11-06 après-midi Formation : Architecture et Administration des Bases de Données - Théorie et Pratique

Architecture et Administration des Bases de Données : Théorie et Pratique

Introduction : L'écosystème des systèmes d'information modernes

Dans le paysage complexe des technologies de l'information contemporaines, la gestion des bases de données constitue un pilier fondamental de toute infrastructure informatique. Contrairement à une vision simpliste qui imaginerait une organisation reposant sur une unique base de données

monolithique, la réalité moderne révèle un écosystème sophistiqué et distribué, où chaque composant répond à des besoins spécifiques et complémentaires. Cette complexité n'est pas accidentelle : elle résulte d'une évolution naturelle vers la spécialisation et l'optimisation. À mesure que les organisations croissent et que leurs besoins se diversifient, l'architecture des données se structure selon une logique de distribution fonctionnelle qui permet d'optimiser les performances, la maintenance et la sécurité.

L'architecture distribuée : une nécessité fonctionnelle

Au cœur de toute organisation moderne, qu'il s'agisse d'une institution financière ou d'une entreprise industrielle, on retrouve des **systèmes centraux** qui constituent l'épine dorsale informationnelle. Dans le secteur bancaire, le *core banking* centralise l'ensemble des opérations fondamentales : gestion des comptes clients, traitement des transactions, administration des produits financiers et respect des contraintes réglementaires. Pour les entreprises classiques, l'**ERP** (Enterprise Resource Planning) - qu'il s'agisse de solutions comme SAP ou de développements propriétaires - intègre comptabilité, gestion des stocks, ressources humaines et l'ensemble des processus métier critiques. Ces systèmes centraux s'appuient généralement sur de volumineuses bases de données relationnelles capables de traiter des téraoctets d'informations avec des exigences de cohérence, de disponibilité et de performance extrêmement élevées. Cependant, ils ne fonctionnent jamais de manière isolée. En parallèle, une multitude de bases de données spécialisées coexistent dans l'écosystème informatique : systèmes de gestion de contenu (CMS) pour les sites web d'entreprise, applications métier spécifiques comme les CRM ou les outils de reporting, bases de données analytiques dédiées au business intelligence, systèmes de cache pour l'optimisation des performances, et bien d'autres encore. Cette architecture distribuée présente des avantages considérables : spécialisation fonctionnelle permettant d'optimiser chaque composant pour son usage spécifique, isolation des risques évitant qu'un problème sur un système affecte l'ensemble, facilité de maintenance et d'évolution, et possibilité de faire appel à des technologies différentes selon les besoins.

Le métier de Database Administrator : évolution et responsabilités

Transformation du rôle du DBA

Le métier de Database Administrator (DBA) a connu une évolution remarquable au cours des dernières décennies. Autrefois poste dédié à temps plein dans les grandes organisations, avec des spécialistes exclusivement focalisés sur la gestion des bases de données, il tend aujourd'hui à devenir une **compétence transversale** que les professionnels IT doivent maîtriser en complément d'autres responsabilités. Cette transformation s'explique par plusieurs facteurs convergents : l'automatisation croissante des tâches d'administration grâce aux outils modernes, la démocratisation des systèmes de gestion de bases de données avec des interfaces plus accessibles, et surtout la nécessité d'une approche DevOps qui intègre développement, administration système et gestion des données dans une vision cohérente. Le DBA moderne doit naviguer entre plusieurs domaines d'expertise : une compréhension fine de l'infrastructure (hardware, réseau, stockage), une collaboration étroite avec les équipes de développement pour l'optimisation applicative, et une connaissance approfondie des enjeux métier pour une modélisation optimale des données.

Responsabilités techniques fondamentales

Installation et configuration : approches multiples

Le DBA maîtrise plusieurs méthodologies d'installation selon les contextes et les contraintes. L'**installation native** suit la documentation officielle du SGBD et offre un contrôle maximal sur la configuration, mais nécessite une expertise technique approfondie. L'**installation par gestionnaires de paquets** (APT, YUM, ou autres) simplifie le processus et assure une meilleure intégration avec l'écosystème système, au prix d'une certaine standardisation. La **containerisation** via Docker permet un déploiement rapide avec des configurations pré-établies et une isolation des environnements. Cependant, il convient de noter un point d'attention critique concernant la containerisation des bases de données. Bien que cette approche facilite grandement le déploiement et le test, elle présente des défis spécifiques dans les environnements de production, notamment dans les plateformes d'orchestration comme Kubernetes ou Docker Swarm. Les problématiques de **stockage persistant** et les questions de **performance** dues aux couches d'abstraction supplémentaires rendent souvent préférable un déploiement sur infrastructure dédiée pour les systèmes critiques.

Sécurité et gestion des accès : responsabilités critiques

Le DBA détient des privilèges administrateur complets sur les systèmes de bases de données, ce qui implique des responsabilités particulièrement lourdes. Il doit mettre en place une **gestion granulaire des droits d'accès** par utilisateur et par objet, assumant une **responsabilité de confidentialité** absolue sur les données sensibles. Dans le contexte réglementaire actuel, notamment avec l'entrée en vigueur du RGPD, le DBA peut endosser le rôle de Data Protection Officer ou du moins jouer un rôle central dans la conformité réglementaire. Il doit implémenter les politiques de sécurité incluant chiffrement, audit et traçabilité, tout en maintenant l'équilibre délicat entre sécurité et performance.

Haute disponibilité : enjeux critiques

La garantie de disponibilité constitue l'un des défis majeurs du métier de DBA, particulièrement dans les environnements industriels critiques. L'exemple souvent cité de l'aciérie illustre parfaitement ces enjeux : lorsqu'une poche de métal en fusion est en cours de traitement, un arrêt de la base de données contrôlant le processus peut avoir des conséquences dramatiques, allant de la perte économique massive aux risques pour la sécurité des personnes. Cette criticité impose la mise en place de systèmes **résilients** avec des mécanismes sophistiqués : sauvegardes à chaud permettant la sauvegarde sans interruption de service, réplication en temps réel pour assurer la redondance, plans de reprise d'activité (PRA) et de continuité d'activité (PCA) testés régulièrement, et monitoring proactif pour la détection précoce des incidents. Dans de tels contextes, les DBA sont souvent soumis à des astreintes 24h/7j.

Optimisation et tuning : l'art de la performance

L'optimisation constitue l'aspect le plus technique et le plus gratifiant du métier de DBA. Elle englobe plusieurs dimensions complémentaires : l'optimisation des requêtes par l'analyse des plans d'exécution et l'indexation intelligente, le tuning infrastructure incluant la configuration des I/O, de la mémoire et des processeurs, la gestion sophistiquée du stockage avec l'optimisation des systèmes RAID et SAN, et la conception d'architectures haute performance utilisant clustering et partitioning. Cette expertise technique place le DBA au cœur de la performance globale du système d'information, avec un impact direct sur l'expérience utilisateur et l'efficacité opérationnelle de l'organisation.

Architecture interne des bases de données : le modèle Oracle

Séparation logique et physique : un principe fondamental

Pour comprendre les mécanismes internes des bases de données modernes, le modèle architectural d'Oracle offre un exemple particulièrement éclairant de la séparation entre abstraction logique et réalité physique. Cette séparation constitue l'un des piliers conceptuels qui permettent aux SGBD d'offrir flexibilité et performance.

Structure logique hiérarchique

La structure logique s'organise selon une hiérarchie claire et cohérente : Au niveau le plus élevé, la **base de données** représente l'entité logique globale, regroupant l'ensemble des informations d'une application ou d'une organisation. Cette base se subdivise en **tablespaces**, espaces de stockage logiques qui regroupent des objets selon leur fonction ou leur criticité. Chaque tablespace

contient des **segments**, représentations logiques d'objets spécifiques comme les tables ou les index. Ces segments sont eux-mêmes composés d'**extents**, ensembles contigus de blocs alloués dynamiquement selon les besoins. Enfin, au niveau le plus fin, les **data blocks** constituent la plus petite unité de stockage logique, généralement de quelques kilo-octets. Cette organisation hiérarchique permet une gestion granulaire et optimisée des ressources, où chaque niveau peut être configuré et optimisé indépendamment selon les besoins spécifiques.

Structure physique correspondante

La structure physique mappe cette organisation logique sur les supports de stockage réels. Les **datafiles** constituent les fichiers physiques stockés sur le système de fichiers ou directement sur les dispositifs de stockage. Le mapping entre tablespaces logiques et datafiles physiques offre une flexibilité considérable pour l'optimisation des performances et la gestion de l'espace. Pour les environnements à très hautes exigences de performance, Oracle propose également l'accès aux **raw devices**, permettant un accès direct aux blocs physiques sans passer par le système de fichiers. Cette approche, bien que complexe à gérer, peut apporter des gains de performance significatifs dans des contextes spécifiques.

Stratégies d'optimisation avancées

Séparation fonctionnelle des données

L'architecture en tablespaces permet une optimisation fine en fonction des caractéristiques d'usage des données. Un **tablespace Index** peut être configuré sur des supports rapides comme les SSD pour optimiser les accès aux index, tandis que le **tablespace Data** contenant les données principales peut résider sur un stockage standard offrant un bon compromis performance/coût. Pour les données historiques peu consultées, un **tablespace Archive** peut utiliser un stockage lent mais très capacitif.

Stratégie Hot/Warm/Cold

Cette approche classifie les données selon leur fréquence d'accès et adapte la stratégie de stockage en conséquence. Les données **Hot**, fréquemment accédées, bénéficient d'un stockage haute performance. Les données **Warm**, occasionnellement utilisées, sont placées sur un stockage standard. Les données **Cold**, rarement consultées mais devant rester accessibles, sont archivées sur un stockage économique haute capacité.

Raw Devices : optimisation extrême

Pour les environnements à exigences de performance exceptionnelles, l'utilisation de raw devices permet d'éliminer les couches d'abstraction du système de fichiers et d'accéder directement aux blocs physiques. Cette approche trouve sa justification dans des contextes très spécifiques : trading haute fréquence où quelques millisecondes déterminent la rentabilité, calcul scientifique

intensif traitant des téraoctets de données (CERN, simulations climatiques), ou ERP critiques gérant des processus industriels continus. Les gains de performance peuvent être spectaculaires, mais la complexité de gestion s'accroît considérablement, nécessitant une expertise très pointue.

Gestion des transactions et garantie de cohérence

Architecture des logs de transaction : universalité du concept

Tous les SGBD modernes implémentent un système de journalisation sophistiqué pour garantir la cohérence des données, bien que les terminologies varient selon les éditeurs. PostgreSQL utilise le **WAL** (Write Ahead Log), MySQL s'appuie sur le **Binary Log** (bin log), Oracle maintient des **Transaction Logs** et **Redo Logs**, tandis que SQL Server gère ses propres **Transaction Logs**.

Mécanisme de fonctionnement

Le principe de fonctionnement repose sur l'**écriture préalable** : toute modification est d'abord écrite dans le journal avant d'être appliquée aux données. L'**écriture différée** permet ensuite d'écrire les données en mémoire puis sur disque de manière asynchrone, optimisant ainsi les performances. En cas d'incident, le processus de **recovery** utilise le journal pour reconstruire un état cohérent. Cette architecture garantit les propriétés **ACID** (Atomicité, Cohérence, Isolation, Durabilité) fondamentales aux bases de données transactionnelles, permettant de maintenir l'intégrité des données même en cas de panne système.

Processus de récupération

En cas de panne système, le processus de recovery suit une séquence précise et rigoureuse. L'**analyse du log** identifie d'abord les transactions incomplètes et détermine l'état de chaque transaction au moment de la panne. La phase de **Redo** rejoue ensuite les transactions validées mais non encore écrites sur disque, garantissant que toutes les modifications confirmées sont bien persistées. La phase d'**Undo** annule les transactions non validées, restaurant la cohérence. Enfin, une **vérification de cohérence** contrôle l'intégrité globale des données.

Applications avancées : réplication et architectures modernes

Le journal de transaction trouve des applications particulièrement élégantes dans la **réplication** entre serveurs. Dans une architecture Master-Slave, le serveur maître n'envoie que son journal de transaction, que le serveur esclave applique à son propre datafile. Cette approche est infiniment plus efficace que la réplication complète des fichiers, car seules les modifications effectives sont transmises, les opérations de lecture et les transactions échouées n'étant pas incluses. Cette séparation entre données et journal ouvre également la voie à des architectures avancées comme **CQRS** (Command Query Responsibility Segregation), où les opérations de lecture et d'écriture sont physiquement séparées pour optimiser les performances.

Configuration et optimisation des SGBD

Paramètres réseau et connectivité

Configuration de l'adresse de liaison

Le paramètre **bind address** détermine les interfaces réseau sur lesquelles le SGBD accepte les connexions. Une configuration sur **localhost** limite l'accès à la machine locale, offrant une sécurité maximale mais une flexibilité réduite. Une configuration sur **0.0.0.0** permet l'accès depuis n'importe quelle adresse IP, offrant une flexibilité maximale mais nécessitant une vigilance sécuritaire accrue. Le choix dépend fondamentalement de l'architecture : développement local versus accès multi-machines.

Dimensionnement des connexions : coordination critique

La configuration du nombre maximum de connexions nécessite une **coordination étroite** entre administrateurs de bases de données et équipes applicatives. Les problématiques courantes incluent le sous-dimensionnement applicatif (application configurée pour 10 connexions maximum, base acceptant 200 connexions) créant un goulot d'étranglement au niveau applicatif, ou le sous-dimensionnement base (application configurée pour 2000 connexions, base limitée à 200) provoquant des erreurs de connexion. Les bonnes pratiques recommandent d'inclure les connexions de service (backup, maintenance) dans le dimensionnement, de surveiller les métriques de pool de connexions, et d'ajuster dynamiquement selon la charge observée.

Optimisation mémoire : shared buffers et stratégies de cache

Les **shared buffers** constituent le cache principal du SGBD, où les blocs de données fréquemment accédés restent en mémoire pour réduire drastiquement les accès disque. L'amélioration des performances est directement proportionnelle à la taille du cache, dans la limite des ressources disponibles. La configuration peut adopter une approche de **pool unique** pour la simplicité de gestion avec allocation globale, ou de **pools multiples** permettant une optimisation fine par type d'opération.

Espaces de travail pour jointures et stratégies d'optimisation

La mémoire de travail dédiée aux jointures influence directement les **stratégies d'optimisation** choisies par le SGBD. Deux approches principales existent : Les **Nested Loops** (boucles imbriquées) suivent une logique simple de double boucle, avec une complexité O(n×m), une faible consommation mémoire, mais des performances dégradées sur gros volumes. Les **Hash Join** créent des tables de hachage en mémoire, permettent une parallélisation sur plusieurs cœurs, offrent des performances supérieures mais avec une consommation mémoire importante. L'optimiseur choisit automatiquement la stratégie en fonction des ressources disponibles et des caractéristiques des données.

Optimisation des requêtes : l'art de la performance

L'optimiseur automatique : intelligence du SGBD

L'**optimiseur de requêtes** constitue le cerveau du SGBD, responsable de transformer une requête SQL en plan d'exécution optimal. Ses critères d'optimisation incluent la taille des tables impliquées, les statistiques de distribution des données, les index disponibles, et les ressources système disponibles.

Intervention manuelle : hints et optimisation ciblée

Lorsque l'optimiseur automatique produit des plans sous-optimaux, les **hints** permettent d'orienter ses décisions. Ces indications peuvent porter sur l'ordre de traitement des tables, la sélection d'index spécifiques, ou le choix de stratégie de jointure. Le processus d'optimisation suit généralement ces étapes : analyse du plan d'exécution initial, identification des goulots d'étranglement, application de hints ciblés, et comparaison des métriques de performance.

Outils d'analyse avancés

Les SGBD modernes intègrent des **advisors automatiques** qui détectent proactivement les requêtes problématiques, proposent des suggestions d'optimisation chiffrées, et offrent des interfaces graphiques pour l'aide à la décision. Cette sophistication justifie économiquement l'investissement dans des solutions commerciales pour les charges complexes impliquant des requêtes de 60-80 lignes sur des centaines de tables.

Gestion des utilisateurs et contrôle d'accès

Architecture hiérarchique des permissions

Fondements conceptuels

La gestion des droits dans les systèmes de bases de données s'organise selon une hiérarchie logique qui répond à des impératifs pratiques de simplification administrative. Au niveau le plus basique, les **droits unitaires** constituent les autorisations élémentaires d'effectuer une action spécifique sur un objet particulier. Bien que cette granularité fine offre un contrôle précis, elle devient rapidement ingérable dans des environnements complexes. Les **rôles** regroupent des ensembles cohérents de droits selon une logique fonctionnelle ou métier. Un rôle "bibliothécaire" pourrait inclure des droits en lecture sur toutes les tables de livres, des droits en écriture sur la table des emprunts, et des droits spécifiques sur la gestion des adhérents. Les **groupes d'utilisateurs** permettent une gestion collective, où les rôles peuvent être assignés à des groupes, bénéficiant automatiquement à tous leurs membres.

Implémentation technique

La syntaxe SQL standardisée suit des principes similaires dans tous les SGBD. La création d'entités utilise les commandes `CREATE USER` et `CREATE ROLE`. L'attribution des droits s'effectue avec la commande `GRANT` qui associe systématiquement privilège, objet et bénéficiaire. La fonctionnalité `WITH GRANT OPTION` permet la délégation de l'attribution des droits, particulièrement utile pour la gestion décentralisée. La commande `REVOKE` permet l'annulation des privilèges précédemment accordés.

Intégration avec les systèmes d'authentification d'entreprise

Modes d'authentification multiples

Les bases de données d'entreprise supportent généralement plusieurs modes d'authentification. L'**authentification native** utilise les mécanismes propres au SGBD, offrant un contrôle total mais nécessitant une administration dédiée. L'**intégration Active Directory** permet de centraliser la gestion des identités, avec les comptes définis dans AD et l'authentification déléguée au contrôleur de domaine. L'**authentification intégrée Windows/Kerberos** représente le niveau d'intégration le plus poussé, offrant une expérience Single Sign-On transparente mais avec des défis techniques importants concernant la propagation des tokens et la gestion des erreurs.

Considérations économiques : impact des licences

Les modèles de licencing influencent significativement les choix d'architecture. Le **licencing par utilisateur nommé** impose un coût pour chaque utilisateur accédant à la base, tandis que le **licencing par cœur de processeur** facture selon la puissance du serveur, indépendamment du nombre d'utilisateurs. Face aux coûts élevés des licences propriétaires, les entreprises adoptent souvent des stratégies de **comptes de service applicatifs**, où une application utilise un compte unique et gère elle-même les autorisations par utilisateur, réduisant drastiquement le nombre de licences nécessaires.

Stratégies de sauvegarde et de restauration

Principe du moindre privilège : prévention fondamentale

Le **principe du moindre privilège** constitue un pilier fondamental de la sécurité informatique. Il stipule qu'un utilisateur ne doit disposer que des privilèges strictement nécessaires à l'accomplissement de ses fonctions légitimes. Dans le contexte des bases de données, cela signifie éviter de se connecter systématiquement avec des comptes administrateurs pour des tâches courantes. L'exemple classique d'une commande Linux illustre l'importance de ce principe : la différence entre `rm -rf toto/*` et `rm -rf toto/ *` (un simple espace) peut avoir des conséquences dramatiques si exécutée avec des privilèges root. Les statistiques montrent que près de la moitié des incidents en production résultent de commandes exécutées avec des privilèges excessifs.

Types de sauvegarde selon la disponibilité

Sauvegarde à froid vs. sauvegarde à chaud

La **sauvegarde à froid** nécessite l'arrêt complet du système, garantissant une cohérence parfaite mais avec une interruption de service. Elle convient aux entreprises disposant de fenêtres de maintenance régulières. La **sauvegarde à chaud** maintient la disponibilité du service mais introduit un risque de perte de données ou d'incohérence. Pour les grandes plateformes comme Netflix, la sauvegarde est un processus continu créant un cycle perpétuel de protection des données.

Stratégies techniques pour la sauvegarde à chaud

Le **snapshot** capture l'état des données à un instant T, offrant une cohérence parfaite si la capture s'effectue entre deux transactions. La **sauvegarde par réplication** utilise une architecture maître-esclave où la réplication est interrompue sur l'esclave pour effectuer la sauvegarde, puis reprend pour rattraper les modifications manquantes.

Approches de sauvegarde : logique vs. physique

Sauvegarde logique (dump)

La sauvegarde logique exporte les données sous forme de requêtes SQL reconstituables. Elle offre une flexibilité maximale, une lisibilité du format, une portabilité entre versions et plateformes, et une granularité permettant la restauration sélective. Cependant, elle présente des limitations en termes de performance et de volumétrie.

Sauvegarde physique

La sauvegarde physique implique la copie directe des fichiers de données et des journaux de transactions. Elle nécessite un **mode backup** pour garantir la cohérence, figeant les datafiles en lecture seule pendant la copie et appliquant ensuite les logs pour la mise à jour.

Sauvegarde par snapshot

Le snapshot utilise les capacités de l'infrastructure sous-jacente pour créer une image instantanée au niveau du système de fichiers. Cette approche offre un impact minimal sur la base en fonctionnement, une rapidité de création quasi-instantanée, et pas d'interruption de service.

Stratégies de sauvegarde selon le contenu

Types de sauvegarde

La **sauvegarde complète** capture l'intégralité des données, constituant la référence pour toute stratégie. La **sauvegarde différentielle** capture tous les changements depuis la dernière sauvegarde complète, facilitant la restauration mais avec une taille croissante. La **sauvegarde incrémentale** capture uniquement les changements depuis la dernière sauvegarde, réduisant la taille mais complexifiant la restauration.

Point-in-Time Recovery (PITR)

Le PITR permet de restaurer une base de données à un moment précis en combinant une sauvegarde physique et l'application des journaux de transactions jusqu'au moment souhaité. Cette technique s'avère particulièrement utile pour les investigations forensiques et la récupération après incidents.

Métriques et objectifs

Indicateurs de performance

Le **RPO** (Recovery Point Objective) définit la perte de données maximale acceptable, influençant directement la fréquence des sauvegardes. Le **RTO** (Recovery Time Objective) spécifie le temps maximal acceptable pour restaurer le service, déterminant le choix de la stratégie de restauration.

Politiques de rétention

La politique de rétention définit la durée de conservation des sauvegardes et doit prendre en compte les besoins métier, les contraintes légales (RGPD, enquêtes fiscales), et les coûts de stockage. Elle nécessite une collaboration étroite entre équipes techniques et métier.

Optimisation des données historisées : le processus de freeze

Principe fondamental

L'optimisation des données historisées représente un enjeu crucial dans la gestion moderne des bases de données. Le processus de **freeze** (gel) constitue une technique d'optimisation particulièrement efficace pour traiter les données anciennes qui ne subissent plus de modifications.

Mécanisme de transformation

Le freeze repose sur la **transformation de données mutables en données immuables**. Lorsque des données atteignent un certain âge ou que leur période de modification active se termine, elles sont marquées comme étant en **lecture seule**. Cette transformation entraîne plusieurs optimisations automatiques : élimination des logs de transaction puisqu'aucune modification future n'est attendue, optimisation du cache avec mise en cache agressive des données immuables, et restructuration physique pour optimiser les accès en lecture.

Applications pratiques et bénéfices

Dans le secteur bancaire, les transactions de plus d'un exercice fiscal peuvent être gelées, permettant une consultation rapide de l'historique sans impacter les performances des opérations courantes. Les logs d'activité système, une fois consolidés, bénéficient grandement de cette optimisation. Cette technique s'inscrit dans une approche de **séparation des charges OLTP et OLAP**, où les données gelées se rapprochent naturellement du modèle OLAP optimisé pour l'analyse et la consultation.

Évolutions et perspectives

Impact du cloud et de l'intelligence artificielle

Les environnements cloud modernes permettent d'optimiser cette approche en déplaçant automatiquement les données gelées vers des supports de stockage moins coûteux mais toujours accessibles. L'intelligence artificielle peut aider à déterminer automatiquement le moment optimal pour geler des données, en analysant les patterns d'accès et les contraintes métier.

Défis et limitations

La gestion de la transition entre états mutable et gelé doit être soigneusement orchestrée. Il faut prévoir des procédures pour les cas exceptionnels où des données gelées doivent être modifiées, impliquant généralement un "dégel" temporaire contrôlé. L'état des données doit être tracé et auditable, particulièrement dans des contextes réglementés.

Schéma récapitulatif

https://archive.qoyri.fr/mindmaps/cours06.11.25.2.html

Concepts architecturaux principaux

Architecture des systèmes d'information :

- Systèmes centraux (Core Banking, ERP) et bases périphériques
- Approche distribuée et spécialisée
- Séparation fonctionnelle des données **Architecture interne des SGBD :**
- Structure logique : Base → Tablespaces → Segments → Extents → Blocks
- Structure physique : Datafiles et Raw devices
- Stratégies d'optimisation Hot/Warm/Cold

Rôles et responsabilités

Database Administrator (DBA) :

- Installation et configuration (native, paquets, conteneurs)
- Sécurité et conformité (RGPD, accès granulaire, audit)
- Haute disponibilité (sauvegardes, réplication, astreintes)
- Optimisation (requêtes, infrastructure, stockage) **Gestion des utilisateurs :**
- Hiérarchie Droits → Rôles → Groupes
- Commandes GRANT/REVOKE avec WITH GRANT OPTION
- Intégration Active Directory et authentification Kerberos

Stratégies de sauvegarde et performance

Types de sauvegarde :

- Par disponibilité : Cold backup vs Hot backup
- Par contenu : Complète, Différentielle, Incrémentale
- Par méthode : Logique (dump), Physique, Snapshot **Métriques et objectifs :**
- RPO (Recovery Point Objective) : perte de données acceptable
- RTO (Recovery Time Objective) : temps de restauration maximal
- PDMA (Perte de Données Maximale Admissible)

Technologies et optimisation

Gestion transactionnelle :

- Logs de transaction (WAL, bin log, transaction logs)
- Mécanismes de recovery (Redo/Undo)
- Propriétés ACID garanties **Optimisation avancée :**
- Freeze des données historisées
- Séparation OLTP/OLAP
- Point-in-Time Recovery (PITR)

Mots-clés essentiels

`DBA`, `ERP`, `Core Banking`, `Tablespace`, `Datafiles`, `Raw devices`, `WAL`, `RGPD`, `Hot/Warm/Cold storage`, `ACID`, `Recovery`, `High Availability`, `Shared Buffers`, `Connection Pool`, `Nested Loops`, `Hash Join`, `Hints`, `Optimiseur`, `Rollback`, `Commit`, `RBAC`, `Information Schema`, `PITR`, `RPO`, `RTO`, `Freeze`, `OLTP/OLAP`, `Snapshot`, `Réplication`, `` `Principe du moindre privilège ``

Revision #1
Created 6 November 2025 16:15:50 by qoyri
Updated 6 November 2025 16:20:21 by qoyri