



Cours PL/SQL

Programmation Oracle

Pr. Lamia ZIAD

Ecole Supérieure de Technologie d'Essaouira

Contents

1	Introduction a PL/SQL	3
1.1	Architecture PL/SQL	3
1.2	Structure d'un Bloc PL/SQL	3
1.3	Avantages de PL/SQL	4
2	Variables et Types de Donnees	5
2.1	Declaration de Variables	5
2.2	Types de Donnees Scalarres	6
2.3	Anchored Datatypes	6
2.4	Types Composites	7
3	Structures de Controle	8
3.1	Structure IF-THEN-ELSE	8
3.2	Boucle LOOP Basique	9
3.3	Boucle WHILE	9
3.4	Boucle FOR	9
3.5	Boucle FOR avec Cursor	10
4	Les Cursors	11
4.1	Cursor Implicite	11
4.2	Cursor Explicite	12
4.3	Cursor avec Parametres	13
4.4	Cursor FOR Loop	13
5	Gestion des Exceptions	15
5.1	Exceptions Predefinies	15
5.2	Exceptions Utilisateur	16
5.3	RAISE_APPLICATION_ERROR	17

6 Procedures et Fonctions	18
6.1 Creation de Procedure	18
6.2 Appel d'une Procedure	20
6.3 Creation de Fonction	20
6.4 Utilisation d'une Fonction	22
7 Les Packages	23
7.1 Specification du Package	23
7.2 Body du Package	25
7.3 Utilisation du Package	26
8 Les Triggers	27
8.1 Trigger sur INSERT	28
8.2 Trigger sur UPDATE	29
8.3 Trigger sur DELETE	30
8.4 Trigger d'Instruction	30
9 Gestion des Transactions	32
9.1 COMMIT et ROLLBACK	32
9.2 AUTONOMOUS_TRANSACTION	33
10 SQL Dynamique	35
10.1 EXECUTE IMMEDIATE	35
10.2 Dynamic SQL avec DDL	37
10.3 DBMS_SQL Package	37

Chapter 1

Introduction à PL/SQL

Definition

PL/SQL (Procedural Language/Structured Query Language) est le langage de programmation procedurale d'Oracle. Il étend SQL avec des structures de contrôle, des boucles, des variables et bien plus.

1.1 Architecture PL/SQL

- **Bloc anonyme** : Exécute immédiatement, non stocké
- **Procedures** : Sous-programmes stockés dans la base
- **Fonctions** : Retournent une valeur
- **Packages** : Groupement de procédures et fonctions
- **Triggers** : Déclenchés automatiquement par événements

1.2 Structure d'un Bloc PL/SQL

Important

Un bloc PL/SQL se compose de trois parties : DECLARE, BEGIN, EXCEPTION.

Exemple

```
1 DECLARE
2     -- Section declaration (optionnelle)
3     v_nombre NUMBER := 10;
4     v_nom VARCHAR2(50);
5 BEGIN
6     -- Section executable (obligatoire)
7     v_nom := 'Jean Dupont';
8     DBMS_OUTPUT.PUT_LINE('Bonjour ' || v_nom);
9     DBMS_OUTPUT.PUT_LINE('Nombre: ' || v_nombre);
10 EXCEPTION
11     -- Section exception (optionnelle)
12     WHEN OTHERS THEN
13         DBMS_OUTPUT.PUT_LINE('Erreur: ' || SQLERRM);
14 END;
15 /
```

1.3 Avantages de PL/SQL

- **Performance** : Execution groupee reduit les appels reseau
- **Productivite** : Integration directe avec SQL
- **Portabilite** : Fonctionne sur toutes les plateformes Oracle
- **Securite** : Gestion des privileges via la base

Chapter 2

Variables et Types de Donnees

2.1 Declaration de Variables

Important

Syntaxe : nom_variable [CONSTANT] type [NOT NULL] [:= valeur];

Exemple

```
1 DECLARE
2     v_salaire NUMBER(8,2) := 2500.50;
3     v_nom VARCHAR2(50) NOT NULL := 'Martin';
4     c_taux CONSTANT NUMBER := 1.20;
5     v_date DATE DEFAULT SYSDATE;
6     v_actif BOOLEAN := TRUE;
7 BEGIN
8     DBMS_OUTPUT.PUT_LINE('Salaire: ' || v_salaire);
9     DBMS_OUTPUT.PUT_LINE('Nom: ' || v_nom);
10 END;
11 /
```

2.2 Types de Donnees Scalarres

Type	Description
NUMBER	Numerique (precision, echelle)
VARCHAR2	Chaine variable (max 32767 en PL/SQL)
CHAR	Chaine fixe
DATE	Date et heure
BOOLEAN	Vrai/Faux/NULL
BINARY_INTEGER	Entier signé

2.3 Anchored Datatypes

Important

Utilisez %TYPE pour lier le type d'une variable à une colonne de table.

Exemple

```

1  DECLARE
2      v_emp_name employees.last_name%TYPE;
3      v_emp_salary employees.salary%TYPE;
4      v_dept_name departments.department_name%TYPE;
5  BEGIN
6      SELECT last_name, salary INTO v_emp_name, v_emp_salary
7      FROM employees WHERE employee_id = 100;
8
9      DBMS_OUTPUT.PUT_LINE(v_emp_name || ' gagne ' || v_emp_salary);
10 END;
11 /

```

2.4 Types Composites

Exemple

```
1 -- Record Type
2 DECLARE
3     TYPE emp_record IS RECORD (
4         id employees.employee_id%TYPE ,
5         nom employees.last_name%TYPE ,
6         salaire employees.salary%TYPE
7     );
8     v_emp emp_record;
9
10 -- Table Type
11    TYPE nom_table IS TABLE OF VARCHAR2(50) INDEX BY BINARY_INTEGER;
12    v_noms nom_table;
13 BEGIN
14     v_emp.id := 100;
15     v_emp.nom := 'Dupont';
16     v_emp.salaire := 5000;
17
18     v_noms(1) := 'Alice';
19     v_noms(2) := 'Bob';
20 END;
21 /
```

Chapter 3

Structures de Contrôle

3.1 Structure IF-THEN-ELSE

Important

Permet l'exécution conditionnelle de blocs de code.

Exemple

```
1 DECLARE
2     v_salaire NUMBER := 3000;
3     v_grade VARCHAR2(20);
4 BEGIN
5     IF v_salaire > 5000 THEN
6         v_grade := 'A';
7     ELSIF v_salaire BETWEEN 3000 AND 5000 THEN
8         v_grade := 'B';
9     ELSE
10        v_grade := 'C';
11    END IF;
12
13    DBMS_OUTPUT.PUT_LINE('Grade: ' || v_grade);
14 END;
15 /
```

3.2 Boucle LOOP Basique

Exemple

```
1 DECLARE
2     v_compteur NUMBER := 1;
3 BEGIN
4     LOOP
5         DBMS_OUTPUT.PUT_LINE('Iteration: ' || v_compteur);
6         v_compteur := v_compteur + 1;
7         EXIT WHEN v_compteur > 5;
8     END LOOP;
9 END;
10 /
```

3.3 Boucle WHILE

Exemple

```
1 DECLARE
2     v_compteur NUMBER := 1;
3 BEGIN
4     WHILE v_compteur <= 5 LOOP
5         DBMS_OUTPUT.PUT_LINE('Compteur: ' || v_compteur);
6         v_compteur := v_compteur + 1;
7     END LOOP;
8 END;
9 /
```

3.4 Boucle FOR

Important

La boucle FOR est ideal quand le nombre d'iterations est connu.

Exemple

```
1 BEGIN
2     -- FOR avec compteur
3     FOR i IN 1..5 LOOP
4         DBMS_OUTPUT.PUT_LINE('Iteration: ' || i);
5     END LOOP;
6
7     -- FOR reverse
8     FOR i IN REVERSE 1..5 LOOP
9         DBMS_OUTPUT.PUT_LINE('Reverse: ' || i);
10    END LOOP;
11 END;
12 /
```

3.5 Boucle FOR avec Cursor

Exemple

```
1 BEGIN
2     FOR emp IN (SELECT employee_id, last_name, salary
3                   FROM employees
4                   WHERE department_id = 50)
5     LOOP
6         DBMS_OUTPUT.PUT_LINE(emp.last_name || ' : ' || emp.salary);
7     END LOOP;
8 END;
9 /
```

Chapter 4

Les Cursors

Definition

Un cursor est un pointeur vers une zone memoire contenant le resultat d'une requete SQL. Il permet de traiter les lignes une par une.

4.1 Cursor Implicite

Important

Oracle cree automatiquement un cursor implicite pour les instructions SQL.

Exemple

```
1 BEGIN
2   UPDATE employees SET salary = salary * 1.1
3   WHERE department_id = 50;
4
5   DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' lignes mises a jour');
6
7   IF SQL%FOUND THEN
8     DBMS_OUTPUT.PUT_LINE('Mise a jour reussie');
9   END IF;
10 END;
11 /
```

4.2 Cursor Explicite

Important

Les cursors explicites donnent plus de contrôle sur le traitement des résultats.

Exemple

```
1 DECLARE
2     CURSOR c_emp IS
3         SELECT employee_id, last_name, salary
4             FROM employees
5                 WHERE department_id = 50;
6
7     v_emp_id employees.employee_id%TYPE;
8     v_nom employees.last_name%TYPE;
9     v_salaire employees.salary%TYPE;
10
11    BEGIN
12        OPEN c_emp;
13
14        LOOP
15            FETCH c_emp INTO v_emp_id, v_nom, v_salaire;
16            EXIT WHEN c_emp%NOTFOUND;
17
18            DBMS_OUTPUT.PUT_LINE(v_emp_id || ' - ' || v_nom || ' - ' ||
19                v_salaire);
20        END LOOP;
21
22        CLOSE c_emp;
23    END;
24    /
```

4.3 Cursor avec Parametres

Exemple

```
1 DECLARE
2     CURSOR c_emp(p_dept_id NUMBER) IS
3         SELECT last_name, salary
4             FROM employees
5                 WHERE department_id = p_dept_id;
6
7     v_nom employees.last_name%TYPE;
8     v_salaire employees.salary%TYPE;
9 BEGIN
10    OPEN c_emp(50); -- Departement 50
11
12    LOOP
13        FETCH c_emp INTO v_nom, v_salaire;
14        EXIT WHEN c_emp%NOTFOUND;
15        DBMS_OUTPUT.PUT_LINE(v_nom || ' : ' || v_salaire);
16    END LOOP;
17
18    CLOSE c_emp;
19 END ;
20 /
```

4.4 Cursor FOR Loop

Important

Simplifie l'utilisation des cursors avec gestion automatique.

Exemple

```
1 DECLARE
2     CURSOR c_emp IS
3         SELECT last_name, salary, department_id
4             FROM employees
5             WHERE salary > 5000;
6 BEGIN
7     FOR emp_rec IN c_emp LOOP
8         DBMS_OUTPUT.PUT_LINE(
9             emp_rec.last_name || ' dans departement ' ||
10            emp_rec.department_id || ' gagne ' || emp_rec.salary
11        );
12    END LOOP;
13 END;
14 /
```

Chapter 5

Gestion des Exceptions

Definition

Une exception est une erreur qui survient pendant l'execution du programme. PL/SQL permet de capturer et traiter ces erreurs gracieusement.

5.1 Exceptions Predefinies

Exception	Description
NO_DATA_FOUND	SELECT INTO ne retourne aucune ligne
TOO_MANY_ROWS	SELECT INTO retourne plusieurs lignes
DUP_VAL_ON_INDEX	Violation de contrainte d'unicite
INVALID_NUMBER	Conversion numerique invalide
ZERO_DIVIDE	Division par zero

Exemple

```
1 DECLARE
2     v_emp_name employees.last_name%TYPE;
3     v_salary employees.salary%TYPE;
4 BEGIN
5     SELECT last_name, salary INTO v_emp_name, v_salary
6     FROM employees WHERE employee_id = 9999; -- ID inexistant
7
8 EXCEPTION
9     WHEN NO_DATA_FOUND THEN
10        DBMS_OUTPUT.PUT_LINE('Aucun employe trouve avec cet ID');
11     WHEN TOO_MANY_ROWS THEN
12        DBMS_OUTPUT.PUT_LINE('Plusieurs employes trouves');
13     WHEN OTHERS THEN
14        DBMS_OUTPUT.PUT_LINE('Erreur: ' || SQLCODE || ' - ' || SQLERRM);
15 END;
16 /
```

5.2 Exceptions Utilisateur

Important

Vous pouvez definir vos propres exceptions pour gerer des cas specifiques.

Exemple

```

1  DECLARE
2      e_salaire_trop_bas EXCEPTION;
3      PRAGMA EXCEPTION_INIT(e_salaire_trop_bas, -20001);
4      v_salaire employees.salary%TYPE := 800;
5  BEGIN
6      IF v_salaire < 1000 THEN
7          RAISE e_salaire_trop_bas;
8      END IF;
9
10 EXCEPTION
11     WHEN e_salaire_trop_bas THEN
12         DBMS_OUTPUT.PUT_LINE('Erreur: Le salaire est trop bas');
13         DBMS_OUTPUT.PUT_LINE('Le salaire minimum est 1000');
14 END;
15 /

```

5.3 RAISE_APPLICATION_ERROR**Exemple**

```

1  DECLARE
2      v_count NUMBER;
3  BEGIN
4      SELECT COUNT(*) INTO v_count
5      FROM employees WHERE department_id = 99;
6
7      IF v_count = 0 THEN
8          RAISE_APPLICATION_ERROR(-20002,
9              'Le departement 99 n existe pas');
10     END IF;
11
12 EXCEPTION
13     WHEN OTHERS THEN
14         DBMS_OUTPUT.PUT_LINE('Code erreur: ' || SQLCODE);
15         DBMS_OUTPUT.PUT_LINE('Message: ' || SQLERRM);
16 END;
17 /

```

Chapter 6

Procedures et Fonctions

6.1 Creation de Procedure

Important

Les procedures sont des sous-programmes qui executent une action sans retourner de valeur.

Exemple

```
1 CREATE OR REPLACE PROCEDURE augmenter_salaire (
2     p_emp_id IN employees.employee_id%TYPE,
3     p_pourcentage IN NUMBER
4 ) IS
5     v_ancien_salaire employees.salary%TYPE;
6     v_nouveau_salaire employees.salary%TYPE;
7 BEGIN
8     -- Recuperer l'ancien salaire
9     SELECT salary INTO v_ancien_salaire
10    FROM employees WHERE employee_id = p_emp_id;
11
12    -- Calculer le nouveau salaire
13    v_nouveau_salaire := v_ancien_salaire * (1 + p_pourcentage/100);
14
15    -- Mettre a jour
16    UPDATE employees SET salary = v_nouveau_salaire
17    WHERE employee_id = p_emp_id;
18
19    COMMIT;
20
21    DBMS_OUTPUT.PUT_LINE('Salaire de ' || p_emp_id ||
22        ' augmente de ' || p_pourcentage || '%');
23
24 EXCEPTION
25     WHEN NO_DATA_FOUND THEN
26         DBMS_OUTPUT.PUT_LINE('Employe non trouve');
27     WHEN OTHERS THEN
28         ROLLBACK;
29         RAISE;
30 END augmenter_salaire;
31 /
```

6.2 Appel d'une Procedure

Exemple

```
1 -- Appel de la procedure
2 BEGIN
3     augmenter_salaire(100, 10); -- Augmentation de 10% pour l'employé
4     100
5 END;
6 /
7 -- Ou directement
8 EXECUTE augmenter_salaire(101, 15);
```

6.3 Creation de Fonction

Important

Les fonctions retournent une valeur et peuvent être utilisées dans des expressions SQL.

Exemple

```
1 CREATE OR REPLACE FUNCTION calculer_salaire_annuel (
2     p_emp_id IN employees.employee_id%TYPE
3 ) RETURN NUMBER IS
4     v_salaire_mensuel employees.salary%TYPE;
5     v_commission employees.commission_pct%TYPE;
6     v_salaire_annuel NUMBER;
7 BEGIN
8     SELECT salary, NVL(commission_pct, 0)
9     INTO v_salaire_mensuel, v_commission
10    FROM employees WHERE employee_id = p_emp_id;
11
12    v_salaire_annuel := (v_salaire_mensuel * 12) * (1 + v_commission)
13    ;
14
15
16 EXCEPTION
17     WHEN NO_DATA_FOUND THEN
18         RETURN NULL;
19     WHEN OTHERS THEN
20         RAISE;
21 END calculer_salaire_annuel;
22 /
```

6.4 Utilisation d'une Fonction

Exemple

```
1 -- Appel dans un bloc PL/SQL
2 DECLARE
3     v_salaire_annuel NUMBER;
4 BEGIN
5     v_salaire_annuel := calculer_salaire_annuel(100);
6     DBMS_OUTPUT.PUT_LINE('Salaire annuel: ' || v_salaire_annuel);
7 END;
8 /
9
10 -- Utilisation dans une requete SQL
11 SELECT employee_id, last_name,
12       calculer_salaire_annuel(employee_id) as salaire_annuel
13 FROM employees
14 WHERE department_id = 50;
```

Chapter 7

Les Packages

Definition

Un package est un regroupement logique de procedures, fonctions, variables et cursors.
Il se compose d'une specification (interface) et d'un body (implementation).

7.1 Specification du Package

Exemple

```
1 CREATE OR REPLACE PACKAGE gestion_employees AS
2     -- Variables publiques
3     g_taux_augmentation CONSTANT NUMBER := 1.1;
4
5     -- Cursor public
6     CURSOR c_emp_par_dept(p_dept_id NUMBER) RETURN employees%ROWTYPE;
7
8     -- Procedures
9     PROCEDURE afficher_employes_dept(p_dept_id NUMBER);
10    PROCEDURE augmenter_salaire_dept(p_dept_id NUMBER, p_pourcentage
11                                     NUMBER);
12
13    -- Fonctions
14    FUNCTION compter_employes_dept(p_dept_id NUMBER) RETURN NUMBER;
15    FUNCTION salaire_moyen_dept(p_dept_id NUMBER) RETURN NUMBER;
16
17 END gestion_employees;
/
```


7.2 Body du Package

Exemple

```
1 CREATE OR REPLACE PACKAGE BODY gestion_employes AS
2
3     -- Implementation du cursor
4     CURSOR c_emp_par_dept(p_dept_id NUMBER) RETURN employees%ROWTYPE
5     IS
6         SELECT * FROM employees
7         WHERE department_id = p_dept_id;
8
9     -- Procedure pour afficher les employes
10    PROCEDURE afficher_employes_dept(p_dept_id NUMBER) IS
11        BEGIN
12            FOR emp IN c_emp_par_dept(p_dept_id) LOOP
13                DBMS_OUTPUT.PUT_LINE(
14                    emp.employee_id || ' - ' ||
15                    emp.last_name || ' - ' ||
16                    emp.salary
17                );
18            END LOOP;
19        END afficher_employes_dept;
20
21    -- Procedure pour augmenter les salaires
22    PROCEDURE augmenter_salaire_dept(p_dept_id NUMBER, p_pourcentage
23                                     NUMBER) IS
24        BEGIN
25            UPDATE employees
26            SET salary = salary * (1 + p_pourcentage/100)
27            WHERE department_id = p_dept_id;
28            COMMIT;
29        END augmenter_salaire_dept;
30
31    -- Fonction pour compter les employes
32    FUNCTION compter_employes_dept(p_dept_id NUMBER) RETURN NUMBER IS
33        v_count NUMBER;
34        BEGIN
35            SELECT COUNT(*) INTO v_count
36            FROM employees WHERE department_id = p_dept_id;
37            RETURN v_count;
38        END compter_employes_dept;
39
40    -- Fonction pour le salaire moyen
41    FUNCTION salaire_moyen_dept(p_dept_id NUMBER) RETURN NUMBER IS
42        v_moyenne NUMBER;          25
43        BEGIN
44            SELECT AVG(salary) INTO v_moyenne
45            FROM employees WHERE department_id = p_dept_id;
```

7.3 Utilisation du Package

Exemple

```
1 BEGIN
2     -- Utiliser les fonctions et procedures du package
3     DBMS_OUTPUT.PUT_LINE('Nombre employes dept 50: ' ||
4         gestion_employes.compter_employes_dept(50));
5
6     DBMS_OUTPUT.PUT_LINE('Salaire moyen dept 50: ' ||
7         gestion_employes.salaire_moyen_dept(50));
8
9     -- Afficher les employes
10    gestion_employes.afficher_employes_dept(50);
11
12    -- Augmenter les salaires
13    gestion_employes.augmenter_salaire_dept(50, 10);
14 END;
15 /
```

Chapter 8

Les Triggers

Definition

Un trigger (declencheur) est un bloc PL/SQL qui s'execute automatiquement lorsqu'un evenement specifique se produit sur une table ou une vue.

8.1 Trigger sur INSERT

Exemple

```
1 CREATE OR REPLACE TRIGGER tr_audit_insert_emp
2   BEFORE INSERT ON employees
3   FOR EACH ROW
4 DECLARE
5   v_username VARCHAR2(30);
6 BEGIN
7   -- Recuperer l'utilisateur courant
8   SELECT USER INTO v_username FROM DUAL;
9
10  -- Inserer dans la table d'audit
11  INSERT INTO audit_employees (
12    audit_id, operation, user_name,
13    employee_id, timestamp
14 ) VALUES (
15    audit_seq.NEXTVAL, 'INSERT', v_username,
16    :NEW.employee_id, SYSDATE
17 );
18 END;
19 /
```

8.2 Trigger sur UPDATE

Exemple

```
1 CREATE OR REPLACE TRIGGER tr_verif_salaire
2   BEFORE UPDATE OF salary ON employees
3   FOR EACH ROW
4 BEGIN
5   -- Verifier que le nouveau salaire est superieur a l'ancien
6   IF :NEW.salary < :OLD.salary THEN
7     RAISE_APPLICATION_ERROR(-20003,
8       'Le nouveau salaire ne peut pas etre inferieur a l ancien
9     ');
10  END IF;
11
12  -- Verifier le salaire minimum
13  IF :NEW.salary < 1000 THEN
14    RAISE_APPLICATION_ERROR(-20004,
15      'Le salaire minimum est 1000');
16  END IF;
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
498
499
499
500
```

8.3 Trigger sur DELETE

Exemple

```
1 CREATE OR REPLACE TRIGGER tr_archiver_emp
2   BEFORE DELETE ON employees
3   FOR EACH ROW
4 BEGIN
5   -- Archiver l'employe supprime
6   INSERT INTO employees_archive (
7     employee_id, last_name, email,
8     hire_date, salary, delete_date
9   ) VALUES (
10    :OLD.employee_id, :OLD.last_name, :OLD.email,
11    :OLD.hire_date, :OLD.salary, SYSDATE
12  );
13 END;
14 /
```

8.4 Trigger d'Instruction

Important

Les triggers d'instruction s'exécutent une fois par instruction, pas par ligne.

Exemple

```
1 CREATE OR REPLACE TRIGGER tr_verif_operations
2   BEFORE INSERT OR UPDATE OR DELETE ON employees
3 DECLARE
4   v_jour VARCHAR2(10);
5 BEGIN
6   -- Recuperer le jour de la semaine
7   SELECT TO_CHAR(SYSDATE, 'DY') INTO v_jour FROM DUAL;
8
9   -- Interdire les operations le weekend
10  IF v_jour IN ('SAT', 'SUN') THEN
11    RAISE_APPLICATION_ERROR(-20005,
12      'Operations interdites le weekend');
13  END IF;
14
15  -- Interdire les operations apres 18h
16  IF TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) > 18 THEN
17    RAISE_APPLICATION_ERROR(-20006,
18      'Operations interdites apres 18h');
19  END IF;
20 END;
21 /
```

Chapter 9

Gestion des Transactions

9.1 COMMIT et ROLLBACK

Important

PL/SQL herite des commandes de transaction SQL : COMMIT, ROLLBACK, SAVEPOINT.

Exemple

```

1  DECLARE
2      v_success BOOLEAN := TRUE;
3  BEGIN
4      -- Point de sauvegarde
5      SAVEPOINT avant_transaction;
6
7      -- Premiere operation
8      UPDATE comptes SET solde = solde - 1000
9      WHERE numero_compte = 123;
10
11     -- Deuxieme operation
12     UPDATE comptes SET solde = solde + 1000
13     WHERE numero_compte = 456;
14
15     -- Simulation d'une verification
16     IF v_success THEN
17         COMMIT;
18         DBMS_OUTPUT.PUT_LINE('Transaction validee');
19     ELSE
20         ROLLBACK TO avant_transaction;
21         DBMS_OUTPUT.PUT_LINE('Transaction annulee');
22     END IF;
23
24 EXCEPTION
25     WHEN OTHERS THEN
26         ROLLBACK TO avant_transaction;
27         DBMS_OUTPUT.PUT_LINE('Erreur - Transaction annulee: ' ||
28             SQLERRM);
28
29 /

```

9.2 AUTONOMOUS_TRANSACTION

Important

Permet d'executer une transaction independante dans une procedure ou fonction.

Exemple

```
1 CREATE OR REPLACE PROCEDURE journaliser_operation (
2     p_message IN VARCHAR2
3 ) IS
4     PRAGMA AUTONOMOUS_TRANSACTION;
5 BEGIN
6     INSERT INTO journal_operations (
7         id, message, date_operation, utilisateur
8     ) VALUES (
9         journal_seq.NEXTVAL, p_message, SYSDATE, USER
10    );
11
12     COMMIT; -- Commit independant de la transaction principale
13 END journaliser_operation;
14 /
15
16 -- Utilisation
17 BEGIN
18     UPDATE employes SET salaire = salaire * 1.1;
19
20     -- Journalisation qui commit independamment
21     journaliser_operation('Augmentation generale des salaires de 10%', );
22
23     -- Le commit principal n'affecte pas la journalisation
24     -- et vice versa
25     ROLLBACK; -- N'annule pas la journalisation
26 END;
27 /
```

Chapter 10

SQL Dynamique

Definition

Le SQL dynamique permet de construire et executer des instructions SQL à l'exécution.

10.1 EXECUTE IMMEDIATE

Important

EXECUTE IMMEDIATE est utilisé pour exécuter des instructions SQL dynamiques.

Exemple

```
1 DECLARE
2     v_table_name VARCHAR2(30) := 'employees';
3     v_column_name VARCHAR2(30) := 'salary';
4     v_emp_id NUMBER := 100;
5     v_value NUMBER;
6     v_sql VARCHAR2(1000);
7 BEGIN
8     -- Construction dynamique de la requete
9     v_sql := 'SELECT ' || v_column_name ||
10            ' FROM ' || v_table_name ||
11            ' WHERE employee_id = :1';
12
13     -- Execution avec parametre
14     EXECUTE IMMEDIATE v_sql INTO v_value USING v_emp_id;
15
16     DBMS_OUTPUT.PUT_LINE('Valeur: ' || v_value);
17 END;
18 /
```

10.2 Dynamic SQL avec DDL

Exemple

```
1 CREATE OR REPLACE PROCEDURE creer_table_backup (
2     p_table_name IN VARCHAR2
3 ) IS
4     v_sql VARCHAR2(1000);
5 BEGIN
6     -- Construire la commande DDL dynamique
7     v_sql := 'CREATE TABLE ' || p_table_name || '_backup AS ' ||
8             'SELECT * FROM ' || p_table_name || ' WHERE 1=0';
9
10    -- Executer la DDL
11    EXECUTE IMMEDIATE v_sql;
12
13    DBMS_OUTPUT.PUT_LINE('Table backup creee: ' || p_table_name || '_backup');
14 END creer_table_backup;
15 /
16
17 -- Utilisation
18 BEGIN
19     creer_table_backup('employees');
20 END;
21 /
```

10.3 DBMS_SQL Package

Important

DBMS_SQL offre plus de flexibilité pour le SQL dynamique complexe.

Exemple

```
1  DECLARE
2      v_cursor INTEGER;
3      v_rows_processed INTEGER;
4      v_emp_id employees.employee_id%TYPE;
5      v_last_name employees.last_name%TYPE;
6  BEGIN
7      -- Ouvrir un cursor
8      v_cursor := DBMS_SQL.OPEN_CURSOR;
9
10     -- Parse la requete
11     DBMS_SQLPARSE(v_cursor,
12         'SELECT employee_id, last_name FROM employees WHERE
13         department_id = 50',
14         DBMS_SQL.NATIVE);
15
16     -- Definir les colonnes de retour
17     DBMS_SQL.DEFINE_COLUMN(v_cursor, 1, v_emp_id);
18     DBMS_SQL.DEFINE_COLUMN(v_cursor, 2, v_last_name, 50);
19
20     -- Executer
21     v_rows_processed := DBMS_SQL.EXECUTE(v_cursor);
22
23     -- Parcourir les resultats
24     LOOP
25         EXIT WHEN DBMS_SQL.FETCH_ROWS(v_cursor) = 0;
26         DBMS_SQL.COLUMN_VALUE(v_cursor, 1, v_emp_id);
27         DBMS_SQL.COLUMN_VALUE(v_cursor, 2, v_last_name);
28         DBMS_OUTPUT.PUT_LINE(v_emp_id || ' - ' || v_last_name);
29     END LOOP;
30
31     -- Fermer le cursor
32     DBMS_SQL CLOSE_CURSOR(v_cursor);
33 END;
/
```