Data Advanced

H5. PL SQL Introductie

Koen Bloemen

Sander De Puydt

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

H5. PL SQL Introductie

PL SQL combineert SQL met **p**rocedural **l**anguage. Het is een programmeertaal die veel meer functionaliteit op tafel brengt voor SQL (voorwaardes, lussen, functies, ...).

- Herbruikbare code
  - We gebruiken het om triggers te schrijven (die hergebruikt worden)
- Foutafhandeling
- Dynamic SQL: dynamisch SQL statements opbouwen en uitvoeren
- ...

H5. PL SQL Introductie

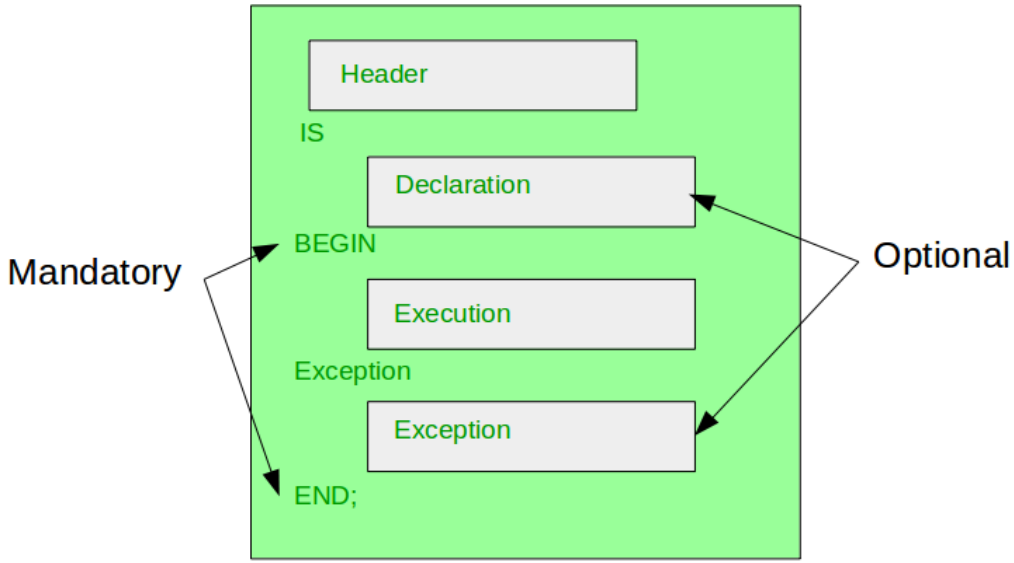
- Block Structure
- Data Types en Variables
- Operatoren
- IF statement

Block Structure

- BEGIN + END
- DECLARE
- EXCEPTION

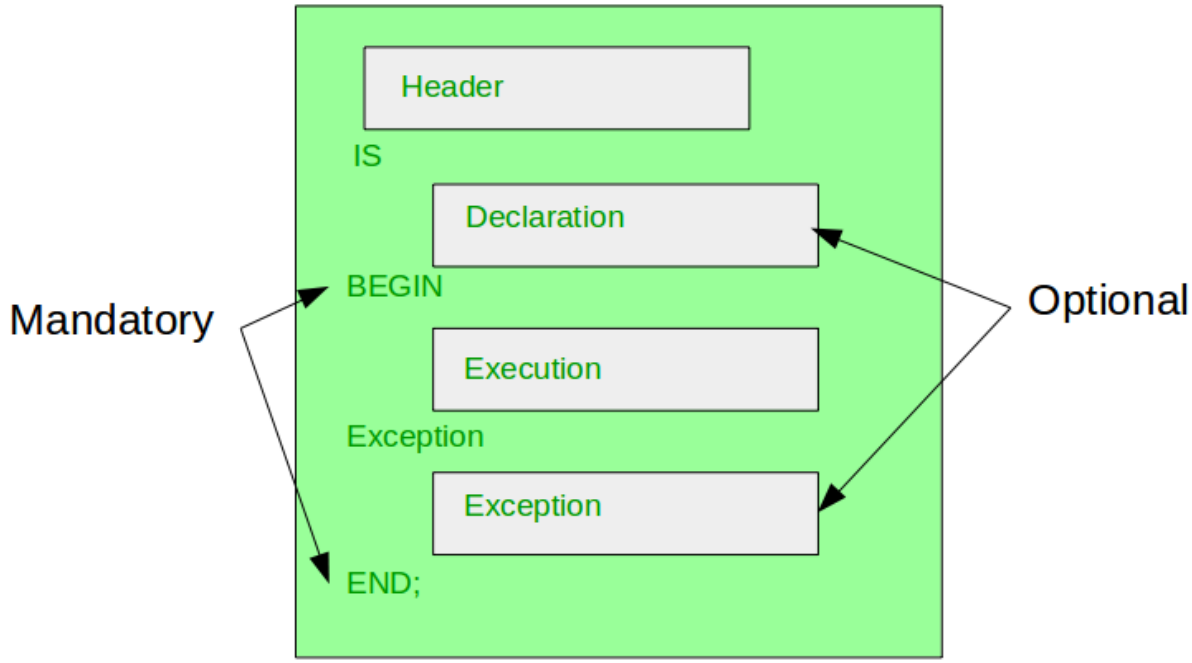
Block Structure

- Een PL SQL block bestaat uit drie delen:
  - Een **declaratie** sectie (optioneel)
  - Een **executie** sectie
  - Een **exception** sectie (optioneel)
- De executie sectie moet altijd aanwezig zijn.



Block Structure – BEGIN + END

- De executie sectie start altijd met BEGIN en **moet** eindigen met END.
- Het moet minstens één uitvoerbaar statement bevatten (wat ook het NULL command mag zijn).



Block Structure – DECLARE

- Met DECLARE kunnen we variabelen aanmaken. Dit onderdeel is optioneel.
- We zien later in dit hoofdstuk variabelen en datatypes die we in de DECLARE sectie kunnen gebruiken.

- Om het "Hello World"-voorbeeld in PL SQL uit te voeren, gaan we eerst serveroutput aan zetten.

> *SET SERVEROUTPUT ON*

- Zonder de serveroutput in te schakelen zien we het resultaat van de *dbms_output.put_line()* methode niet

Block Structure – DECLARE

- Met DECLARE kunnen we variabelen aanmaken. Dit onderdeel is optioneel.

"Hello World"-voorbeeld:

```
DECLARE
    message varchar2(20):= 'Hello World';
BEGIN
    dbms_output.put_line(message);
END;
/
```

Block Structure – EXCEPTION

- In PL SQL worden alle errors behandeld als exception
  - Hier zijn enkele vaak voorkomende fouten: https://docs.oracle.com/cd/E35622_01/html/821-1225/common-error-codes.html
  - Hier kan je een volledig overzicht van alle fouten vinden: https://www.dba-oracle.com/t_error_code_list.htm
  - Je kan echter ook de fouten op een natuurlijke manier ontdekken door de output te lezen van je code.

Block Structure – EXCEPTION

- Voorbeeld van Errors ontdekken die je kan opvangen:

```
DECLARE
   l_name medewerkers.NAAM%TYPE;
BEGIN
   -- als mnr = 7369, dan output: CASPERS
   SELECT NAAM INTO l_name FROM Medewerkers WHERE mnr = 7369;
   dbms_output.put_line('Employee name is ' || l_name);
END;
/
```

Block Structure – EXCEPTION

- Voorbeeld van Errors ontdekken die je kan opvangen: NO DATA FOUND

```
DECLARE
   l_name medewerkers.NAAM%TYPE;
BEGIN
   /* als mnr = 1, dan:
   *  Error report -
   *  ORA-01403: no data found
   */
   SELECT NAAM INTO l_name FROM Medewerkers WHERE mnr = 1;
   dbms_output.put_line('Employee name is ' || l_name);
END;
/
```

Block Structure – EXCEPTION

- Voorbeeld van Errors ontdekken die je kan opvangen:
  - ORA-01403 NO DATA FOUND is een voorgedefinieerde fout.
  - De code "dbms_output.put_line('Employee name is ' || l_name);" is nooit uitgevoerd, omdat de exception getriggered is.
  - We kunnen deze fout opvangen door "WHEN NO_DATA_FOUND" toe te voegen als voorwaarde aan de EXCEPTION sectie

Block Structure – EXCEPTION

- Voorbeeld van NO DATA FOUND opvangen in EXCEPTION sectie:

```
DECLARE
   l_name medewerkers.NAAM%TYPE;
BEGIN
   -- als mnr = 7369, dan output: CASPERS
   SELECT NAAM INTO l_name FROM Medewerkers WHERE mnr = 7369;
   dbms_output.put_line('Employee name is ' || l_name);
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         dbms_output.put_line('Customer ' || l_customer_id || ' does not exist');
END;
/
```

Block Structure

- In de PL SQL block structure kunnen we commentaar toevoegen met de volgende symbolen:

- Single line: --
- Multi line: /* ... */

```
DECLARE
    -- variabelen aanmaken
    message varchar2(20):= 'Hello World';
BEGIN
    /*
    *   PL SQL executable statements
    */
    dbms_output.put_line(message);
END;
/
```

Data Types en Variables

- Data Types
- Declaratie
- Initialisatie

Data Types en Variables – Data Types

- Er zijn vier soorten van scalar data types in pl sql:
  - Getallen
  - Boolean
  - Characters
  - Datetime
- PL SQL data types omvatten ook de **SQL data types**, zoals: number en binary_float
- Er zijn ook **PL SQL specifieke** data types, zoals **PLS_INTEGER** dat een 32-bit integer representeert.

Data Types en Variables – Data Types

- Nummerische data types:

Data Type	Beschrijving
PLS_INTEGER	Een geheel getal van 32 bits
BINARY_FLOAT	Kommagetal van 32 bits
BINARY_DOUBLE	Kommagetal van 64 bits
NUMBER(prec, scale)	Een getal met een gegeven precision en scale: • Precision: het aantal cijfers dat het getal groot is • Scale: De hoeveelheid getallen na de komma NUMBER(5,2) kan bijvoorbeeld 123,45 zijn.

Data Types en Variables – Data Types

- Character data types:

Data Type	Beschrijving
CHAR	Fixed-length tekst
VARCHAR2	Een tekst van variabele lengte
NCHAR	Fixed-length tekst die karakters kan bevatten van andere schriften
NVARCHAR2	Een tekst van variabele lengte die karakters kan bevatten van andere schriften

Data Types en Variables – Data Types

- Boolean data type:

Data Type	Beschrijving
BOOLEAN	Het logische data type dat TRUE, FALSE of NULL kan zijn **Let op!** Aangezien SQL geen data type heeft equivalent aan BOOLEAN kan je de volgende dingen niet doen met BOOLEAN: • Het gebruiken in een SQL statement • Het gebruiken in Built-in SQL functions, zoals TO_CHAR

- DataTime data type:

Data Type	Beschrijving
DATE	Een datetime die achterliggend de dag opslaat en de hoeveelheid seconden na middernacht

Data Types en Variables – Declaratie

- Wanneer je een declaratie doet van een variabele, dan gebruik je de volgende syntax:

Variable_name [CONSTANT] datatype;

```
DECLARE
   customer_name VARCHAR2(25);
   price NUMBER(10,2);
BEGIN
   NULL;
END;
/
```

Data Types en Variables – Declaratie

- Je kan variabelen ook het datatype geven van een specifieke kolom met *[schema_name].table_name.column_name%TYPE*

```
DECLARE
   employee_name student.medewerkers.naam%TYPE;
   employee_birth_date student.medewerkers.gbdatum%TYPE;
BEGIN
   NULL;
END;
/
```

Data Types en Variables – Initialisatie

- Een initialisatie doe je met het ":=" toewijzingssymbool.

Variable_name [CONSTANT] datatype := value;

```
DECLARE
   customer_name VARCHAR2(25) := 'Guido Gezelle';
   price NUMBER(10,2) := 99999,99;
   -- declaratie voor constante
   pi CONSTANT NUMBER := 3.141592654;
BEGIN
   NULL;
END;
/
```

Data Types en Variables – Initialisatie

- Je kan waardes uit een tabel opslaan in een variabelen met *SELECT INTO*

```
DECLARE
   employee_name student.medewerkers.naam%TYPE;
   employee_birth_date student.medewerkers.gbdatum%TYPE;
BEGIN
   SELECT naam, gbdatum INTO employee_name, employee_birth_date
   FROM student.medewerkers
   WHERE MNR = '7369';
   DBMS_OUTPUT.PUT_LINE('naam = ' || employee_name || ' birth = ' || employee_birth_date);
END;
/
```

Operatoren

- Rekenkundig
- Relationeel
- Vergelijkend
- Logisch

Operatoren – Rekenkundig

- De gekende rekenkundige operatoren zijn nog steeds bruikbaar in PL SQL.

Operator	Beschrijving
+	optelling
-	aftrekking
*	vermenigvuldiging
/	deling
**	Exponent operator voor een machtsverheffing

Operatoren – Relationeel

- Je kan zowel != als <> gebruiken om te controleren of twee waardes niet hetzelfde zijn.

Operator	Beschrijving
=	Test gelijkheid tussen twee variabelen
!= Of <>	Test of twee variabelen niet hetzelfde zijn
>	Test of linkse operator groter is dan de rechtse operator
<	Test of linkse operator kleiner is dan de rechtse operator
>=	Test of linkse operator groter is dan of gelijk is aan de rechtse operator
<=	Test of linkse operator kleiner is dan of gelijk is aan de rechtse operator

Operatoren – Vergelijkend

- Het resultaat van een vergelijkende operator is altijd: TRUE, FALSE of NULL

Operator	Beschrijving
LIKE	Vergelijkt een waarde of patroon, zoals 'Mehmet Ali' LIKE 'M% A_I' geeft TRUE terug.
BETWEEN … AND …	Vergelijkt of een waarde tussen twee limieten ligt
IN	Test of een waarde deel is van een set
IS NULL	Test of een waarde NULL is

Operatoren – Logisch

- Een logische operator werkt met BOOLEAN waardes en geeft een BOOLEAN waarde als resultaat

Operator	Beschrijving
AND	Controleert of beide operanden waar zijn
OR	Controleert of minstens één operand waar is
NOT	Keert de waarde om van TRUE naar FALSE en andersom

IF statement

- IF THEN
- ELSE en ELSIF
- CASE
- Nested

Operatoren – IF THEN

- Je kan een if statement schrijven met de volgende syntax:

IF voorwaarde THEN
statements;

END IF;

Voorbeeld:

```
BEGIN
   IF 1 != 0 THEN
      dbms_output.put_line('ongelijk');
   END IF;
END;
/
```

Operatoren – ELSE ELSIF

- Indien je voorwaarde niet waar is, dan kan je dit gedrag opvangen met ELSE

IF voorwaarde THEN
statements;
ELSE
else_statements;
END IF;

```
BEGIN
   IF 1 != 0 THEN
      dbms_output.put_line('ongelijk');
   ELSE
      dbms_output.put_line('gelijk');
   END IF;
END;
/
```

Operatoren – ELSE ELSIF

- Indien je meerdere voorwaardes wil controleren, dan doe je dit met ELSIF

IF condition_1 THEN
*   statements_1*
ELSIF condition_2 THEN
*   statements_2*
[ELSIF condition_3 THEN
*   statements_3] ...*
[ELSE
*   else_statements]*
END IF;

```
BEGIN
   IF 1 < 0 THEN
      dbms_output.put_line('kleiner');
   ELSIF 1>0 THEN
      dbms_output.put_line('groter');
   ELSE
      dbms_output.put_line('gelijk');
   END IF;
END;
/
```

Operatoren – CASE

- Er zijn twee soorten van CASE statements die je kan gebruiken: een **simple case** en een searched case.
- Een simple case gebruikt een selector variabele en controleert de waarde ervan.
- Je kan een ELSE toevoegen die wordt uitgevoerd, indien er geen waarde matcht met de selector

```
-- Voorbeeld van Simple CASE
CASE selector
WHEN selector_value_1
   THEN statements_1
WHEN selector_value_1
   THEN statement_2 ...
ELSE
   else_statements
END CASE;
```

Operatoren – CASE

- Een **searched case** gebruikt selector variabele en controleert de waarde ervan.
- De condities worden van boven naar onder geëvalueerd
- Enkel de eerste conditie die TRUE teruggeeft wordt uitgevoerd (ook al zijn er meerdere condities TRUE)
- Indien er geen conditie TRUE is en er geen ELSE gebruikt is, dan krijg je een **CASE_NOT_FOUND** exception

```
-- Voorbeeld van Searched CASE
CASE
   WHEN condition_1 THEN statements_1
   WHEN condition_2 THEN statements_2
...
   WHEN condition_n THEN statements_n
   [ ELSE else_statements ]
END CASE;
```

Operatoren – Nested

- Net zoals de gekende if statements van andere programmeertalen, kan je met PL SQL ook je if statements nesten:

```
IF condition_1 THEN
    IF condition_2 THEN
        nested_if_statements;
    END IF;
ELSE
    else_statements;
END IF;
```

Oefeningen!