Cours de bases de données relationnelles

BD et SGBD : Table des Matières

Introduction

Les bases de données sont actuellement au cœur du système d'information des entreprises. Les systèmes de gestion de bases de données, initialement disponibles uniquement sur des "mainframes", peuvent maintenant être installés sur tous les types d'ordinateurs y compris les ordinateurs personnels. Mais attention, souvent on désigne, par abus de langage, sous le nom "bases de données" des ensembles de données qui n'en sont pas.
Qu'est-ce donc qu'une base de données? Que peut-on attendre d'un système de gestion de bases de données? C'est à ces questions, entre autres, que cet ouvrage essaie d'apporter des réponses.
Dans un premier temps, et de façon informelle, on peut considérer une Base de Données (BD) comme une grande quantité de données, centralisées ou non, servant pour les besoins d'une ou plusieurs applications, interrogeables et modifiables par un groupe d'utilisateurs travaillant en parallèle. Quant au Système de Gestion de Bases de Données (SGBD), il peut être vu comme le logiciel qui prend en charge la structuration, le stockage, la mise à jour et la maintenance des données ; c'est, en fait, l'interface entre la base de données et les utilisateurs ou leurs programmes.

Les limites à l'utilisation des fichiers

L' utilisation de fichiers impose d'une part, à l'utilisateur de connaître l'organisation (séquentielle, indexée, ...) des fichiers qu'il utilise afin de pouvoir accéder aux informations dont il a besoin et, d'autre part, d'écrire des programmes pour pouvoir effectivement manipuler ces informations. Pour des applications nouvelles, l'utilisateur devra obligatoirement écrire de nouveaux programmes et il pourra être amené à créer de nouveaux fichiers qui contiendront peut-être des informations déjà présentes dans d'autres fichiers.
De telles applications sont :

- rigides,
- contraignantes,
- longues et coûteuses à mettre en œuvre.

Les données associées sont :

- mal définies et mal désignées,
- redondantes,
- peu accessibles de manière ponctuelle,
- peu fiables.

La prise de décision est une part importante de la vie d'une société. Mais elle nécessite d'être bien informé sur la situation et donc d'avoir des informations à jour et disponibles immédiatement.
Les utilisateurs, quant à eux, ne veulent plus de systèmes d'information constitués d'un ensemble de programmes inflexibles et de données inaccessibles à tout non spécialiste ; ils souhaitent des systèmes d'informations globaux, cohérents, directement accessibles (sans qu'ils aient besoin soit d'écrire des programmes soit de demander à un programmeur de les écrire pour eux) et des réponses immédiates aux questions qu'ils posent. On a donc recherché des solutions tenant compte à la fois des désirs des utilisateurs et des progrès techniques. Cette recherche a abouti au concept de base de données.
Définition (base de données) : Une base de données est un ensemble d'informations sur un sujet qui est :

- exhaustif,
- non redondant,
- structuré,
- persistant.

Définition (système de gestion de base de données) : Un système de gestion de base de données est un logiciel qui permet de :

- décrire,
- modifier,
- interroger,
- administrer,

les données d'une base de données.

Objectifs des systèmes de gestion de bases de données

Les bases de données et les systèmes de gestion de bases de données ont été créés pour répondre à un certain nombre de besoins et pour résoudre un certain nombre de problèmes.
Des objectifs principaux ont été fixés aux systèmes de gestion de bases de données dès l'origine de ceux-ci et ce, afin de résoudre les problèmes causés par la démarche classique.
Ces objectifs sont les suivants :

Il faut pouvoir accéder aux données sans savoir programmer ce qui signifie des langages "quasi naturels".

Efficacité des accès aux données

Ces langages doivent permettre d'obtenir des réponses aux interrogations en un temps "raisonnable". Ils doivent donc être optimisés et, entre autres, il faut un mécanisme permettant de minimiser le nombre d'accès disques. Tout ceci, bien sur, de façon complètement transparente pour l'utilisateur.

Administration centralisée des données

Des visions différentes des données (entre autres) se résolvent plus facilement si les données sont administrées de façon centralisée.

Non-redondance des données

Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.

Cohérence des données

Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Elles doivent pouvoir être exprimées simplement et vérifiées automatiquement à chaque insertion, modification ou suppression des données.

Partageabilité des données

Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment. Si ce problème est simple à résoudre quand il s'agit uniquement d'interrogations et quand on est dans un contexte mono-utilisateur, cela n'est plus le cas quand il s'agit de modifications dans un contexte multi-utilisateurs. Il s'agit alors de pouvoir :

- permettre à deux (ou plus) utilisateurs de modifier la même donnée "en même temps" ;
- assurer un résultat d'interrogation cohérent pour un utilisateur consultant une table pendant qu'un autre la modifie.

Sécurité des données

Les données doivent pouvoir être protégées contre les accès non autorisés. Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.

Résistance aux pannes

Que se passe-t-il si une panne survient au milieu d'une modification, si certains fichiers contenant les données deviennent illisibles? Les pannes, bien qu'étant assez rares, se produisent quand même de temps en temps. Il faut pouvoir, lorsque l'une d'elles arrive, récupérer une base dans un état "sain". Ainsi, après une panne intervenant au milieu d'une modification deux solutions sont possibles : soit récupérer les données dans l'état dans lequel elles étaient avant la modification, soit terminer l'opération interrompue.

Malheureusement, ces objectifs ne sont pas toujours atteints.

Concepts de base

Pour assurer ces objectifs (surtout les deux premiers), trois niveaux de description des données ont été définis par la norme ANSI/SPARC.

Niveau interne

> Description du stockage des données au niveau des unités de stockage, des fichiers, ... On appelle cette description le schéma interne.

Niveau conceptuel

> Description de la structure de toutes les données qui existent dans la base, description de leurs propriétés (relations qui existent entre elles) c'est-à-dire de leur sémantique inhérente, sans soucis d'implémentation physique ni de la façon dont chaque groupe de travail voudra s'en servir. On appelle cette description le schéma conceptuel.

Niveau externe

> Description pour chaque utilisateur de sa perception des données. On appelle cette description le schéma externe ou vue.

Le résultat de la conception d'une base de données sera une description des données. Par description on entend définir les propriétés d'ensembles d'objets modélisés dans la base de données et non pas d'objets particuliers. Les objets particuliers sont définis par les programmes d'applications lors des insertions et des mises à jour des données. Ils doivent alors vérifier les propriétés des ensembles auxquels ils appartiennent.
Cette description des données sera effectuée en utilisant un modèle de données. Ce dernier est un outil intellectuel utilisé pour comprendre l'organisation logique des données. C'est un ensemble de concepts et de règles pour les utiliser, permettant de construire avec des types de données une représentation de la réalité.
Un système de gestion de bases de données est caractérisé par le modèle de description des données qu'il supporte. Les données sont décrites sous la forme de ce modèle, grâce à un langage de description des données. Cette description est appelée schéma. Les modèles utilisés sont : réseau, relationnel, objet, ...
Une fois la base de données spécifiée, on peut y insérer des données, les récupérer, les modifier et les détruire. C'est ce qu'on appelle manipuler les données. Les données peuvent être manipulées non seulement par un langage spécifique de manipulation des données mais aussi par des langages de programmation "classiques".

Composants des systèmes de gestion de bases de données

Un système de gestion de bases de données va donc posséder un certain nombre de composants logiciels et ce, quel que soit le modèle de données qu'il supporte. On trouve donc des composants chargés de :

La description des données

> Cette partie sera constituée des outils (en gros des langages) permettant de décrire la vision des données de chaque utilisateur et l'intégration dans une vision globale. On y trouve aussi les outils permettant de décrire le stockage physique des données.

La récupération des données

> Cette partie prend en charge l'interrogation et la modification des données et ce, de façon optimisée. Elle est composée de langages de manipulation de données spécifiques et d'extensions de langages "classiques". Elle gère aussi les problèmes de sécurité.

La sauvegarde et la récupération après pannes

> Cette partie comporte des outils permettant de sauvegarder et de restaurer de façon explicite une base de données. Elle comporte aussi des mécanismes permettant, tant qu'une modification n'est pas finie, de pouvoir revenir à l'état de la base avant le début de

cette modification.

Les accès concurrents aux données

C'est la partie chargée du contrôle de la concurrence des accès aux données. Elle doit être telle que chaque utilisateur attende le moins possible ses données tout en étant certain d'obtenir des données cohérentes en cas de mises à jour simultanées de la base.

Chacun de ces composants est décrit de façon plus détaillée par la suite.

Un peu d'histoire

1960

Uniquement des systèmes de gestion de fichiers plus ou moins sophistiqués.

1970

Début des systèmes de gestion de bases de données réseaux et hiérarchiques proches des systèmes de gestion de fichiers. Ces systèmes de gestion de bases de données avaient rempli certains des objectifs précédents mais on ne pouvait pas interroger une base sans savoir où était l'information recherchée (on "naviguait") et sans écrire de programmes.

Sortie du papier de CODD sur la théorie des relations, fondement de la théorie des bases de données relationnelles.

1980

Les systèmes de gestion de bases de données relationnels apparaissent sur le marché.

1990

Les systèmes de gestion de bases de données relationnels dominent le marché.

Début des systèmes de gestion de bases de données orientés objet.

Le modèle relationnel

Le modèle relationnel a été formalisé par CODD en 1970. Quelques exemples de réalisation en sont : DB2(IBM), INFORMIX, INGRES, ORACLE.
Dans ce modèle, les données sont stockées dans des tables, sans préjuger de la façon dont les informations sont stockées dans la machine. Un ensemble de données sera donc modélisé par un ensemble de tables.
Le succès du modèle relationnel auprès des chercheurs, concepteurs et utilisateurs est dû à la puissance et à la simplicité de ses concepts. En outre, contrairement à certains autres modèles, il repose sur des bases théoriques solides, notamment la théorie des ensembles et la logique mathématique(théorie des prédicats d'ordre 1).
Les objectifs du modèle relationnel :

- proposer des schémas de données faciles à utiliser,
- améliorer l'indépendance logique et physique,
- mettre à la disposition des utilisateurs des langages de haut niveau pouvant éventuellement être utilisés par des non informaticiens,
- optimiser les accès à la base de données,
- améliorer l'intégrité et la confidentialité,
- fournir une approche méthodologique dans la construction des schémas.

De façon informelle, on peut définir le modèle relationnel de la manière suivante :

- Les données sont organisées sous forme de tables à deux dimensions, encore appelées relations et chaque ligne n-uplet ou tuple,
- les données sont manipulées par des opérateurs de l'algèbre relationnelle,
- l'état cohérent de la base est défini par un ensemble de contraintes d'intégrité.

Au modèle relationnel est associée la théorie de la normalisation des relations qui permet de se débarrasser des incohérences au moment de la conception d'une base de données.

Définitions

Définition (Domaine) : Ensemble de valeurs.
Définition (Relation) : Sous-ensemble du produit cartésien d'une liste de domaines caractérisé par un nom. En d'autres termes, une relation n'est ni plus ni moins qu'une table dans laquelle chaque colonne correspond à un domaine et porte un nom ce qui rend leur ordre sans aucune importance.
Définition (Attribut) : Colonne d'une relation caractérisée par un nom.
Définition (Schéma de relation) : Nom de la relation, suivi de la liste des attributs avec leurs domaines.
Définition (Base de données relationnelles) : Base de données dont le schéma est un ensemble de schémas de relations et dont les occurrences sont les tuples de ces relations.
Définition (Système de gestion de bases de données relationnel) : C'est un logiciel supportant le modèle relationnel, et qui peut manipuler les données avec des opérateurs relationnels.

Opérateurs relationnels

Définition (Projection) : Opération qui consiste à supprimer des attributs d'une relation et à éliminer les tuples en double apparaissant dans la nouvelle relation. Cette opération est notée Π.
Définition (Restriction) : Opération qui consiste à supprimer les tuples d'une relation ne satisfaisant pas la condition précisée.
Définition (Jointure) : Opération qui consiste à faire le produit cartésien de deux relations, puis à supprimer les tuples ne satisfaisant pas une condition portant sur un attribut de la première relation et sur un attribut de la seconde.
Définition (Union) : Opération portant sur deux relations ayant le même schéma et construisant une troisième relation constituée des tuples appartenant à chaque relation. Les tuples en double sont éliminés.
Définition (Différence relationnelle) : Opération portant sur deux relations ayant le même schéma et construisant une troisième relation dont les tuples sont constitués de ceux ne se trouvant que dans une seule relation.
Définition (Intersection) : Opération portant sur deux relations ayant le même schéma et construisant une troisième relation dont les tuples sont constitués de ceux appartenant aux deux relations.

Formes normales

Dépendance fonctionnelle

Définition (dépendance fonctionnelle) : Soit $R(A_1,A_2,...A_n)$ un schéma de relation, et X et Y des sous-ensembles de $\{A_1,A_2,...A_n\}$. On dit que X détermine Y ou que Y dépend fonctionnellement de X si, et seulement si, des valeurs identiques de X impliquent des valeurs identiques de Y. On le note : X -> Y
Définition (dépendance fonctionnelle élémentaire) : C'est une dépendance fonctionnelle de la forme X -> Y, où A est un attribut unique n'appartenant pas à X et où il n'existe pas X' inclus dans X tel que X -> Y.

Notion de clé

Définition (clé de relation) : Soit R(A_1, A_2, ..., A_n) un schéma de relation, et X un sous-ensemble de (A_1, A_2, ..., A_n), X est une clé si, et seulement si, :

- X -> (A_1, A_2, ..., A_n)
- X est minimal

Formes normales

Définition (Première forme normale) : Une relation est en première forme normale si et seulement si tout attribut contient une valeur atomique.
Définition (Deuxième forme normale) : Une relation est en deuxième forme normale si et seulement si :

- elle est en première forme normale ;
- tout attribut n'appartenant pas à une clé ne dépend pas que d'une partie de cette clé.

Définition (Troisième forme normale) : Une relation est en troisième forme normale si et seulement si :

- elle est en deuxième forme normale ;
- tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non-clé.

Définition (Forme normale de BOYCE-CODD) : Une relation est en Forme normale de BOYCE-CODD (BCNF) si, et seulement si, les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clé détermine un attribut.

Langages de manipulation de données relationnelles

Ces langages, dits assertionnels, sont basés sur la logique des prédicats d'ordre 1 et permettent de spécifier les données que l'on souhaite obtenir, sans dire comment y accéder. On doit y trouver des opérations permettant de : [la modification]

la recherche

: retrouver des tuples vérifiant certains critères,

l'insertion

: ajouter des tuples,

la suppression

: enlever des tuples vérifiant certains critères,

la modification

: modifier des tuples vérifiant certains critères.

Un langage de manipulation de données n'est pas utilisable à lui seul, il doit aussi pouvoir être incorporable dans un langage de programmation classique.
On peut distinguer trois grandes classes de langages :

- Les langages algébriques basés sur l'algèbre relationnelle de CODD dans lesquels les requêtes sont exprimées comme l'application des opérateurs relationnels sur des relations. C'est dans cette catégorie que l'on trouve le langage sql(structured query language), standard pour l'interrogation de bases de données.
- Les langages basés sur le calcul relationnel de tuples construits à partir de la logique des prédicats dans lesquels les variables manipulées sont des tuples.
- Les langages basés sur le calcul relationnel de domaines, construit aussi à partir de la logique des prédicats mais en faisant varier les variables sur les domaines des relations.

Le langage sql (Structured Query Language) comprend à lui seul l'ensemble des instructions nécessaires à la spécification et à l'utilisation d'une base de données relationnelle. C'est un langage de type déclaratif c'est-à-dire que l'on spécifie les propriétés des données que l'on recherche et pas, comme dans un langage impératif, comment les retrouver.

Le langage sql est un langage normalisé, la dernière version de la norme date de 92 et, souvent, on y fait référence en parlant de sql-92. La prochaine version de la norme est en cours de rédaction afin d'intégrer, entre autres, la notion de types abstraits algébriques, on la désigne sous le nom de sql3.
C'est à la fois :

- un langage d'interrogation de données (LID) : `SELECT` ;
- un langage de manipulation de données (LMD) : `UPDATE`, `INSERT`, `DELETE` ;
- un langage de définition des données (LDD) : `ALTER`, `CREATE`, `DROP`;
- un langage de contrôle des données et des utilisateurs (LCD) : `GRANT`, `REVOKE`.

Remarques

Beaucoup de systèmes de gestion de données (et non pas de gestion de bases de données) sont vendus comme étant relationnels, souvent parce qu'ils présentent les données sous forme de tables.
Un système est dit minimalement relationnel s'il satisfait aux conditions suivantes :

- toute information dans la base est représentée par des valeurs dans des tables,
- il n'y a pas de pointeurs visibles par l'utilisateur entre les tables,
- le système doit supporter au moins les opérateurs relationnels de restriction, projection, jointure naturelle.

Un système est dit complètement relationnel s'il satisfait, en plus, aux conditions suivantes :

- il supporte tous les opérateurs de l'algèbre relationnelle,
- il supporte la contrainte d'unicité de clé d'une relation,
- il supporte les contraintes référentielles qui permettent de s'assurer que la valeur d'une donnée d'une relation existe dans une autre relation (notion de foreign key).

En dépit de sa simplicité et de son élégance le modèle relationnel n'apporte pas une réponse satisfaisante à tous les problèmes des applications. Il faut :

- Pouvoir prendre en compte des "objets" structurés ainsi que les opérations qui leur sont associées (bases de données orientées objet).
- Prendre en compte des données peu structurées : textes, sons, images, graphiques (bases de données multimédia).
- Faire le pont avec l'intelligence artificielle afin de pouvoir déduire de nouvelles données à partir de celles existant déjà (bases de données déductives)

L'optimiseur de requêtes

Avec des langages tels que sql, l'utilisateur précise les propriétés des données qui l'intéressent sans fournir d'algorithme d'accès. Les optimiseurs de requête sont là pour cela, leurs objectifs sont de :

- vérifier la correction syntaxique de la question,
- rechercher des questions équivalentes plus simples,
- déterminer l'ordre des opérations élémentaires d'accès en vue de :
    - réduire le nombre d'entrées/sorties,
    - réduire le temps cpu,
    - réduire la taille des ressources mémoires nécessaires,
    - optimiser en premier les questions les plus fréquentes.

Chaque sgbd contient un optimiseur de requêtes qui peut être légèrement différent d'un sgbd à l'autre. Les algorithmes utilisés sont rarement connus en détail mais les grandes étapes de l'optimisation sont en général :

1. Représenter la requête sous une forme interne et la décomposer en une séquence d'opérations élémentaires.
2. Transformer la requête par :
    - simplification : remplacement d'une question par une question équivalente plus simple,
    - ordonnancement des opérations élémentaires.
3. Construire un ensemble de plans d'exécution candidats.
4. Calculer le coût de chaque plan et choisir le meilleur.
5. Exécuter la requête.

Réécriture des requêtes

La reformulation de la requête par l'optimiseur a pour but de minimiser le nombre de calculs effectués. Voici quelques exemples de transformations possibles :

- Les expressions et conditions faisant intervenir des constantes sont, si c'est possible, évaluées. Ainsi, `âge > 60 - 10` sera transformé en `âge > 50` mais `âge + 10 > 60` ne sera pas réécrit en `âge > 50`.
- un `BETWEEN` sera remplacé par une expression contenant les symboles >= et <= séparés par `AND`.
- Une expression logique complexe préfacée par `NOT` sera réécrite afin que celui-ci soit mis devant chacune de ses sous-expressions. Le `NOT` devant une expression simple sera, si c'est possible, supprimé. Ainsi `NOT âge > 50` deviendra `âge <= 50`.
- Les requêtes comportant des `OR` sont réécrites en utilisant des `UNION`.
- Les sous-requêtes sont remplacées par des jointures.

Choix des chemins d'accès

L'optimiseur d'interrogation d'oracle est chargé de déterminer le chemin d'accès optimal pour exécuter un `SELECT`.
Le choix porte essentiellement sur l'utilisation des index portant sur les colonnes mentionnées dans la clause `WHERE`, sachant qu'en l'absence d'index utilisable une interrogation portant sur une table nécessitera le balayage total de la table.

Requête portant sur une seule table

L'optimiseur d'interrogation classe les différents types de prédicats en fonction de leur syntaxe dans l'ordre suivant ; du plus sélectif au moins sélectif :

- `WHERE rowid` = constante
- `WHERE` clé de cluster = constante
- `WHERE` colonne de `CONNECT BY` indexée = constante
- `WHERE` colonne indexée (unique) = constante
- `WHERE` colonne indexée (non unique) = constante
- `WHERE` colonne indexée = constante
- `WHERE` colonne indexée `BETWEEN` val1 `AND` val2 ou `WHERE` colonne indexée `LIKE` 'c%'
- `WHERE` colonne indexée > ou < constante
- `WHERE MAX` ou `MIN` de colonne indexée
- `WHERE` colonne non indexée = constante
- `WHERE` colonne is null ou is not null
- `WHERE` colonne ! = constante
- `WHERE` colonne indexée `LIKE '%c'` où `WHERE` colonne indexée `LIKE '_c'`

oracle peut dans certains cas utiliser plusieurs index sur une même table (5 au max), y compris pour évaluer des prédicats lies par `OR`.
Un `SELECT` avec plusieurs prédicats indexés liés par `AND` sera exécuté comme une intersection du résultat de plusieurs `SELECT` bénéficiant chacun d'un index, si les prédicats sont des égalités.

Exemple :

```
SELECT *
FROM emp
WHERE nom = 'MARTIN'
AND n_dept = 20;
```

De même un `SELECT` avec plusieurs prédicats liés par `OR` sera exécuté comme une union du résultat de plusieurs `SELECT` bénéficiant chacun d'un index. Dans ce cas, il faut que tous les prédicats bénéficient d'un index.

Exemple :

```
SELECT *
FROM emp
WHERE nom = 'MARTIN'
OR n_dept = 20 ;
```

Dans ces cas il se peut que l'utilisation de l'un des index soit pénalisante. Ce peut être le cas d'un index peu sélectif, alors que les autres index utilisables sont très sélectifs. Dans ce cas l'on peut empêcher oracle d'utiliser les index inadéquats, en formulant les conditions de telle façon que le critère de recherche soit une fonction de la colonne indexée et non pas la colonne indexée elle-même (dans ce cas oracle n'utilise pas l'index).

Exemple :

```
SELECT *
FROM emp
WHERE nom = 'MARTIN'
AND n_dept + 0 = 20;
```

où les champs `nom` et `n_dept` sont indexés. Le fait d'ajouter 0 à la colonne `n_dept` empêche d'utiliser l'index sur `n_dept`, qui ne ferait que ralentir la requête car il est peu sélectif. L'index sur `nom`, qui est très sélectif, suffit.

De la même façon, l'on peut concaténer une chaîne vide à une colonne de type caractère pour empêcher oracle d'utiliser l'index sur cette colonne :

```
SELECT *
FROM emp
WHERE nom = 'MARTIN'
AND fonction ||''= 'directeur' ;
```

Jointures sans index

Dans le cas de jointure sans index, oracle classe au préalable chaque table selon le critère de jointure. L'algorithme est symétrique et l'ordre dans lequel les tables sont mentionnées n'a donc pas d'influence sur les performances.

Jointures avec index

Dans le cas où le critère de jointure est indexé au moins dans l'une des tables, oracle choisit une table directrice : c'est cette table qui sera lue en premier et qui pilotera la jointure. Si le critère de jointure est indexé dans une seule des tables, oracle choisit l'autre table comme table directrice. Si le critère de jointure est indexé dans les deux tables, la table directrice est celle sur laquelle portent les prédicats les moins sélectifs(compte tenu de la classification des prédicats ci-dessus). Si les prédicats portant sur chaque table sont équivalents, la table directrice sera la dernière table mentionnée dans la clause `FROM`.

Ainsi, dans l'exemple suivant, l'optimiseur considère que les restrictions portant sur chacune des tables sont équivalentes, et la table directrice sera la table `dept` qui est citée en dernier dans la clause `FROM`.

```
SELECT *
FROM emp, dept
WHERE emp.n_dept = dept.n_dept
AND dept.nom = 'vente'
AND fonction ='directeur' ;
```

`ORDER BY`

L'optimiseur d'interrogation d'oracle décidera dans certains cas de passer par l'index pour sélectionner les lignes d'une table selon un certain ordre. Pour cela il faut que :

- le critère de classement soit le contenu d'une colonne indexée
- cette colonne ait l'attribut not null (obligatoire)
- la sélection porte sur toute la table (pas de `WHERE`)

Exemple : Le `SELECT` suivant utilisera l'index sur la colonne `salaire` si cette colonne ne contient pas de valeurs `NULL`.

```
SELECT nom, salaire
FROM emp
ORDER BY salaire ;
```

Cohérence des interrogations et accès concurrents

Ce chapitre étudie les conditions de bon fonctionnement d'un système de gestion de bases de données travaillant dans un contexte multi-utilisateurs multi-tâches. Comme un système d'exploitation classique gère, partage les ressources et synchronise les processus, un système de gestion de bases de données doit assurer (entre autres) le partage des données. On retrouve, ici, les problèmes classiques de gestion de ressources partagées.
Les problèmes à éviter se classent, en gros, en deux catégories : les pertes d'opérations et la récupération d'une bases de données incohérente. De façon plus détaillée on peut avoir des :

- lectures inconsistantes
- lectures non reproductibles
- des lectures de données fantômes
- des modifications perdues
- des données incohérentes
- des accès à des données inexistantes

Il est facile de réaliser un système de gestion de bases de données avec un contrôle de concurrence simple et efficace consistant, par exemple, à verrouiller la base entière pendant l'exécution de chaque transaction. Les performances d'un tel système se dégraderaient rapidement avec l'augmentation du nombre d'usagers et il deviendrait très rapidement inutilisable.
Un système réaliste doit assurer que :

- lectures et écritures ne mettant pas en jeu les mêmes données peuvent s'exécuter en parallèle ;
- les lectures ne sont pas en attente de fin d'écriture ou de fin de lecture des mêmes données ;
- les écritures ne sont pas en attente de fin de lecture des mêmes données ;
- les écritures attendent uniquement la fin d'écriture des mêmes données.

Il faut trouver une solution qui préserve l'intégrité des données sans pénaliser les performances. Cette solution repose sur les notions de :

- fichier image avant, "before image" ou "rollback segment", tous ces mots désignant un, ou plusieurs, fichiers contenant la valeur des données avant la modification ainsi que celle-ci.
- base cohérente et de transaction (ces deux notions sont définies ci-dessous).

Définition (contrainte d'intégrité) : assertion(propriété, prédicat) qui doit être vérifiée par certaines données à des instants déterminés.
Définition (bases de données cohérente) : bases de données dont toutes les contraintes d'intégrité sont respectées.
Les contraintes d'intégrité peuvent non seulement porter sur une donnée unique : contrainte de domaine de variation, contrainte de plage de valeurs, mais aussi sur des données référençant d'autres données .
Définition (transaction) : unité de traitement séquentiel exécutée pour le compte d'un usager qui, appliquée à une base cohérente restitue une base cohérente.

Interrogation

Cohérence d'une interrogation

Un utilisateur qui interroge une table (même très grande) est garanti de voir toutes les données telles qu'elles étaient au moment du début de l'interrogation, même si d'autres utilisateurs modifient la table et valident leurs modifications pendant ce temps.
Les sgbd dont oracle utilisent alors le fichier image avant pour assurer cette cohérence. Le `COMMIT` des utilisateurs modifiant la table n'est pas différé à la fin de l'interrogation.
Remarque : Dès que l'on interroge une table, un verrou est placé sur la définition de la table, c'est à dire qu'un autre utilisateur ne peut pas détruire la table, l'indexer, la mettre en cluster ou modifier sa définition, jusqu'à ce que l'interrogation soit terminée.

Cohérence de plusieurs interrogations successives

Si l'utilisateur désire que l'on ne modifie pas une table pendant une session de travail, celui-ci peut verrouiller la table en mode partagé au moyen de l'ordre sql suivant :

```
LOCK TABLE  nom_table IN SHARE MODE NOWAIT;
```

où l'option `NOWAIT`, qui peut s'adjoindre à toutes les commandes de verrouillage, spécifie que le process qui demande le verrou n'est pas mis en attente si celui-ci n'est pas disponible.
La table n'est alors accessible aux autres utilisateurs qu'en lecture jusqu'à la fin de la transaction de celui qui l'a verrouillée (les autres utilisateurs peuvent aussi verrouiller la table en share mode).
Les modifications des autres utilisateurs seront suspendues.

Mise à jour

Pour s'assurer l'accès exclusif en modification à une table, l'on peut verrouiller cette table en mode exclusif par la commande :

```
LOCK TABLE  nom_table IN EXCLUSIVE MODE NOWAIT;
```

La table n'est alors accessible aux autres utilisateurs qu'en lecture et ils ne peuvent plus la verrouiller en mode exclusif, ni en mode mise à jour partagée, ni en mode partagé jusqu'à la fin de la transaction. Ce verrou est également obtenu automatiquement dès que l'on effectue un `UPDATE`, `INSERT` ou `DELETE` sur une table. C'est le mode par défaut de mise à jour d'une table.

Contrôle des accès à la base et sécurité des données

Les systèmes de gestion de bases de données permettent à plusieurs utilisateurs de travailler en toute sécurité sur la même base ou sur des bases différentes.
Chaque donnée peut être définie, soit comme confidentielle et accessible à un seul utilisateur, soit comme étant partageable entre plusieurs utilisateurs.
Les ordres `GRANT` et `REVOKE` du langage sql permettent de définir les droits de chaque utilisateur sur les objets de la base.
Tout utilisateur doit posséder un nom d'utilisateur et un mot de passe pour pouvoir accéder à la base. C'est ce nom d'utilisateur qui déterminera les droits d'accès aux objets de la base.

- Droits d'accès aux tables

Droits d'accès aux tables

La protection des objets d'une base de données est décentralisée : c'est le créateur d'un objet qui possède tous les droits de lecture, de modification et de suppression de cet objet. Les autres utilisateurs n'ont aucun droit d'accès à cet objet, à moins que le créateur ne les leur accorde explicitement par une commande `GRANT` :

```
GRANT privilege ON {nom_table | nom_vue} TO nom_utilisateur
WITH GRANT OPTION ;
```

Les privilèges pouvant être accordés sont entre autres les suivants : [update(col,...)]

`SELECT`

consultation

`INSERT`

rajout des lignes

`UPDATE(col,...)`

modification des lignes (peut-être restreint à certaines colonnes)

`DELETE`

suppression des lignes

`ALTER`

modification de la définition de la table

`ALL`

tous les droits ci-dessus

Un utilisateur ayant reçu un privilège avec l'option `GRANT` peut le transmettre à son tour (`WITH GRANT OPTION`).
Exemple : L'utilisateur `scott` peut autoriser l'utilisateur `douglas` à lire sa table `emp`.

```
GRANT SELECT ON emp TO douglas ;
```

Les droits peuvent être accordés à tous les utilisateurs en employant le mot réservé `PUBLIC` à la place du nom d'utilisateur.

```
GRANT SELECT, UPDATE ON emp TO PUBLIC;
```

Un utilisateur ayant accordé un privilège peut le reprendre à l'aide de l'ordre `REVOKE`.

```
REVOKE PRIVILèGE ON {nom_table | nom_vue} FROM  nom_utilisateur;
```

Le propriétaire d'une table peut donner des droits de lecture non pas sur la table entière, mais sur une vue basée sur la table. Ainsi, seules les informations faisant partie de la vue seront accessibles.

```
CREATE VIEW emp1 AS
SELECT nom, fonction, embauche
FROM scott.emp
WHERE n_dept != 10 ;
```

```
GRANT SELECT ON emp1 TO PUBLIC;
```

Tous les utilisateurs pourront sélectionner les employés à travers la vue `emp1` : ils ne verront que les colonnes `nom`, `fonction`, `embauche` de la table `emp` et n'auront pas accès aux employés du département 10. Avec l'option de contrôle (`CHECK OPTION`), les vues peuvent servir à réaliser les contrôles lors des mises à jour.
Exemple : La vue suivante ne permettrait pas d'insérer dans la table des employés `emp` un employé dont le numéro de département ne figurerait pas dans la table des départements `dept` :

```
CREATE VIEW majemp AS
SELECT *
FROM emp
WHERE n_dept IN (SELECT n_dept
        FROM dept)
WITH CHECK OPTION ;
```

Comme il est impossible de faire une mise à jour sur une vue comportant une jointure il faut transformer le critère de jointure en une sous-interrogation.
Pour oracle , le nom complet d'une table ou d'une vue est le nom donné par le créateur, préfixé par le nom du créateur. Par exemple `scott.emp` est le nom complet de la table `emp` créée par l'utilisateur `scott` . Ceci permet à plusieurs utilisateurs de créer des objets de même nom sans qu'il y ait confusion. En revanche, pour accéder à un objet dont on n'est pas le créateur, il faut le désigner par son nom complet incluant le nom du créateur.

Stockage des données

La taille des données stockées dans une bases de données va de l'ordre de plusieurs centaines de Mega-octets pour des bases moyennes à des dizaines ou des centaines de Giga-octets pour des bases importantes. La plus grosse base connue à ce jour atteint le Téra-octet.
Ces donnés, bien évidemment, vont être stockées sur un ou plusieurs disques de façon complètement transparente pour l'utilisateur final. Les temps d'accès disques étant très pénalisants, il va falloir essayer de les minimiser.
Il existe deux possibilités liées au stockage des données de minimiser les accès disques :

Les index

Selon le modèle relationnel les sélections peuvent être faites en utilisant le contenu de n'importe quelle colonne et les lignes sont stockées dans n'importe quel ordre.
Considérons le `SELECT` suivant :

```
SELECT *
FROM emp
WHERE nom = 'MARTIN'
```

Un moyen de retrouver la ou les lignes pour lesquelles `nom` est égal à `MARTIN` est de balayer toute la table.
Un tel moyen d'accès conduit à des temps de réponse prohibitifs pour des tables dépassant quelques centaines de lignes.
Une solution offerte par tous les systèmes de gestion de bases de données est la création d'index, qui permettra de satisfaire aux requêtes les plus fréquentes avec des temps de réponse acceptables.
Un index sera matérialisé par la création de blocs disque contenant des couples (valeurs d'index, numéro de bloc) donnant le numéro de bloc disque dans lequel se trouvent les lignes correspondant à chaque valeur d'index.

Utilisation des index

L'adjonction d'un index à une table ralentit les mises à jour (insertion, suppression, modification de la clé) mais accélère beaucoup la recherche d'une ligne dans la table.
L'index accélère la recherche d'une ligne à partir d'une valeur donnée de clé, mais aussi la recherche des lignes ayant une valeur d'index supérieure ou inférieure à une valeur donnée, car les valeurs de clés sont triées dans l'index.
Exemple : Les requêtes suivantes bénéficieront d'un index sur le champ `n_dept`.

```
SELECT * FROM emp WHERE num = 16034  ;
SELECT * FROM emp WHERE num >= 27234 ;
SELECT * FROM emp WHERE num BETWEEN 16034  AND 27234;
```

Un index est utilisable même si le critère de recherche est constitué seulement du début de la clé.
Exemple : La requête suivante bénéficiera d'un index sur la colonne `nom`.

```
SELECT *
FROM emp
WHERE nom LIKE 'M
```

Par contre si le début de la clé n'est pas connu, l'index est inutilisable.
Exemple : La requête suivante ne bénéficiera pas d'un index sur le champ `nom`.

```
SELECT *
FROM emp
WHERE ename LIKE '
```

Valeurs `NULL`

Elles ne sont pas représentées dans l'index, ceci afin de minimiser le volume nécessaire pour stocker l'index. En contrepartie, l'index ne sera d'aucune utilité pour retrouver les valeurs `NULL` lorsque le critère de recherche est du type `IS NULL`.

Conversions

L'index n'est utilisable que si le critère de sélection est le contenu de la colonne indexée, sans aucune transformation. Par exemple un index sur `salaire` ne sera pas utilisé pour la requête suivante :

```
SELECT * FROM emp
WHERE salaire * 12 > 300000 ;
```

Attention en particulier aux conversions de type qui peuvent empêcher l'utilisation de l'index.
sql est un langage typé, chaque type de données (numérique, caractère, date) ayant ses propres opérateurs, ses propres fonctions et sa propre relation d'ordre. En conséquence, si dans une expression, figurent à la fois un nombre et une chaîne de caractères, sql convertira la chaîne de caractères en nombre. De même si dans une expression, figurent à la fois une chaîne de caractères et une date, sql convertira la chaîne de caractères en date.
Or, dans un prédicat du type :

```
WHERE fonction(col_indexée) = constante
```

sql ne peut pas utiliser l'index.
Ceci peut se produire , de façon insidieuse, lorsque sql est obligé d'ajouter un appel à une fonction de conversion à cause d'une discordance de type.
Exemple : Le prédicat suivant ne bénéficiera pas d'un index sur le champ `embauche`.

```
SELECT * FROM emp
WHERE embauche LIKE '
```

En effet, sql est obligé d'effectuer une conversion, et le prédicat qui sera évalué est :

```
WHERE TO_CHAR(embauche) LIKE '
```

Le critère de recherche est une fonction de `embauche`, et non le champ `embauche` lui-même, dans ce cas l'index est inutilisable.

Choix des index

Indexer en priorité :

1. les clés primaires
2. les colonnes servant de critère de jointure
3. les colonnes servant souvent de critère de recherche

Ne pas indexer :

1. les colonnes contenant peu de valeurs distinctes (index alors peu efficace)
2. les colonnes fréquemment modifiées

Index comprimé et non comprimé

Les clés dans les index peuvent être comprimées ou non. La compression est une technique permettant de réduire dans des proportions très importantes (d'autant plus que la clé est longue) le volume de l'index.
En contrepartie, il faut parfois un traitement supplémentaire pour recomposer la clé lors des mises à jour de l'index.
Par défaut, les index sont comprimés, les avantages de réduction de taille l'emportant sur les inconvénients dans la plupart des cas.
sql sait exécuter certaines requêtes directement au niveau de l'index sans passer par le segment de données, si l'index est non comprimé et si tous les champs résultats de la requête sont dans l'index.
Exemple : L'index crée par :

```
CREATE INDEX x
ON emp (num, nom)
nocompress ;
```

permettra de répondre à la question :

```
SELECT nom
FROM emp
WHERE num > 17217 ;
```

sans lire la table puisque toutes les informations se trouvent dans l'index et que l'index est non concaténé.

Index concaténé

Un index concaténé est un index portant sur plusieurs colonnes.
Exemple :

```
CREATE INDEX xemp
ON (n_dept,num) ;
```

Les index concaténés peuvent être utilisés pour matérialiser une clé composée de plusieurs colonnes.
sql sait utiliser un index concaténé même si le critère de recherche ne porte pas sur toutes les colonnes présentes dans l'index.
Exemple : L'index ci-dessus est utilisable si l'on ne connaît que le numéro de département.

```
SELECT nom
FROM emp
WHERE n_dept = 20 ;
```

Les clusters

Buts

Le cluster est une organisation physique des données qui consiste à regrouper physiquement (dans un même bloc disque) les lignes d'une ou plusieurs tables ayant une caractéristique commune (une même valeur dans une ou plusieurs colonnes) constituant la clé du cluster.
La mise en cluster a trois objectifs :

- accélérer la jointure selon la clé de cluster des tables mises en cluster,
- accélérer la sélection des lignes d'une table ayant même valeur de clé, par le fait que ces lignes sont regroupées physiquement,
- économiser de la place, du fait que chaque valeur de la clé du cluster ne sera stockée qu'une seule fois.

Le regroupement en cluster est totalement transparent à l'utilisateur : des tables mises en cluster sont toujours vues comme des tables indépendantes.
Par exemple on pourrait mettre en cluster les tables `emp` et `dept` selon `n_dept`. Ces tables seraient réorganisées de la façon suivante : un bloc de cluster serait créé pour chaque numéro de département, ce bloc contenant à la fois les lignes de la table `emp` et de la table `dept` correspondant à ce numéro de département. La jointure entre les tables `emp` et `dept` selon `n_dept` deviendrait alors beaucoup plus rapide, puisqu'elle serait déjà réalisée dans l'organisation physique des tables.
Pour que l'on puisse mettre une table en cluster il faut que l'une au moins des colonnes faisant partie du cluster soit définie comme obligatoire (`NOT NULL`).
On peut indexer les colonnes d'une table en cluster, y compris les colonnes correspondant à la clé ou à une partie de la clé du cluster. La clé elle-même est automatiquement indexée, on peut éventuellement la réindexer pour créer un index unique servant à contrôler son unicité.