2. PL/SQL - Concepte generale

Cuprins

2.1. Ce este *PL/SQL*?

- Un limbaj de programare procedural creat de compania *Oracle.*
- Asigură accesarea datelor unei baze de date şi permite gruparea unei mulţimi de comenzi într-un bloc unic de tratare a datelor.
- Poate fi utilizat doar cu o bază de date *Oracle* sau cu un utilitar *Oracle.*
- Limbaje echivalente dezvoltate de alţi producători

2.2. Legătura cu *SQL*

- *SQL* (***S**tructured **Q**uery **L**anguage*)
  - Este un limbaj non-procedural, denumit şi limbaj declarativ, care permite programatorilor să se axeze preponderent pe input/output şi mai puţin pe paşii programului.
  - Este un limbaj *4GL* (*fourth-generation-programming language*), un limbaj care este mai apropiat de limbajul natural, decât de limbajul de programare.
  - A fost standardizat de *American National Standards Institute* (*ANSI*).
- *PL/SQL* (***P**rocedural **L**anguage/**S**tructured **Q**uery **L**anguage*)
  - Reprezintă **extensia procedurală** a limbajului *SQL*.
  - Include atât instrucţiuni *SQL* pentru prelucrarea datelor şi pentru gestiunea tranzacţiilor, cât şi instrucţiuni proprii.
  - Este un limbaj *3GL* (*third-generation programming language*).

Clasificarea limbajelor

2.3. Caracteristicile limbajului *PL/SQL*

- Este integrat cu *server*-ul *Oracle* şi cu utilitarele *Oracle*.
- Este puternic integrat cu *SQL*:
  - permite utilizarea tuturor comenzilor *SQL* de prelucrare a datelor, de control al tranzacţiilor, a funcţiilor *SQL*, a operatorilor şi pseudo-coloanelor;
  - suportă tipurile de date *SQL*;
  - permite atât *SQL Static* (textul complet al comenzii este cunoscut la momentul compilării), cât şi *SQL Dinamic* (textul complet al comenzii este cunoscut la *run time*);
  - permite procesarea setului de rezultate al unei cereri *SQL* linie cu linie.

- Extinde *SQL* prin construcţii specifice limbajelor procedurale:
  - definirea constantelor și a variabilelor;
  - declararea tipurilor;
  - definirea și utilizarea cursoarelor;
  - utilizarea structurilor de control;
  - definirea procedurilor şi funcţiilor;
  - modularizarea programelor (subprograme, pachete);
  - detectarea şi gestiunea erorilor de execuţie şi a situaţiilor excepţionale;
  - introducerea tipurilor obiect şi a metodelor etc.
- Permite definirea și utilizarea declanşatorilor.
- Asigură securitatea informaţiei.
- Mărește performanța aplicației:
  - permite înglobarea mai multor instrucțiuni într-un singur bloc și trimiterea acestui către baza de date, reducând-se astfel traficul dintre aplicație și baza de date.
- Are suport pentru dezvoltarea aplicaţiilor *Web*.
  - Aplicaţiile *Web* scrise în *PL/SQL* sunt proceduri stocate care interacţionează cu un *browser Web* printr-un protocol *HTTP*.
    - Pentru a facilita dezvoltarea aplicațiilor *Web* sistemul furnizează o serie de pachete predefinite (de exemplu, pachetul *UTL_SMTP* poate fi utilizat pentru a trimite un *mail* dintr-o procedură stocată *PL/SQL*)
    - Pachetele pot fi folosite împreună cu *Oracle Internet Application Server* şi *WebDB.*
  - *PL/SQL Server Pages* (*PSPs*) permite dezvoltarea paginilor *Web* cu conținut dinamic:
    - scripturile *PL/SQL* pot fi integrate în codul sursă *HTML*;
    - scripturile rulează atunci când un *client Web* solicită o pagină;
    - un script poate accepta parametrii, interoga sau actualiza baza de date și a afișa rezultatele într-o pagină personalizată.
- Este portabil
  - Programele *PL/SQL* pot rula pe orice suport pe care există un *server Oracle*.
  - Nu depinde de platformă sau de sistemul de operare.
  - Se pot crea programe, pachete sau librării portabile care pot fi utilizate cu bazele de date *Oracle* în medii diferite.

2.4. Motorul *PL/SQL*

- Compilează și execută codul *PL/SQL*.
- Se află pe *server*-ul *Oracle* sau în unele utilitare *Oracle* (de exemplu, *Oracle Forms*).
  - Unele utilitare *Oracle* au propriul motor *PL/SQL*. Acesta este independent față de motorul *PL/SQL* de pe *server*-ul *Oracle*. Utilitarul transmite blocurile către motorul *PL/SQL* local care execută toate comenzile procedurale.
- Indiferent de mediu, motorul *PL/SQL* execută comenzile procedurale, dar trimite comenzile *SQL* către motorul *SQL* de pe *server*-ul *Oracle*.
  - Dacă nu ar fi incluse în blocuri *PL/SQL* comenzile *SQL* ar fi procesate separat, fiecare implicând câte un apel la *server*-ul *Oracle*.
- Comenzile procedurale pot fi executate pe stația *client* fără interacțiune cu *server*-ul *Oracle* sau în întregime pe *server*-ul *Oracle*.
  - Dacă utilitarul *Oracle* are motor *PL/SQL*, iar blocul *PL/SQL* nu conține comenzi *SQL*, atunci acesta este procesat local.

- ❖ Un subprogram *PL/SQL* stocat este un obiect al bazei de date și poate fi accesat de orice aplicație.
- ❖ Apelurile procedurilor care sunt stocate pe *server* sunt trimise pentru procesare motorului *PL/SQL* de pe *server*.
- ❖ Subprogramele *PL/SQL* (proceduri, funcții) declarate într-o aplicație *Developer Suite* (de exemplu, *Oracle Forms*) sunt diferite de cele stocate în baza de date. Acestea sunt procesate local.

1. Unde se află motorul *SQL*?
2. Unde se află motorul PL/SQL?
3. Se poate scrie cod PL/SQL folosind SQL*Plus sau SQL Developer?
4. Utilitarul SQL*Plus are motor PL/SQL? Dar utilitarul SQL Developer?
5. Comenzile PL/SQL pot fi incluse în cod SQL? Dar invers?
6. *SQL* transmite *server*-ului de baze de date ce să facă (declarativ), nu cum să facă. (adevărat/fals)
7. *PL/SQL* transmite *server*-ului de baze de date cum să facă (procedural). (adevărat/fals)

2.5. Evoluție

În 1977 Larry Ellison împreună cu câțiva prieteni înființează compania *Software Development Laboratories.*

În 1979 numele companiei este schimbat în *Relational Software, Inc*. Primul produs lansat de această companie a fost o bază de date relațională denumită *Oracle.* Deși era prima versiune, din motive de marketing a fost denumită *Oracle V2*. Pentru accesul la date era utilizat *SQL* și nu permitea tranzacții. În acel an *Relational Software, Inc* era singura companie care producea o bază de date compatibilă cu *SQL*.

În 1982 numele companiei a fost schimbat în *Oracle Corporation.* Limbajul ales pentru baza de date a fost modelat pe *ADA*, un limbaj de programare orientat obiect de nivel înalt care este extins din *Pascal*. *Oracle* a denumit acest limbaj *PL/SQL*. Fiind descendent al limbajelor *ADA* și *Pascal*, *PL/SQL* este un limbaj bazat pe structură de blocuri.

- *Oracle V3* (1983)
  - funcționalitățile *commit* și *rollback* pentru tranzacții
- *Oracle V4* (1984)
  - consistență la citire
- *Oracle V5* (1985)
  - arhitectura *client-server*
  - suport pentru cererile distribuite
- *Oracle V6* (1988) – prima versiune *Oracle* care suportă *PL/SQL*
  - limbaj limitat, bazat pe *script*-uri, nu pe proceduri stocate
  - gestiunea erorilor primitivă
  - *SQL* complet integrat
  - blocare la nivel de linie
- *Oracle V7* (1992)
  - proceduri stocate
  - integritate referențială
  - *trigger*-i
  - tablourile *PL/SQL*
  - pachetul *UTL_FILE* pentru a putea accesa fișierele sistemului de operare
  - pachetul *DBMS_SQL* pentru *SQL* dinamic
  - *Oracle Advanced Queuing*
  - *Oracle Enterprise Manager*
  - distribuire

- *Oracle V8* (1997)
  - orientarea obiect (tipuri obiect și metode)
  - rutine externe
  - suport pentru cereri stea
  - tipul *LOB*
  - tipuri colecție (vectori și tablouri imbricate)
  - aplicații multimedia
  - capacitatea de a realiza apeluri *HTTP* din baza de date
- *Oracle V8i* (1999) – devine bază de date comercială
  - *drop column*
  - partiționare și subpartiționare
  - *XML*
  - *WebDB*
  - funcții analitice în *SQL*
  - indexare *online*
  - tranzacții autonome
  - rutine externe *Java*
  - *SQL* dinamic nativ
  - operaţii *bulk bind*
  - *trigger*-i bază de date
  - îmbunătățire a performanței *SQL* și *PL/SQL*
  - baza de date încorporează *Java Virtual Machine* (*Oracle JVM* cunoscut și sub numele de *Aurora*)
- *Oracle V9i* (2001)
  - tabele externe
  - compilare nativă (*PL/SQL* la *C*)
  - *Oracle Streams*
  - *XML DB*
  - îmbunătățiri la nivel de *data warehouse* și *BI*
  - extindere și îmbunătățiri pentru *SQL analitic*
  - comenzi şi expresii *CASE*
  - tipuri noi de date în *SQL* şi *PL/SQL* (*DATATIME*, *TIMESTAMP*, colecţii pe mai multe niveluri)
  - ierarhii de tipuri şi subtipuri

- funcții tabel
- expresii CURSOR
- claselor *Java* în baza de date
- *Real Application Cluster*

- *Oracle V10g* (2003)
  - *recyclebin* (recuperare obiecte șterse)
  - permanentizări asincrone
  - criptarea transparentă a datelor (criptare automată la nivel de coloană)
  - îmbunătățire *SQL* analitic
  - *SQL Tuning Advisor*
  - îmbunătățire *OLAP*
  - *SQL Model Clause*
  - proceduri stocate .*Net*
- *Oracle V11g* (2007)

- Procedurile *PL/SQL* se execută mai rapid prin compilarea lor într-un cod nativ.
  - Procedurile sunt translatate în cod *C*, compilate cu ajutorul unui compilator *C* şi apoi automat preluate în procese *Oracle*.
  - Această tehnică, care nu cere restaurarea bazei de date, poate fi utilizată pentru proceduri şi pachete *Oracle*.
- *Oracle* furnizează soluţii (interfeţe *client-side* şi *server-side*, utilitare, *JVM* integrată cu *server*-ul *Oracle*) dezvoltatorilor de aplicaţii pentru crearea, gestionarea şi exploatarea aplicaţiilor *Java*.
- Procedurile *Java* stocate pot fi apelate dintr-un pachet *PL/SQL*, iar proceduri *PL/SQL* existente pot fi invocate din proceduri *Java*. Datele *SQL* pot fi accesate prin două interfeţe (*API*): *JDBC* şi *SQLJ*. Astfel:
  - pentru a invoca o procedură *Java* din *SQL* este nevoie de interfaţa *Java Stored Procedures*,
  - pentru a invoca dinamic comenzi *SQL* complexe este folosit *JDBC*,
  - pentru a utiliza comenzi *SQL* statice, simple (referitoare la un tabel ale cărui coloane sunt cunoscute) dintr-un obiect *Java* este folosit *SQLJ*.

2.6. Comparație cu alte limbaje

	PL/SQL	C	JAVA
Necesită BD sau utilitar Oracle	da	nu	nu
Orientat obiect	câteva caracteristici	nu	da
Performanță asupra BD Oracle	foarte eficient	mai puțin eficient	mai puțin eficient
Portabil pe diferite SO	da	oarecum	da
Ușor de învățat	relativ ușor	mai dificil	mai dificil

2.7. Diagrama bazei de date utilizată în exemple

- Gestiunea activității unei companii comerciale
  Vezi diagrame curs
- Schema simplificată
  - Entități

DEPOZITE(id_depozit#, denumire, adresa, oras, judet, orar, capacitate, valoare, *id_director*, *id_tara*)

SECTOARE(id_sector#, descriere, id_depozit)

ZONE (id_zona#, descriere, capacitate_maxima, capacitate_folosita, id_sector)

PRODUSE (id_produs#, denumire, descriere, stoc_curent, stoc_impus, pret_unitar, greutate, volum, tva, id_zona, id_um, id_categorie, data_crearii, data_modificarii, activ)

CARACTERISTICI (#id_caracteristica, denumire, descriere)

UNITATI_MASURA(id_um#, denumire, descriere)

CATEGORII (id_categorie#, denumire, nivel, id_parinte)

FACTURI(id_factura#, data, status, id_casa, id_client, id_adresa_livrare, id_adresa_facturare, id_tip_livrare, id_tip_plata, interval_livrare)

CASE(id_casa#, nume, serie, parola)

TIP_LIVRARE(id_tip_livrare#, denumire, tarif, *id_firma_t*)

ADRESE(id_adresa#, strada, oras, tara, cod_postal, id_client)

TIP_PLATA(id_tip_plata#, cod, descriere)

CLIENTI(id_client#, telefon, email, tip, oras, data_crearii, data_modificarii)

- Subentități

PERSOANE_FIZICE(id_client_f#, nume, prenume, cnp)

PERSOANE_JURIDICE(id_client_j#, denumire, persoana_contact, cui, cont, banca, cod_fiscal, numar_inregistrare)

- Tabele asociative

CLIENTI_AU_PRET_PREFERENTIAL(id_pret_pref#, id_categorie, id_client_j, discount, data_in, data_sf)

PRODUSE_AU_CARACTERISTICI (id_produs#, id_caracteristica#, valoare)

FACTURI_CONTIN_PRODUSE (id_factura#, id_produs#, cantitate, pret_facturare)

2.8. De ce este utilizat *PL/SQL*?

- În practică există numeroase situații în care *SQL* se dovedește a fi limitat.
- Exemplu – vezi explicații curs
  - Clienții companiei pot avea prețuri personalizate în funcție de anumite criterii. Comenzile se realizează *online*, clienții consultând cataloage cu prețuri personalizate.
  - În cazul în care prețul personalizat se calculează raportat la categoriile de produse este nevoie de o clasificare a clienților în funcție de categorie și numărul de produse cumpărate din categoria respectivă.
  - Tabelul *CLASIFIC_CLIENTI* (*id_client*, *id_categorie*, *nr_produse*, *clasificare*) conținea deja o clasificare a clienților.
  - Se decide o nouă clasificare a acestora conform tabelului de mai jos.

id_categorie	clasificare	număr produse cumpărate
1	A	> 1000
	B	≥ 500
	C	≥ 0
2	A	> 2000
	B	≥ 1000
	C	≥ 200
	D	≥ 0

- Soluție *SQL* posibilă

Varianta1

```
UPDATE clasific_clienti
SET    clasificare = 'A'
WHERE  nr_produse > 1000
AND    id_categorie = 1;


UPDATE clasific_clienti
SET    clasificare = 'B'
WHERE  nr_produse BETWEEN 500 AND 1000
AND    id_categorie = 1;
…
```

1. Câte comenzi *UPDATE* sunt necesare pentru datele din tabelul anterior?

2. Este eficientă o astfel de abordare?

3. Există alte metode de abordare utilizând *SQL*?

Varianta2

```
UPDATE clasific_clienti
SET    clasificare = CASE WHEN nr_produse>1000
                          THEN 'A'
                       WHEN nr_produse BETWEEN 500
                          AND 1000 THEN 'B'
                       ELSE 'C' END
WHERE id_categorie = 1;


…
```

Varianta3

```
UPDATE clasific_clienti
SET    clasificare =
          CASE WHEN nr_produse>1000
                       AND id_categorie=1 THEN 'A'
               WHEN nr_produse BETWEEN 500 AND 1000
                       AND id_categorie = 1 THEN 'B'
               WHEN nr_produse BETWEEN 0 AND 499
                       AND id_categorie=1 THEN 'C'
               WHEN nr_produse>2000
                       AND id_categorie=2 THEN 'A'
               WHEN nr_produse BETWEEN 1000 AND 2000
                       AND id_categorie = 2 THEN 'B'
               WHEN nr_produse BETWEEN 200 AND 999
                       AND id_categorie=2 THEN 'C'
               ELSE 'D' END;
```

4. Ce se întâmplă atunci când există mai multe categorii de produse și mai multe niveluri de clasificare pentru fiecare categorie?

- Soluție *PL/SQL* posibilă

Varianta1

```
DECLARE
    CURSOR info IS
           SELECT id_client, id_categorie, nr_produse
           FROM   clasific_clienti;
    v_clasific clasific_clienti.clasificare%type;
BEGIN
    FOR i IN info LOOP
        CASE WHEN i.nr_produse>1000
                  AND i.id_categorie=1
                  THEN v_clasific := 'A';
             WHEN i.nr_produse BETWEEN 500 AND 1000
                  AND i.id_categorie = 1
                  THEN v_clasific := 'B';
             WHEN i.nr_produse BETWEEN 0 AND 499
                  AND i.id_categorie=1
                  THEN v_clasific := 'C';
             WHEN i.nr_produse>2000
                  AND i.id_categorie=2
                  THEN v_clasific := 'A';
             WHEN i.nr_produse BETWEEN 1000 AND 2000
                  AND i.id_categorie = 2
                  THEN v_clasific := 'B';
             WHEN i.nr_produse BETWEEN 200 AND 999
                  AND i.id_categorie=2
                  THEN v_clasific := 'C';
             ELSE v_clasific := 'D';
        END CASE;
        UPDATE clasific_clienti
        SET    clasificare = v_clasific
        WHERE  id_client = i.id_client
        AND    id_categorie = i.id_categorie;
    END LOOP;
END;
/
```

1. De câte ori se execută comanda *UPDATE*?
2. Este eficientă o astfel de abordare?
3. Cât timp este blocată resursa?
4. Ar fi utilă o cheie primară artificială?

Varianta2

```
DECLARE
    CURSOR info IS
           SELECT id_client, id_categorie, nr_produse
           FROM   clasific_clienti
    FOR UPDATE;

    v_clasific clasific_clienti.clasificare%type;
BEGIN
    FOR i IN info LOOP
        CASE WHEN i.nr_produse>1000
                  AND i.id_categorie=1
                  THEN v_clasific := 'A';
             WHEN i.nr_produse BETWEEN 500 AND 1000
                  AND i.id_categorie = 1
                  THEN v_clasific := 'B';
             WHEN i.nr_produse BETWEEN 0 AND 499
                  AND i.id_categorie=1
                  THEN v_clasific := 'C';
             WHEN i.nr_produse>2000
                  AND i.id_categorie=2
                  THEN v_clasific := 'A';
             WHEN i.nr_produse BETWEEN 1000 AND 2000
                  AND i.id_categorie = 2
                  THEN v_clasific := 'B';
             WHEN i.nr_produse BETWEEN 200 AND 999
                  AND i.id_categorie=2
                  THEN v_clasific := 'C';
             ELSE v_clasific := 'D';
        END CASE;

        UPDATE clasific_clienti
        SET    clasificare = v_clasific
        WHERE  CURRENT OF info;
    END LOOP;
END;
/
```

5. Este mai eficientă această abordare? Cât timp este blocată resursa?

6. Există şi alte metode de rezolvare a problemei?

7. Numărul de produse cumpărate de client trebuie să fie în permanență actualizat şi să corespundă situației prezente. Se poate realiza *realtime* acest lucru cu *SQL*?

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database 11g*: *PL/SQL Fundamentals*, *Student Guide*, Oracle University (2009)
4. *A Mini-History of Oracle and PL/SQL*, L. Cunningham (2012) (*http://www.dba-oracle.com/t_edb_pl_sql_features_release.htm*)
5. *Oracle Database*, Wikipedia (2012) (*http://en.wikipedia.org/wiki/Oracle_Database*)