# 3

PL/SQL

- Curseurs explicites
- Exceptions
- Déclencheurs (Triggers)

# Curseurs explicites

## SELECT RETOURNE PLUS D'UNE LIGNE

```
51  DECLARE
52    CURSOR cur_shw IS
53      SELECT shw_id, shw_title, shw_date
54        FROM show_shw
55        WHERE EXTRACT (MONTH FROM shw_date) = 10;
56    v_shw_id show_shw.shw_id%TYPE;
57    v_shw_title show_shw.shw_title%TYPE;
58    v_shw_date show_shw.shw_date%TYPE;
59
60  BEGIN
61    OPEN cur_shw;
62    LOOP
63      FETCH cur_shw INTO v_shw_id, v_shw_title, v_shw_date;
64      EXIT WHEN cur_shw%NOTFOUND;
65      DBMS_OUTPUT.PUT_LINE(v_shw_title || ' ' || TO_CHAR(v_shw_date, 'DD Month'))
66    END LOOP;
67
68    CLOSE cur_shw;
69  END;
```

```
DECLARE
  CURSOR cur_shw IS
    SELECT *
      FROM show_shw
      WHERE EXTRACT (MONTH FROM shw_date) = 10;
  v_shw_record cur_shw%ROWTYPE;

BEGIN
  OPEN cur_shw;
  LOOP
    FETCH cur_shw INTO v_shw_record;
    EXIT WHEN cur_shw%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_shw_record.shw_title || ' '
                         || TO_CHAR(v_shw_record.shw_date, 'DD Month'));
  END loop;

  CLOSE cur_shw;
END;
```

# Attributs des Curseurs

%ISOPEN

- TRUE si le curseur est ouvert

%ROWCOUNT

- Retourne le nombre de lignes récupérées jusque là

%FOUND

- TRUE si le dernier FETCH a récupéré une ligne

%NOTFOUND

- Le contraire de %FOUND
- TRUE si le dernier FETCH n’a pas retourné une ligne

# Plus simplement

```
DECLARE
  CURSOR cur_shw IS
    SELECT *
      FROM show_shw
      WHERE EXTRACT (MONTH FROM shw_date) = 10;
BEGIN
  FOR v_shw_record IN cur_shw LOOP
    DBMS_OUTPUT.PUT_LINE( v_shw_record.shw_title ||
                          ' ' ||
                          TO_CHAR(v_shw_record.shw_date, 'DD Month'));
  END LOOP;
END;
```

Oracle Student Guide 5.3 p 5

# Cursor FOR Loops

- Compare the cursor FOR loop (on the left) with the cursor code you learned in the previous lesson.
- The two forms of the code are logically identical to each other and produce exactly the same results.

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name
      FROM employees
      WHERE department_id = 50;
BEGIN
  FOR v_emp_rec IN cur_emps LOOP
    DBMS_OUTPUT.PUT_LINE(…);
  END LOOP;
END;
```

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name
      FROM employees
      WHERE department_id = 50;
  v_emp_rec  cur_emps%ROWTYPE;
BEGIN
  OPEN cur_emps;
  LOOP
    FETCH cur_emps INTO v_emp_rec;
    EXIT WHEN cur_emps%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(…);
  END LOOP;
  CLOSE cur_emps;
END;
```

Oracle Student Guide 5.3 p 5

# Cursor FOR Loops Using Subqueries

- Again, compare these two forms of code.
- They are logically identical, but which one would you rather write – especially if you hate typing!

```
BEGIN
  FOR v_dept_rec IN (SELECT *
      FROM departments) LOOP
    DBMS_OUTPUT.PUT_LINE(…);
  END LOOP;
END;
```

```
DECLARE
  CURSOR cur_depts IS
    SELECT * FROM departments;
  v_dept_rec
    cur_depts%ROWTYPE;
BEGIN
   OPEN cur_depts;
   LOOP
     FETCH cur_depts INTO
       v_dept_rec;
     EXIT WHEN
       cur_depts%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(…);
   END LOOP;
   CLOSE cur_depts;
END;
```

# Curseurs avec paramètres

Curseur utilisé plusieurs fois avec des valeurs différentes pour la clause WHERE

Curseurs imbriqués

```
3   DECLARE
4     CURSOR cur_shw (p_month NUMBER) IS
5       SELECT *
6         FROM show_shw
7         WHERE EXTRACT (MONTH FROM shw_date) = p_month;
8     v_month NUMBER;
9
10  BEGIN
11    SELECT MIN(EXTRACT (MONTH FROM shw_date))
12      INTO v_month
13      FROM show_shw;
14    FOR v_shw_record IN cur_shw(v_month) LOOP
15      DBMS_OUTPUT.PUT_LINE(v_shw_record.shw_title || ' '
16                           || TO_CHAR(v_shw_record.shw_date, 'DD Month'));
17    END loop;
18
19  END;
```

# PL/SQL : Exceptions

# Gestion des exceptions

Position dans le bloc PL/SQL

Types d’erreurs – types d’exceptions

Oracle Student Guide 7.1

# Exceptions / Gestion des erreurs

## Erreurs

- Erreurs déclenchées par oracle
- Erreurs définies par le programmeur

## 2 possibilités

- Traitée dans la section exception
- Le bloc se termine par une erreur et l'exception est propagée (Oracle section 7_4)

## Catégories d'exceptions

- Erreurs se déclenchant implicitement
  - Erreurs prédéfinies
  - Erreurs Oracle non prédéfinies
- Erreurs se déclenchant explicitement

# Erreurs prédéfinies

## Prédéfinies

- ZERO_DIVIDE
- NO_DATA_FOUND
  - SELECT … INTO ne retourne aucune ligne
- TOO_MANY_ROWS
  - SELECT … INTO concerne plus d'une ligne
- DUP_VAL_ON_INDEX
  - Doublon sur une colonne « UNIQUE »
- Etc.
- Liste des erreurs prédéfinies – PL/SQL user's guide and reference

# Erreur oracle prédéfinie

```
DECLARE
v_tsh_name type_show_tsh.tsh_name%TYPE;

BEGIN
  SELECT tsh_name
    INTO v_tsh_name
    FROM type_show_tsh
    WHERE tsh_id = 200;

  DBMS_OUTPUT.PUT_LINE(v_tsh_name)
END;
```

```
Rapport d'erreur -
ORA-01403: no data found
ORA-06512: at line 4
01403. 00000 -  "no data found"
*Cause:    No data was found from the
objects.
*Action:   There was no data from the
objects which may be due to end of
fetch.
```

```
BEGIN
  SELECT tsh_name
      INTO v_tsh_name
       FROM type_show_tsh
      WHERE tsh_id = 200;

 DBMS_OUTPUT.PUT_LINE(v_tsh_name);

EXCEPTION
WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('No such type of show');

WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error occurred');
END;
```

# Erreurs oracle non prédéfinies

**INSERT INTO type_show_tsh**

**VALUES (7, NULL);**

Erreur commençant à la ligne: 35 de la commande -

INSERT INTO type_show_tsh

VALUES (7, NULL)

Rapport d'erreur -

Erreur SQL : ORA-01400: cannot insert NULL into ("BDD1"."TYPE_SHOW_TSH"."TSH_NAME")

01400. 00000 - "cannot insert NULL into (%s)"

*Cause: An attempt was made to insert NULL into previously listed objects.

*Action: These objects cannot accept NULL values.

# Erreurs oracle non prédéfinies

Peut être récupérée par WHEN OTHERS

Peut être nommée : EXCEPTION_INIT

Faire apparaître la nature de l'erreur

- SQLCODE
  - Retourne le code de la dernière erreur
  - A utiliser uniquement dans la section exception
  - sql_err := SQLCODE;
- SQLERRM
  - Retourne le message d'erreur de la dernière erreur
  - sql_msg := SQLERRM;

```
DECLARE
e_insert EXCEPTION;
PRAGMA EXCEPTION_INIT(e_insert, -01400);
v_error_code NUMBER;
v_error_msg VARCHAR2(255);

BEGIN
INSERT INTO type_show_tsh
VALUES (7, NULL);

EXCEPTION
WHEN e_insert THEN
  DBMS_OUTPUT.PUT_LINE('Error on INSERT');
  v_error_code := SQLCODE;
  v_error_msg := SQLERRM;
  INSERT INTO error_log(e_user, e_date, error_code, error_msg)
  VALUES(USER, SYSDATE, v_error_code, v_error_message);
END;
```

Déclarer l'exception

Associer numéro de l'exception oracle et exception déclarée

Gérer l'exception

Insérer code d'erreur et message dans une table de log

# Erreurs définies par le programmeur

Les salaires ne peuvent pas être diminués

Les notes ne peuvent pas être mises à jour le dimanche

Une erreur de saisie ne provoque pas la mise à jour attendue

Une instruction UPDATE ou DELETE ne modifie/supprime aucune ligne

```
1  CREATE OR REPLACE PROCEDURE del_bkg_noexcep(
2      p_bkg_id IN booking_bkg.bkg_id%TYPE )
3   AS
4  BEGIN
5    DELETE FROM booking_bkg
6      WHERE bkg_id = p_bkg_id;
7
8    DBMS_OUTPUT.PUT_LINE('Booking num : ' ||
9                          p_bkg_id ||
10                         ' has been successfully removed');
11
12 END del_bkg_noexcep ;
```

```
SELECT * FROM booking_bkg;
```

| | BKG_ID | BKG_DATE | BKG_TOTAL_SEAT | BKG_CST_ID | BKG_SHW_ID | BKG_TPR_ID |
|---|---|---|---|---|---|---|
| 1 | 100 | 29/09/12 | 2 | 1 | 10 | 1 |
| 2 | 101 | 29/09/12 | 4 | 3 | 10 | 2 |
| 3 | 102 | 28/08/12 | 1 | 2 | 14 | 1 |

```
BEGIN
 del_bkg_noexcep(100);
 del_bkg_noexcep(10);
 del_bkg_noexcep(101);
 COMMIT;
END;
```

```
Booking num : 100 has been successfully removed
Booking num : 10 has been successfully removed
Booking num : 101 has been successfully removed
```

```
SELECT * FROM booking_bkg;
```

| | BKG_ID | BKG_DATE | BKG_TOTAL_SEAT | BKG_CST_ID | BKG_SHW_ID | BKG_TPR_ID |
|---|---|---|---|---|---|---|
| 1 | 102 | 28/08/12 | 1 | 2 | 14 | 1 |

```
15  CREATE OR REPLACE PROCEDURE del_bkg(
16      p_bkg_id IN booking_bkg.bkg_id%TYPE)
17   AS
18     e_bkg_id_entry EXCEPTION;
19  BEGIN
20     DELETE FROM booking_bkg
21       WHERE bkg_id = p_bkg_id;
22     IF (SQL%NOTFOUND) THEN
23        RAISE e_bkg_id_entry;
24     END IF;
25     DBMS_OUTPUT.PUT_LINE('Booking num : ' ||
26                          p_bkg_id ||
27                          ' has been successfully removed');
28     EXCEPTION
29       WHEN e_bkg_id_entry THEN
30           DBMS_OUTPUT.PUT_LINE('no such booking ' || p_bkg_id);
31  END del_bkg ;
32  /
```

```
SELECT * FROM booking_bkg;
```

| | BKG_ID | BKG_DATE | BKG_TOTAL_SEAT | BKG_CST_ID | BKG_SHW_ID | BKG_TPR_ID |
|---|---|---|---|---|---|---|
| 1 | 100 | 29/09/12 | 2 | 1 | 10 | 1 |
| 2 | 101 | 29/09/12 | 4 | 3 | 10 | 2 |
| 3 | 102 | 28/08/12 | 1 | 2 | 14 | 1 |

```
BEGIN
 del_bkg(100);
 del_bkg(10);
 del_bkg(101);
 COMMIT;
END;
```

```
Booking num : 100 has been successfully removed
no such booking 10
Booking num : 101 has been successfully removed
```

```
SELECT * FROM booking_bkg;
```

| | BKG_ID | BKG_DATE | BKG_TOTAL_SEAT | BKG_CST_ID | BKG_SHW_ID | BKG_TPR_ID |
|---|---|---|---|---|---|---|
| 1 | 102 | 28/08/12 | 1 | 2 | 14 | 1 |

# Bonnes pratiques

Toujours penser à la section exception

Nommer les exceptions plutôt que tout traiter avec « OTHERS »

Décider si la partie exception doit exécuter commit, rollback, ou laisser la transaction se poursuivre

Student
Guide
7.1

# Triggers

# Triggers (déclencheurs)

Un trigger (déclencheur) est un programme PL/SQL

- Associé à un évènement
- Qui s'exécute automatiquement lorsque l'évènement se produit
- Est stocké

En particulier pour :

- Valider certaines contraintes
- Tracer les modifications réalisées sur les tables
- Générer automatiquement certaines valeurs de colonnes

Mais pas pour se substituer à des contraintes

- FOREIGN KEY
- CHECK…

# Evènements possibles

Instructions LMD

- Sur une table
- Sur une vue

Instructions LDD

- CREATE, ALTER

Evènements système

- Connexion d’un utilisateur, arrêt de la base

# Types de triggers

Niveau ligne (row-level)

- Se déclenche pour chacune des lignes affectées par l'instruction qui a déclenché le trigger

Niveau instruction (statement-level)

- Se déclenche une unique fois pour l'instruction qui a déclenché le trigger

# Exemple de trigger LMD

```
1  CREATE OR REPLACE TRIGGER log_hpr_price_inc
2
3    AFTER UPDATE OF hpr_seat_price ON has_price_hpr FOR EACH ROW
4
5    WHEN (NEW.hpr_seat_price < OLD.hpr_seat_price)
6
7  BEGIN
8    INSERT INTO log_price(l_user, l_date,
9                          l_hpr_id, l_shw_id,
10                         l_new_price, l_old_price)
11   VALUES(USER, SYSDATE,
12          :NEW.hpr_tpr_id, :NEW.hpr_shw_id,
13          :NEW.hpr_seat_price, :OLD.hpr_seat_price);
14 END;
```

# Trigger LMD (DML trigger)

S’exécute automatiquement lorsqu’une instruction LMD (INSERT, UPDATE, DELETE) est exécutée

2 classifications possibles

- Quand s’exécute-t-il ?
  - BEFORE, AFTER, INSTEAD OF l’instruction LMD qui le déclenche
- Combien de fois s’exécute-t-il ?
  - Une seule fois – trigger niveau instruction
  - Pour chaque ligne affectée par l’évènement déclencheur – trigger niveau ligne

# Spécifier l’évènement

Timing

- AFTER, BEFORE, INSTEAD OF

Type d’évènement

- AFTER UPDATE
- BEFORE UPDATE
- AFTER INSERT OR UPDATE OR DELETE

Spécification de la table concernée

- ON customer_cst
- ON booking_bkg

Pour les ordres UPDATE : spécification de la colonne

- OF cst_phone ON customer_cst

# Exemples

```
18  CREATE OR REPLACE TRIGGER hpr_price_upd_trigg
19    BEFORE UPDATE OF hpr_seat_price ON has_price_hpr
20    BEGIN ... END;
21
22  CREATE OR REPLACE TRIGGER hpr_del_trigg
23  BEFORE DELETE ON has_price_hpr
24    BEGIN ... END;
25
26  CREATE OR REPLACE TRIGGER hpr_del_trigg
27    AFTER INSERT OR DELETE ON has_price_hpr
28    BEGIN ... END;
29
30  CREATE OR REPLACE TRIGGER cst_upd_trigg
31    AFTER UPDATE OF cst_last_name, cst_phone ON customer_cst
32    BEGIN ... END;
```

# Niveau ligne ou niveau instruction

FOR EACH ROW

- Le trigger se déclenche pour chaque ligne affectée par l'ordre SQL
- Trigger niveau ligne (row-level trigger)

Ne rien mentionner

- Trigger niveau instruction (statement-level trigger)

# Trigger niveau instruction

Se déclenche une fois ( même si aucune ligne n'a été affectée)

N'accède pas aux valeurs des colonnes des lignes affectées

```
3  CREATE OR REPLACE TRIGGER log_cst_changes_trigg
4  AFTER UPDATE ON customer_cst
5  BEGIN
6       INSERT INTO log_cst_table (who, when_done)
7       VALUES (USER, SYSDATE);
8  END;
9  /
```

# Déclenchement du trigger

```
16 UPDATE customer_cst
17 SET cst_first_name = 'Robert'
18 WHERE cst_id >= 1;
```

AFTER UPDATE =>

- Les modifications sont réalisées
- Puis le trigger se déclenche (une unique fois)

# Exemple de trigger LMD – niveau ligne

```
1   CREATE OR REPLACE TRIGGER log_hpr_price_inc
2
3     AFTER UPDATE OF hpr_seat_price ON has_price_hpr FOR EACH ROW
4
5     WHEN (NEW.hpr_seat_price < OLD.hpr_seat_price)
6
7   BEGIN
8     INSERT INTO log_price(l_user, l_date,
9                           l_hpr_id, l_shw_id,
10                          l_new_price, l_old_price)
11    VALUES(USER, SYSDATE,
12          :NEW.hpr_tpr_id, :NEW.hpr_shw_id,
13          :NEW.hpr_seat_price, :OLD.hpr_seat_price);
14  END;
```

# Référence aux anciennes et/ou nouvelles valeurs des lignes

Uniquement dans les triggers niveau ligne

INSERT

- :NEW seulement

DELETE

- :OLD seulement

UPDATE

- :OLD et/ou :NEW

# Exemple de trigger LMD – niveau ligne

```
1   CREATE OR REPLACE TRIGGER log_hpr_price_inc
2
3     AFTER UPDATE OF hpr_seat_price ON has_price_hpr FOR EACH ROW
4
5     WHEN (NEW.hpr_seat_price < OLD.hpr_seat_price)
6
7   BEGIN
8     INSERT INTO log_price(l_user, l_date,
9                           l_hpr_id, l_shw_id,
10                          l_new_price, l_old_price)
11    VALUES(USER, SYSDATE,
12          :NEW.hpr_tpr_id, :NEW.hpr_shw_id,
13          :NEW.hpr_seat_price, :OLD.hpr_seat_price);
14  END;
```

# WHEN

Optionnel

Permet de préciser une condition pour le déclenchement effectif

WHEN (NEW.hpr_seat_price < OLD.hpr_seat_price)

# Trigger LMD – contrôle

```
21 CREATE OR REPLACE TRIGGER ck_hpr_trigg
22   BEFORE INSERT ON has_price_hpr FOR EACH ROW
23 BEGIN
24     IF TO_CHAR(SYSDATE, 'DY') = 'DIM.' THEN
25         raise_application_error(-20201, 'Pas d''insertion de prix le dimanche');
26     END IF;
27 END;
28 /
```

raise_application_error fait échouer le trigger

Lorsque qu'un trigger échoue, l'instruction LMD qui l'a déclenchée est annulée automatiquement

L'insertion n'est pas réalisée

# Tester le type d'évènement

Déclenchement sur plusieurs évènements

- INSERTING
- DELETING
- UPDATING

IF INSERTING THEN ….

```
CREATE OR REPLACE TRIGGER ck_hpr_price
  BEFORE INSERT OR UPDATE OF hpr_seat_price ON has_price_hpr FOR EACH ROW
BEGIN
    IF INSERTING THEN
       IF TO_CHAR(SYSDATE, 'DY') = 'DIM.' THEN
         raise_application_error(-20201, 'Pas d''insertion de prix le dimanche');
       END IF;
    ELSIF :NEW.hpr_seat_price < :OLD.hpr_seat_price THEN
        raise_application_error(-20202, ‘Update : Pas de baisse de prix');
    END IF;
END;
```

# UPDATE has_price_hpr
 set hpr_seat_price = hpr_seat_price - 1;

```
Erreur commençant à la ligne: 16 de la commande -
UPDATE has_price_hpr
  set hpr_seat_price = hpr_seat_price - 1
Rapport d'erreur -
Erreur SQL : ORA-20202: Pas de baisse de prix
ORA-06512: at "BDD1.CK_HPR_PRICE", line 7
ORA-04088: error during execution of trigger 'BDD1.CK_HPR_PRICE'
```

# Nommer / Activer / Supprimer

- Nom : Faire apparaître le nom de la table
  - EMP_CK_SALARY
- Activer / désactiver
  - ALTER TRIGGER emp_ck_salary DISABLE;
  - ALTER TRIGGER emp_ck_salary ENABLE;
- Supprimer
  - DROP TRIGGER emp_ck_salary;
- Lister
  - SELECT trigger_name, status FROM USER_TRIGGERS;

# Transactions

COMMIT, ROLLBACK

- Ne sont pas autorisés dans les triggers