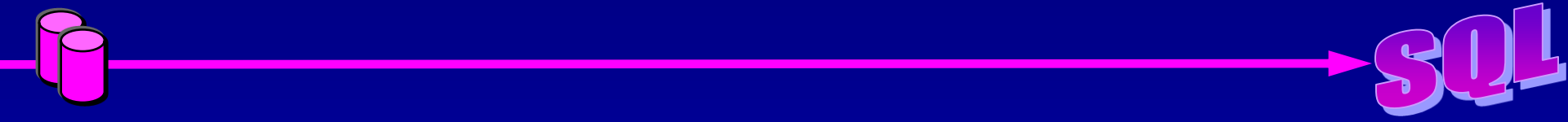



# Le Langage de BLOC

# PL/SQL

# Le Langage de Bloc
# PL/SQL # SQL

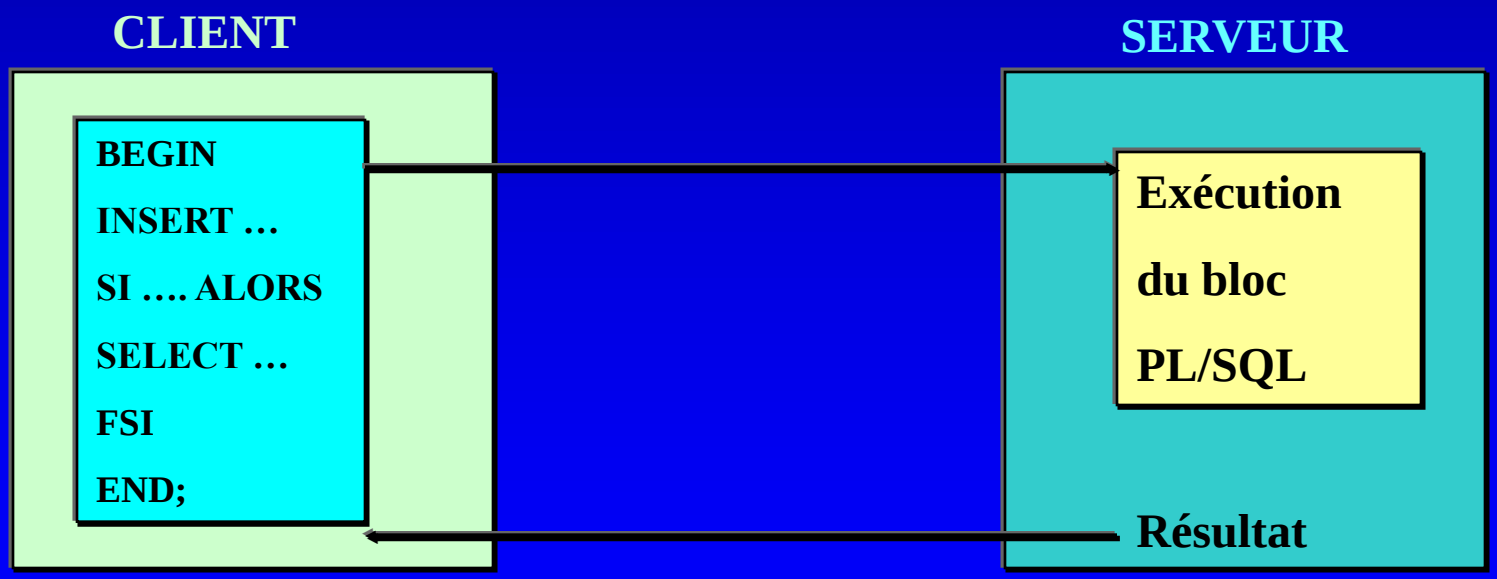

- **SQL : langage ensembliste**
  - **Ensemble de requêtes distinctes**
  - **Langage de 4ème génération : on décrit le résultat sans dire comment il faut accéder aux données**
  - **Obtention de certains résultats : encapsulation dans un langage hôte de 3ème génération**
- **PL/SQL**
  - **'Procedural 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
```

# 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**

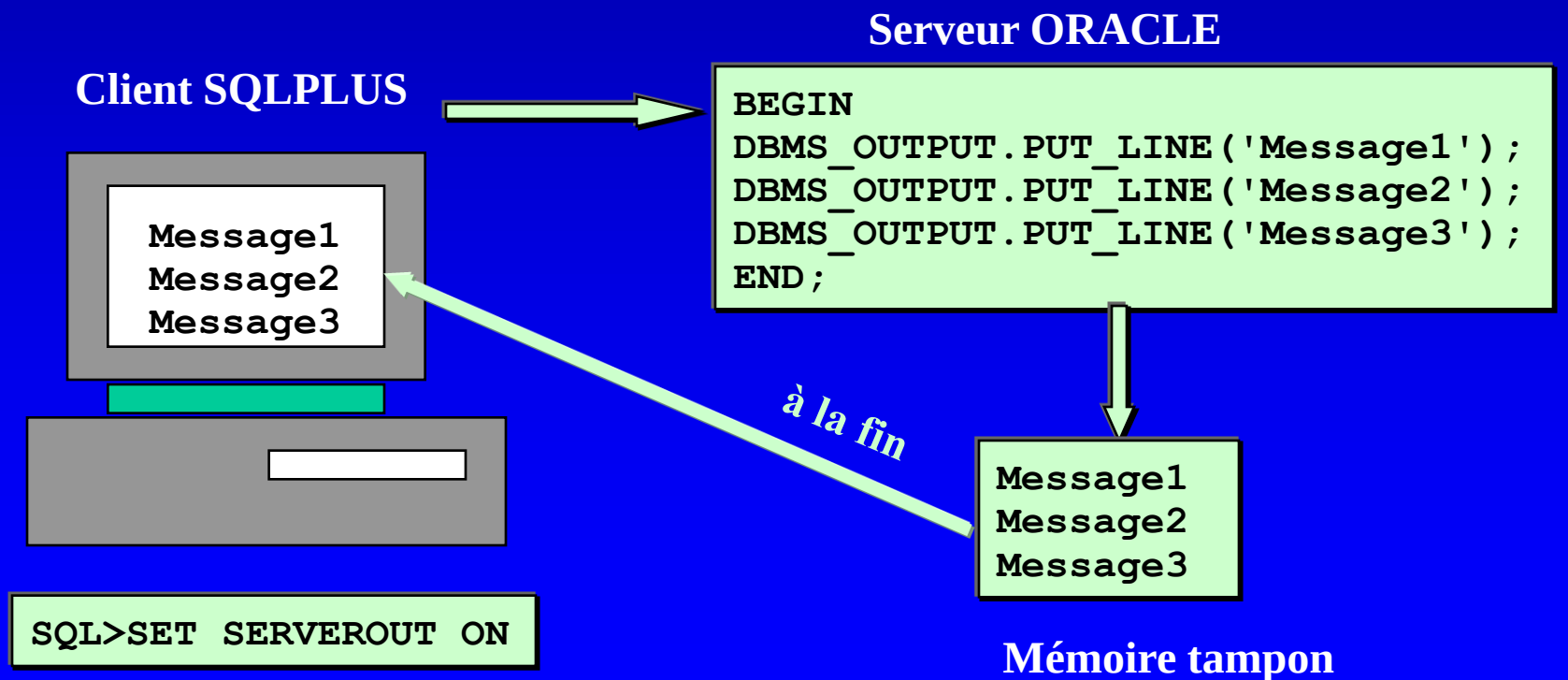
```
FOR (indice IN [REVERSE] borne1..borne2) LOOP
      instructions;
END LOOP;
```

- **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);
……
```

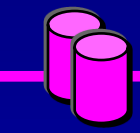
# Sélection multi – ligne : les CURSEURS
## Principe des curseurs


- 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





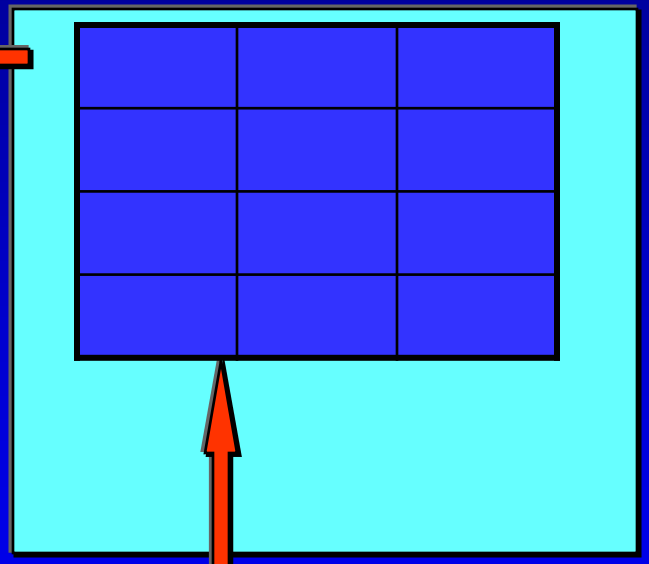

**Programme PL/SQL**

variables

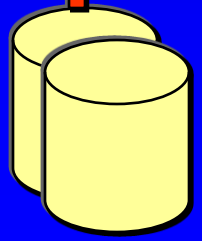
```
DECLARE
CURSOR c1 IS SELECT ……;
BEGIN
   OPEN c1;
   FETCH c1 INTO ……;
   WHILE (c1%FOUND) LOOP
   ……
   ……
      FETCH c1 INTO ……;
   END LOOP;
   CLOSE c1;
END;
```

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 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 (2)

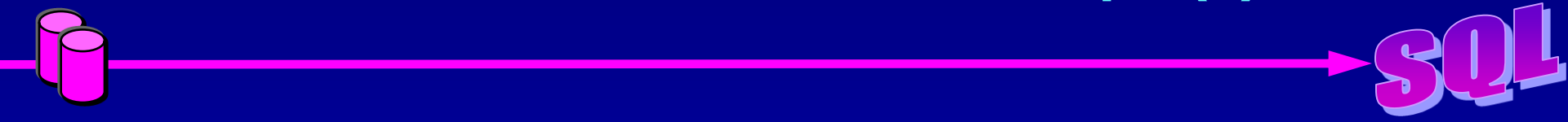

```
DECLARE
enfant_sans_parent  EXCEPTION;
PRAGMA EXCEPTION_INIT(enfant_sans_parent,-2291);
BEGIN
       INSERT INTO fils VALUES ( ……. );

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


# Procédures Stockées

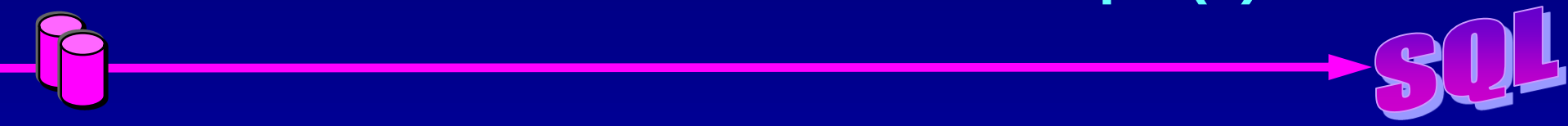
# Fonctions

# 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, …)

# 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**

# 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);
```

# 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

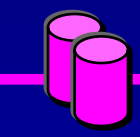
# Déclaration d'une fonction stockée

```
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 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;
```