# Cours PL/SQL

E.Coquery

emmanuel.coquery@liris.cnrs.fr

# Programmation au sein du SGBD

Pourquoi ?

- Les contraintes prédéfinies ne sont pas toujours suffisantes.
  - Exemple : tout nouveau prix pour un CD doit avoir une date de début supérieure à celle des autres prix pour ce CD.
- L'insertion, la suppression ou la mise à jour de certaines données peut nécessiter des calculs sur la base.
- Utilisation de fonction propres à l'application dans des requêtes.

# PL/SQL

- Procedural Language extensions to SQL
- Objectif : bonne intégration du langage avec SQL
- Syntaxe proche du Pascal
- Permet de définir des fonctions, des procédures ou encore des déclencheurs (triggers).

# Instructions de contrôle

`BEGIN ... END ;` définit le début et la fin d'un bloc d'instructions

`/` si seul sur une ligne : fin d'une définition (déclenche l'évaluation)

*variable* `:=` *valeur* `;` affectation

`IF` *condition* `THEN ... ELSIF` *condition* `THEN ... ELSE ... END IF ;`

`WHILE` *condition* `LOOP ... END LOOP ;`

`FOR` *compteur* `IN` *min*`..`*max* `LOOP ... END LOOP ;`

`LOOP ... EXIT WHEN` *condition* `END LOOP ;`

# Variables

```
DECLARE
    variable1 type1;
    variable2 type2;
    variable3 type3  := val;
BEGIN
    ...
END;
```

`type` : tout type SQL.

# SELECT INTO

Variation permettant de stocker le résultat d'une requête dans une (ou des) variable(s) :

SELECT val1, val2, . . .INTO var1, var2, . . .
FROM . . .
;

La requête doit renvoyer exactement 1 n-uplet

# Fonctions

```
CREATE FUNCTION nom (arg1 IN type1, arg2 OUT type2,
                     arg3 IN OUT type3, ...)
RETURN type AS
    variable4 type4;
    ...
BEGIN
    ...
    RETURN valeur; END;
/
show errors
```

# Procédures

```
CREATE PROCEDURE nom (arg1 IN type1, arg2 OUT type2,
                      arg3 IN OUT type3, ...)
AS
    variable1 type1;
    ...
BEGIN
    ...
END;
/
show errors
```

CALL *nom*(*valeur1*, *variable2*, *variable3*, ...);

# Triggers : INSERT

Exécutés lors d'une insertion

CREATE OR REPLACE TRIGGER *nom*
BEFORE *(ou bien AFTER)* INSERT ON *table*
FOR EACH ROW
DECLARE
    ...
BEGIN
    ...
END ;
/
show errors

:NEW représente le n-uplet inséré (modifiable si BEFORE)

# Triggers : UPDATE

Exécutés lors d'une mise à jour

CREATE OR REPLACE TRIGGER *nom*
BEFORE *(ou bien AFTER)* UPDATE ON *table*
FOR EACH ROW
DECLARE
    ...
BEGIN
    ...
END ;
/
show errors

:`NEW` représente le nouveau n-uplet (modifiable si BEFORE)
:`OLD` représente l'ancien n-uplet

# Triggers : DELETE

Exécutés lors d'une mise à jour

CREATE OR REPLACE TRIGGER *nom*
BEFORE *(ou bien AFTER)* DELETE ON *table*
FOR EACH ROW
DECLARE
    ...
BEGIN
    ...
END ;
/
show errors

:OLD représente l'ancien n-uplet

# Triggers : exemple

```
CREATE OR REPLACE TRIGGER verifie_date
BEFORE UPDATE ON prix
FOR EACH ROW
DECLARE
    date_max DATE;
BEGIN
    SELECT MAX(date) INTO date_max
    FROM prix
    WHERE code_barre = :NEW.code_barre;

    IF :NEW.date ¡ date_max
    THEN
        RAISE_APPLICATION_ERROR(-1,'Date illégale');
    END IF;
END;
/
show errors
```

# Exceptions - rattrapage

Rattrapage d'exceptions :

```
BEGIN
  ...
EXCEPTION
  WHEN exception1 THEN
      ...
  WHEN exception2 THEN
      ...
  WHEN OTHERS THEN
      ...
END;
```

# Exceptions - définition et utilisation

Déclaration d'exceptions :

```
DECLARE
    nom_exception EXCEPTION;
    ...
BEGIN
    ...
```

Déclenchement d'exceptions :

```
RAISE nom_exception;
```

# Curseurs

Pointeurs associés au résultat d'une requête.

```
...
DECLARE
    CURSOR nom IS
    SELECT ... ;
    ...
BEGIN
    OPEN nom ;
    LOOP
        FETCH nom INTO variable ;
        EXIT WHEN nom%NOTFOUND ;
        ...
    END LOOP ;
END ;
```

# Curseur et FOR

Abbréviation pour un FETCH and une boucle :

FOR *var1*, *var2*, ... IN *nom_curs*
LOOP
    ...
END LOOP ;

# Exemple

```
CREATE OR REPLACE PROCEDURE CORRIGE_ARTISTES() AS
    CURSOR cd_artiste IS
    SELECT code_barre,num_artiste
    FROM cd NATURAL JOIN artiste;

    code INTEGER;
    art INTEGER;
BEGIN
    FOR code,art IN cd_artiste
    LOOP
        DELETE FROM piste_artiste
        WHERE code_barre = code AND num_artiste = art;
    END LOOP;
END;
```

# Références

Site Oracle et PL/SQL :
http://www.techonthenet.com/oracle/

Référence PL/SQL Oracle :
http://download.oracle.com/docs/cd/B19306_01
/appdev.102/b14261/toc.htm