NUMBER,  sal    NUMBER,
      comm   NUMBER,  deptno NUMBER) IS
   BEGIN
      INSERT INTO emp VALUES (empno, ename, job,
         mgr, SYSDATE, sal, comm, deptno);
   END;
   PROCEDURE fire (emp_id NUMBER) IS
   BEGIN
      DELETE FROM emp WHERE empno = emp_id;
   END;
END;
```

Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and control structures
- ✓ Abstract data types
- ✓ Cursors
- ✓ Exceptions
- ✓ Subprograms
- ✓ Stored procedures and packages
- Triggers
- Dynamic SQL

Triggers

- Definition: Stored procedure associated to a given table and automatically fired by events related to actions on the table.
- Triggers complement integrity constraints by allowing to apply complex integrity rules. They are features of active databases.

Types of triggers

	Insertion	Deletion	Update
Before	#1	#2	#3
After	#4	#5	#6

…of a database table

Trigger definition

```
CREATE [OR REPLACE] TRIGGER trig_name
   BEFORE | AFTER
   INSERT | DELETE | UPDATE
| [INSERT] [[OR] DELETE] [[OR] UPDATE]
   ON table_name
   [FOR EACH ROW]
   -- PL/SQL block coding the actions
   -- to process
```


Trigger definition

- Specific variables
  - :NEW.attribute_name : Attribute value after update

    Ex. INSERT INTO CUSTOMER (1, 'NewCust');
    :NEW.CustNum takes value 1.
    :NEW.Name takes value 'NewCust'.
  - :OLD.attribute_name : Attribute value before update

    Ex. DELETE FROM CUSTOMER WHERE CustNum = 1;
    :OLD.CustNum takes value 1.

Sample trigger

```
-- Primary key check on table customer
CREATE OR REPLACE TRIGGER customer_pk
BEFORE INSERT OR UPDATE ON customer
FOR EACH ROW
DECLARE
      n INTEGER;
      key_exists EXCEPTION;
      key_null EXCEPTION;
BEGIN
      -- Is new key null?
      IF :NEW.custnum IS NULL THEN
            RAISE key_null;
      END IF;
```

Sample trigger

```
      -- Does new key already exist?
      SELECT COUNT(custnum) INTO n FROM customer
            WHERE custnum = :NEW.custnum;
      IF n > 0 THEN
            RAISE key_exists;
      END IF;
EXCEPTION
      WHEN key_exists THEN
            RAISE_APPLICATION_ERROR(-20501,
            'Primary key already used!');
      WHEN key_null THEN
            RAISE_APPLICATION_ERROR(-20502,
            'Primary key must have non-null value!');
END;
```


Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and control structures
- ✓ Abstract data types
- ✓ Cursors
- ✓ Exceptions
- ✓ Subprograms
- ✓ Stored procedures and packages
- ✓ Triggers
- Dynamic SQL

Static SQL vs. dynamic SQL

- Examples
  - Stored procedure that updates table EMP
    ⇒ static SQL (query is known at compilation time)
  - Stored procedure that updates a table whose name is a parameter
    ⇒ dynamic SQL (query is not known at compilation time)
- Definition: Building an SQL query on-the-fly in a PL/SQL block

Dynamic query execution

- Dynamic SQL is mandatory for DDL queries that alter database structure (CREATE, DROP, etc.)
- Execution statement

```
EXECUTE IMMEDIATE query
[INTO var1, var2...]
[USING [IN | OUT | IN OUT] param1
            [, [IN | OUT | IN OUT] param2]...
];
```


Sample dynamic queries

```
DECLARE
      q VARCHAR(100);
      my_deptno dept.deptno%TYPE := 50;
      my_dname dept.dname%TYPE := 'Staff';
      my_loc dept.loc%TYPE := 'Lyon';
      emp_tuple emp%ROWTYPE;
      table_name VARCHAR(15) := 'dept';

BEGIN
      -- Parameterized query
      q := 'INSERT INTO dept VALUES (:p1, :p2, :p3)';
      EXECUTE IMMEDIATE q USING my_deptno, my_dname,
                              my_loc;
```


Sample dynamic queries

```
-- Result storage
q := 'SELECT * FROM emp WHERE empno = :id';
EXECUTE IMMEDIATE q INTO emp_tuple USING 5;

-- Without a query variable
EXECUTE IMMEDIATE
      'DELETE FROM dept WHERE deptno = :n'
       USING my_deptno;

-- Using the concatenation operator
EXECUTE IMMEDIATE
      'DELETE FROM ' || table_name ||
      ' WHERE deptno = ' || my_deptno;
```

Dynamic cursors

```
-- Example
DECLARE
       TYPE DynCurs IS REF CURSOR;       -- Type pointer
       emp_cv DynCurs;                   -- Dynamic cursor
       name emp.ename%TYPE;
       salary emp.sal%TYPE := 10000;
BEGIN
       OPEN emp_cv FOR                   -- Open cursor
         'SELECT ename, sal FROM emp
          WHERE sal > ' || salary;
       FETCH emp_cv INTO name, salary;
       WHILE emp_cv%FOUND LOOP
             -- Statements
             FETCH emp_cv INTO name, salary;
       END LOOP;
       CLOSE emp_cv;                     -- Close cursor
```


Outline

- ✓ Introduction
- ✓ Variable and constant declaration
- ✓ Operators and cont[illegible]ructu[illegible]
- ✓ Abstract data typ[illegible]
- ✓ Cursors
- ✓ Exceptions
- ✓ Subprograms
- ✓ Stored procedures [illegible]d packa[illegible]
- ✓ Triggers
- ✓ Dynamic SQL