Système de Gestion de Base de Données - 3IF

Victor Lezaud

16 janvier 2018

Sommaire

Première partie
SQL

1 Gestion des tables

La gestion des schémas de base de données se fait en SQL via les trois opérations ci-dessous :

- Définition du schéma d'une relation `CREATE TABLE`
- Modification du schéma de relation `ALTER TABLE`
- Suppression d'une relation et de son schéma `DROP TABLE`

1.1 CREATE TABLE

1.1.1 Définition des attributs d'une table

On liste les attributs avec la syntaxe suivante `<nom> <type>,` entre les parenthèses de l'opération `CREATE TABLE`.

```
CREATE TABLE Licencie (
    Nom varchar2(15),
    Prenom varchar2(15),
    DateNais date,
    Genre char(1),
    Numlic number(5),
    Niveau varchar(12),
);
```

1.1.2 Créer une table avec une requête

Il est aussi possible de créer la table à partir d'une requête SFW avec la syntaxe suivante :

```
CREATE TABLE <table>
AS
    SELECT
    FROM
    WHERE
```

1.1.3 Contrainte

La force de SQL dans la définition de schéma de données réside dans la déclaration de différant type de contrainte qui permettent de restreindre les modifications des tables afin que celles-ci respectent toujours les dépendances fonctionnelles et une certaine logique (age $>$ 0 par exemple). Nous allons voir comment déclarer ces contraintes.

La clé primaire est une des clés de la table (voir cours MdD) qui a été choisi en fonction de son usage et de sa taille. Exemple : `NumLic` est un excellent choix de clé primaire.

Déclaration d'une clé primaire :

```
[Constraint <NomContrainte>] PRIMARY KEY (<attr1>[,<attr2>]*)
```

```
CREATE TABLE Licencie (
    [...]
    Constraint ClePrimaire PRIMARY KEY (NumLic)
);
```

Déclaration d'une clé étrangère : Pour la définition voir cours MdD. Exemple d'implémentation : l'attribut NumLic de la table Inscription est une clé étrangère référençant l'attribut NumLic de la table Licencie.

```
[Constraint <NomContrainte>] FOREIGN KEY (<attr1>[,<attr2>]*)
                REFERENCES <table>(<attr1>[,<attr2>]*)
```

```
CREATE TABLE Inscription (
    [...]
    Constraint CleEtrangere FOREIGN KEY (NumLic)
    REFERENCES Licencie(NumLic)
);
```

Autres contraintes :

- `NOT NULL`: L'attribut doit être renseigné.
- `UNIQUE` : L'ensemble d'attributs n'a que des valeurs uniques.
- `CHECK` : Contrôle des valeurs que peut prendre l'attribut.
- `DEFAULT` : L'attribut prend la valeur par défaut s'il n'est pas initialisé explicitement (mal placé : plus une technique d'initialisation qu'une contrainte)

Deux méthodes d'implémentation :

- `<AttributeName> <Type> [CONSTRAINT <nom>] CHECK (<condition>):`
- `[CONSTRAINT <nom>] CHECK (<condition>)`

1.2 DROP TABLE

1.2.1 Instruction DROP TABLE

DROP TABLE permet de supprimer une table ainsi que son schéma. Mais qu'en est-il des contraintes ? ? En effet la contrainte de clé étrangère notamment crée des contraintes entre les tables.

```
DROP TABLE <nomTable>[,<nomTable>]*;
```

1.2.2 ON CASCADE

ON CASCADE permet de régler le comportement de la base de données lors de la suppression de table concernant cette contrainte. Pour les clés étrangères par exemples, un attribut d'une table primaire référence un attribut d'une table secondaire. La suppression de la table primaire implique donc une modification de la table secondaire.
Il existe `ON DELETE CASCADE` qui supprime les tuples contenant la valeur supprimé dans la table primaire et `ON DELETE SET NULL` met la valeur à `NULL`.

```
[Constraint <NomContrainte>] FOREIGN KEY (<attr1>[,<attr2>]*)
REFERENCES <table>(<attr1>[,<attr2>]*) ON DELETE {SET NULL|CASCADE}

CREATE TABLE Inscription (
    [...]
    Constraint CleEtrangere FOREIGN KEY (NumLic)
    REFERENCES Licencie(NumLic) ON DELETE {SET NULL|CASCADE}
);
```

1.2.3 CASCADE CONSTRAINTS

CASCADE CONSTRAINTS permet de tout simplement supprimer les contraintes qui peuvent être invalidées par la suppression de la table

```
DROP TABLE <nomTable>[,<nomTable>]* [CASCADE CONSTRAINTS];
```

1.2.4 PURGE

PURGE libère instantanément l'espace de stockage occupé par la table.

```
DROP TABLE <nomTable>[,<nomTable>]* [CASCADE CONSTRAINTS] [PURGE];
```

1.3 ALTER TABLE

1.3.1 ADD

Ajoute un nouvel attribut

```
ALTER TABLE <nomTable> ADD <DefAttribut>
```

1.3.2 DROP COLUMN

Supprime un attribut

```
ALTER TABLE <nomTable> DROP COLUMN <DefAttribut>
```

1.3.3 RENAME COLUMN

Redéfinie le nom d'une colonne

```
ALTER TABLE <nomTable> RENAME COLUMN <nomActuel> TO <nouveauNom>
```

1.3.4 ADD CONSTRAINT :

Ajoute une contrainte à la table

```
ALTER TABLE <nomTable> ADD CONSTRAINT <DefContrainte>
```

1.3.5 DROP CONSTRAINT

Supprime une constraint

```
ALTER TABLE <nomTable> DROP CONSTRAINT <NomContrainte>
```

2 Gestion des tuples

2.1 Insertion de tuples

Il existe trois manières d'insérer des tuples dans une table. On commence toujours par `INSERT INTO <table>`

2.1.1 Technique de base

On ajoute le tuple avec les attributs dans le même ordre que la définition de la table.

```
INSERT INTO <table>
VALUES (V1, V2, ..., Vn);
```

2.1.2 En modifiant l'ordre

Comme la première sauf qu'on précise d'abord l'ordre dans lequel on donne les attributs

```
INSERT INTO <table> (Attr3, Attr1, ... Attrn)
VALUES (V3, V1, ..., Vn);
```

2.1.3 A partir d'une requête SFW :

On récupère directement les résultats d'une requête pour les insérer dans la table

```
INSERT INTO <table>
     SELECT
     FROM
     WHERE
```

2.1.4 Suppression de tuples

On peut supprimer les tuples d'une table avec la syntaxe ci-dessous. Il faut faire attention à garder la cohérence de toute la base de données quand on supprime des données!!

```
DELETE FROM <table>
WHERE <condition>
```

2.1.5 Modification de tuples

De même on peut modifier la valeur de certains attributs.

```
UPDATE <table>
SET <attr> = {<espression>|DEFAULT} [...]*
[WHERE <condition>]
```

2.2 Gestion de transaction

2.2.1 Une transaction

Une transaction est une étape de travail, c'est la liste des modifications qui ont été apportées depuis la fin de la précédente.

2.2.2 Fin de transaction

Tant que la transaction n'est pas finie les modifications de tuples ne sont pas apportés à la Base de Données mais seulement à la session de travail. Il y a deux mot-clés pour terminer une transaction :

- `COMMIT` : valide le travail qui a été effectué et apporte les modifications à la base de données
- `ROLLBACK` : annule tout ce qui a été dans cette transaction

2.2.3 Sauvegardes

Il est possible de créer des points de retour `SAVEPOINT <id>`, afin de revenir en arrière avec un `ROLLBACK TO <id>`

3 Requête d'Interrogation

3.1 La base : SFW

3.1.1 SELECT

Comme son nom l'indique permet de sélectionner les attributs à afficher pour chaque tuple. Cela correspond à la projection en algèbre relationnelle et à la déclaration des variables libres en calcul. Le mot-clé `DISTINCT` permet de supprimer les doublons dans la sélection.

3.1.2 FROM

Spécifie les tables sur lesquelles porte la requête. Cela correspond à la base de données donnée en deuxième argument de la fonction $ans(Q, d)$ du calcul relationnel. `CROSS JOIN`, `NATURAL JOINN` et `JOIN <table> USING(<attr>)` permettent de réaliser le produit cartésien ou la jointure naturelle des tables directement dans le from.

WHERE permet de spécifier les conditions devant être vérifiées par les tuples. Permet de réaliser les jointures et les sélections de l'algèbre. On peut placer dans le where autant de condition logiques que l'on veut séparées par les opérateurs logique AND OR et NOT. L'ordre n'a jamais d'importance dans le where.

```
SELECT [DISTINCT <attr>[, <attri>]*
FROM <table>[[,,CROSS JOIN, NATURAL JOIN] <tablei>]*
[WHERE <condition> [<operateur> <condition>]*]
```

3.2 Opérations ensemblistes

Les opérations ensembliste `UNION, MINUS, INTERSECT` sont utilisables en SQL. Par leur nature ces opérations supprime les doublons. Seule exception possible `UNION ALL`.

```
SELECT ...
FROM ...
WHERE ...
```

```
UNION / MINUS / INTERSECT
SELECT ...
FROM ...
WHERE ...
```

3.3 Requêtes imbriquées

3.3.1 Principe

Les requêtes renvoient des tables en tout point similaires à celles de la base de données il est donc parfaitement possible d'utiliser le résultat d'une requête comme une table de relation de la base. Une requête renvoyant une unique valeur peut-être utilisée comme une constante littérale. Un exemple d'utilisation est tout simplement les opérateur ensembliste juste au-dessus.

3.3.2 Mots-clé utiles

On peut utiliser dans la clause WHERE un certain nombre de mots-clés particulièrement utiles avec les requêtes imbriquées. De plus avec ces mots-clés on peut faire apparaître les attributs de la requête principale dans la secondaire

- `Exists` vérifie s'il existe un tuple dans une autre table vérifiant une condition
- `IN` ou `NOT IN` vérifie si un attribut a une valeur présente dans une autre table
- `ANY` équivalent à `IN`
- `ALL` vérifie si une condition est vraie pour l'ensemble des tuples d'une table.

3.3.3 Exemples

Ces différents mots-clés sont très proche en réalité. Avec les trois premiers on va rechercher les noms et prénoms des joueurs ayant gagné au moins un match, et pour ALL de ceux qui ont gagné tous leurs matchs.

```
SELECT Prenom, Nom
FROM Licencie
WHERE EXISTS (SELECT *
              FROM Matchs
              WHERE Licencie.NumLic = Match.Gagnant)

SELECT Prenom, Nom
FROM Licencie
WHERE NumLic IN (SELECT Gagnant
                 FROM Matchs)

SELECT Prenom, Nom
FROM Licencie
WHERE NumLic = ANY (SELECT Gagnant
                    FROM Matchs)
```

```
SELECT Prenom, Nom
FROM Licencie
WHERE NumLic = ALL (SELECT Gagnant
                    FROM Matchs
                    WHERE EXIST (SELECT *
                                 FROM Matchs
                                 WHERE Licencie.NumLic=Lic1
                                 OR Licencie.NumLic=Lic2)
```

3.4 Agrégation des résultats

3.4.1 Opérateur d'agrégation :

Certains opérateurs tels que SUM, AVG, MIN, MAX, COUNT, etc... renvoient des valeurs qui sont calculés à partir d'un certain nombre de tuples. De base l'opération se fait sur l'ensemble de la table mais il est possible de définir des groupes de tuples avec les commandes GROUP BY et HAVING.

3.4.2 GROUP BY ... [HAVING ...] :

`GROUP BY` permet de créer des groupes de tuples ayant une même valeur pour un ensemble et `HAVING <condition>` permet de sélectionner les groupes vérifiant une condition.

3.4.3 Exemple :

On recherche le nombre moyen de match gagné par les joueurs ayant gagné au moins 3 matchs.

```
SELECT Gagnant, AVG(COUNT(*))
FROM Match
GROUP BY Gagnant
HAVING COUNT(*)>3
```

3.5 WITH statique : création de table

3.5.1 WITH

Est un mot-clé de SQL qui permet de créer une table uniquement le temps d'une requête afin de simplifier et d'alléger l'écriture des requêtes, surtout quand on plusieurs utilise des requêtes imbriqués semblables

3.5.2 Exemple

Recherche des participants dont la somme des points est égal au maximum de points obtenus par au moins candidat sur la compétition 1.

```
SELECT L.NumLic, SUM(Points)
FROM PointMatch P, Licencie L
WHERE P.NumLic = L.NumLic AND P.IdC = 1
GROUP BY L.NumLic
HAVING SUM(POINTS) = (SELECT MAX(SUM(Points))
```

```
                    FROM PointMatch P, Licencie L
                    WHERE P.NumLic = L.NumLic
                         AND P.IdC = 1
                    GROUP BY L.NumLic);
```

Avec `WITH` :

```
WITH pointsParticipants(NumLic, Spts) as (
    SELECT NumLic, SUM(Points)
    FROM PoitnMatch P, Licencie L
    WHERE P.NumLic = L.NumLic AND P.IdC = 1
    GROUP BY L.NumLic
)

SELECT NumLic, Spts
FROM pointsParticipants
WHERE Spts = (SELECT MAX(Spts)
              FROM pointsParticipants)
```

3.5.3 Efficacité :

Ici la table pointsParticipants n'est calculée qu'une seule fois (lors du WITH) contre deux dans l'exemple précédent, l'utilisation du WITH améliore donc l'efficacité de la requête.

3.6 WITH récursif

3.6.1 Problème

On souhaite à partir d'une table ParentsEnfants(Parents, Enfants) trouver l'ensemble des descendants de Jean. Le problème est que l'on ne sait pour l'instant que faire l'union de l'ensemble des enfants avec celui des petits-enfants des arrières-petits-enfants etc... Mais Le nombre de génération pris en compte est défini lors de la requête alors si on ne connaît pas la table cela devient problématique.

3.6.2 Le WITH récursif

On a vu juste au-dessus que l'on peut créer des tables lors des requêtes avec le mot-clé WITH. Il est possible de l'utiliser de manière récursive. Il faut donc faire attention aux cycles : la requête doit absolument avoir un nombre fini de réponses ! !

3.6.3 Solution du problème

```
WITH Descendants(p, d) as (
     SELECT Parents ad p, Enfants as d
     FROM ParentsEnfants as PE
     UNION ALL
     SELECT D.p, PE.Enfants as d
     FROM Descendants as D, ParentsEnfants as PE
     WHERE D.d = PE.Parents
```

```
)

SELECT *
FROM Descendants
WHERE p='Jean'
```

Tout d'abord la table Descendants copie la table ParentsEnfants puis on prend chaque tuple où un d dans descendants apparaît comme parent dans ParentsEnfants et on ajoute le tuple associant chaque ancêtre du parent avec l'enfant.

3.7 Requête hiérarchique

3.7.1 Autre Solution

La requête hiérarchique est une autre des solutions au problème précédent. Comme son nom l'indique elle permet de parcourir la table en descendant dans un ordre précisé par un lien hiérarchique entre deux attributs tel qu'un lien de parenté. Dans notre exemple on va chercher les enfants de la génération n+1 en cherchant les tuples ou la génération n apparaît comme parent.

3.7.2 Syntaxe :

```
START WITH <attr1> = cte
CONNECT BY PRIOR <attr1>=<attr2>
```

Cela réalise l'algo suivant :

1. sélectionner les tuples où attr1=cte
2. récupérer l'ensemble des valeurs de attr2 dans les tuples de l'étape précédente
3. sélectionner les tuples où attr1 appartient à l'ensemble des valeurs de attr2 de l'étape précédente
4. récupérer l'ensemble des valeurs de attr2 dans les tuples de l'étape précédente, si l'ensemble est nul arrêter sinon retourner en 3

Le reste de la requête s'applique à l'ensemble des tuples qui ont été sélectionnés au cours de l'algorithme.

3.8 Les vues

Une vue correspond à une requête nommée. On peut donc l'utiliser comme une nouvelle table. C'est la requête et non la table qui stockée, en utilisant une vue.

```
{CREATE|REPLACE} VIEW <nomVue>
AS <SFW...>
[WITH CHECK OPTION]
[WITH READ ONLY]
```

Deuxième partie

Procedural Language / SQL

4 Pourquoi faire ?

Le PL/SQL permet de créer des procédures automatisés capables de :
- vérifier des contraintes d'intégrité entre les tables lors de la saisie
- modifier la base de données (passer les noms en majuscules, augmenter les prix de 10%)
- réaliser une maintenance de la base de données

5 Bloc

Le PL/SQL s'organise en bloc de la manière suivante :

```
DECLARE
     types
     variables
     constantes
     curseurs
     exceptions utilisateurs]
BEGIN [<NomBloc>]
     instructions PL/SQL
     possibilité de blocs imbriqués
[EXCEPTION
     traitement des exceptions]
END [<NomBloc>];
```

5.1 Declaration

Types et déclaration de variables : On retrouve tous les types de SQL. Il existe deux type supplémentaires : les records qui sont des tuples et les curseurs qui permettent de parcourir les tuples du résultat d'une requête.
Déclaration
- `<nomVar> [CONSTANT] <nomType> [:= <valeur>];`
- `<nomVar> <nomAttr>%TYPE;`
- Record : `TYPE <nomType> IS RECORD(<nomAttr> <typeAttr>, ...);` Puis `<nomVar> <nomType>`
- Record : `<nomVar> <nomTable>%ROWTYPE;`
- Curseur : `CURSOR <nomCurseur> IS <SFW...>`

5.2 Affectation

Il existe deux méthodes pour affecter une valeur à une variable. L'affectation classique : `<nomVar> := <expression>;` ou par requête SQL. Cette dernière permet de stocker le résultat d'une requête dans une ou plusieurs variables. Attention : la requête doit avoir un seul tuple résultat ! ! Voici deux exemples de la syntaxe.

```
SELECT Nom, Prenom, NumLic INTO Nom$, Prenom$, NumLic$
FROM Inscrition
WHERE NumLic = 8;

SELECT * INTO RecordMembre
FROM Inscription
WHERE NumLic = 8;
```

5.3 Condition et Boucle

5.3.1 Blocs IF et CASE :

Fonctionne comme le if et le switch des autres langages

```
IF (<condition>)
THEN
    <instructions>;
[ELSIF (<condition)
    <instructions>;]
[ELSE
    <instructions>;]
END IF;

CASE <variable>
WHEN <valeur> THEN <operation>;
WHEN <valeur> THEN
    BEGIN
        <seqOp>;
    END
ELSE ...
END CASE;
```

5.3.2 While et For :

Exactement comme les autres langages. Possible de parcourir une requête avec les curseurs

```
WHILE <condition>
LOOP
    ...
ENDLOOP;

FOR <varDeBoucle> IN [REVERSE] <borneInf>..<borneSup>
LOOP
     ...
END LOOP

DECLARE
   CURSOR <nomCurseur> IS SFW...;
   ...
BEGIN
```

```
...
FOR <var> IN <nomCurseur>
LOOP
    ...
END LOOP;
```

5.4 Traitement des exceptions

5.4.1 Gestion d'exception

```
DECLARE
    <MonException> EXCEPTION;

BEGIN
    ...
    RAISE <MonException>;
    ...
EXCEPTION
    WHEN <MonException> THEN ...
    ...
    WHEN OTHERS ...
END;
```

Pour le cas particulier des erreur on utilise :

```
RAISE_APPLICATION_ERROR(<code>,<Message>);
```

5.4.2 Erreurs standards

SQL possède un grand nombre d'erreurs standards. On peut utiliser `SQLCODE` pour obtenir le code de l'erreur et `SQLERRM` pour obtenir le message de l'erreur

5.4.3 Stocker le message dans une table

```
CREATE TABLE audit_table(code NUMBER, message VARCHAR2(64), date TIMESTAMP)

DECLARE
     ...
BEGIN
     ...
EXCEPTION
     WHEN OTHERS THEN
          code := SQLCODE;
          msg := SQLERRM;
          INSERT INTO audit_table
          VALUES(code, msg, SYSTIMESTAMP)
END
```

6 Bloc nommés

Il existe trois types de bloc nommés :

— Procédure
— Fonction
— Package

6.1 Procédure

6.1.1 Définition

Une procédure est un bloc nommé possédant un certain nombre d'argument et pouvant être appelé autre part dans le code.

6.1.2 Création/Modification :

```
CREATE OR REPLACE PROCEDURE <NomProc>
   [(<nomParam> {IN|OUT|IN OUT} <Type> [NOCOPY])]
IS
    [DECLARATION]
BEGIN
    [CORPS]
END;
```

6.1.3 Utilisation

```
<NomProc>(<parametres>)
```

6.1.4 Suppression

```
DROP PROCEDURE <NomProc>
```

6.2 Fonction

6.2.1 Une fonction

Une fonction est une procédure qui renvoie une valeur à la fin de son exécution

6.2.2 Création/Modification :

```
CREATE OR REPLACE FUNCTION <NomFct>
   [(<nomParam> {IN|OUT|IN OUT} <Type> [NOCOPY])] RETURN <Type>
IS
    [DECLARATION]
BEGIN
    [CORPS]
    RETURN(<valeur>)
END;
```

6.2.3 Utilisation

```
[<variable> :=] <NomFct>(<parametres>)
```

6.2.4 Suppression

```
DROP FUNCTION <NomProc>
```

6.3 Package

6.3.1 Définition

Un package est un regroupement de fonctions et de procédures. Cela permet de regrouper les procédures et fonctions qui sont fonctionnent ensemble. Un package est composé d'une interface (équivalent à un .h en C/C++) et d'un corps (équivalent au .c/.cpp).

6.3.2 Syntaxe :

```
CREATE OR REPLACE PACKAGE <NomPck>
IS
    <NomVar> <Type>;
    FUNCTION <NomFct>[(<nomParam> {IN|OUT|IN OUT}
                <Type> [NOCOPY])] RETURN <Type>;
    PROCEDURE <NomProc> [(<nomParam> {IN|OUT|IN OUT}
                <Type> [NOCOPY])]
END <NomPck>


CREATE OR REPLACE PACKAGE BODY <NomPck>
IS
    <NomVar> <Type>;

    FUNCTION <NomFct>[(<nomParam> {IN|OUT|IN OUT}
                <Type> [NOCOPY])] RETURN <Type>
    IS
         ...
    END <NomFct>

    PROCEDURE <NomProc> [(<nomParam> {IN|OUT|IN OUT}
                <Type> [NOCOPY])]
    IS
         ...
    END <NomProc>
END <NomPck>
```

7 Triggers

7.1 Principe

7.1.1 Définition

Un trigger est un morceau de code PL/SQL qui est appelé lorsque les tuples d'une table sont modifiés. Cela permet par exemple de vérifier des contraintes d'intégrité entre plusieurs tables distinctes.

7.1.2 Caractéristiques d'un trigger

— Possède un nom et du code PL/SQL
— Déclenché par un événement (INSERT, UPDATE, DELETE)
— Réalisé avant ou après cet événement
— Surveille une seule table
— N'est pas modifiable, mais supprimable ou désactivable

7.1.3 Possibilité d'un trigger

Dans un trigger l'accès à la table surveillée est interdit, on ne peut donc pas effectuer de requête dessus. On ne peut ni utiliser COMMIT et ROLLBACK, ni créer de table dans un trigger. Par contre on a accès aux anciennes et aux nouvelles valeurs des tuples concernés par l'événement grâce à `:OLD.<nomAttribut>` et `:NEW.<nomAttribut>`. On peut connaître la nature de l'événement grâce aux constantes booléennes `INSERTING`,`UPDATING` et `DELETING`

7.2 Syntaxe

7.2.1 Création

```
CREATE OR REPLACE TRIGGER <nomTrigger>
{BEFORE|AFTER} <evt> [OR <evt>]* ON <NomTable>
[FOR EACH ROW [WHEN <condition>]]
    <Bloc PL/SQL>
```

7.2.2 Activer / Désactiver

```
ALTER TRIGGER <NomDeclencheur> {ENABLE|DISABLE};

ALTER TABLE <NomTable> DISABLE ALL TRIGGERS;
```

7.2.3 Supprimer

```
DROP TRIGGER <NomDeclencheur>;
```

Troisième partie
Complexité

8 Index

8.1 Principe

En base de données, un index est une structure de données permettant d'accéder rapidement à un tuple si l'on connaît certains de ses attributs. L'index est stocké et entretenu par le SGBD. Il fait partie du schéma physique de la base de données. Il simplifie et accélère sélection, jointures, l'agrégation.

8.2 Différents types d'index

8.2.1 B-Arbres

Avantages :
- Permet d'accélérer les traitements
- Peut être utilisé pour des requêtes portant sur des égalités et des inégalités

Inconvénients :
- les index peuvent devenir plus volumineux que la table initiale.
- Oracle : Attention au null : ne permet pas de faire des recherches sur l'absence ou la présence de null

8.2.2 BitMap

Principe : pour chaque valeur présente dans la table, indiquer la liste des tuples présentant cette valeur. Encoder les valeurs pour aller plus vite. Encodage le moins volumineux possible.

Avantages :
- Indexage le moins volumineux possible
- Permet de faire des calculs (count(),sum(),avg()) sur les attributs indexés sans prendre en compte les autres valeurs des tuples
- Modification plus simple

Inconvénients :
- Perd son intérêt avec le nombre de valeurs possibles
- Ne fonctionne que sur les égalités

8.2.3 Hash

Avantages :
- Très performant si la fonction de hachage est bien construite

Inconvénients :

- Ne fonctionne que sur les égalités
- Nécessite une réorganisation périodique ou hachage dynamique, sinon risque de perte de performances ou de gâchis d'espace

8.3 Choix de l'indice

Il faut trouver un juste équilibre entre performances des requêtes et performances de mise à jour et espace de stockage. Certains SGBD ne permettent l'utilisation pas tous les index. Le B-Arbre est le plus courant.

8.4 Implémentation

8.4.1 Implicite

Un index est créé automatiquement sur la clé primaire d'une table.

8.4.2 Explicite

Il est d'usage d'en créer sur chaque clef étrangère pour optimiser les jointures

```
CREATE [BITMAP] INDEX <nomIndex> ON
<nomTable> (<attribut>[,<attribut]*)
```

9 Optimisation de requête

9.1 Analyse lexicale et syntaxique

- Vérification de la syntaxe SQL
- Vérification des types :
  - Présence des attributs et relations dans le schéma
  - Compatibilité dans les types

9.2 Simplification

On simplifie les conditions de sélection dans le WHERE (cf cours Archi circuit numérique pour l'algèbre booléenne) et on supprime les opérateurs sans effet (DISTINCT sur des attributs UNIQUE).

9.3 Normalisation

On passe les conditions de sélection en forme normale conjonctive soit une conjonction (OU) de disjonction (ET).

9.4 Restructurer

Dans cette étape on traduit la requête SQL en algèbre relationnelle (cf cous Modélisation de données. Il y a beaucoup d'arbre de requête équivalents, comment choisir le bon arbre ?

9.5 Plan d'exécution

Problème : Non seulement il y a des arbres équivalents mais pour un arbre il y a plusieurs algorithmes possibles car chaque opération algébrique peut être implémenter de différentes manières. Il s'agit donc de trouver le plus rapide

Le coût d'un noeud dépend du nombre de données qu'il a à traiter et de l'algorithme utilisé. Il s'agit donc de faire en premier les opérations les plus sélectives. Les sélections et les jointures sont plus efficaces avec des index il peut donc être intéressant de les déplacer en bas de l'arbre.

L'impact des données ne doit pas être négligé en effet certains algorithme ont une complexité qui varie en fonction des données qui sont les tables (l'efficacité de l'index Bitmap ou Hash par exemple).

Conclusion Le choix du plan d'exécution est complexe et dépend des données présentes dans la table. Les SGBD stockent donc des statistiquement sur les données des tables pour prendre les meilleurs décisions. Le choix doit être pris rapidement car il serait ridicule que l'optimisation soit plus longue que l'exécution de la requête.

Quatrième partie
Administration

10 Data Base Administrator

Le Data Base Administrator (DBA) doit :
- Installer le SGBD et le paramétrer en lien avec les applications clientes. En mode centralisé, cluster ou largement distribué.
- Créer la base de données en faisant les bons choix (taille d'un bloc, emplacements des fichiers sur disque, nombre de fichiers de données, taille des fichiers, multiplexage, espace logique de stockage (tablespace), etc...)
- Gérer les comptes utilisateurs
- Assurer la cohérence et la sécurité des données
- Amélioration des performances lors de l'exploitation
- Gestion des mises à jour et des évolutions du SGBD en exploitation

11 Stockage des informations

11.1 Stockage des données sous Oracle

Oracle fonctionne avec deux niveaux de stockage de données :
- niveau physique : les fichiers
- niveau logique : permet aux utilisateurs de s'abstraire des fichiers en utilisant les tablespaces.

La base de données est constituée de fichiers de types différents

11.2 Niveau Physique : les fichiers

11.2.1 Les fichiers sous Oracle

Les fichiers d'Oracle se divisent en deux catégories :
- Les fichiers de configuration de la base
- Les fichiers stockant les informations de la base

Fichiers de configuration :
- Fichier d'initialisation (paramètres de démarrage de la base)
- Fichier de contrôle (nom de la base, date de création, structure physique de la base, emplacement des fichiers, etc...)
- Fichier des mots de passe
- ...

Fichiers d'information :
- Fichier de données (table, vues, procédures stockées, ...)
- Fichier de reprise redo-log (historique des modifications effectuées sur la base)
- Fichier de contrôle (schéma physique de la base de données : tablespace, fichiers, ...)

11.2.2 Fichier de contrôle

Un fichier de contrôle est un fichier binaire. Il décrit le schéma physique de la BD : son nom, les noms et situations des autres fichiers, les informations sur les tablespaces,... Il est mis à jour automatiquement lors de la modification du schéma physique de la BD.

11.2.3 Fichiers de données

Les fichiers de données servent à stocker les données de la BD (tables, vues, procédures, etc...). Ils sont dimensionnés à la création mais leur taille peut évoluer dynamiquement. Il peut y avoir un ou plusieurs fichiers de données.

11.3 Niveau Logique : les tablespaces

11.3.1 Définition d'un tablespace

Pour l'utilisateur un tablespace est assimilable à un espace de stockage, cela lui permet de s'abstraire des fichiers physiques. Oracle fait le lien entre les fichiers physique et les objets du schéma (tables, vues, index, ...) via les tablespaces.

11.3.2 Utilisation des tablespaces

Une BD possède au moins un tablespace, mais il est possible d'en rajouter avec :

```
CREATE TABLESPACE users2
DATAFILE 'data3.dat'
```

Un tablespace n'appartient qu'à une seule BD.

11.3.3 Tablespaces spécifiques

— SYSTEM : informations système, table du dictionnaire de données, programmes PL/SQL... C'est le seul tablespace obligatoire
— USERS : Pour les objets des utilisateurs
— ...

12 Gestion des utilisateurs

12.1 Les utilisateurs

12.1.1 SYS

L'utilisateur SYS est toujours présent. Son mot de passe est définit à la création de la table. Il possède tous les droits. Il est propriétaire du dictionnaire, relations, vues dans le schéma SYS.

12.1.2 SYSTEM

De même SYSTEM est toujours préset est son mot de passe est définit à la création de la table. Il possède beaucoup de droits mais moins que SYS.

12.1.3 Utilisateurs standards

On peut définir autant d'utilisateurs que l'on veut à l'aide des instructions :

```
CREATE USER ...;
DROP USER ...;
```

12.2 Les privilèges

Lorsqu'un utilisateur est créé il n'a aucun privilège, il faut lui en donner avec le mot-clef `GRANT`.

12.2.1 Privilèges système

Parmi les privilèges système on trouve le droit de :
— modifier la BD
— modifier les paramètres systèmes
— créer des tables, des index, des vues, des triggers, ...
— donner des privilèges
— ...

12.2.2 Privilèges objets

Parmi les privilèges objet on trouve le droit de :
— modifier une table
— utiliser les mots-clefs `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `EXECUTE`, etc...
— ...

12.3 Les rôles

12.3.1 Définition et syntaxe

Un rôle est un ensemble de droits, on peut ensuite donner un rôle à un utilisateur pour lui donner l'ensemble de ces droits avec les syntaxes suivantes :
— Créer un rôle (`CREATE ROLE...`)
— Donner/retirer des privilège à un rôle (`GRANT/REVOKE`)
— Donner/retirer un rôle à un utilisateur ou à un autre rôle (`GRANT/REVOKE`)

12.3.2 Conseil d'organisation

Pour gérer les droits des utilisateurs on va créer 2 niveaux de rôles :
— Rôle d'application : Chaque tache d'application aura son rôle d'application avec les droits nécessaires
— Rôle d'utilisateur : Chaque type d'utilisateur aura son rôle d'utilisateur créer par combinaison des rôles d'application liés aux taches réaliser par ce type d'utilisateur

Enfin chaque utilisateur aura un unique rôle d'utilisateur. Aucun droit ne sera donné individuellement.

12.4 Les profils

Un profil est un ensemble nommé de limite de ressources comme un nombre max de connexion simultanées ou une limite de validité du mot de passe etc... On gère un profil avec les syntaxes :

- créer un profil `CREATE PROFILE p LIMIT ...`
- associer un profil à un utilisateur :
  - à la création `CREATE USER ... PROFILE ...`
  - après `ALTER USER ...`

Cinquième partie
Transaction et Isolation

13 Définition et but d'une transaction

Définition : Une transaction est une séquence qui fait passer une BD d'un état cohérent à un autre état cohérent.

But : Les transactions servent à maintenir la cohérence des informations.

Dangers :
- Concurrence
  - Perte d'écritures
  - Introduction d'incohérence
- Panne de transaction
  - Erreur en cours d'exécution
  - Nécessité de défaire les mises à jour déjà effectuées
- Panne système
  - Perte de mémoire centrale
  - -> Transactions en cours doivent être refaite
- Panne disque
  - Perte mémoire de masse
  - ->Redémarrage à partir d'une sauvegarde

14 Structure d'une transaction

14.1 Début et Fin d'une transaction Oracle

Début : Une transaction débute par n'importe quelle opération SQL ou par la fin de la transaction précédente.

Fin : La transaction courante est terminé lors d'un COMMIT ou ROLLBACK, de la déconnexion de l'utilisateur, de d'une opération de définition de données ou l'échec du processus.

14.2 Opérations importantes

14.2.1 Restriction de l'étude

On restreint l'étude des transactions aux opérations pouvant poser problèmes : LECTURE et ECRITURE, et les opérations spécifiques : DEBUT, VALIDER et ABANDONNER

14.2.2 Notations

On note r la lecture et w l'écriture.
On note T_i la transaction i.
Ainsi on note $r_i(d)$ la lecture d'une donnée d durant la transaction i

Et $w_i(d)$ l'écriture d'une donnée d durant la transaction i On peut ajouter la valeur lu ou écrite x avec $r_i(d : x)$ et $w_i(d : x)$

15 Gestionnaire de transaction

Le gestionnaire de transaction est une partie du SGBD qui contrôle l'exécution des transactions demandées par les différents utilisateurs et assure les propriétés ACID :

— Atomicité
— Cohérence
— Isolation
— Durabilité

15.1 Propriétés ACID

15.1.1 Atomicité

La transaction est indivisible ! ! L'ensemble des actions est validé ou annulé tout entier jamais en partie. En cas de validation toutes les opérations doivent être validées. En cas d'annulation toutes les opérations de la transaction sont annulées. L'annulation peut être décidée par l'utilisateur ou le gestionnaire de transaction.

15.1.2 Cohérence

Une transaction doit accéder à une base de données dans un état cohérent et retourner une base de données dans un état cohérent. Cette propriétés doit toujours être vérifiée que la transaction soit validée ou annulée.

15.1.3 Isolation

Le degré d'isolation définit comment les effets d'un transaction sont perçus par les autres et comment elle perçoit les effets des autres.
Tant qu'elle n'a pas été validée les effets d'une transaction ne doivent pas être visibles par les autres et donc une transaction ne doit percevoir les effets que des transactions validées.

15.1.4 Durabilité

Les mises à jours réalisées par une transaction validée doivent persister.
Par exemple en cas de panne, le système garantit que les effets des transactions validées seront toujours présent au redémarrage. Pour les autres il faudra les relancées.

15.1.5 But des propriétés ACID

— Gestion des pannes :
  — Atomicité (transaction indivisible)
  — Durabilité (Produit un état persistant)
— Gestion de la concurrence :
  — Cohérence (Part et produit une BD dans un état cohérent)

— Isolation (Effet invisibles des autres avant validation)

15.2 Gestion de la concurrence

15.2.1 Objectifs et Principes

— Permettre à plusieurs utilisateurs de travailler en même temps
— Maintenir un bon niveau de performance (parallélisation des transactions)
— Maintenir la cohérence de la BD (Ordre d'exécution des opérations)
— simplicité pour l'utilisateur

15.2.2 Problèmes à éviter

— Introduction d'incohérences (non respect des contraintes du schéma)
— Perte d'écriture (écriture refusée car rendu impossible)
— Lecture sale (lecture d'une valeur définie par une transaction annulée plus tard)
— Lecture non répétable (lectures successives donnant des résultats différents)
— Lectures fantômes (insertion de nouveaux tuples par une transaction qui fausse les calculs d'une autre transaction)

15.2.3 Eléments théoriques

Exécution sérielle

Définition : Une exécution de transactions est dite sérielle si et seulement si il n'y a aucun entrelacement entre les opérations des transactions

Théorème : Comme les transactions maintiennent individuellement la cohérence, si la BD est initialement cohérente, le résultat d'une exécution sérielle est un état cohérent.

Exécution / Ordonnancement

Définition : une exécution est un ordonnancement des opérations réalisées par un ensemble de transactions.

Objectifs : définir ce qu'est une exécution correcte et avoir un système qui s'y conforme.

Exécutions équivalentes

Définition : Deux exécutions sont équivalentes si et seulement si elles incluent les même transactions et ont le même résultat indépendamment de l'état initial de la BD.

Utilité : Permet de trouver des exécutions équivalentes à celle souhaité : exécution sérielle.

Opérations conflictuelles

Définition : Des opérations sont conflictuelles si elles accèdent aux même données et au moins l'une d'entre elles est une écriture

Exécutions équivalentes vis-à-vis des conflits

Théorème : deux exécutions d'un ensemble de transactions sont équivalentes si et seulement si :
— l'ordre des opérations de chaque transaction est identique
— l'ordre des opérations conflictuelles validées sont identiques

Théorème : En partant d'une exécution, permuter deux opérations adjacentes, de transaction différente et non conflictuelles donne une exécution équivalente à la première.

Exécution sériable

Définition : Une exécution est dite sériable s'il existe une exécution sérielle équivalente vis-à-vis des conflits.

Attention : Deux exécutions sérielles du même ensemble de Transactions peut donner des résultats différents.

15.2.4 Solution par verrouillage

Contrôle des accès concurrents par verrouillage :

Principe : Avant d'effectuer une opération (lire/écrire), une transaction doit en acquérir le contrôle : verrou. Le verrou doit être relâché quand la transaction a finit. En lecture le verrou peut-être partagé, en écriture il doit être exclusif. Lorsqu'une transaction est interdite d'accès à une donnée par un verrou , elle est mise en attente et réactivée quand le verrou est relâché.

Opérations importantes d'une transaction : Il faut maintenant considérer dans les transaction les lectures et les écritures mais aussi les poses et les levées de verrous.

Problème : D'une part cette solution ne résolve pas tous les problèmes de plus elle ajoute un problème d'interblocage : si deux transactions posent un verrous sur une données dont l'autre a besoin ultérieurement.

Verrouillage en deux phases 2PL

Transaction bien formée : Pose un verrou partagé sur une donnée avant de la lire. Pose un verrou exclusif sur une donnée avant de l'écrire. Tous les verrous sont libérés à la fin de la transaction.

Deux règles :
— Règle 1 : Deux transactions différentes ne peuvent posséder simultanément des verrous en conflit
— Règle 2 : Toute transaction qui libère un verrou ne peut plus en acquérir d'autres

Sériabilité : Toute exécution par un verrouillage à deux phases est sériable.

Verrouillage en deux phases 2PL strict : En réalité il est difficile de savoir quand on aura plus besoin de nouveaux verrous. On modifie donc la Règle 2 et on supprime l'ensemble des verrous en même temps à la fin de la transaction

Avantage de cette méthode : Permet d'éviter les pertes de mises à jour, pas de lecture non reproductible, pas de lecture sale. Adapté aux cas où l'on a beaucoup de transaction de courte durée.

16 Isolation

16.1 Niveaux d'isolation SQL92

	Read-Uncomm	Read Comm	Repeat read	Serial
Lecture sale	x			
Lecture non répétable	x	x		
Lecture Fantome	x	x	x	

16.2 Niveaux d'isolation sous Oracle

Oracle propose 3 niveaux d'isolation.

16.2.1 read commited (par défaut)

Chaque requête ne perçoit que les changements qui ont étés commité avant le début de la requête en cours (pas la transaction).

16.2.2 serializable

Ne sont perceptibles que les changements qui avaient étés commités avant le début de la transaction plus les changements de la transaction elle-même.

16.2.3 read only (spécifique Oracle)

Au sein d'une transaction ne sont perceptibles que les changements qui ont été commités avant que la transaction ne commence et ne permet pas de modifier les données.

16.3 Spécifier le niveau d'isolation

On peut spécifier le niveau d'isolement :
— pour une transaction : `SET TRANSACTION ISOLATION LEVEL ...;`
— pour toute une session : `ALTER SESSION SET isolation_level = ...;`

Sixième partie

Conception d'une BD distribué Top-Down

17 Fragmentation

17.1 Propriétés recherchés

— On ne perd pas de données
— On peut retrouver les liens entre les données, on peut reconstituer la BD centralisée.
— On duplique un minimum de données

17.2 Critère de fragmentation

On souhaite optimiser : l'usage fréquent qui est fait sur les données. La fragmentation est donc faite en fonction des usages. Les usages sont définis par les applications. Deux applications peuvent utiliser fréquemment la même table mais pas les même données.

17.3 Fragmentation Horizontale Primaire

17.3.1 Trouver les prédicats discriminants

La première étape consiste à trouver les prédicats que vérifient les tuples utilisés par une application : on les nomme prédicats discriminants.

17.3.2 Trouver les prédicats composés

Un tuple est nécessairement qualifié par une combinaison des prédicats discriminants (avec les opérateurs AND et NOT). On peut simplifier la liste des combinaisons par impossibilité logique ou due au contraintes d'intégrité de la table. On peut ensuite simplifier les combinaisons (par exemple $x = 2 \; and \; x \neq 3 \;\Leftrightarrow\; x = 2$). La liste ainsi obtenus donne les prédicats composé. Tous les tuples vérifient un unique prédicat composé et chaque prédicat composé peut être vérifiés par un tuple de la table.
Les prédicats composés correspondent chacun à un nouveau fragment formé de l'ensemble des tuples vérifiant ce prédicat.

17.3.3 Vérification

A cette étape il convient de vérifier que la fragmentation vérifie l'ensemble des propriétés recherchées.

17.4 Fragmentation Horizontale Dérivée

17.4.1 Principe

Que faire des tables qui référencent la table fragmentée ? Il serait logique que les informations liées restent ensemble lors de la fragmentation. On procède alors à une fragmentation horizontale dérivée.

17.4.2 Réalisation

Soit A une table fragmentée horizontalement et B une table tel que la jointure $A \bowtie B$ est utilisé fréquemment. Soit A_i les fragments de A, on trouve les fragments B_i de B avec le relation suivante :

$$B_i = \pi_B(A_i \bowtie B)$$

17.4.3 Propriétés

La fragmentation ainsi obtenue :
— est disjointe (si la primaire l'est)
— permet la reconstruction

17.5 Fragmentation verticale

La fragmentation verticale est définie par la projection. Elle doit vérifier deux règles :
— La clef appartient à tous les fragments
— les autres attributs n'appartiennent à un et un seul fragment
Ainsi elle est disjointe (hormis la clef) et permet la reconstruction.

17.6 Fragmentation mixtes

Il est possible de combiner les fragmentations horizontales e verticales. On obtient ainsi des fragmentations mixtes. Les propriétés d'une fragmentation mixte correspond à l'intersection des fragmentations qui la composent.

17.7 Démarche générale

1. Fragmentations horizontales
   (a) Fragmentations principales
      i. Identification des prédicats discriminants
      ii. Construction des prédicats composés
   (b) Fragmentations dérivées
2. Fragmentations verticales
3. Compositions des fragmentations. Attention à l'ordre des fragmentations ! !

18 Répartition

18.1 Contrainte d'allocation des fragments

- Un fragment est sur un et un seul site
- Optimisation en communication, stockage et performance dépend de :
  - BD (tailles des fragments, contraintes d'intégrité)
  - Application (nb lecture/écriture fragment/site)
  - Réseau (vitesse communication)
  - Site (coût unitaire stockage et calcul)

18.2 Solution

Il n'y a pas de solution absolue. Il faut donc faire une approche heuristique avec des approximations.

19 Réplication

Réplication vs Duplication

Lors d'une duplication la gestion des doublons doit être réalisée par les applications. La réplication crée un lien explicite dans le SGBD, le maintien de la cohérence est donc géré par le SGBD et le DBA.

19.1 Vocabulaire

- Réplication = processus consistant à copier et maintenir des objets dans différentes BD
- Objet répliqué, réplicat = objet qui existe dans plusieurs BD
- Groupe de réplicats = collection de réplicats logiquement liés
- Site de réplication = site qui héberge un groupe de réplicat
- Site Maître pour le réplicat r = site à partit duquel il est possible de modifier r.

19.2 Différent types de réplication

- Pas de réplication
  - Stockage sur un site et accès réseau pour les autres
  - Pb de performance et disponibilité
- Réplication 'manuelle'
  - Propagation des mises à jour manuellement
  - Passage par un protocole 2PC (Two Phases Commit) - cher et bloquant
- Réplication automatique et transparente
  - Mono-Maître
  - Multi-Maîtres

19.3 Réplication Mono-Maître

Les réplicats ne peuvent être modifiés qu'à partir du site Maître, les autres sites ne servent qu'en lecture

19.4 Réplication Multi-Maîtres

Les réplicats peuvent être modifiés à partir de toutes les sites

19.5 Cohérence des données

Il peut y avoir des conflits entre différentes valeurs pour les réplicats. Certains sont résolus de manière automatique. D'autres attendent une intervention humaine dans la file des erreurs. L'administrateur est responsable de la résolution des conflits manuellement.

20 Bilan général de l'approche Top-Down

1. Conception du schéma
2. Fragmentation en fonction des usages
3. Placement des fragments en fonctions des usages
4. Eventuelle réplication en fonction des usages

Septième partie

Base de données distribuée sous ORACLE

21 Connexion à une base de données distante

21.1 Pré-requis

Pour accéder à une base de données dustante il faut :
- Qu'elle ait un nom global unique
- Que ce nom soit connu par la base locale (rôle de l'administrateur)
- Avoir un compte sur la base distante
- Utiliser ses informations pour créer un lien de base de données

21.2 Lien de base de données

Un lien est déclaré localement sur une base de données et fait référence à une autre distante. Un lien peut être privé (limité à l'utilisateur qui le déclare) ou public (utilisable par tous). Il existe trois types de connexion :
- connected user link : chaque utilisateur se connecte avec son propre login. Les logins doivent être identiques sur les deux sites.
- fixed user link : Les utilisateurs se connectent avec le login définit dans le lien
- current user link : dans les procédures stockées

21.3 En SQL

21.3.1 Création du lien

```
CREATE [SHARED] DATABASE LINK nomLien CONNECT TO {user IDENTIFIED
BY password | CURRENT USER} USING 'nomBD';
```

21.3.2 Accès aux objets

A l'aide du lien on peut accéder et manipuler la base de données distante comme si on y était connecté localement avec le même utilisateur. Il suffit de rajouter le suffixe `@nomLien` après les objets manipulés.

21.3.3 Fermer ou supprimer un lien

```
ALTER SESSION CLOSE DATABASE LINK nomLien;

DROP DATABASE LINK nomLien;
```

21.4 Transparence de la distribution

Pour éviter les écritures trop longues on peut rendre transparent l'utilisation du lien avec :
- une vue locale de la table distante

— un synonyme (préférable à une vue de la table entière)

21.5 Collocated Inline View

Dans un environnement distribué les accès à distance peuvent être très couteux, d'autant que le latences réseaux ne sont pas prévisibles. Il faut donc au privilégier l'exécution locale. Pour cela il convient d'utiliser des Collocated Inline Views :

— Collocated = sur le même site
— Inline View = Select dans un From

21.6 Réplication sous ORACLE

21.6.1 Réplication multi-maîtres

— Chaque site est maître est peut donc le modifier
— Lors d'une modification, le site informe les autres
— Oracle gère automatiquement les modifications

21.6.2 Réplication par vue matérialisée

Vue matérialisée

Définition est une vue dont le résultat est stocké physiquement. Le résultat peut-être traité comme une table. Une vue matérialisée répliquée est une copie d'un objet que l'on appelle Maître sur un site distant. Le Maître peut-être une table ou une vue matérialisée. Une vue matérialisée peut-être :

— read-only
— updatable
— writeable

Exemple de syntaxe

```
CREATE MATERIALIZED VIEW ... AS SFW;
```

Quelques contraintes : La requête qui définit une vue matérialisée doit référencer la clé primaire de la relation maître. Une vue matérialisée ne peut-être définie sur une table qui contient des données cryptées en utilisant un cryptage transparent.

Le rafraîchissement

Quand rafraîchir ?

— Asynchrone (par défaut)
  — Un changement local sur un objet est stocké dans la file d'attente locale des modifications.
  — Régulièrement, les éléments de cette file sont communiqués aux autres sites
— Synchrone

- Tout changement doit être communiqué à tous et validé par tous pour appliqué en local
- Bloquant tant que l'on n'a pas la réponse de tous

Comment rafraîchir ?
- Transmettre toutes les données
- Méthode incrémentale : ne transmettre que les modifications

Types de rafraîchissement asynchrone :
- Complete Refresh
  - Toutes les données sont supprimées de la réplique
  - Toutes les données sont transmises par le maître et insérées dans la réplique
  - Recommandé lorsque le nombre le nombre de tuples à modifier/inséré dépasse 50% de la cardinalité de la table.
- Fast Refresh
  - Le maître transmet les modifications à apporter à la réplique.
  - Nécessite l'usage d'un log pour mémoriser les modifications effectuées sur le maître et les appliquer sur la réplique
- Force Refresh
  - Oracle tente un fast refresh si c'est possible, sinon il fait un fast refresh.

21.6.3 Les conflits

Types de conflits :
- Conflits en mise à jour
- Unicité (violation de contrainte d'intégrité)
- Conflits en suppression

Détection des conflits :
- Sur un site maître
- Utilisation des clés primaires des tables pour identifier chaque ligne de manière unique
- Les applications ne sont pas autorisées à faire des mises à jour sur une clé primaire

Résolution des conflits : Oracle ne cherche pas à savoir qui a tord ou raison. Plusieurs méthodes dont latest timestamps (priorité au plus récent), overwrite et 6 autres.

Limites : Les mécanismes d'Oracle ne permettent de régler qu'une partie des conflits. Pour les autres, ils sont déclarés en 'Unresolved Conflicts' et sont stockés dans une file d'attente d'erreurs. C'est un rôle de l'administrateur de données que de résoudre manuellement ces conflits.