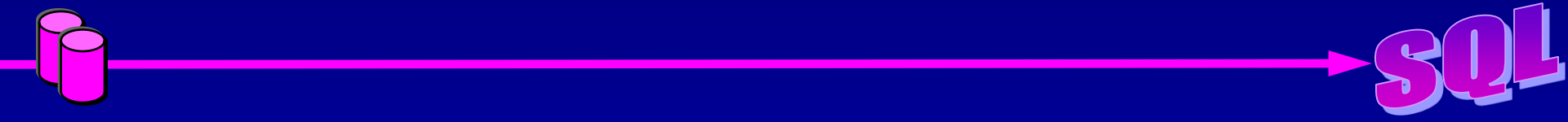
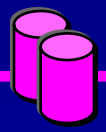


Le Langage de BLOC

PL/SQL

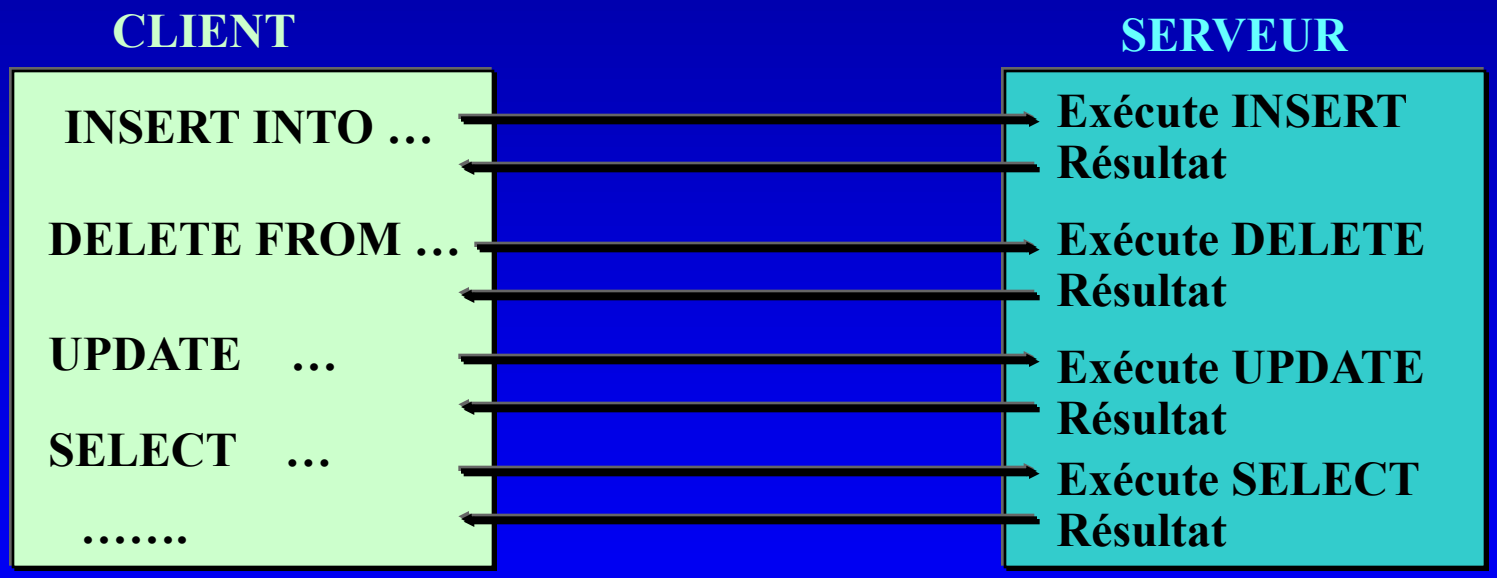
Le Langage de Bloc
PL/SQL # SQL



- SQL : langage ensembliste
  - Ensemble de requêtes distinctes
  - Langage de 4ème génération : on décrit le résultat sans dire comment il faut accéder aux données
  - Obtention de certains résultats : encapsulation dans un langage hôte de 3ème génération
- PL/SQL
  - 'Procédural Language' : sur-couche procédurale à SQL, boucles, contrôles, affectations, exceptions, ….
  - Chaque programme est un bloc (BEGIN – END)
  - Programmation adaptée pour :
    - Transactions
    - Une architecture Client - Serveur

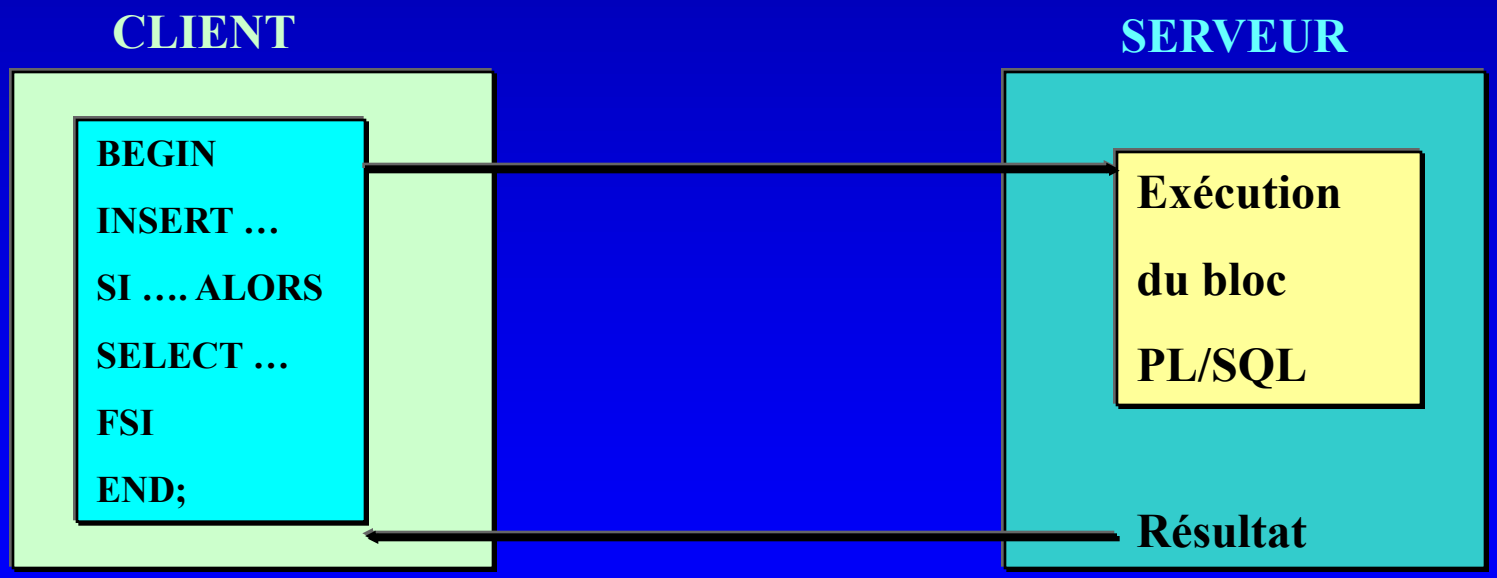
Requêtes SQL

- Chaque requête ‘client’ est transmise au serveur de données pour être exécutée avec retour de résultats

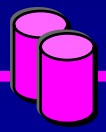


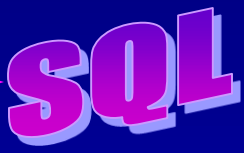
Bloc PL/SQL

- Le bloc de requêtes est envoyé sur le serveur. Celui-ci exécute le bloc et renvoie 1 résultat final.



Format d'un bloc PL/SQL





- Section DECLARE : déclaration de
  - Variables locales simples
  - Variables tableaux
  - cursors
- Section BEGIN
  - Section des ordres exécutables
  - Ordres SQL
  - Ordres PL
- Section EXCEPTION
  - Réception en cas d'erreur
  - Exceptions SQL ou utilisateur

```
DECLARE
        --déclarations
BEGIN
        --exécutions
EXCEPTION
        --erreurs
END;
/
```

Variables simples

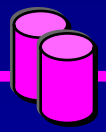
- Variables de type SQL

```
nbr      NUMBER(2)  ;
nom      VARCHAR(30)  ;
minimum  CONSTANT  INTEGER  := 5  ;
salaire  NUMBER(8,2)  ;
debut    NUMBER  NOT  NULL  ;
```

- Variables de type booléen (TRUE, FALSE, NULL)

```
fin      BOOLEAN  ;
reponse  BOOLEAN  DEFAULT  TRUE  ;
ok       BOOLEAN  := TRUE;
```

Variables faisant référence au dictionnaire de données



- Référence à une colonne (table, vue)

```
vsalaire    employe.salaire%TYPE;
vnom        etudiant.nom%TYPE;
Vcomm       vsalaire%TYPE;
```

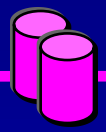
- Référence à une ligne (table, vue)

```
vemploye    employe%ROWTYPE;
vetudiant   etudiant%ROWTYPE;
```

  - Variable de type ‘struct’
  - Contenu d’une variable : variable.colonne

```
vemploye.adresse
```

Tableaux dynamiques



- **Déclaration d'un type tableau**

```
TYPE  <nom du type du tableau>
IS TABLE OF <type de l'élément>
INDEX BY BINARY_INTEGER;
```

- **Affectation (héritage) de ce type à une variable**

```
<nom élément>  <nom du type du tableau>;
```

- **Utilisation dans la section BEGIN : un élément du tableau :**

```
<nom élément>(rang dans le tableau)
```

Tableaux dynamiques variables simples

- Déclaration d'un tableau avec des éléments numériques

```
TYPE  type_note_tab
IS TABLE OF NUMBER(4,2)
INDEX BY BINARY_INTEGER;
tab_notes type_note_tab;
i NUMBER;
```

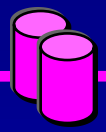
```
i:=1;
tab_notes(i):= 12.50;
```

- Déclaration d'un tableau avec des éléments caractères

```
TYPE  type_nom_tab
IS TABLE OF VARCHAR(30)
INDEX BY BINARY_INTEGER;
tab_noms type_nom_tab;
i NUMBER;
```

```
i:=1;
tab_noms(i):= 'Asma';
```

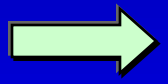
Tableaux dynamiques variables simples avec héritage



- **Tableau avec éléments hérités**

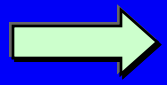
```
TYPE  type_note_tab
IS TABLE OF examen.note%TYPE
INDEX BY BINARY_INTEGER;
tab_notes type_note_tab;
i NUMBER;
```

```
i:=1;
tab_notes(i) := 12.50;
```



```
TYPE  type_nom_tab
IS TABLE OF etudiant.nom%TYPE
INDEX BY BINARY_INTEGER;
tab_noms type_nom_tab;
i NUMBER;
```

```
i:=1;
tab_noms(i):= 'Asma';
```



Tableaux dynamiques avec des éléments de type RECORD

- Type RECORD : plusieurs variables dans un élément

```
TYPE  type_emp_record
(idEmp NUMBER,
 nomEmp VARCHAR(30),
 adrEmp VARCHAR(80));
```

```
TYPE  type_emp_tab
IS TABLE OF type_emp_record
INDEX BY BINARY_INTEGER;
tab_emps type_emp_tab;
i NUMBER;
```

```
i:=1;
tab_emps(i).idEmp:= 100;
tab_emps(i).nomEmp:= 'Asma';
tab_emps(i).adrEmp:= 'Alger';
```

Tableaux dynamiques avec des éléments de type ROW

- Type ROW : chaque élément est une variable ‘struct’

```
TYPE  type_emp_tab
IS TABLE OF employe%ROWTYPE
INDEX BY BINARY_INTEGER;
tab_emps type_emp_tab;
i NUMBER;
```

```
i:=1;
tab_emps(i).idE:= 100;
tab_emps(i).nom:= 'Asma';
tab_emps(i).adresse:= 'Alger';
```

Variables paramétrées lues sous SQLPLUS : &

- Variables lues par un ACCEPT …. PROMPT

+

```
ACCEPT plu PROMPT 'Entrer la valeur : '
```

PL

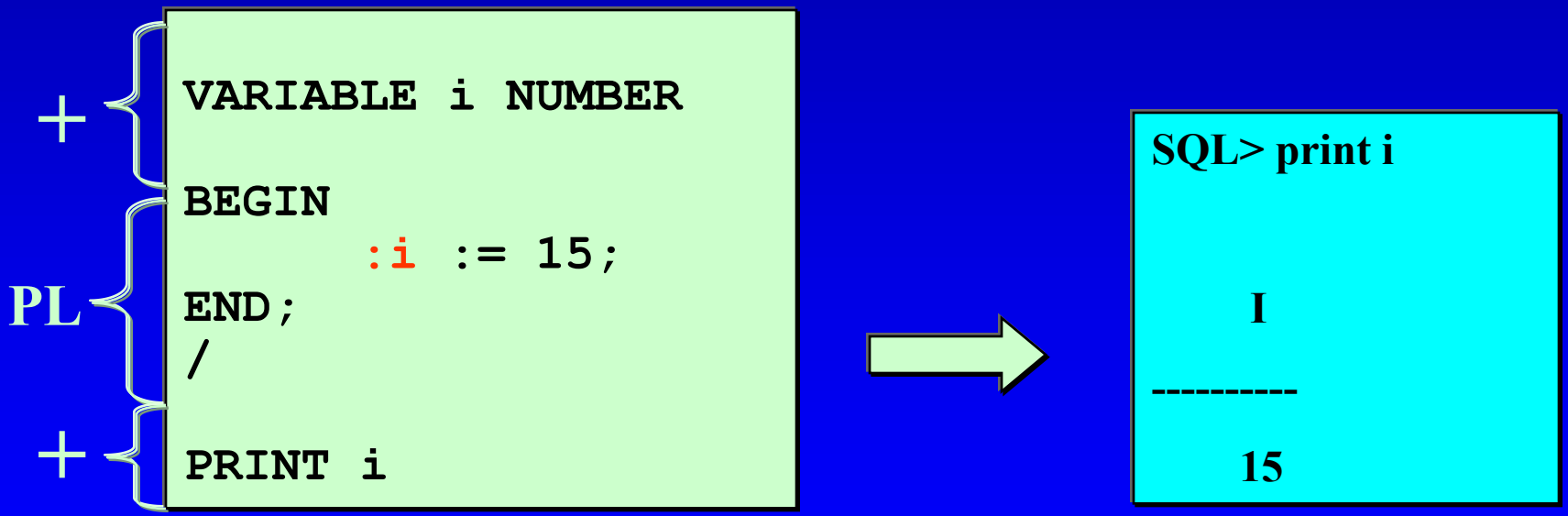
```
DECLARE
      -- déclarations
BEGIN
      -- travail avec le contenu de plu :
      -- &plu si numérique
      -- '&plu' si caractère
END;
/
```

+

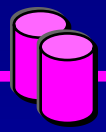
```
-- Ordre SQL .....
```

Variables en sortie sous SQLPLUS : :

- Variable déclarée sous sqlplus , utilisée dans le bloc PL puis affichée sous sqlplus



Instructions PL



- **Affectation (:=)**
  - A := B;
- **Structure alternative ou conditionnelle**
  - Opérateurs SQL : >,<,….,OR,AND,….,BETWEEN,LIKE,IN
  - IF …. THEN ….. ELSE ……END IF;

```
IF condition THEN
      instructions;
      ELSE
      instructions;
      IF condition THEN instructions;
            ELSIF condition THEN instructions;
                  ELSE instructions;
END IF;
```

Structure alternative : CASE (1)

- **Choix selon la valeur d'une variable**

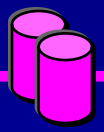
```
CASE variable

      WHEN valeur1 THEN action1;
      WHEN valeur2 THEN action2;
      ……
      ELSE  action;

END CASE;
```

Structure alternative : CASE (2)

- **Plusieurs choix possibles**

```
CASE

      WHEN expression1 THEN   action1;
      WHEN expression2 THEN   action2;
      ……
      ELSE  action;

END CASE;
```

Structure itérative



- **LOOP**

```
LOOP
      instructions;
      EXIT WHEN (condition);
END LOOP;
```

- **FOR**

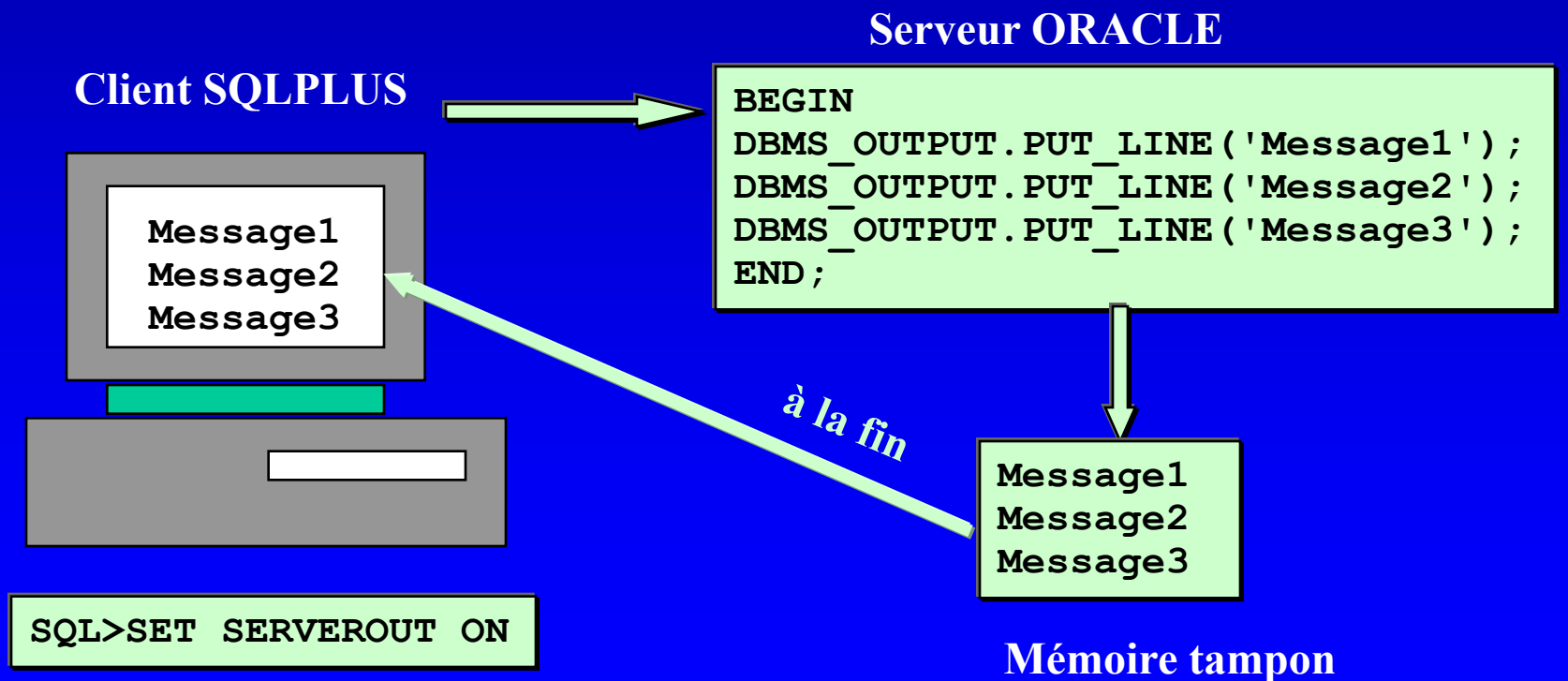
```
FOR (indice IN [REVERSE] borne1..borne2) LOOP
      instructions;
END LOOP;
```

- **WHILE**

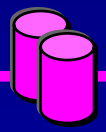
```
WHILE (condition) LOOP
      instructions;
END LOOP;
```

Affichage de résultats intermédiaires Package DBMS_OUTPUT

- Messages enregistrés dans une mémoire tampon côté serveur
- La mémoire tampon est affichée sur le poste client à la fin



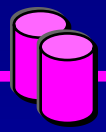
Le package DBMS_OUTPUT



- Écriture dans le buffer avec saut de ligne
  - DBMS_OUTPUT.PUT_LINE(<chaîne caractères>);
- Écriture dans le buffer sans saut de ligne
  - DBMS_OUTPUT.PUT(<chaîne caractères>);
- Écriture dans le buffer d'un saut de ligne
  - DBMS_OUTPUT.NEW_LINE;

```
DBMS_OUTPUT.PUT_LINE('Affichage des n premiers ');
DBMS_OUTPUT.PUT_LINE('caractères en ligne ');
FOR i IN 1..n LOOP
      DBMS_OUTPUT.PUT(tab_cars(i));
END LOOP;
DBMS_OUTPUT.NEW_LINE;
```

Sélection mono – ligne
SELECT …. INTO



- Toute valeur de colonne est rangée dans une variable avec INTO

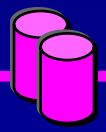
```
SELECT nom,adresse,tel INTO vnom,vadresse,vtel
FROM etudiant WHERE ide=&nolu;
```

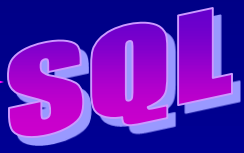
```
SELECT nom,adresse,libDip INTO vnom,vadresse,vdip
FROM etudiant e, diplôme d WHERE ine=&nolu
AND e.idDip=d.idDip;
```

- Variable ROWTYPE

```
SELECT * INTO vretud FROM etudiant WHERE ine=&nolu;
……
DBMS_OUTPUT.PUT_LINE('Nom étudiant : '||vretud.nom);
……
```

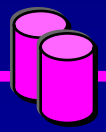
Sélection multi – ligne : les CURSEURS
Principe des curseurs





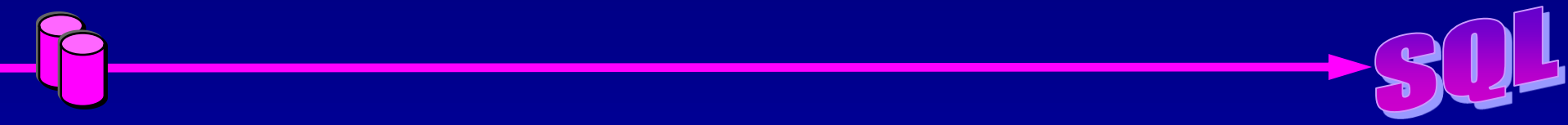
- Obligatoire pour sélectionner plusieurs lignes
- Zone mémoire (SGA : Share Global Area) partagée pour stocker les résultats
- Le curseur contient en permanence l'@ de la ligne courante
- Curseur implicite
  - SELECT t.* FROM table t WHERE ……
  - t est un curseur utilisé par SQL
- Curseur explicite
  - DECLARE CURSOR →

Démarche générale des curseurs



- **Déclaration du curseur : DECLARE**
  - Ordre SQL sans exécution
- **Ouverture du curseur : OPEN**
  - SQL ‘monte‘ les lignes sélectionnées en SGA
- **Sélection d’une ligne : FETCH**
  - Chaque FETCH ramène une ligne dans le programme client
  - Tant que ligne en SGA : FETCH
- **Fermeture du curseur : CLOSE**
  - Récupération de l’espace mémoire en SGA

Traitement d'un curseur



Programme PL/SQL

variables

```
DECLARE
CURSOR c1 IS SELECT ……;
BEGIN
   OPEN c1;
   FETCH c1 INTO ……;
   WHILE (c1%FOUND) LOOP
   ……
   ……
       FETCH c1 INTO ……;
   END LOOP;
   CLOSE c1;
END;
```

FETCH

SGA

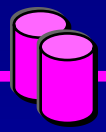
OPEN

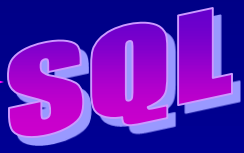
BD

Gestion ‘classique’ d’un curseur

```
DECLARE
CURSOR c1 IS SELECT nom,moyenne FROM etudiant;
vnom            etudiant.nom%TYPE;
vmoyenne        etudiant.moyenne%TYPE;
e1 ,e2 NUMBER;
BEGIN
        OPEN c1;
        FETCH c1 INTO vnom,vmoyenne;
        WHILE c1%FOUND LOOP
        IF vmoyenne < 10 THEN e1:=e1+1;
                INSERT INTO liste_refus VALUES(vnom);
        ELSE    e2:=e2+1;
                INSERT INTO liste_recus VALUES(vnom);
        END IF;
        FETCH c1 INTO vnom,vmoyenne;
        END LOOP;
        CLOSE c1;
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(e2)||'Reçus ');
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(e1)||‘Refus ');
        COMMIT;
END;
```

Les variables système des Curseurs





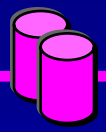
- Curseur%FOUND
  - Variable booléenne
  - Curseur toujours ‘ouvert’ (encore des lignes)
- Curseur%NOTFOUND
  - Opposé au précédent
  - Curseur ‘fermé’ (plus de lignes)
- Curseur%COUNT
  - Variable number
  - Nombre de lignes déjà retournées
- Curseur%ISOPEN
  - Booléen : curseur ouvert ?

Gestion ‘automatique’ des curseurs

```
DECLARE
CURSOR c1 IS SELECT nom,moyenne FROM etudiant;
-- PAS DE DECLARATION DE VARIABLE DE RECEPTION
e1 ,e2 NUMBER;
BEGIN
        --PAS D’OUVERTURE DE CURSEUR
        --PAS DE FETCH
        FOR c1_ligne IN c1 LOOP
        IF c1_ligne.moyenne < 10 THEN e1:=e1+1;
        INSERT INTO liste_refus VALUES(c1_ligne.nom);
        ELSE   e2:=e2+1;
        INSERT INTO liste_recus VALUES(c1_ligne.nom);
        END IF;
        END LOOP;
        --PAS DE CLOSE
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(e2)||'Reçus ');
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(e1)||‘Refus ');
        COMMIT;
END;
```

Variable STRUCT de réception

Curseurs et Tableaux
exemple final




```
DECLARE
CURSOR c1 IS SELECT nom,moyenne FROM etudiant
WHERE moyenne>=10 ORDER BY nom DESC;
TYPE type_nom_tab IS TABLE OF etudiant.nom%TYPE
INDEX BY BINARY_INTEGER;
tab_noms type_nom_tab;
i,j NUMBER;
BEGIN  /* Remplissage tableau */
       i:=1;
       FOR c1_ligne IN c1 LOOP
       tab_noms(i):= c1_ligne.nom;
       i:=i+1;
       END LOOP; /* Affichage du tableau */
       FOR j IN 1..i-1 LOOP
       DBMS_OUTPUT.PUT_LINE('Rang : '||TO_CHAR(j)||
       'Etudiant : '||tab_nom(j));
       END LOOP;
END;
```

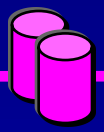
Gestion des Exceptions
Principe

- Toute erreur (SQL ou applicative) entraîne automatiquement un débranchement vers le paragraphe EXCEPTION :

```
BEGIN
        instruction1;
        instruction2;
        ……
        instructionn;
EXCEPTION
        WHEN exception1 THEN
        ……
        WHEN exception2 THEN
        ……
        WHEN OTHERS THEN
        ……
END;
```

Débranchement involontaire (erreur SQL)
ou volontaire (erreur applicative)

Deux types d'exceptions



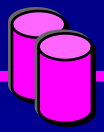
- **Exceptions SQL**
  - **Déjà définies (pas de déclaration)**
    - **DUP_VAL_ON_INDEX**
    - **NO_DATA_FOUND**
    - **OTHERS**
  - **Non définies**
    - **Déclaration obligatoire avec le n° erreur (sqlcode)**

```
nomerreur EXCEPTION;
PRAGMA EXCEPTION_INIT(nomerreur,n°erreur);
```

- **Exceptions applicatives**
  - **Déclaration sans n° erreur**

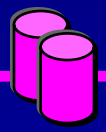
```
nomerreur EXCEPTION;
```

Exemple de gestion d'exception (1)

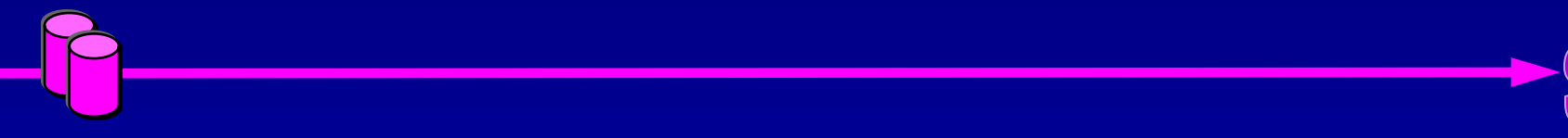



```
DECLARE
tropemprunt   EXCEPTION;
i NUMBER;
BEGIN
        i:=1;
        SELECT ……
        i:=2;
        SELECT ……
        IF …… THEN RAISE tropemprunt; ……
EXCEPTION
        WHEN NO_DATA_FOUND THEN
        IF i=1 THEN ……
                ELSE
        END IF;
        WHEN tropemprunt THEN
        ……
        WHEN OTHERS THEN
        ……
END;
```

Exemple de gestion d'exception (2)



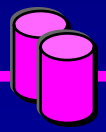

```
DECLARE
enfant_sans_parent  EXCEPTION;
PRAGMA EXCEPTION_INIT(enfant_sans_parent,-2291);
BEGIN
       INSERT INTO fils VALUES ( ……. );

EXCEPTION
       WHEN enfant_sans_parent THEN
       ……..
       WHEN OTHERS THEN
       ……..
END;
```



Procédures Stockées

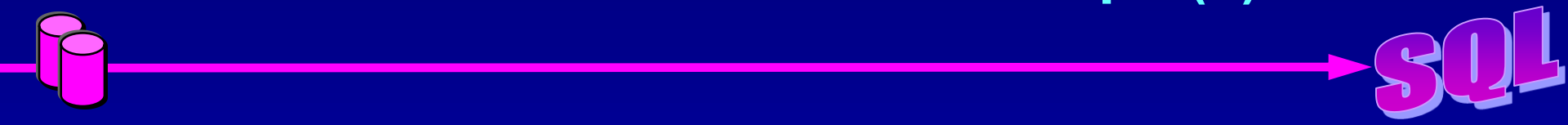
Fonctions

Procédures Stockées : Principe (1)



- Programme (PL/SQL) stocké dans la base
- Le programme client exécute ce programme en lui passant des paramètres (par valeur)
- Si le code est bon , le SGBD conserve le programme source (USER_SOURCE) et le programme compilé
- Le programme compilé est optimisé en tenant compte des objets accélérateurs (INDEX, CLUSTER, PARTITION, …)

Procédures Stockées : Principe (2)



CLIENT **SERVEUR**

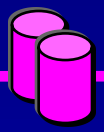
EXECUTE P(p1, p2);

PROCEDURE P(v1,v2) AS
BEGIN
Ordre SQL et PL/SQL
……..
END P;

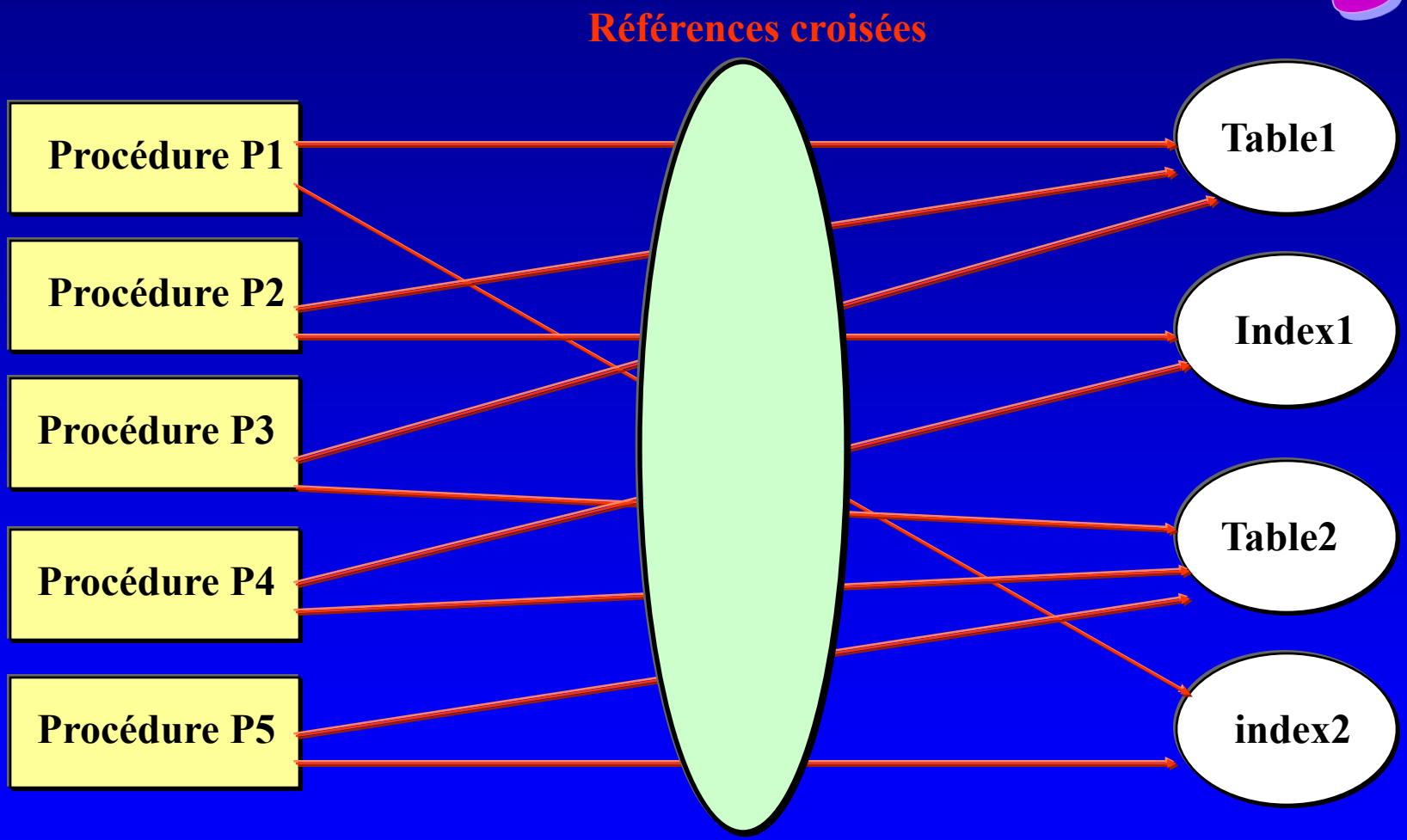
P

Retour résultats

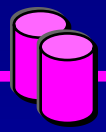
Optimisation des procédures liens avec les objets

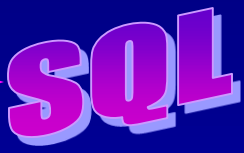


SQL



Optimisation des procédures

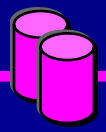




- Recompilation automatique d'une procédure si un objet est modifé
- Recompilation manuelle possible

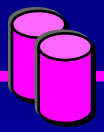
```
ALTER PROCEDURE <nom_procédure> COMPILE;
```

Avantages des procédures stockées



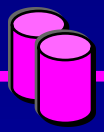
- Vitesse : programme compilé et optimisé
  - Une requête SQL normale est interprétée et optimisée à chaque exécution
- Intégrité : encapsulation des données
  - Vers le modèle Objet
  - Droit d'exécution et plus de manipulation
  - Les règles de gestion sont données sur le serveur en un seul exemplaire
- Performance : moins de transfert réseau
  - Plus de transfert de bloc de programme
  - Une procédure pour plusieurs utilisateurs
- Abstraction : augmentation du niveau d'abstraction des développeurs Client
- Performance :
  - Extensibilité, Modularité, Réutilisation, Maintenance

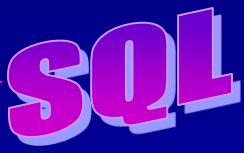
Déclaration d'une procédure stockée




```
CREATE [OR REPLACE] PROCEDURE <nom_procédure>
[(variable1 type1, ..., variablen typen [OUT])] AS
...
-- déclarations des variables et
-- curseurs utilisées dans le corps de la procédure
BEGIN
....
-- instructions SQL ou PL/SQL
EXCEPTION
....
END;
/
```

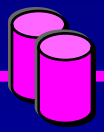
Exemple 1 de procédure stockée inscription d'un étudiant






```
CREATE PROCEDURE inscription (ide varchar2(10),pnom
 varchar2(30),spec varchar2(30),ann_ins number)
AS

BEGIN

DBMS_OUTPUT.PUT_LINE('Début inscription de ||pnom');
INSERT INTO etudiant VALUES(ide,pnom,spec);
INSERT INTO inscrire VALUES(ide,ann_ins);
DBMS_OUTPUT.PUT_LINE('Transaction réussie');
COMMIT;

END;
/
```

Exemple 1 : appel de la procédure



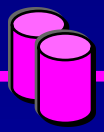
- **A partir de sqlplus**

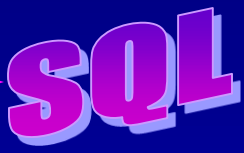
```
ACCEPT vide PROMPT 'Entrer le matricule : '
……
EXECUTE inscription(‘&ide’,'&vnom',’&an_ins‘,
&spec’);
```

- **A partir de PL/SQL**

```
inscription(ide,nom,an_ins, spec);
```

Exemple 2 : avec retour de valeurs suppression d'un étudiant

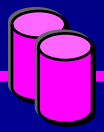


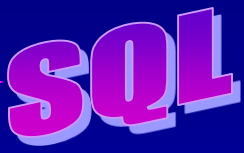



```
CREATE PROCEDURE suppression (pidEtu NUMBER,
retour OUT NUMBER) AS
inscriptions EXCEPTION;
PRAGMA EXCEPTION_INIT(inscriptions,-2292);
vnom etudiant.nom%TYPE;
BEGIN
SELECT nom INTO vnom FROM etudiant WHERE idEtu=pidEtu;
DELETE FROM etudiant WHERE idEtu=pidEtu;
DBMS_OUTPUT.PUT_LINE('Etudiant '||vnom||' supprimé');
COMMIT;
retour:=0;
```

../..

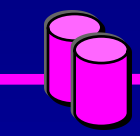
Exemple 2 : avec retour de valeurs suppression d'un étudiant (suite)

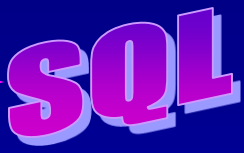





```
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Etudiant'||TO_CHAR(pidEtu)||'inconnu);
retour:=1;
WHEN inscriptions THEN
DBMS_OUTPUT.PUT_LINE('Encore des inscriptions');
retour:=2;
……
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(SQLERRM);
retour:=9;
END;
/
```

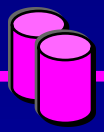
Exemple 2 : appel avec retour

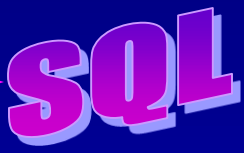





```
VARIABLE ret NUMBER
ACCEPT vnom PROMPT 'Entrer le nom : '
……
EXECUTE inscription('&vnom',……, '&vdip',:ret);
PRINT ret
```

Les Fonctions stockées





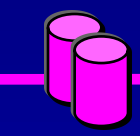
- Comme une procédure mais qui ne retourne qu'un seul résultat
- Même structure d'ensemble qu'une procédure
- Utilisation du mot clé **RETURN** pour retourner le résultat
- Appel possible à partir de :
  - Une requête SQL normale
  - Un programme PL/SQL
  - Une procédure stockée ou une autre fonction stockée

Déclaration d'une fonction stockée

```
CREATE [OR REPLACE] FUNCTION nom_fonction
[(paramètre1 type1, …… , paramètren typen)]
RETURN type_résultat IS
-- déclarations de variables,curseurs et exceptions
BEGIN
-- instructions PL et SQL

RETURN(variable);
END;
/
```

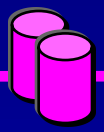
1 ou plusieurs RETURN

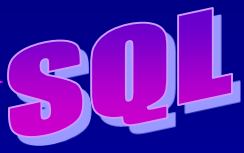
Exemple 1 de fonction stockée




```
CREATE OR REPLACE FUNCTION moy_points_marques
(eqj joueur.ideq%TYPE)
RETURN NUMBER IS
moyenne_points_marques NUMBER(4,2);
BEGIN
SELECT AVG(totalpoints) INTO moyenne_points_marques
FROM joueur WHERE ideq=eqj;
RETURN(moyenne_points_marques);
END;
/
```

Utilisation d'une fonction





- A partir d'une requête SQL

```
SELECT moy_points_marques('e1') FROM dual;
```

```
SELECT nomjoueur FROM joueur WHERE
totalpoints > moy_points_marques('e1');
```

- A partir d'une procédure ou fonction

```
BEGIN
……
IF moy_points_marques(equipe) > 20 THEN ……

END;
```

Exemple 2 de fonction stockée

```
CREATE OR REPLACE FUNCTION bon_client
(pidclient NUMBER, pchiffre NUMBER)
RETURN BOOLEAN IS
total_chiffre NUMBER;
BEGIN
SELECT SUM(qte*prix_unit) INTO total_chiffre
FROM commande WHERE idclient=pidclient;
IF total_chiffre > pchiffre THEN
      RETURN(TRUE);
ELSE  RETURN(FALSE);
END IF;
END;
```

```
BEGIN
……
IF bon_client(client,10000) THEN ……
……
```