Administration des bases de données (Oracle et PostgreSQL)

Fabien De Marchi

Faculté des Sciences et Technologies
Université de Lyon

Prérequis et objectifs

- Prérequis
  - Modèle relationnel (structure, contraintes, SQL)
- Objectifs
  - Connaître les tâches d'un DBA
  - Connaître les concepts et points clés de l'architecture
  - Savoir effectuer quelques opérations de base

Pour en savoir plus...

- PostrgreSQL Architecture et notions avancées (G. Lelarge)
- Mastering PostGreSQL 10 (H-J Schonig)
- docs.postgresql.fr
- www.oracle.com

Les métiers autour des bases de données

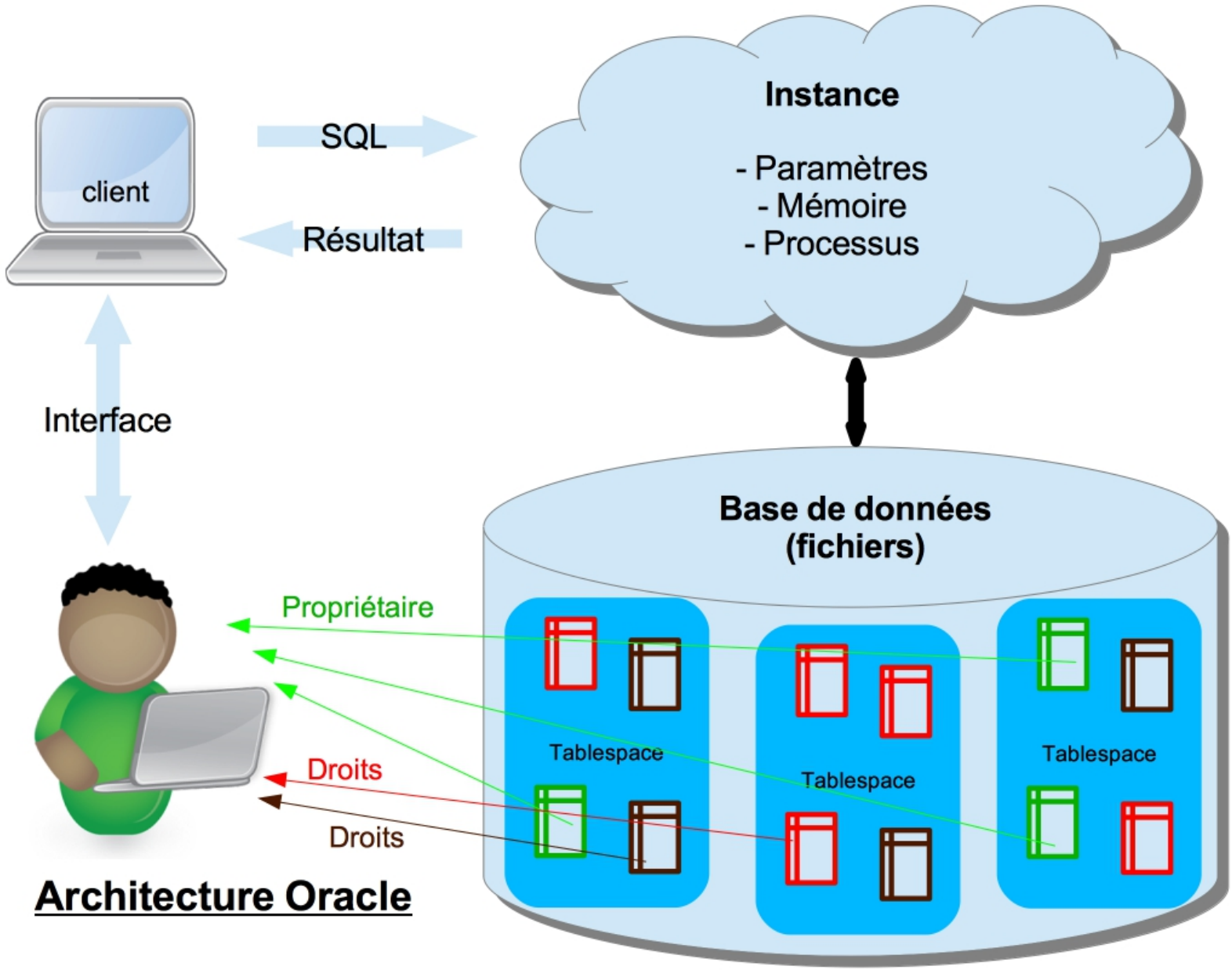
- Administrateur
- Responsable de la sécurité
- Administrateur réseaux
- Développeurs d'application
- Administrateurs d'application
- Utilisateurs : modifier les données, créer des rapports

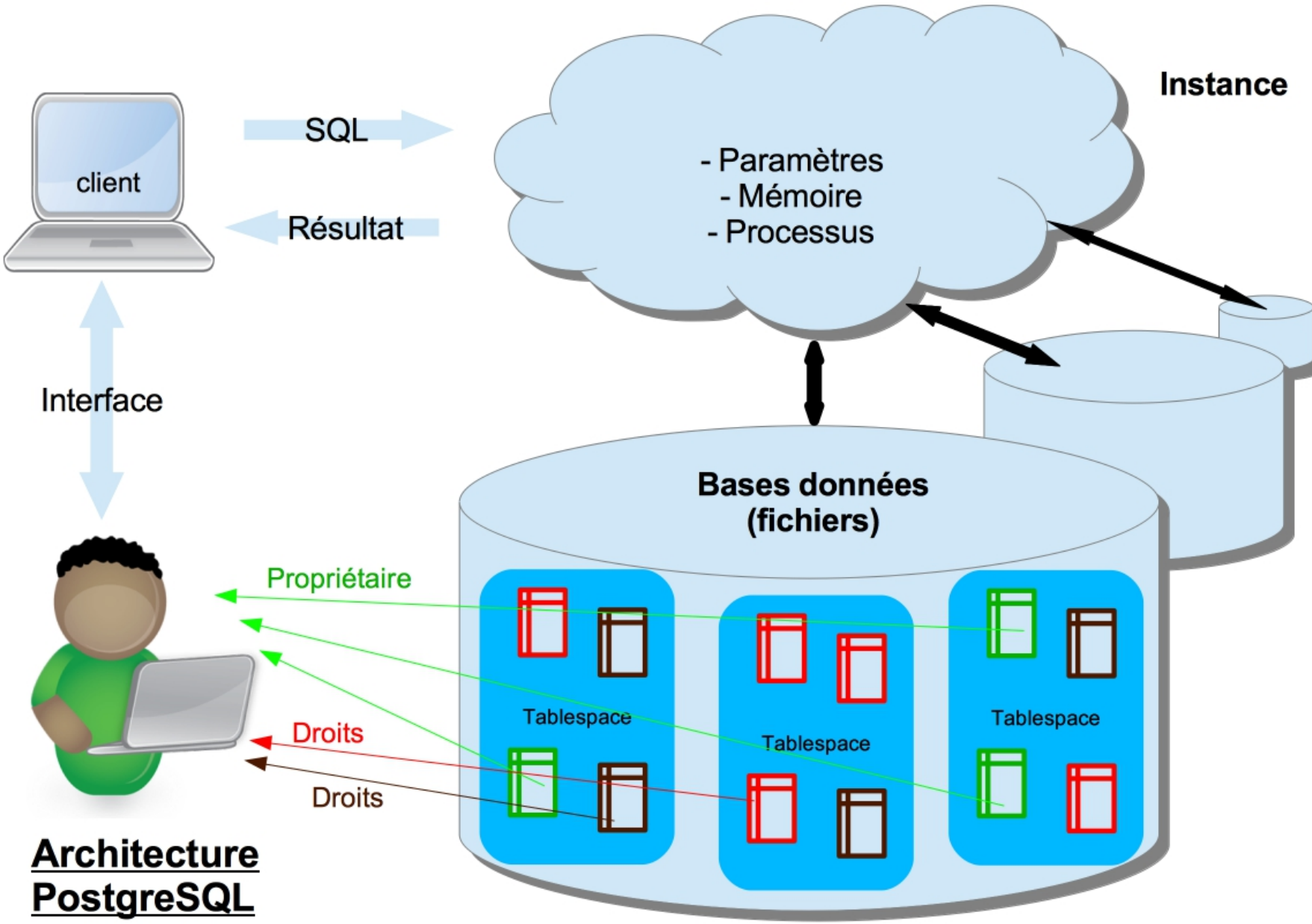
Note

Dans des environnements de petite taille, l'administrateur peut jouer quasiment tous les rôles

Rôles du DBA

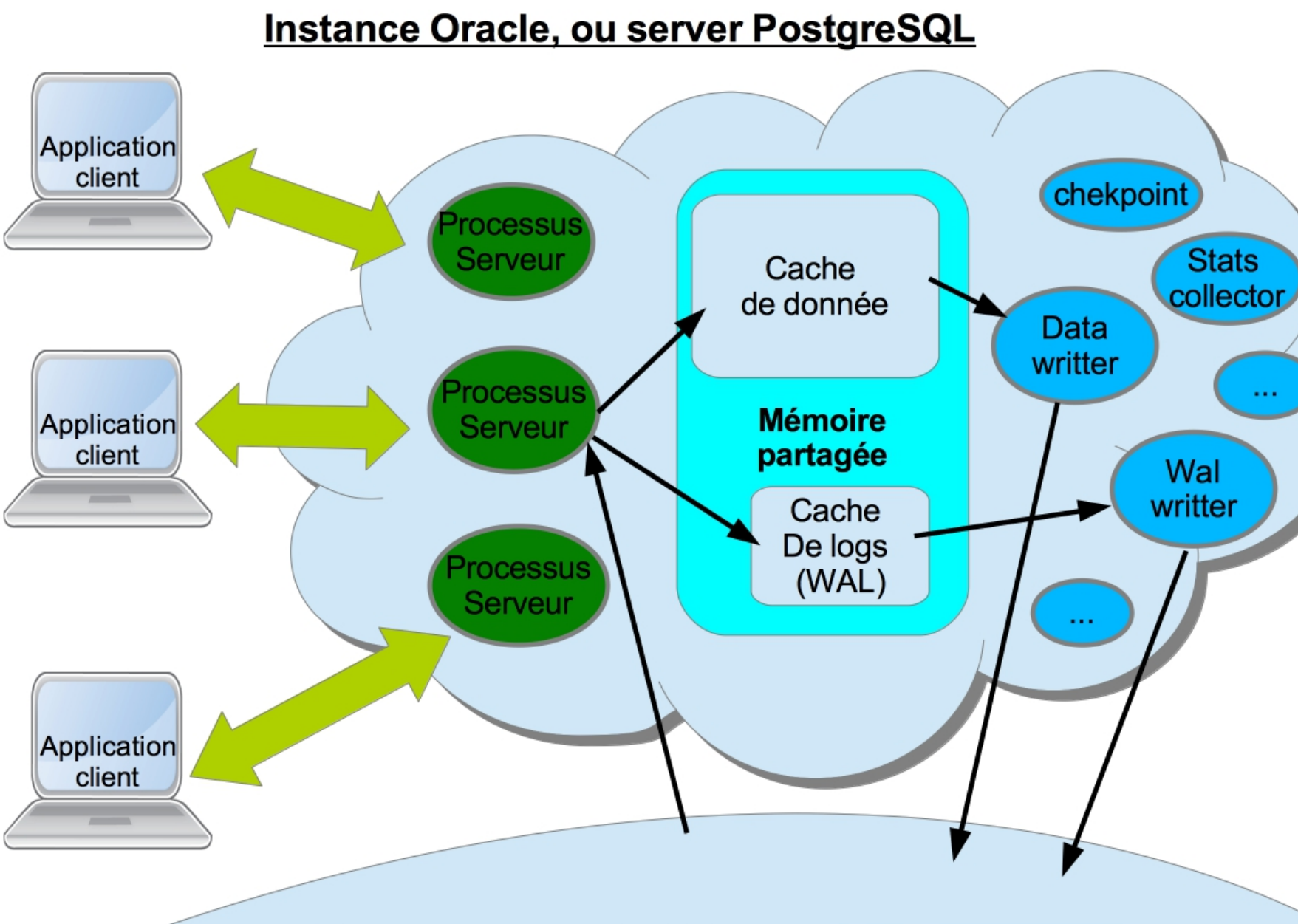
- Installer le SGBD
  - un serveur, des applications clientes,
  - En fonction de l'OS et des paramètres systèmes
  - composants réseaux, modules
- Planifier et créer des bases de données
- Gérer l'espace et implanter les schémas des données
- Assurer la sécurité, l'intégrité et la pérennité des données
- Effectuer des réglages pour optimiser les performances
- Résoudre les problèmes...


client
SQL
Résultat
Instance
- Paramètres
- Mémoire
- Processus
Interface
Base de données
(fichiers)
Propriétaire
Droits
Droits
Tablespace
Tablespace
Tablespace
Architecture Oracle


client
SQL
Résultat
Instance
- Paramètres
- Mémoire
- Processus
Interface
Bases données
(fichiers)
Propriétaire
Tablespace
Tablespace
Tablespace
Droits
Droits
Architecture
PostgreSQL

Architecture PostgreSQL (cont.)

- Les BD sont des espaces séparés
- Un utilisateur ne peut se connecter que sur une BD à la fois
- Les tablespaces et les rôle sont transversaux aux BD


Instance Oracle, ou server PostgreSQL
Application client
Application client
Application client
Processus Serveur
Processus Serveur
Processus Serveur
Cache de donnée
Mémoire partagée
Cache De logs (WAL)
chekpoint
Stats collector
Data writter
...
Wal writter
...

Enchaînement type (1)

- Instance démarrée sur le serveur
- Une application cliente établit une connexion et ouvre une session
- Le serveur détecte la requête de connexion et crée un processus serveur dédié
- L'utilisateur lance une requête SQL et un commit
- Le processus serveur analyse et exécute la requête (lecture des fichiers de données, écriture en cache)
  - Exploitation du cache pour éviter des E/S
  - Peuvent accéder aux fichiers de données en lecture
  - Transaction inscrite en cache de log
  - Modifications exécutées en cache de données

Enchaînement type (2)

- Les processus *walwriter* et *writer* déchargeront dans les fichiers
- Le résultat, ou une confirmation, est envoyé au processus client

Les transactions

- Groupement atomique de requêtes
- Etat visible à l'issu de la transaction
- Deux états de terminaison possible
  - COMMIT : validation
  - ROLLBACK : annulation (volontaire ou en cas d'erreur)
- Explicite (BEGIN ... END) ou implicite.
- Implicite (auto-commit)

Les transactions

- Propriétés ACID
  - Atomicité : pas d'état intermédiaire visible
  - Cohérence : vérifie les contraintes **après** la fin
  - Isolation : des transactions entre elles (différents niveaux)
  - Durabilité : validation = enregistrement sur le disque.

Les tablespaces

- Permettent de gérer :
  - la localisation des données
  - la séparation de données (performances, organisation logique)
- Les objets sont créés :
  - Dans le tablespace spécifié à la création de l'objet
  - Sinon : dans le tablespace par défaut de l'utilisateur (Oracle)
  - Sinon : dans le tablespace par défaut de la BD
  - Sinon : dans le tablespace par défaut de l'installation
    - Tablespace "system" sous Oracle (fichier system.dbf)
    - Tablespace "*pg_default*" sous Postgre (répertoire **base**)

Les schémas

- Permettent une organisation logique des données
  - Selon des besoins applicatifs différents
  - Facilitent le contrôle d'accès
  - Sous Oracle, pas de distinction entre les SCHEMAS et les Utilisateurs
  - sous Postgre, les utilisateurs n'ont pas forcément de schéma.

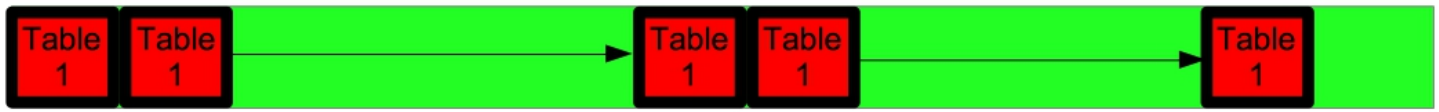
Fichier de contrôle

- Le fichier de contrôle : chef d'orchestre
- Point de cohérence pour "ouvrir la base" au lancement de l'instance
- Numéro de version, points de synchronisation, taille des blocks, codage des entiers etc...
- La perte de ce fichier est à éviter à tout prix !
  - Oracle permet le "multiplexage" : maintenance à chaud de plusieurs fichiers jumeaux
  - Emplacement PostgreSQL : dans le tablespace *pg_global* (base/global/*pg_control*)

Organisation des données sous ORACLE

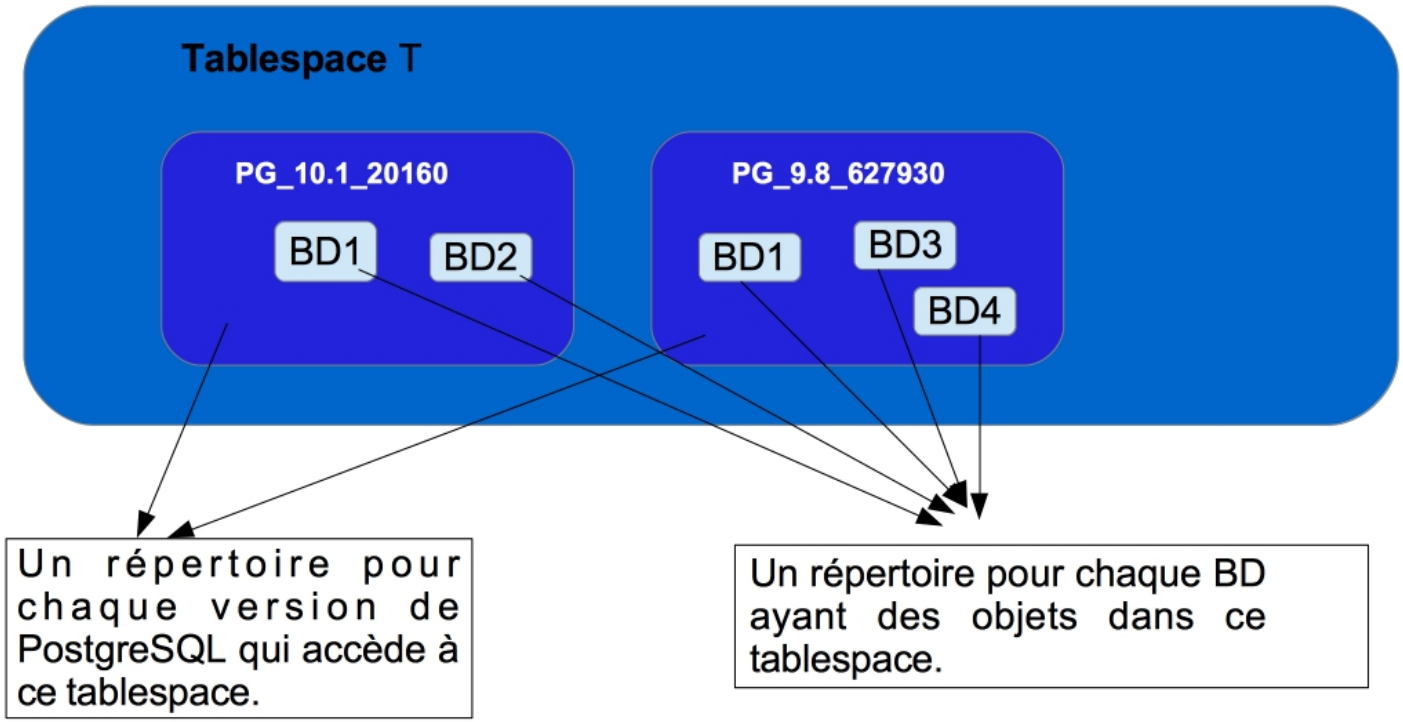
Tablespace = un ou plusieurs fichiers de données : organisation logique des donnée, point de vue utilisateur. Peuvent être stockés n'importe où, y compris sur un disque distant.

Tout objet (relation, trigger, contrainte, index...) **appartient à un seul tablespace,** et correspond à un **segment** (liste chaînée de blocs).



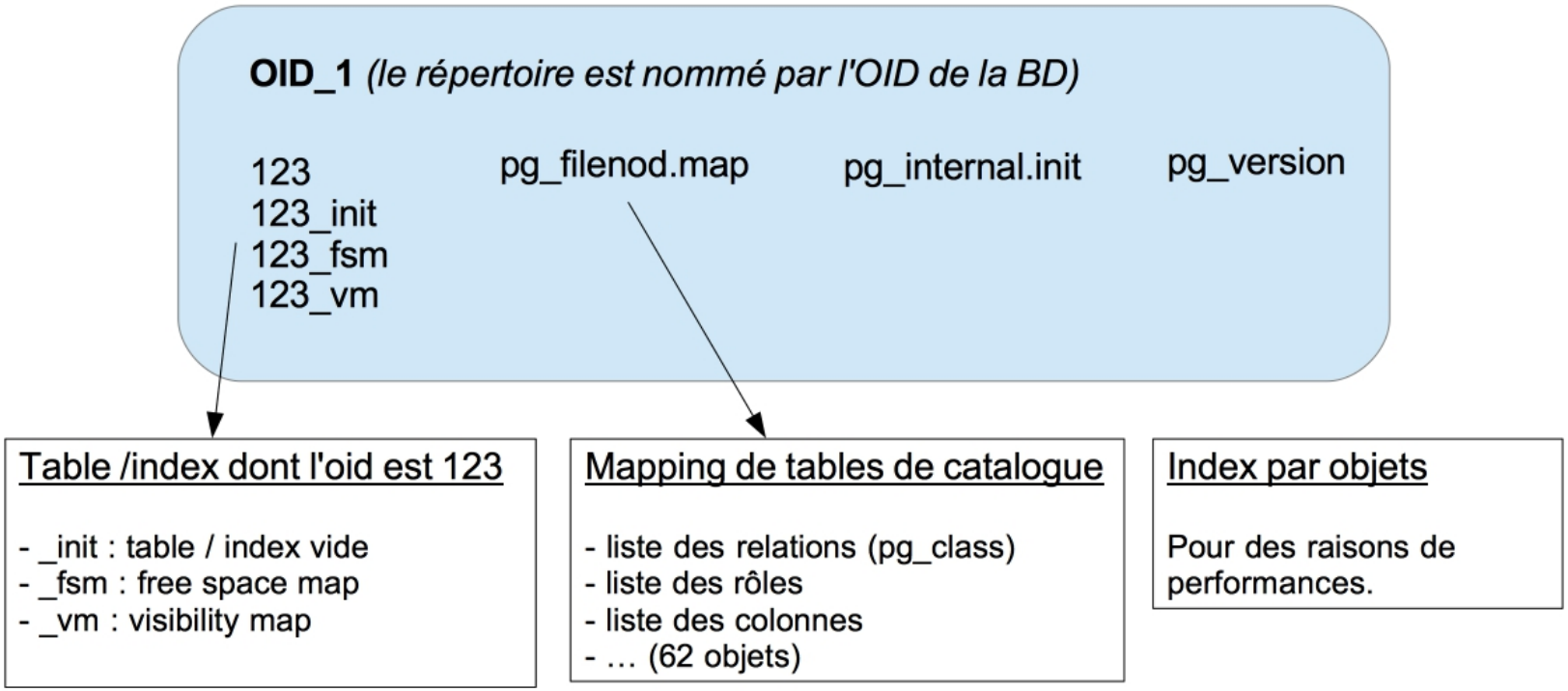
Bloc de données : élément d'E/S minimal.
8Ko par défaut : paramétrable à la création de la base **uniquement**.

Les tablespaces sous PostgreSQL



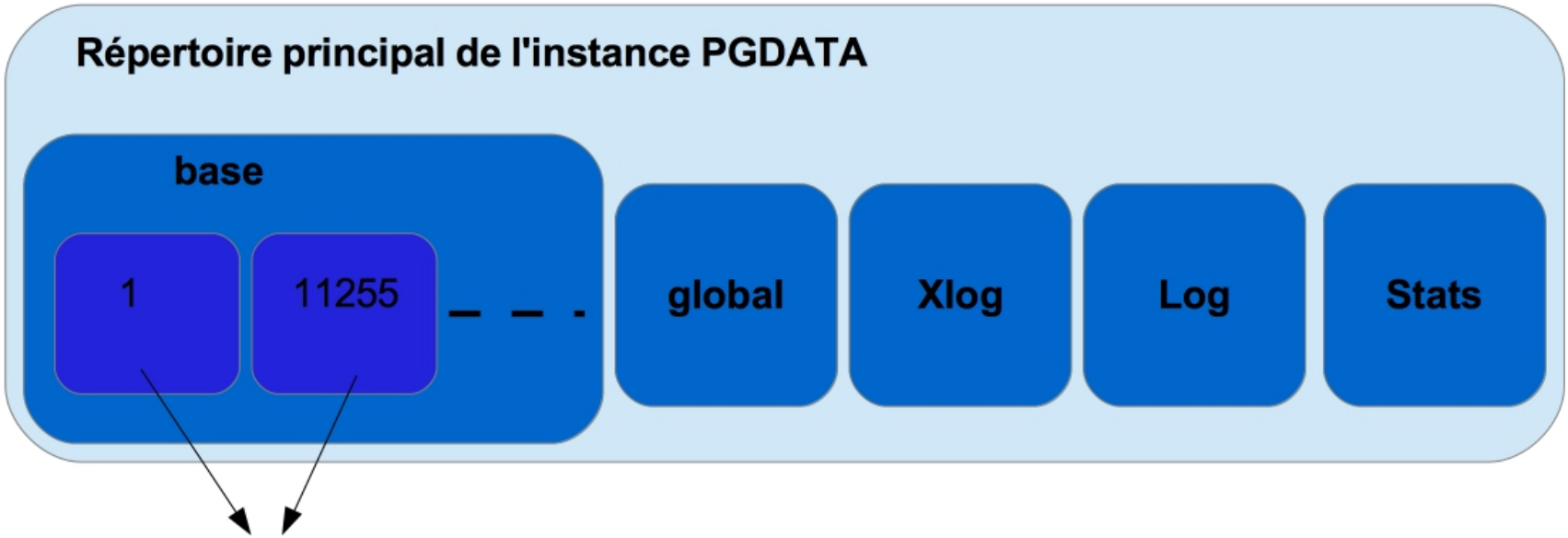
Conseil : les tablespaces doivent être créés en dehors de PGDATA.

Contenu des répertoires bases de données



SELECT oid, datname FROM pg_database

Organisation des données sous PostgreSQL



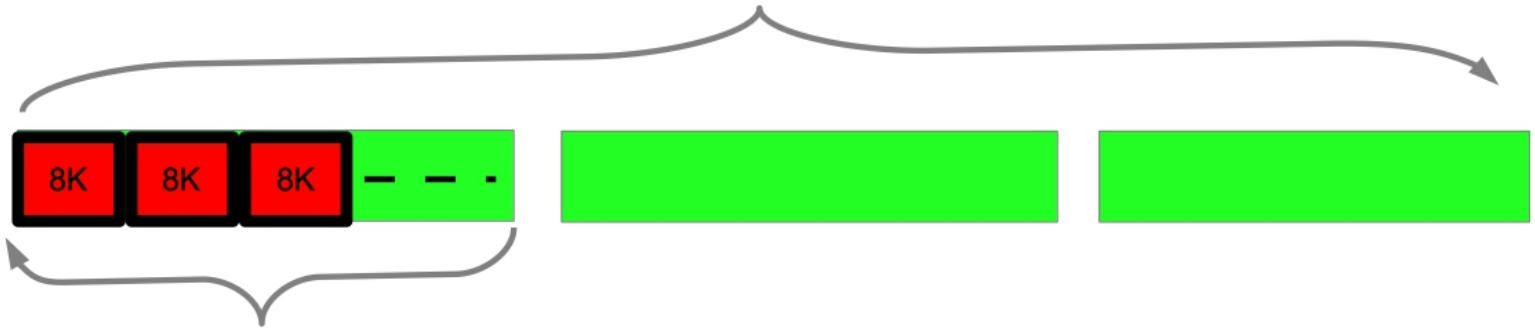
Chaque **base de données** est un ensemble de fichiers à l'intérieur du répertoire « **base** »; dont le nom est son OID dans le système. « **base** » est le répertoire **tablespace** par défau (pg_default).

Sous PostgreSQL, les tablespaces dont des répertoires. Tout objet peut être stocké dans un tablespace au choix (pour des raisons de place ou de performances).

Le répertoire **« global »** (tablespace « pg_global ») stocke les données communes à toute l'instance : la liste et description des bases de donnée, des rôles, des tablespaces.

Les fichiers de données

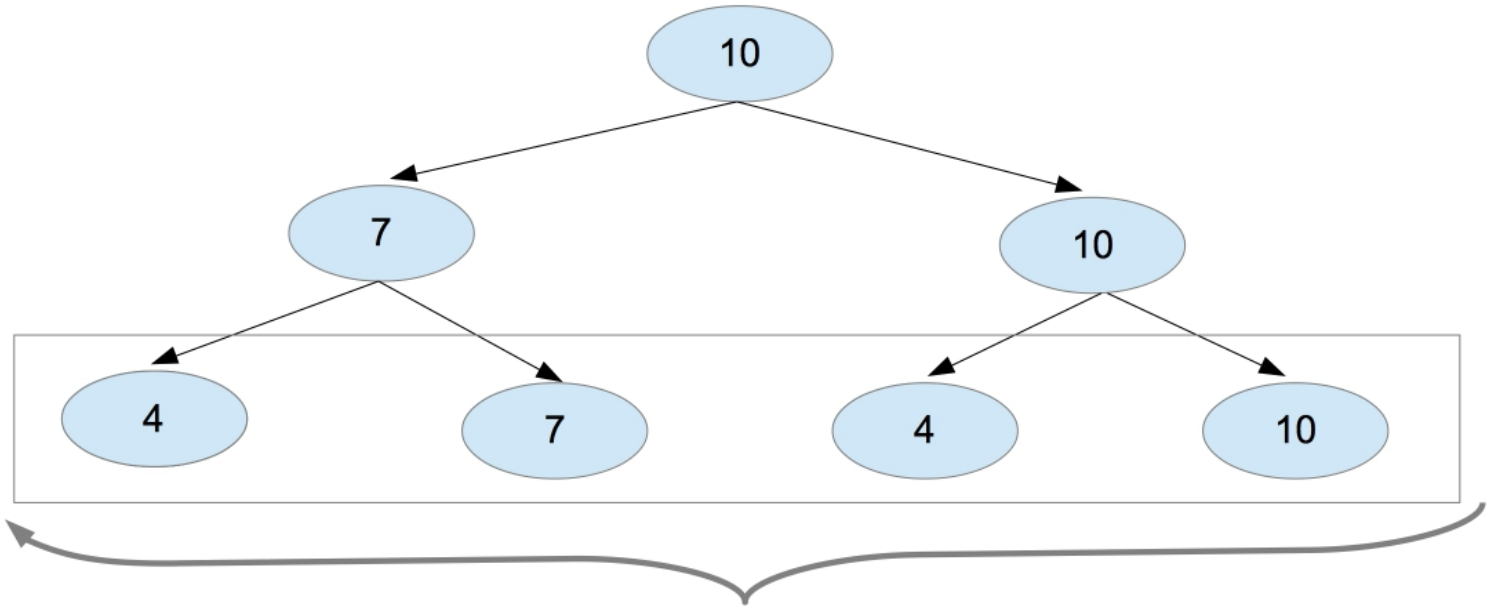
Les relations sont stockées par segments (fichiers) de 1 G.



Allocation par blocs de 8ko.

Taille des blocs et taille des segments configurables à l'installation de PostgreSQL
==> choix définitif !

FSM : Free Space Map

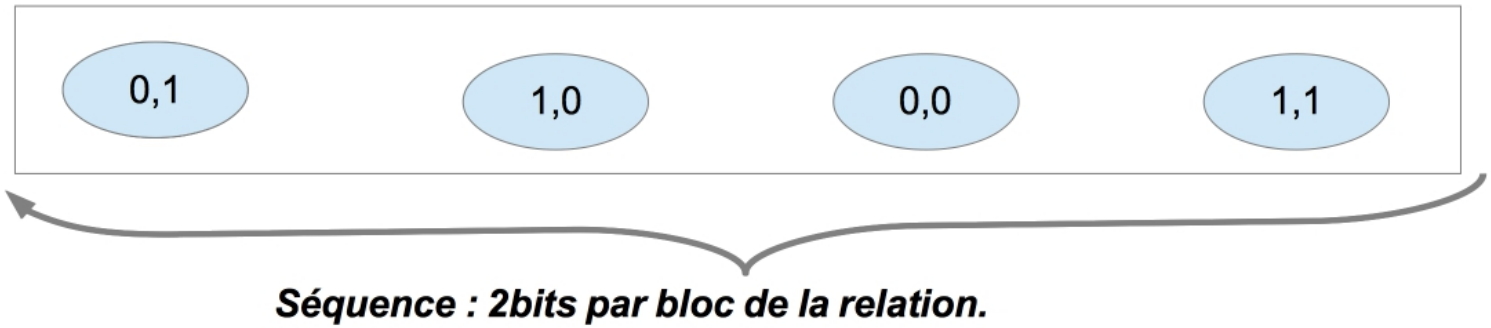


Feuilles = séquence des blocs avec leur place libre.

Arbre binaire ; permet de trouver rapidement dans quels blocs insérer les nouveaux tuples. Unité : 1/256e d'un bloc (32o pour des blocks de 8ko).

VM : Visibility Map

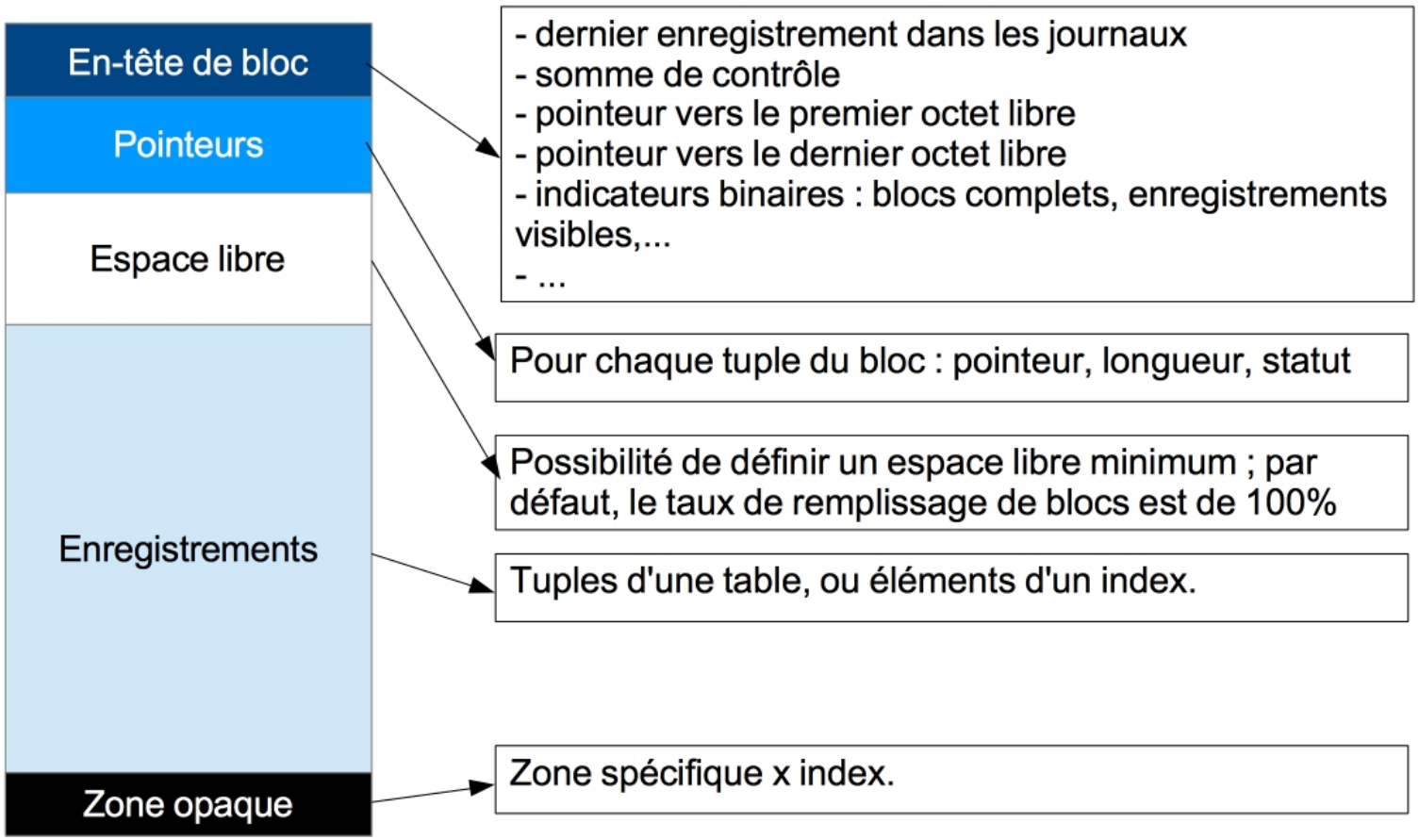
Un enregistrement modifié ou supprimé est noté « invisible »



Séquence : 2bits par bloc de la relation.

Premier bit : Tous les enregistrement du bloc sont-ils visibles ?
Deuxième bit : Est-ce que tous les enregistrements sont gelés ?

Bloc de données



Gestion de l'espace libre sous PostgreSQL

- Les lignes supprimées ou obsolètes ne sont pas détruites
- commande SQL VACUUM permet de libérer l'espace. Ne verrouille pas la table.
- Option "FULL" : déplace des enregistrements pour optimiser l'espace. Verrouille la table.
- S'applique à toutes les tables, ou sur une table ciblée.
- Suivie souvent de la commande ANALYSE (statistiques)
- Tâche de fond automatisée : *autovacuumworker*

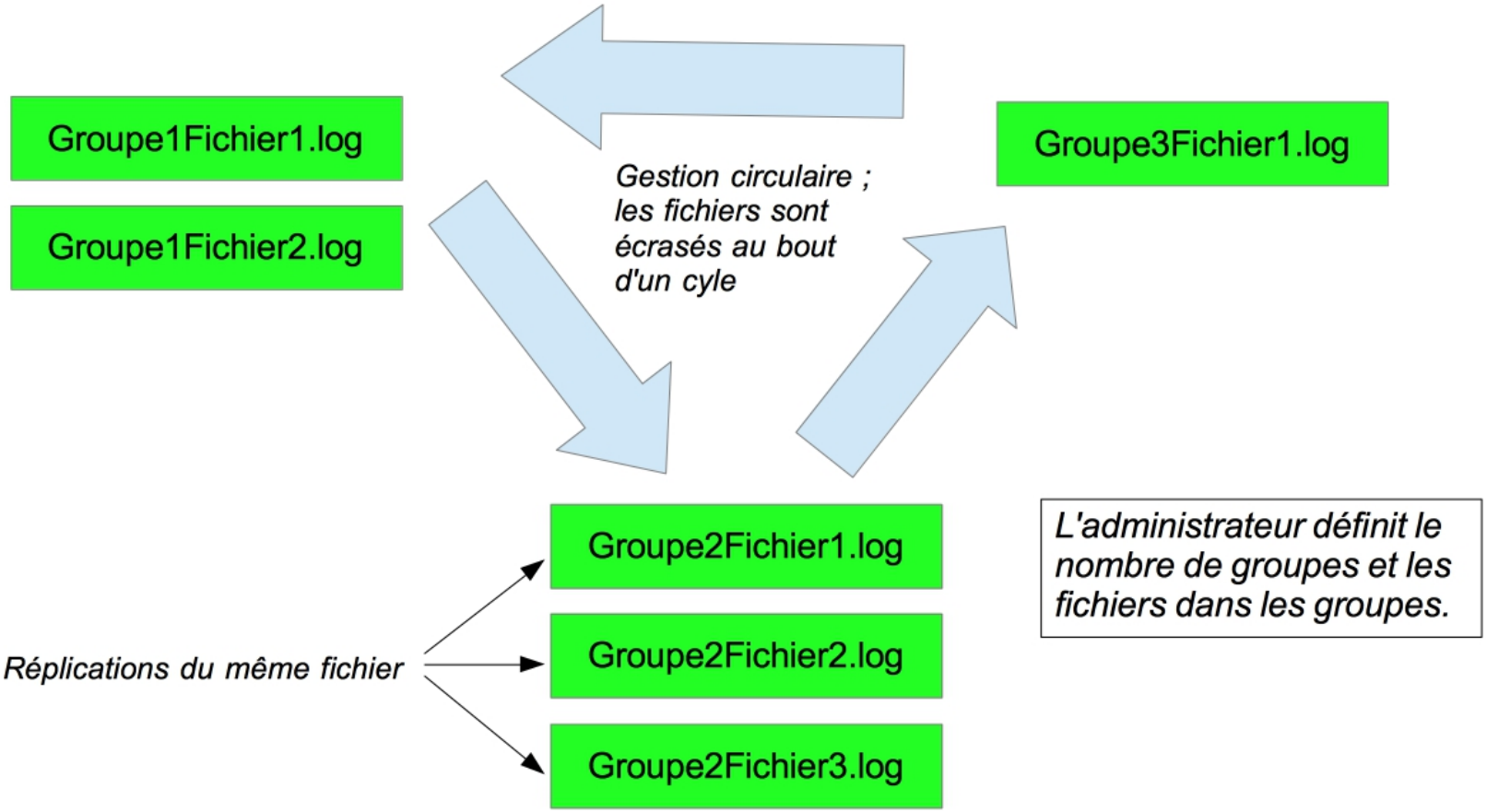
Les fichiers de log

- Principe Write-Ahead Logging : les **transactions** sont journalisées AVANT d'être appliquées
- Oracle
  - Fichiers logfiles (.log)
  - orchestré par le processus LogWriter
- PostgreSQL
  - Fichiers WAL dans le répertoire **xlog**
  - Orchestré par le processus WalWriter
- "Rejoués" pour restauration ou reprise après panne
- Durée de vie limitée, paramétrable
- Archivage = stockage permanent de tous les journaux

Les fichiers de log

- transactions écrites en cache par les processus serveur
- écrites dans les fichiers de log par les processus dédiés
- Espace de log paramétrable :
  - nbre et taille des groupes sous Oracle
  - *Min_wal_size* et *Max_wal_size* sous PostgreSQL

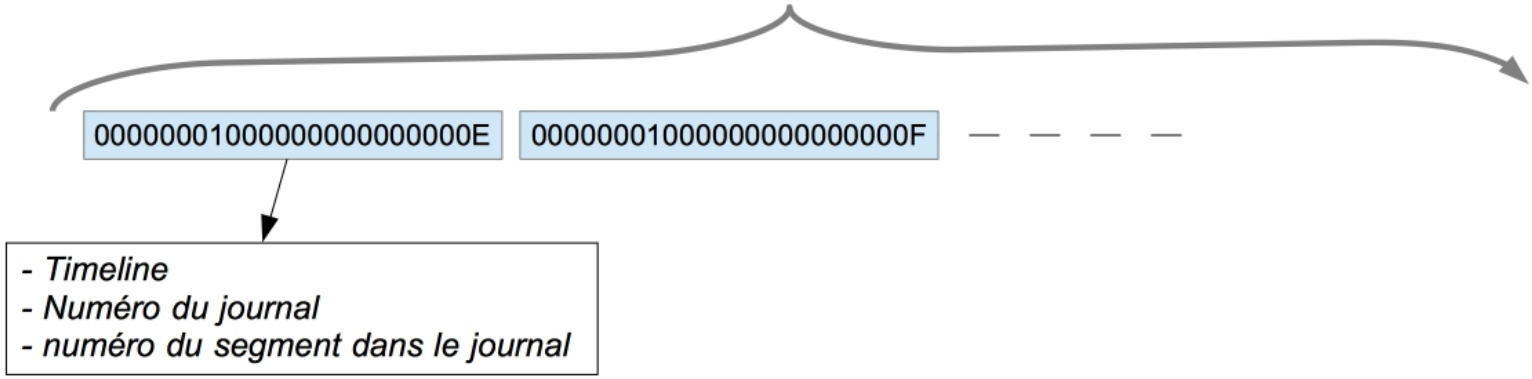
Gestion de la journalisation (logs) sous Oracle



Journalisation des transactions sous PostgreSQL

Ensemble de fichiers dans le répertoire **pg_xlog** – gestion par le processus *wal writer*

Journal de transactions : 4Go, sous la forme de 256 fichiers de 16Mo



Un nouveau journal est créé :
- en cas de changement de timeline (restauration arrière)
- lorsque le journal est plein

Pour déplacer les logs sur un autre disque (recommandé), la seule solution est de créer un lien symbolique pour le répertoire « pg_xlog ».

Les fichiers de log

- d'autres répertoires utilisés par PostgreSQL
- répertoire **clog**
  - stocke l'état des transactions
  - validée / annulée / en cours / sous-transaction
  - bitmap : 2 bits par transaction
- Répertoire *pg_commit_ts* : horodatage des transactions
- Répertoire *pg_subtrans_ts* : gestion des sous-transactions

Planifier la base

Phase de reflexion avant création, pour faire les bons choix.

- Réfléchir aux tables et indexes à venir, estimer leur taille
- Planifier la distribution de ses fichiers, l'espace libre dans les blocks
- Choisir l'encodage des caractères (peut-être surchargé par les clients)
- Déterminer la taille des blocs de données
- Déterminer la stratégie de sauvegarde et reprise après panne

Créer une base

- Oracle
  - Lancer une instance “inactive” à partir de paramètres
  - Commande CREATE DATABASE
- commande “initdb” pour PostgreSQL
  - Crée les répertoire, et deux BD de départ *template*1 et *template*0
  - Les choix sont à faire à l'installation de PostgreSQL : taille des blocs, segments, etc...
  - CREATE DATABASE permet de créer de nouvelles BD, en s'appuyant sur un exemple
  - OU BIEN : *createdb*

```
CREATE DATABASE mynewdb
USER SYS IDENTIFIED BY pz6r58
USER SYSTEM IDENTIFIED BY y1tz5p
LOGFILE GROUP 1 ('/u01/oracle/oradata/mynewdb/redo01.log') SIZE 100M,
GROUP 2 ('/u01/oracle/oradata/mynewdb/redo02.log') SIZE 100M,
GROUP 3 ('/u01/oracle/oradata/mynewdb/redo03.log') SIZE 100M
MAXLOGFILES 5
MAXLOGMEMBERS 5
MAXLOGHISTORY 1
MAXDATAFILES 100
MAXINSTANCES 1
CHARACTER SET US7ASCII
NATIONAL CHARACTER SET AL16UTF16
DATAFILE '/u01/oracle/oradata/mynewdb/system01.dbf' SIZE 325M REUSE
EXTENT MANAGEMENT LOCAL
SYSAUX DATAFILE '/u01/oracle/oradata/mynewdb/sysaux01.dbf' SIZE 325M REUSE
DEFAULT TABLESPACE tbs_1
DEFAULT TEMPORARY TABLESPACE tempts1
TEMPFILE '/u01/oracle/oradata/mynewdb/temp01.dbf'
SIZE 20M REUSE
UNDO TABLESPACE undotbs
DATAFILE '/u01/oracle/oradata/mynewdb/undotbs01.dbf'
SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

Démarrer une base de données

- ORACLE : Commande STARTUP
  - NOMOUNT : base fermée et non montée
  - MOUNT : base fermée et montée
  - FORCE : ouvre de force, en tuant une éventuelle instance démarrée
- PostgreSQL : *pg_ctl* start
  - Lancement du processus *postmaster*
  - Celui-ci lance un processus *startup*
  - Reprend la main si tout va bien
- Si besoin, les journaux sont rejoués (Recovering)

Fermer une base de données

- Oracle : commande SHUTDOWN
  - NORMAL : attend la déconnexion de tous les utilisateurs
  - IMMEDIATE : annule toutes les transactions non validées et tue les sessions en cours
  - TRANSACTIONAL : attend la fin des transactions puis tue les sessions
  - ABORT : Arrêt brutal, n'annule pas les transactions non validées
- PostgreSQL : *pg_ctl* stop
  - SMART : attend la déconnexion de tous les utilisateurs
  - FAST : annule toutes les transactions non validées et tue les sessions en cours
  - IMMEDIATE : Arrêt brutal, n'annule pas les transactions non validées

Introduction

- Il faut toujours avoir à l'esprit une politique de sécurité
- Assurée par le DBA, ou un administrateur dédié
- Principales tâches :
  - Gérer les utilisateurs
  - Affecter les ressources : tablespaces, quotas,...
  - Gérer les privilèges et les rôles
  - Surveiller l'usage de la base de données (Audit)
- Deux niveaux de sécurité "base de données"
  - Les comptes utilisateurs : login et mot de passe
  - Rôles, privilèges et profils : contrôle l'accès aux objets et aux commandes systèmes
- Ne pas oublier la sécurité du SE, et la sécurité "physique" des serveurs...

Introduction

1. N'installer que ce qui est nécessaire

Introduction

1. N'installer que ce qui est nécessaire
2. Sécuriser les compte par défaut après installation

Introduction

1. N'installer que ce qui est nécessaire
2. Sécuriser les compte par défaut après installation
3. Utiliser des mots de passe sécurisés, renouveler régulièrement

Introduction

1. N'installer que ce qui est nécessaire
2. Sécuriser les compte par défaut après installation
3. Utiliser des mots de passe sécurisés, renouveler régulièrement
4. Toujours accorder le minimum de privilèges aux utilisateurs

Introduction

1. N'installer que ce qui est nécessaire
2. Sécuriser les compte par défaut après installation
3. Utiliser des mots de passe sécurisés, renouveler régulièrement
4. Toujours accorder le minimum de privilèges aux utilisateurs
5. S'assurer de la sécurité du SE

Introduction

1. N'installer que ce qui est nécessaire
2. Sécuriser les compte par défaut après installation
3. Utiliser des mots de passe sécurisés, renouveler régulièrement
4. Toujours accorder le minimum de privilèges aux utilisateurs
5. S'assurer de la sécurité du SE
6. S'assurer de la sécurité du réseau (Ex. : SSL)

Introduction

1. N'installer que ce qui est nécessaire
2. Sécuriser les compte par défaut après installation
3. Utiliser des mots de passe sécurisés, renouveler régulièrement
4. Toujours accorder le minimum de privilèges aux utilisateurs
5. S'assurer de la sécurité du SE
6. S'assurer de la sécurité du réseau (Ex. : SSL)
7. Appliquer les correctifs de sécurité

Introduction

1. N'installer que ce qui est nécessaire
2. Sécuriser les compte par défaut après installation
3. Utiliser des mots de passe sécurisés, renouveler régulièrement
4. Toujours accorder le minimum de privilèges aux utilisateurs
5. S'assurer de la sécurité du SE
6. S'assurer de la sécurité du réseau (Ex. : SSL)
7. Appliquer les correctifs de sécurité
8. Signaler les failles

Au niveau du système

1. PostgreSQL ne peut pas être lancé par un administrateur système
2. Par défaut : super utilisateur “postgres”, propriétaire des fichiers
3. Les fichiers ne sont pas cryptés ; possible avec l'extension *pgcrypto*
4. Pas de somme de contrôle sur les fichiers données par défaut
5. Les communications, par défaut, sont en clair (SSL configurable)
6. Attention à l'injection d'SQL dans les applications

Rôles

1. Permettent de définir des types d'utilisateurs
2. Ce sont des ensembles de droits, avec login et mot de passe
3. Le rôle *postgres* a tout les droits
4. Tous les utilisateurs ont le rôle *public* : permet de désactiver des droits à tout le monde
5. Les rôles sont globaux ; non spécifiques à une BD

Rôles

1. Tous les rôles ont des droits par défaut (à retirer si besoin)
   1. Droit de connexion sur toutes les BD ;
   2. Droit de créer des objets sur tous les schémas *publics* des BD
   3. Droit de créer des objets temporaires
   4. Droit de créer des fonctions SQL et PL/pgSQL
   5. Droit d'utliliser les fonctions créées par d'autres dans le schéma *public*
2. Pas de droit de modification et d'accès aux objets des autres !
3. Pas le droit de créer une base, ou un rôle.

Rôles

1. Les droits sont accordés avec GRANT et retirés avec REVOKE
2. Portent sur des objets spécifiques, ou des ensembles d'objets
3. Un rôle peut être affecté à un autre rôle, comme un droit
4. Possibilité de définir des politiques de sécurité sur les tables
5. On peut supprimer un rôle s'il ne possède pas d'objets.

Droits sur les tuples

1. “Politiques de sécurité” en postgreSQL (seulement)
2. Possibilité de passer par des fonctions ou procédures

Différence Oracle / PostgreSQL

1. Sous Oracle, on affecte des *ROLE* aux *USER*.
2. PostgreSQL : un *ROLE* peut lui-même être un utilisateur. Pas de mot clé *USER*.
3. On peut donc créer des rôles “génériques” qu’on affecte à des rôles “utilisateurs simples”
4. Dans tous les cas : éviter d’attribuer des droits/privilèges directement à des utilisateurs, sans passer par des rôles.

Sauvegarde à froid

- Arrêter le serveur
- Sauvegarder la totalité des fichiers
  - Répertoire principal, et tous les autres
  - Repérer tous les tablespaces avec une requête sur *pg_tablespace*
- Redémarrer le serveur (sur un OS identique !)

Sauvegarde à chaud

- Export logique avec *pg_dump*
  - Une seule BD
  - Ne sauvegarde pas les objets globaux
  - en SQL, .tar, custom,
- Export logique *pg_dumpall*
  - Tout le cluster PostgreSQL
  - totale ou partielle
  - format SQL
- Sauvegarde continue à chaud
  - Idem sauvegarde à froid, en utilisant *pg_start_backup* et *pg_stop_backup*
  - nécessite que l'archivage soit activé, pour lever les incohérences.
- Effectuer des tests de restauration ! (voir doc pour procédure détaillée)

Réplication

- Cluster de réplication (2 noeuds minimum)
- Le noeud primaire accepte les modifications
- La charge de lecture peut être partagée
- Un noeud secondaire peut être promu en primaire
- Possibilité de réplication en cascade
- Réplication physique
  - Transmission et “re-jeu” des journaux, au fil de l’eau
  - Réplication de l’instance entière uniquement
  - Les serveurs ont la même architecture
- Réplication logique lève ces problèmes

- Réglage de l'instance, des paramètres, de la répartition des fichiers
- Réglage des requêtes et accès aux données

Concevoir et développer pour la performance

- Eviter absolument les conflits et limites de ressource
- Ne pas penser que l'investissement en matériel va assurer les performances
- Penser en terme de "passage à l'échelle"
  - Comportement linéaire dans la charge de travail
  - Spécificités internet
    - disponibilité 24/24
    - nombre d'accès imprévisible
    - souplesse des requêtes
    - Volatilité et exigence des utilisateurs (7s. d'attente au maximum)
    - Concevoir/développer vite et bien !
  - Causes de mauvaises performances
    - Mauvaise conception, ou mauvaise implémentation
    - Mauvais dimensionnement matériel
    - Limitations logicielles : application, DBMS ou SE

Concevoir et développer pour la performance (cont.)

- Savoir répondre aux questions suivantes
  - Combien d'utilisateurs à supporter ? très peu, peu à beaucoup, une infinité
  - Quelle mode d'interaction ? Navigateur web ou application cliente personnalisée
  - Où sont les utilisateurs ? (Temps de transfert réseaux)
  - Quelle charge d'accès, combien de données en lecture seule ?
  - Quel est le temps de réponse requis par les utilisateurs ?
  - Quelle disponibilité requise par les utilisateurs ?
  - Mises à jour en temps réel ?
  - Quel taille à prévoir pour les données ?
  - Quelles sont les contraintes budgétaires ?

Principes pour la conception

- Ne pas faire compliqué quand on peut faire simple
  - Eviter les schémas ou requêtes incompréhensibles (utiliser des vues si besoin)
  - Eviter les superpositions de couches logicielles
- Soigner la modélisation des données pour les parties principales
- Implémenter un schéma en 3NF au moins pour assurer la flexibilité
  - Optimiser avec vues matérialisées, clusters, colonnes calculées
  - Bien organiser les index
- Organiser des campagnes de tests crédibles facilitera le déploiement

- SQL = langage déclaratif
- Optimiseur explore plusieurs plans d'exécution
- Un coût est estimé pour chaque plan "exploré" en utilisant les statistiques sur les données
- Deux modes d'optimisation possibles
  - Coût de récupération de la première ligne (clusters)
  - ou de la récupération de toutes les lignes
- Oracle permet les *hints* (conseils) et stocke les résultats d'optimisation en cache.

Accès aux données

- Plusieurs façons d'accéder aux données
- *Seqscan* parcours séquentiel des blocs d'une table
  - Seul possible si aucun index
  - Le plus efficace pour petites tables
  - *FULLTABLESCAN* sous Oracle.
- *BitmapIndexScan*
  - Parcours de l'index, génération d'un bitmap, puis parcours de la table.
- *IndexOnlyScan*
  - Lorsque l'index contient toutes les informations
  - Utilise le fichier VM, mais pas la table

Index : quels colonnes ?

- Quelles colonnes faut-il indexer ?
  - Attributs utilisés fréquemment dans les clause WHERE (jointures ou sélections)
  - L'efficacité augmente avec la sélectivité de l'attribut
  - Automatique pour les clés primaires, unique
  - Effet sur les performances en cas de maj
- Inutile si la clé d'indexation est passée en paramètre d'une fonction
- On peut faire des index composés de plusieurs colonnes
  - Si utilisées ensemble avec une clause AND
  - Placer en premier les attributs les plus fréquemment utilisés
  - Sinon, placer en premier celui sur lequel est ordonnée la table

Les différents types de jointure

- Jointure imbriquée : double boucle
  - Lorsque peu de lignes doivent être jointes à droite, ou présence d'un index
  - O(NxM)
- jointure par hachage : la table la plus petite est "hachée" en mémoire
  - Lorsque la table hachée tient en mémoire
  - Un seul parcours de chaque table
  - Ne supporte que l'égalité
- Jointure par tri fusion : tri puis fusion de chaque opérande
  - Lorsque les sources sont déjà triées, ou avec index B-arbre
  - La condition de jointure est une inégalité
- Jointure cartésienne : produit cartésien, pas de condition de jointure
- Toutes ces variantes existent en jointure externe.

Quelques “plus” d’Oracle

- Possibilité de créer des clusters de tables
- index de fonctions
- Optimisation des *N* premières réponses (N=10,100,1000)
- Possibilité d’utiliser *HINT*

Statistiques et Explain

- les statistiques sont récoltées régulièrement par *autovacuum*
- A lancer à la main en cas de batch
- Commande *EXPLAIN* pour visualiser le plan d'exécution et les coûts estimés
- option ANALYSE pour exécuter réellement la requête et récupérer les mesures.
- Outils graphiques avec *pgAdmin*

Conseils

- Bien planifier ses index
- Jointures inutiles (SQL généré dans l'application)
- Attention aux connexions multiples (application)
- Privilégier les clauses AND
- Privilégier UNION ALL sur UNION
- Pas de DISTINCT inutile
- Utiliser JOIN (norme SQL) pour éviter les produites cartésiens
- Bonne utilisation des connecteurs EXISTS, IN.

Etapes pour la résolution des problèmes

1. Vérifications préliminaires (avant les problèmes)
   1. Récolter les impressions de base, les projets des utilisateurs
   2. Récolter le maximum de statistiques (SE, DB, applications) lorsque les performances sont bonnes et lorsqu'elles sont mauvaises
   3. Vérifier régulièrement les SE des utilisateurs (matériel, ressources...)
2. Comparer les symptômes avec les "10 erreurs fréquemment commises"
3. Réaliser une modélisation conceptuelle du système lors de l'apparition des symptôme
4. Lister toutes les solutions et les appliquer une à une jusqu'à l'obtention du résultat, ou l'identifiaction des contraintes extérieures conduisant à l'échec.

Traitement des urgences...

- Bien souvent, un problème doit-être traité dans l'urgence avant une résolution rigoureuse
- Les étapes sont alors "raccourcies" :
  1. Faire l'inventaire des problèmes, des symptômes, des changements récents
  2. Vérifier l'état du matériel : CPU, disques, mémoire, réseau de chaque tier
  3. Déterminer si le problème est au niveau du CPU ou de l'attente d'évènement. Utiliser les vues dynamiques sur les performances du catalogue.
  4. Appliquer des mesures d'urgence pour stabiliser le systèmes : suspendre une application, réduire la charge, tuer un processus...
  5. vérifier la stabilité du système, récolter des statistiques, et suivre la procédure complète de résolution