Partie III : Concepts avancés en bases de données

- Vues et droits d'accès
- Transactions, résistance aux pannes et concurrence d'accès
- Procédures stockées et déclencheurs

VUES ET DROITS D'ACCÈS

VUES

Vue : Définition

- Une vue est une **table virtuelle** au sens où ses instances n'existent pas physiquement.
- Une vue est une **table logique** pointant sur une table physique.
- Un utilisateur peut suivre l'évolution d'une table physique via une vue.
- Chaque appel à une vue correspond à l'exécution d'une requête SELECT.

Vue : Avantages

- **Optimisation :** donner un nom à une requête longue pour l'utiliser souvent.
- **Simplification :** réduire des tables complexes à des ensembles de vues plus simples.
- **Sécurité et confidentialité :** masquer certaines données (lignes ou colonnes) aux utilisateurs.

Vue : Type

Les vues sont manipulées, interrogées et mises à jour comme n'importe quelle BD conceptuelle (tables), mais cela dépend de l'implémentation choisie :

1. **Vues virtuelles**
2. **Vues matérialisées**

Vues virtuelles

- Les relations de la vue ne sont pas stockées seule sa définition est stockée
- Le SGBD doit traduire les requêtes et mises à jour sur la vue en requêtes et mises à jour sur la BD conceptuelle

Vues matérialisées

- Stockées physiquement (ex: entrepôts de données)
- Pour des raisons de performances, on peut avoir intérêt à volontairement enregistrer le résultat de la vue, on parle alors de vue matérialisée.

  CREATE MATERIALIZED VIEW…

- Attention à la taille des vues matérialisées qui peut être conséquente (en présence de jointure)

Création d'une vue

CREATE [OR REPLACE] [TEMP] VIEW nom
[(nom_colonne [,…])]
AS requête
[WITH [CASCADED | LOCAL] CHECK OPTION]

- **temp :** supprimée en fin de session
- **check option :** les conditions de la création doivent être respectées lors des INSERT et des UPDATE
- **local :** uniquement sur cette vue
- **cascaded :** sur toutes les vues-filles

Exemple simple

```
CREATE VIEW comedies AS
SELECT *
FROM films
WHERE genre = 'Comédie‘;

SELECT *
FROM comedies
WHERE sortie = 2010;
```

Exemple avec jointure

```
CREATE VIEW tous AS
SELECT e.nom as Employe, d.nom as departement
FROM Employes e, Departement d
WHERE e.departement = d.id

SELECT * FROM tous;
```

Exemple avec héritage

```
CREATE VIEW tous AS
SELECT e.nom as Employe,
d.nom as Departement
FROM Employes e, Departement d
WHERE e.departement = d.id

CREATE VIEW les_dupont AS
SELECT *
FROM tous
WHERE Employe = 'Dupont';
```

Modification d’une vue

- la vue doit être sans jointure
- pas de renommage des colonnes
- pas d’opérateur d’agrégation

Suppression d'une vue

DROP VIEW nom [,…]

[CASCADE | RESTRICT]

- **Cascade :** supprime aussi les objets qui dépendent de la vue
- **restrict :** refuse de supprimer la vue si un objet en dépend (valeur par défaut)

Renommer une vue

RENAME ancien_nom

TO nouveau_nom;

VUES ET DROITS D'ACCÈS

droits d'accès

Pourquoi des droits ?

- **De nombreuses menaces**
  - Omniprésence des bases de données
  - Informations de valeur croisées ou non
  - Utilisées par des prestataires externes
- **Des négligences**
  - protection des bases de données
  - les SGBD sont les 1er cibles d'attaques
  - 49% des attaques sont internes

Droits et SGBD

Dans les SGBD, il existe un système d'autorisation et de protection de la BD contre des accès non autorisés.

- Le système de gestion de droits à deux principes :
  - Accorder des droits
  - Révoquer des droits

Droits / Privilèges

- Niveau objets
  - objet = table, vue, fonction, procédures
  - des droits sur les objets (privilèges)
- Niveau utilisateurs (cf. systèmes) :
  - Utilisateur **connect** :
  - utiliser les tables de la BD,
  - créer des vues,
  - transmettre des droits
  - Utilisateur **resources** :
  - connect + créer des tables
  - Utilisateur **DBA** :
  - resources + créer des utilisateurs

Droits sur les objets

	Tables	Vues	Fonctions
SELECT	X	X	
UPDATE	X	X	
DELETE	X	X	
INSERT	X	X	
ALTER	X		
REFERENCES	X		
EXECUTE			X

Droits sur les tables

GRANT

{SELECT | INSERT | UPDATE | DELETE |

RULE | REFERENCES | TRIGGER |ALL

[PRIVILEGES]}

ON [TABLE] ma_table [,…]

TO {user | GROUP name | PUBLIC} [,…]

[WITH GRANT OPTION]

Exemples simples

GRANT
SELECT
ON client **TO** PUBLIC

GRANT
INSERT
ON TABLE client **TO** Alice

GRANT
UPDATE, DELETE
ON TABLE client **TO** Bob
WITH GRANT OPTION *# Héritage de droits*

Exemples fins

GRANT
INSERT, UPDATE (nom, adresse)
ON TABLE client **TO** Alice
 # La clé doit être insérée par un déclencheur

 # Pas de privilèges sur une colonne via SELECT
CREATE VIEW personne **AS**
SELECT nom, adresse
FROM client

GRANT SELECT **ON** personne **TO** Bob

Droit de créer des tables

GRANT {

{CREATE | TEMPORARY | TEMP } [,…]

| ALL [PRIVILEGES]}

ON DATABASE ma_base [,…]

TO {user | GROUP name | PUBLIC } [,…]

[WITH GRANT OPTION]

Droit super-utilisateur

- Utilisateur de niveau **DBA**
- Dans la plupart des systèmes, le super-utilisateur par défaut est celui qui a créé la base de donnée.
- Un super-utilisateur a deux droits :
  - Créer des utilisateurs
  - Créer des bases de données

Création d'utilisateur

CREATE USER nom [[WITH] option [...]]

Options :

SYSID uid *(choisir l'identifiant)*

| CREATEDB | NOCREATEDB

| CREATEUSER | NOCREATEUSER

| IN GROUP nomgroupe [,...]

| [ENCRYPTED | UNENCRYPTED] PASSWORD mdp

| VALID UNTIL temps_absolu

Exemples

- Un utilisateur sans mot de passe

```
CREATE USER Alice;
```

- Un utilisateur avec un mot de passe

```
CREATE USER Paul WITH PASSWORD 'jw8s0F4';
```

- Un utilisateur avec un mot de passe valide jusqu'à la fin 2018 *(après 1 seconde dans 2019, il est invalidé)*

```
CREATE USER Claire WITH PASSWORD 'jw8s0F4‘
VALID UNTIL '2018-01-01';
```

Exemples avancés

- Un utilisateur pouvant créer des bases de données

```
CREATE USER manuel WITH PASSWORD
'jw8s0F4' CREATEDB;
```

- Un utilisateur pouvant créer des utilisateurs

```
CREATE USER manuel WITH PASSWORD
'jw8s0F4' CREATEUSER;
```

Utilisation de rôles

- **À la base :** il faut créer des groupes puis les associer aux utilisateurs.
- **Maintenant :** il faut créer des rôles puis les faire hériter aux utilisateurs *(un utilisateur est un rôle particulier)*.

Exemples de rôles

- **GROUPES**

CREATE GROUP vendeur **WITH** Alice, Bob;

GRANT SELECT ON client **TO GROUP** vendeur;

CREATE Claire **IN GROUP** vendeur;

- **ROLES**

CREATE ROLE vendeur

GRANT SELECT ON client **TO GROUP** vendeur;

CREATE ROLE Alice **WITH INHERIT IN ROLE** vendeur;

Révocation

Suppression de droits

- Mis en place par un utilisateur
- Le **pouvoir de révocation est limité par les autorisations** de l'utilisateur en question.

Exemples :

- Le super-utilisateur peut faire toute les révocations qu'il souhaite.
- Paul qui a offert le droit d'INSERT sur une table X à Alice ne peut que lui retirer ce droit (et pas un autre).

Supprimer des droits

REVOKE [GRANT OPTION FOR]

{{SELECT | INSERT | UPDATE | DELETE | RULE |

REFERENCES | TRIGGER} | ALL [PRIVILEGES]}

ON [TABLE] nom_table [, ...]

FROM {user | GROUP name | PUBLIC} [,…]

[CASCADE | RESTRICT]

Révoquer la création de table

REVOKE [GRANT OPTION FOR]

{{CREATE | TEMPORARY | TEMP } [,…]

| ALL [PRIVILEGES]}

ON DATABASE ma_base [,…]

FROM {user | GROUP name | PUBLIC} [,…]]

[CASCADE | RESTRICT]

Supprimer le droit d'administration

REVOKE [ADMIN OPTION FOR]
rôle [,…] **FROM** utilisateur [,…]
[CASCADE | RESTRICT]

Exemples

- Retire au groupe PUBLIC le droit d'insertion dans la table FILMS :

  REVOKE INSERT ON films FROM PUBLIC;

- Retire tous les droits de l'utilisateur Bob sur la vue GENRES :

  REVOKE ALL PRIVILEGES ON genres FROM Bob;

  ! Révoque juste les droits qui ont été donnés !

- Retire le rôle ADMINS à Alice :

  REVOKE Admins FROM Alice;

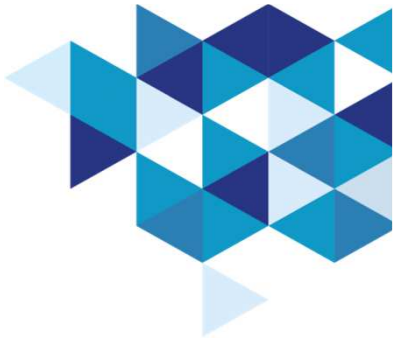
Conclusion

L'administration d'une base de données peut se faire par l'utilisation conjointe de **vues** et de **droits**.

Exemple

interdire l'accès aux tables mais autoriser l'accès des vues dérivées de ces tables.

- Les vues présentent les données indépendamment des tables.
- Les droits organisent les utilisateurs en fonction de groupes et de rôles.



TRANSACTIONS, RÉSISTANCE AUX PANNES ET CONCURRENCE D'ACCÈS

Transactions

Notion de transaction

Une transaction est une suite d'opérations interrogeant la BD, pour laquelle l'ensemble des opérations doit être, soit validé, soit annulé.

Toute transaction est réalisé ou rien ne l'est :

- **Validation :** toute la transaction est prise en compte,
- **Annulation :** la transaction n'a aucun effet.

Validation d'une transaction

Explicites : **COMMIT;**

Implicites (Oracle) :

- Commande de déconnexion en mode interactif tout ordre de mise à jour du schéma (create, drop, alter…)
- Commande « grant »
- Toute mise à jour des données en mode de confirmation automatique (autocommit on)

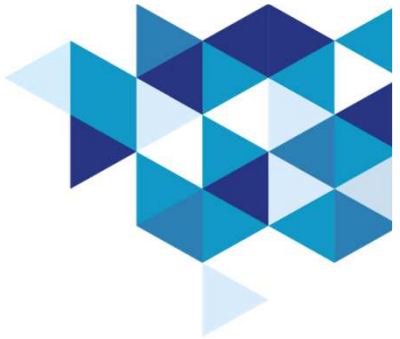
Effet : confirme toutes les mises à jour depuis le début de la transactions (i.e. depuis la dernière confirmation ou annulation)

Annulation d'une transaction

Explicites : ROLLBACK

Implicites : déconnexion anormale (autre que « exit »)

Effet : annule toutes les mises à jour depuis le début de la transactions (i.e. depuis la dernière confirmation ou annulation)



TRANSACTIONS, RÉSISTANCE AUX PANNES ET CONCURRENCE D'ACCÈS

Résistance aux pannes

Résistance aux pannes

Le SGBD doit permettre de :

- Minimiser le travail perdu
- Assurer un retour à des données cohérentes

A quoi sont dues les pannes ?

- Erreur humaine
- Erreur de programmation
- Défaillance matérielle

Types de pannes

- **La panne sur une action :** lorsqu'une commande SGBD est mal exécutée
- **La panne de transaction :** erreur de programmation, accès concurrents, dead-lock…
- **La panne système :** nécessite le redémarrage du système (erreur logicielle, coupure de courant…)
- **La panne mémoire secondaire :** suite à une défaillance matérielle ou logicielle impliquant de mauvaises écritures

Reprise sur panne

Le SGBD doit fournir un protocole aux applications permettant de :

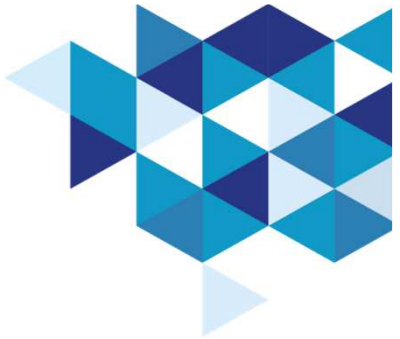
- Faire une transaction
- Défaire une transaction
- Refaire une transaction

Trois moyens à conjuguer :

- La journalisation
- Les sauvegardes
- La réplication

En cas de panne :

- On reprend l'état sauvegardé de la base
- On ré-exécute toutes les actions du journal
- La réplication permet de limiter les interruptions de service



TRANSACTIONS, RÉSISTANCE AUX PANNES ET CONCURRENCE D'ACCÈS

Concurrence d'accès

Gestion de la concurrence d'accès

- Accès concurrent
  - Il y a un accès concurrent lorsque plusieurs utilisateurs (transactions) accèdent en même temps à la même donnée dans une base de données.
- Gestion des accès concurrents (contrôle de concurrence)
  - S'assurer que l'exécution simultanée des transactions produit le même résultat que leur exécution séquentielle (l'une puis l'autre)

Accès concurrents

- Problèmes posés par les accès concurrents
  - Perte de mise à jour
  - Lecture impropre
  - Lecture non reproductible
  - Objets fantômes

Perte de mise à jour

T1 et T2 modifient simultanément A

T_1	T_2	BD
		$A = 10$
`read A`		
	`read A`	
`A = A + 10`		
`write A`		$A = 20$
	`A = A + 50`	
	`write A`	$A = 60$

Les modifications effectuées par T1 sont perdues

Lecture impropre

T_1	T_2	*BD*
		$A + B = 200$
		$A = 120$ $B = 80$
`read A`		
`A = A - 50`		
`write A`		$A = 70$
	`read A`	
	`read B`	
	`display A + B` (150 est affiché)	
`read B`		
`B = B + 50`		
`write B`		$B = 130$

T2 lit une valeur de A non validée, affiche une valeur incohérence

Lecture impropre (suite)

T_1	T_2	BD
		$A = 50$
	`A = 70`	
	`write A`	$A = 70$
`read A` (70 est lu)		
	`rollback` (La valeur initiale de A est restaurée)	$A = 50$

T1 lit une valeur de A non confirmée

Lecture non reproductible

T_1	T_2	BD
		$A = 10$
	`read A` (10 est lu)	
`A = 20`		
`write A`		$A = 20$
	`read A` (20 est lu)	

T2 lit deux valeurs de A différentes

Objet fantôme

T_1	T_2	BD
		$E = \{1, 2, 3\}$
`display card(E)` 3 est affiché		
	`insert 4 into E`	$E = \{1, 2, 3, 4\}$
`display card(E)` 4 est affiché		

Contrôle des accès

- Verrouillage :
  - Le verrouillage est la technique la plus classique pour résoudre les problèmes dus à la concurrence:
    - Avant de lire ou écrire une donnée, une transaction peut demander un verrou sur cette donnée pour interdire aux autres transactions d'y accéder.
    - Si ce verrou ne peut être obtenu, parce qu'une autre transaction en possède un, la transaction demandeuse est mise en attente.
  - Afin de limiter les temps d'attente, on peut jouer sur :
    - La granularité du verrouillage : pour restreindre la taille de la donnée verrouillée (n-uplet, une table)
    - Le mode de verrouillage: pour restreindre les opérations interdites sur la donnée verrouillé.

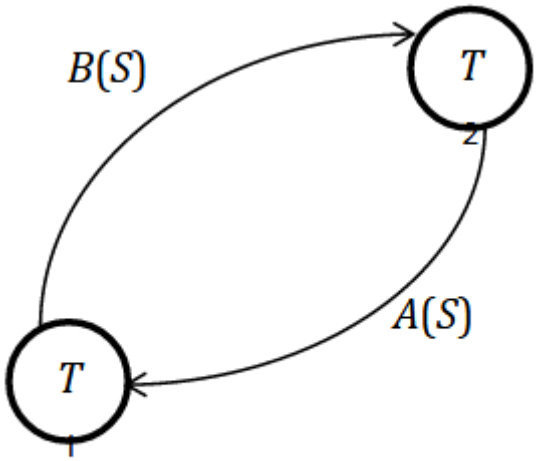
Exemple verrouillage

T1	T2	Résultat A=10
Read A avec verrou		
A=A+10	Read A avec verrou	
	attente	A=20
Write A Commit;		A=20
	Read A avec verrou	20
	A=A+50	50
	Write A commit	A=70

Problème de verrouillage : interblocage

- L'impasse générée par deux transactions (ou plus) qui attendent, l'une, que des verrous se libèrent, alors qu'ils sont détenue par l'autre :

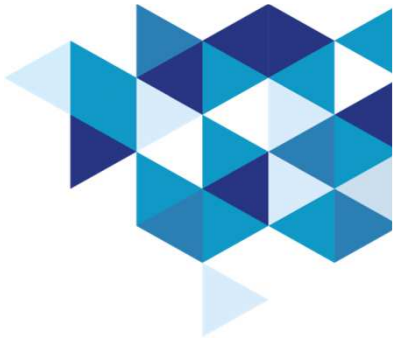
T_1	T_2
`lock X A`	
	`lock X B`
`lock S B`	
attente	`lock S A`
attente	*attente*



graphe d'attente

Résolution de l'interblocage

- Deux approches :
  - Prévention :
    - Toutes les ressources nécessaires à la transaction sont verrouillées au départ
    - Problème : cas des transactions qui ne démarrent jamais !
    - Méthode peu utilisée aujourd'hui
  - Détection :
    - On inspecte à intervalles réguliers le graphe d'attente pour détecter si un interblocage s'est produit. Dans ce cas, on défait l'une des transactions bloquées et on la relance un peu plus tard.
    - On annule une transaction dont le temps d'attente dépasse un certain seuil, et on la relance un peu plus tard.



PROCÉDURES STOCKÉES ET DÉCLENCHEURS

Procédures stockées

Limites du SQL

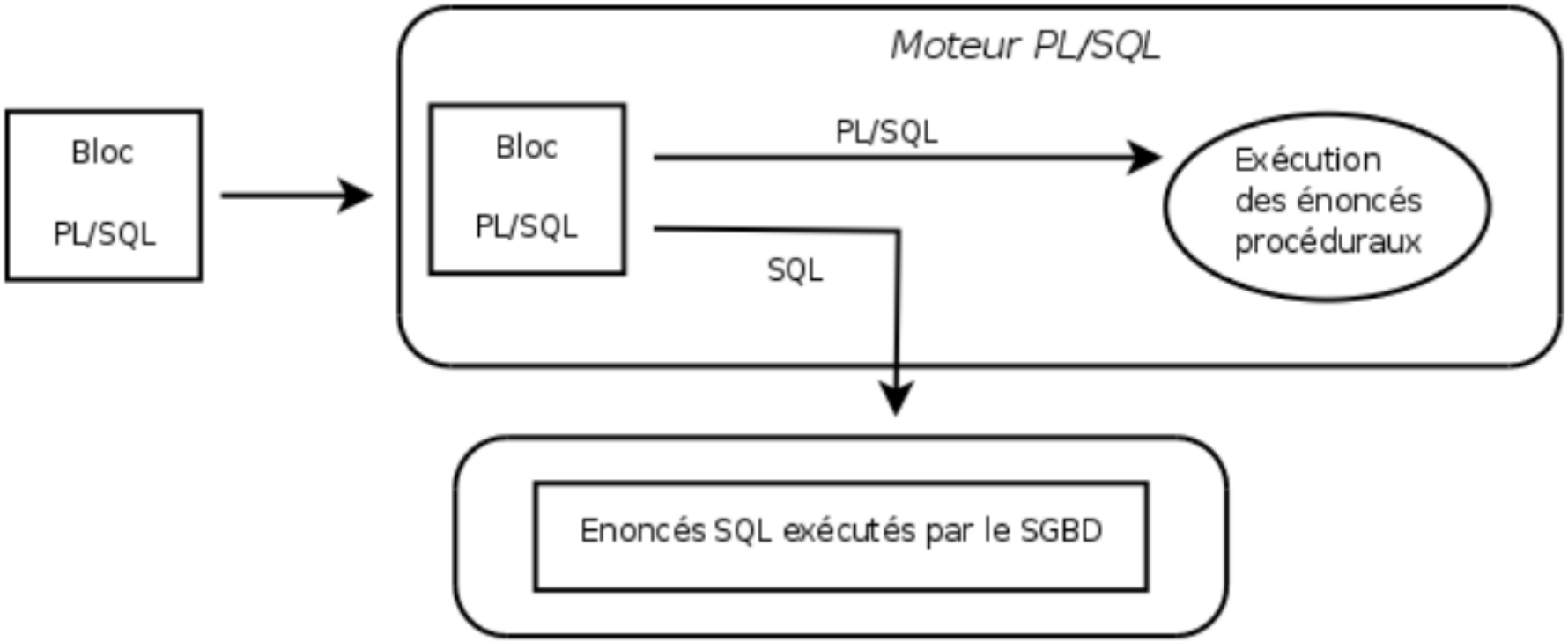
- **Langage non procédural**
- Il n’a pas de :
  - Variables
  - Itérations
  - Branchements conditionnels
- Impossible de lier plusieurs requêtes SQL :

regrouper un bloc de commandes et le soumettre au noyau

PL/SQL

- Langage fournissant une interface procédurale au SGBD Oracle
- Intégration du langage SQL en lui apportant une dimension procédurale
- Réalisation de traitements algorithmiques (ce que ne permet pas SQL)
- Mise à disposition de la plupart des mécanismes classiques de programmation des langages hôtes tels que C, C++, JAVA …

Environnement PL/SQL



Avantage de PL/SQL

- Structures itératives : WHILE … LOOP, FOR … LOOP, LOOP …
- Structures conditionnelles : IF … THEN … ELSE | ELSEIF … ENDIF, CASE …
- Déclaration des curseurs et des tableaux
- Déclaration de variables
- Affectation de valeurs aux variables
- Branchements : GOTO, EXIT
- Exceptions : EXCEPTION

Utilisation de PL/SQL

Le PL/SQL peut être utilisé sous trois formes :

- Un bloc de code, exécuté comme une unique commande SQL, via un interpréteur standard (SQLplus ou iSQL*Plus)
- Un fichier de commande PL/SQL
- Un programme stocké (procédure, fonction, trigger)

Langage PL/SQL : Blocs

- Un programme est structuré en bloc d'instructions qui peuvent être de 3 types :
  - procédures anonymes
  - procédures nommées
  - fonctions nommées

- Un bloc peut contenir d'autres blocs

Anatomie d’un bloc

DECLARE

Déclarations de constantes et de variables

BEGIN

Commandes exécutables

END;

Sous-bloc DECLARE (1)

- Une variable, c'est :
  - 30 caractères au plus
  - commence par une lettre
  - peut contenir _, $ et #
  - insensible à la casse
  - portée habituelle des langages à blocs
  - doit être déclarée avant utilisation

Sous-bloc DECLARE (2)

- Déclaration d'un type d'un attribut
  - < variable > table.attribut%type
- Déclaration d'un type n-uplet
  - < variable > table%rowtype
  - < variable > record
- Déclaration d'une date
  - < variable > date
- Déclaration d'un entier
  - < variable > integer
- Déclaration d'une constante
  - < variable > CONSTANT := constante

Sous-bloc DECLARE (3)

```
employe emp%ROWTYPE;
nom emp.nom.%TYPE;

select * INTO employe
from emp
where matricule = 900;

nom := employe.nome;
employe.dept := 20;
...
insert into emp values employe;
```

Sous-bloc DECLARE (4)

Utilisation du type record:

TYPE nomRecord IS RECORD (

champ1 type1,

champ2 type2

);

- Dans tous les cas :
  - Déclarations multiples **interdites**
  - Si une variable porte le même nom qu'une colonne d'une table, c'est la **colonne qui l'emporte**

Sous-bloc BEGIN ... END;

- Partie BEGIN ... END :
  - Affectation
  - Branchement conditionnel
  - Itération

Branchement conditionnel

IF condition THEN

ELSE IF condition THEN

ELSE IF condition THEN

ELSE instruction

END IF;

END IF;

END IF;

Choix (type simple)

```
CASE expression
WHEN expr1 THEN instruction1 ;
WHEN expr2 THEN instruction2 ;
...
ELSE instructions ;
END CASE;
```

Itérations (1)

```
LOOP
instructions ;
EXIT [WHEN condition] ;
instructions ;
END LOOP;

WHILE condition LOOP
instructions ;
END LOOP;
```

Itérations (2)

FOR element IN [REVERSE] domaine

LOOP

instructions ;

END LOOP;

Domaines :

- intervalle comme 1..100
- éléments d'une table (SELECT)

Procédures et fonctions

- Offrir aux programmeurs la possibilité de créer des blocs de traitements.
- Introduire quelques bases de la programmation dans les moteurs SQL
- Les procédures et les fonctions sont stockées dans la base de données comme les autres objets (tables, requêtes, ...)

Création d'une fonction/procédure

```
CREATE FUNCTION gen_cle_client () RETURNS OPAQUE AS
'
DECLARE
nocli integer;
BEGIN
SELECT nocli INTO max(no_client) FROM client;
IF nocli ISNULL THEN
nocli:=0;
END IF;
NEW.no_client:=nocli+1;
RETURN NEW;
END;
'
LANGUAGE 'plpgsql';
```

CREATION avec paramètres

```
CREATE FUNCTION double (integer)
RETURNS integer
AS
'BEGIN
RETURN 2*$1;
END; '
LANGUAGE 'plpgsql';
```

- Les paramètres sont utilisés via les macros

$1, $2, ..., $x pour les paramètres 1, 2, ..., xème

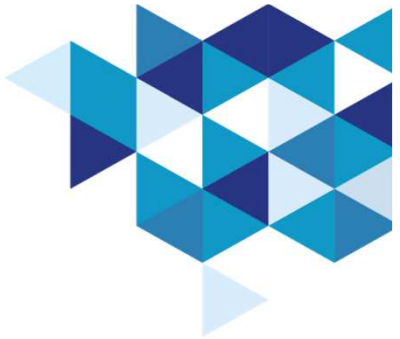
Remplacement

- REPLACE permet de changer le code d'une fonction existante
- En général lors de la création on peut aussi mettre la primitive REPLACE.

```
CREATE OR REPLACE FUNCTION double (integer)
RETURNS integer
AS
'BEGIN
RETURN 2*$1;
END; '
LANGUAGE 'plpgsql';
```

Exemple

```
CREATE OR REPLACE FUNCTION gen_cle_client () RETURNS void AS
'
DECLARE
i RECORD;
total real;
BEGIN
FOR i IN   SELECT nocli, count(qtité*PU)
INTO total
FROM commande GROUP BY nocli
LOOP
IF total > 10000 THEN
INSERT i into table_TB_CLIENT;
END IF;
END LOOP;
END;
'
LANGUAGE 'plpgsql';
```



PROCÉDURES STOCKÉES ET DÉCLENCHEURS

Déclencheurs

Qu'est-ce qu'un trigger ?

- Un programme **stocké** dans une BD
  - associé à une **table** donnée
  - associé à un **événement** se produisant sur cette table
- Le trigger est exécuté lorsque l'événement auquel il est attaché se produit sur la table.

Intérêt

- Maintenance des tables facilitées
- Mise à jour automatique cohérente
- Sécurité renforcée
- Gestion d’un historique
- Gestion événementielle transparente
- Analyse de données pour la décision
- Implémentation des MCT

Les événements déclenchants

Le programme associé au trigger se déclenche lorsque l'un des événements se produit :

- Insertion dans la table : INSERT
- Mise à jour : UPDATE
- Suppression : DELETE

Caractéristique

- Un trigger est associé à une seule table
- S'exécute à l'arrivée de l'événement
- Déclenche un bloc PL/SQL (fonction)
- Détruit avec la destruction d'une table
- Peut être désactivé.

Types de Déclencheurs

- **Triggers table (STATEMENT)**

  Sont exécutés une seule fois lorsque des modifications surviennent sur une ou plusieurs lignes de la table.

  Il n'est pas possible d'avoir accès à la valeur ancienne et la valeur nouvelle (OLD et NEW)

- **Triggers ligne (ROW)**

  Sont exécutés séparément pour chaque ligne modifiée dans la table.

  Il est possible d'avoir accès à la valeur ancienne et la valeur nouvelle grâce aux mots clés OLD et NEW

SYNTAXE Trigger Table

CREATE [OR REPLACE] TRIGGER nom_trigger
{BEFORE | AFTER} événement
ON nom_table
DECLARE
-- Déclarations variables, curseurs, records, …
BEGIN
-- Traitement
EXCEPTION
-- Gestionnaires d'exceptions
END [nom_Trigger]

SYNTAXE Trigger ligne

CREATE [OR REPLACE] TRIGGER nom_trigger
{BEFORE | AFTER} événement
ON nom_table
FOR EACH ROW [WHEN condition]
[REFERINCING {[old [AS] nom_old] | New [AS] nom_new]}]
DECLARE
-- Déclarations variables, curseurs, records, …
BEGIN
-- Traitement
EXCEPTION
-- Gestionnaires d'exceptions
END [nom_Trigger]

Exemple

```
CREATE FUNCTION gen_cle_client () RETURNS OPAQUE AS
'DECLARE
nocli integer;
BEGIN
SELECT nocli INTO max(no_client) FROM client;
IF nocli ISNULL THEN
nocli:=0;
END IF;
NEW.no_client:=nocli+1;
RETURN NEW;
END; '
LANGUAGE 'plpgsql';

CREATE TRIGGER trig_bef_ins_client
BEFORE INSERT
ON client
FOR EACH ROW
EXECUTE PROCEDURE gen_cle_client();
```