DUT STID, Université de la Côte d'Azur

Bases de données avancées

Gestion de BD

Prof. D. Malchiodi (malchiodi@di.unimi.it)

http://malchiodi.di.unimi.it/teaching/STID-BDA

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

UNIVERSITÉ
CÔTE D'AZUR

Ouvrages de références

- G. Gardarin, “Base de données”, édition Eyrolles
  - La référence en BD
- Andreas Meier, « Introduction pratique aux bases de données relationnelles », 2ème édition, Collection IRIS
  - Exemple complet en Access avec exercices corrigés
- Elmasri & Navathe, « Fundamentals of Database Systems », 4th edition, International Edition
- Chris J. Date, « Introduction aux Bases de Données », 7e édition, vuibert informatique
  - Livre très complet avec exercices non corrigés
- K. Williams and al., « XML et les bases de données », édition Eyrolles
- Sur le net
  - http://www.hec.unil.ch/gcampono/index.php/teaching/BDA

Rappels de BD

BD & SGBD

- BD
  - Un ensemble bien structuré de données relatives à un sujet global contenant le moins de redondances possibles et accessible par plusieurs utilisateurs simultanément
- SGBD
  - Ensemble de programmes permettant à des utilisateurs de créer et d'utiliser des BDs
  - Activités
    - Définition d'une BD (spécifications des types de données à stocker)
    - Construction d'une BD (stockage des données proprement dite)
    - Manipulation des données (ajouter, supprimer, retrouver des données …)

BD & SGBD

Que doit permettre un SGBD ?

- Décrire les données indépendamment des applications
  - langage de définition des données (DDL)
- Manipuler les données
  - interroger et mettre à jour les données
  - sans préciser d'algorithme d'accès (dire QUOI sans dire COMMENT)
  - langage de *requêtes* déclaratif
  - ex. : quels sont les noms des produits de prix < 100 Euros ?
  - langage de manipulation des données (DML)

BD & SGBD

- Contrôler les données
  - Intégrité *:* vérification de contraintes d'intégrité ex.: le salaire doit être compris entre 1400 Euros et 10000 Euros
  - Confidentialité *:* contrôle des droits d'accès, autorisation
  - langage de contrôle des données (DCL)

- Partage
  - Une BD peut être partagée entre plusieurs utilisateurs en même temps
  - Nécessité de contrôle des accès concurrents
  - Notion de transation
  - L'exécution d'une transation doit préserver le cohérence de la BD.

Les différentes instructions SQL

- Langage de Définitions de Données (LDD)
  - CREATE TABLE
  - CREATE VIEW
  - ALTER
- Langage de Manipulation De Données (LMD)
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
  - OPEN
  - FETCH
  - CLOSE
- Langage de Contrôle de Données (LCD)
  - GRANT et REVOKE
  - BEGIN et END TRANSACTION
  - COMMIT et ROLLBACK

Administration d’une BD

MySQL

- Même en faisant référence à un type specifique de DB, il y a plusieurs SGBD disponibles.
- Pour cette première partie, nous allons utiliser MySQL (version 5.7.25)

Sécurité des accès

- MySQL implémente un système sophistiqué de contrôle d'accès qui permet de gérer les opérations des clients et d'empêcher les clients non autorisés d'accéder au système
- Lorsqu'un client se connecte au serveur:
  - Vérification de la connexion : un client qui se connecte au serveur doit avoir un nom d'utilisateur et un mot de passe valides, e se connecter d’un hôte valide
  - Vérification de la demande : une fois la connexion établie avec succès, MySQL vérifie, pour chaque instruction émise par le client, si le client dispose des privilèges suffisants pour exécuter cette instruction. MySQL peut vérifier un privilège au niveau de la base de données, de la table et de la colonne.

Création des usagers

- Un usager est crée par la commande CREATE USER

CREATE USER user@host IDENTIFIED BY password ;

- Les mots de passe sont stockées après cryptage
- L'absence de l'hôte signifie « n'importe quel hôte »
- Les guillemets vont utilisés quand une chaîne contient des caractères spéciaux (comme % et _)

```
mysql> CREATE USER momo IDENTIFIED BY 'PasswOrd.';
mysql> CREATE USER 'momo'@'localhost' IDENTIFIED BY
'Password';
```

Création des usagers

- On peut utiliser les métacaractères % (zero ou plus caractères) et _ (n'importe quel caractère), en utilisant des guillemets
- On peut créer plusierus usagers avec un même nom, mais avec differentes permission en fonction de l'hôte de connexion
- La création d'une couple usager+host qui existe déjà déclanche une erreur
- Noms et hôtes doivent être placès séparemment dans guillemets
- Selon l'installation de mySQL, des politiques de mot de passe peuvent être mis en place

Création des usagers

```
mysql> CREATE USER momo@'%' IDENTIFIED BY 'PasswOrd.';
mysql> CREATE USER momo@'%.fr' IDENTIFIED BY
'PasswOrd.';
mysql> CREATE USER 'momo@localhost' IDENTIFIED BY
'PasswOrd.'; # ça marche pas!
```

Permissions des nouveaux usagers

- La commande SHOW GRANTS visualise les permissions d'un usager
- Un usager qui vient d'être crée n'a que la permission USAGE (il peut seulement se connecter)

```
mysql> CREATE USER momo IDENTIFIED BY 'Passw0rd.';
mysql> SHOW GRANTS FOR momo;
+----------------------------------+
| Grants for momo@%                |
+----------------------------------+
| GRANT USAGE ON *.* TO `momo`@`%` |
+----------------------------------+
```

Attention aux métacaractères

- Délimiter une chaîne avec guillemets peut avoir des mauvaises conséquences

```
mysql> CREATE USER ‘momo@localhost’ IDENTIFIED BY ‘PasswOrd.’;
mysql> SHOW GRANTS FOR ‘momo@localhost’;
+----------------------------------------+
| Grants for momo@localhost@%            |
+----------------------------------------+
| GRANT USAGE ON *.* TO `momo@localhost`@`%` |
+----------------------------------------+
```

Sécurité des accès

- La base mysql, automatiquement crée, contient les permissions des usager, organisés dans les tables
  - user (comptes d'utilisateurs / permission à niveau global)
  - db (permissions à niveau de base des données)
  - table_priv et columns_priv (permissions à niveau de table et de colonnes)
  - procs_priv (fonctions d'agrégation / procédures stockées)
- Seul le usager root devrait avoir accès à cette base !

La table «user»

```
mysql> USE mysql;
mysql> DESCRIBE user;
+------------------------+-----------------------+------+-----+---------+-------+
| Field                  | Type                  | Null | Key | Default | Extra |
+------------------------+-----------------------+------+-----+---------+-------+
| Host                   | char(60)              | NO   | PRI |         |       |
| User                   | char(32)              | NO   | PRI |         |       |
| Select_priv            | enum('N','Y')         | NO   |     | N       |       |
| Insert_priv            | enum('N','Y')         | NO   |     | N       |       |
| ...                    | ...                   | ...  | ... | ...     | ...   |
| authentication_string  | text                  | YES  |     | NULL    |       |
| password_expired       | enum('N','Y')         | NO   |     | N       |       |
| password_last_changed  | timestamp             | YES  |     | NULL    |       |
| password_lifetime      | smallint(5) unsigned  | YES  |     | NULL    |       |
| account_locked         | enum('N','Y')         | NO   |     | N       |       |
+------------------------+-----------------------+------+-----+---------+-------+
```

Grant & Revoke

- Permissions
  - Permet de définir des droits d’accès, de modification, de suppression… pour chaque utilisateur
  - Les permissions peuvent être definies aussi en fonction de l’hôte duquel l’utilisateur est connecté
  - Tâche effectuée par l’administrateur de la BD
    - GRANT permet d'accorder des droits à un utilisateur (parfois plusieurs sur certains SGBD)
    - REVOKE permet de retirer des droits à un utilisateur (ou plusieurs sur certains SGBD)

Les permissions MySQL

```
mysql> SHOW PRIVILEGES
```

- SELECT : droit d'effectuer des recherches avec «select»
- INSERT : droit d'effectuer des insertions avec «insert»
- UPDATE : droit d'effectuer des mises à jour avec «update»
- DELETE : droit d'effectuer des destructions d'enregistrement dans des tables
- INDEX : droit de créer ou de détruire des index de table
- ALTER : droit de modifier la structure des tables avec «alter»
- CREATE : droit de créer des bases ou des tables avec «create»
- USAGE : droit de se connecter au serveur, sans rien faire d'autre (utile pour changer le mot de passe de connexion)
- LOCK : droit de verrouiller / déverrouiller des tables

Les permissions MySQL

- DROP : droit de détruire des tables ou des bases
- GRANT : permet d'affecter droits et permissions à un utilisateur
- REFERENCES : droit lié aux «foreign keys»

Privilèges globaux, ne s'appliquent pas à une base particulière

- RELOAD : permission de relancer le serveur mysql et d'écrire les tables sur disque
- SHUTDOWN : droit d'arrêter le serveur mysql
- FILE : droit d'écrire ou lire dans des fichier ASCII avec les commandes «load data» et «into outfile»

SHOW

- SHOW peut être utilisé pour énumérer aussi autres objets :
  - SHOW DATABASES
  - SHOW TABLES

Allouer des permissions: GRANT

- Les permissions de table se manipulent avec les commandes SQL GRANT et REVOKE
- GRANT permet de créer un utilisateur, lui allouer des droits et changer son mot de passe

*GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ON {tbl_name | * | *.* | db_name.*} TO user [IDENTIFIED BY [PASSWORD] 'password']*

```
mysql> GRANT all ON test.* TO momo IDENTIFIED BY ‘PasswOrd.’;
```

```
mysql> GRANT select, insert ON mysql.* TO momo@localhost
IDENTIFIED BY ‘PasswOrd.’;
```

- Les privilèges «file», «reload» et «shutdown» sont globaux pour tout le serveur et non pas pour une base particulière

```
mysql> GRANT file ON *.* TO momo@localhost;
```

Affichage des droits d’accès

- Les exemples précédents insèrent le user «momo» avec son mot de passe, et la machine autorisée, dans la table «user» de la base mysql, avec les droits globaux.

```
mysql> SELECT * FROM user WHERE user='momo';
| localhost | momo  | N | N | N | N | N | N | N
| N | N | Y | N | N | N | N | N | N | N | N
| N | N | N | N | N | N | N | N | N | N |   |
|   |   | 0 | 0 | 0 | 0 | mysql_native_password |
*BE08431E1615FDE206B80F0192AE599DDB88D061
| N | 2019-01-27 23:28:12 | NULL | N |
```

Affichage des droits d’accès

- Les droits de l’usager relatif aux bases sont placés dans la table «db» de la base mysql

```
mysql> SELECT * FROM db WHERE user='momo';
| %         | test  | momo  | Y | Y | Y | Y | Y | Y |
N | Y | Y
| Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| localhost | mysql | momo  | Y | Y | N | N | N | N |
N | N | N
| N | N | N | N | N | N | N | N | N | N |
```

Affichage des droits d’accès

```
mysql> show grants for momo;
+-----------------------------------------+
| Grants for momo@%                       |
+-----------------------------------------+
| GRANT USAGE ON *.* TO 'momo'@'%'        |
| GRANT ALL PRIVILEGES ON `test`.* TO 'momo'@'%' |
+-----------------------------------------+
mysql> show grants for momo@localhost;
+-------------------------------------------------+
| Grants for momo@localhost                       |
+-------------------------------------------------+
| GRANT FILE ON *.* TO 'momo'@'localhost'         |
| GRANT SELECT, INSERT ON `mysql`.* TO 'momo'@'localhost' |
+-------------------------------------------------+
```

Attribution de permissions

- GRANT liste_permissions ON liste_objets TO liste_utilisateurs [WITH GRANT OPTIONS]
  - WITH GRANT OPTION permet de définir si l'utilisateur peut lui-même accorder à un autre utilisateur les permissions qu'on lui accorde sur les éléments
- Authorisation collective
  - PUBLIC en lieu et place de la liste d'utilisateurs permet d'accorder les privilèges sur le ou les objets à l'ensemble des utilisateurs
  - Le mot clé ALL en lieu et place de la liste de permissions permet d'accorder tous les privilèges aux utilisateurs présents dans la liste

Grant

- Exemple

```
mysql> GRANT UPDATE ON etudiants
TO Jerome, Francoise, Georges
WITH GRANT OPTION;
```

  - L’option WITH GRANT OPTION authorise donc plusieurs utilisateurs à accorder des permissions à un même utilisateur
  - Il y a donc des règles à respecter lors du retrait de permissions à un utilisateur

Usages de GRANT

- Allouer des privilèges sur une base

```
mysql> GRANT select ON Ecole.* TO momo@localhost;
```

- Allouer des privilèges sur une table

```
mysql> GRANT update ON Ecole.Etudiants TO momo@localhost;
```

- Allouer des privilèges sur une colonne

```
mysql> GRANT insert (nom, prenom) ON Ecole.Etudiants TO
momo@localhost;
```

Usages de GRANT

- Chaque exécution de GRANT modifie des éventuelles permission déjà alloués
- L'option REQUIRE permet d'allouer une permission seulement au cas où l'usager utilise une spécifique connexion sécurisée

```
mysql> GRANT select ON Ecole.* TO momo@localhost
REQUIRE ssl;
```

Usages de GRANT

- Changer le mot de passe d'un utilisateur

```
mysql> GRANT USAGE TO momo@localhost IDENTIFIED BY
'PasswOrd';
```

- Créer un super-utilisateur (~root)

```
mysql> GRANT ALL PRIVILEGES ON *.* TO momo@localhost
IDENTIFIED BY 'PasswOrd' WITH GRANT OPTION;
```

- NB : ne pas créer (trop de) super-utilisateurs !

Roles

- Les nouvelles versions de MySQL (mais pas celle que nous utilisons) implementent le concept de « rôle »
- Un rôle corresponds à un usager abstrait (développeur, testeur, ...) et est defini en termes des privilèges correspondants
- Les rôles sont attribués aux usagers effectifs
- Ça rend plus facile la gestion les usagers

Ôter des permissions: REVOKE

- REVOKE permet de supprimer des droits pour des utilisateurs

REVOKE priv_type [column_list] [, priv_type [column_list]] ON
{ tbl_name | * | *.* | db_name.* } FROM user [, user]

```
mysql> REVOKE ALL ON *.* FROM momo;
mysql> REVOKE alter, delete, drop ON mysql.* FROM momo;
mysql> REVOKE GRANT OPTION on mysql.* FROM
momo@localhost;
mysql> REVOKE select ON mysql.* FROM
‘momo’@’pcml.dom.fr’
```

REVOKE

- Lorsque on retire un droit D à un utilisateur U, il faut que ce droit soit rétiré aux utilisateurs auquels U a accordé D
- Un utilisateur peut avoir reçu un droit de plusieurs utilisateurs
- Il s’agit donc de retirer les droits des utilisateurs l’ayant obtenu de quelqu’un qui ne l’a plus, en sachant qu’il peut l’avoir de plusierus personnes simultanément...
- La clause REVOKE étant implémentée différemment selon le SGBD, il s’agit de consulter la documentation

Enlever un utilisateur

- À partir de la version 4.1.1 de mySQL il faut enlever tous les privilèges avant de détruire un utilisateur

```
mysql> SHOW GRANTS FOR momo@localhost;
mysql> REVOKE ALL PRIVILEGES FROM momo@localhost;
mysql> DROP USER momo;
```

- L'enlevement d'un usager qui n'existe pas déclenche une erreur, sauf si l'option IF EXISTS ait été spécifiée

Enlever un utilisateur

- Pour les versions antérieures à 4.1.1 il faut agir directement sur la table mysql.user
- Chaque modification via mysql.user requiert de récharger les permissions en mémoire

```
mysql> DELETE FROM mysql.user WHERE user='momo' AND
host='localhost';
mysql> FLUSH PRIVILEGES;
```

FLUSH PRIVILEGES

- La commande FLUSH PRIVILEGES récharge en memoire la table des privilèges
- Il est théoriquement possible ajouter des usager ou changer/ôter leurs permissions en exécutant des query sur les tables de la base mysql...
- ...mais ça a un coût plus haut en termes de temps d’execution

Changer le mot de passe

- Via GRANT (déprécié)

```
mysql> GRANT usage TO momo IDENTIFIED BY ‘PasswOrd.’;
```

- Via UPDATE (requiert aussi FLUSH PRIVILEGES)

```
mysql> UPDATE user SET
authentication_string=PASSWORD(‘PasswOrd.’) WHERE
user=’momo’ AND host=’localhost’;
```

- Via ALTER USER

```
mysql> ALTER USER momo IDENTIFIED BY ‘PasswOrd.’;
```

- Via SET PASSWORD

```
mysql> SET PASSWORD for momo=’PasswOrd.’;
```

Création d'une base

- Via la commande CREATE DATABASE (ou CREATE SCHEMA)

```
mysql> CREATE DATABASE Ecole;
```

- L'option IF NOT EXISTS évite l'eventuel erreur qui aurait lieu au cas où la base n'existait pas

```
mysql> CREATE DATABASE IF NOT EXISTS Ecole;
```

- USE permet de selectionner et utiliser une base

```
mysql> USE Ecole;
```

Création d’une table

- Via CREATE TABLE

CREATE TABLE Nom_de_la_table (Nom_de_colonne1 Type_de_donnée, Nom_de_colonne2 Type_de_donnée, ...);

- On peut utiliser l’option IF NOT EXISTS

Types de données

Type de donnée	Syntaxe	Description
Alphanumérique	CHAR(n)	Chaine de longeur fixe n<16383
Alphanumérique	VARCHAR(n)	Chaîne de longeur maximale n<16383
Numérique	NUMBER(n[, d])	Nombre (n chiffres, d après virgule)
Numérique	SMALLINT	Entier signé (16 bit)
Numérique	INTEGER	Entier signé (32 bits)
Numérique	FLOAT	Nombre à virgule flottante
Horaire	DATE	Date (JJ/MM/AA)
Horaire	TIME	Heure (HH:MM:SS.CC)
Horaire	TIMESTAMP	Date et heure
Texte	TEXT	Chaîne de longeur maximale < 65535
Blob	BLOB	Objet binaire
...	...	...

Contraintes d’integrité

- Clauses permettant de contraindre la modification des tables, afin que les données dans la base soient conformes à règles attendues
- Doivent être exprimés dans la création grâce à mots clé :
  - DEFAULT
  - NOT NULL : champ obligatoire
  - UNIQUE : chaque valeur doit être différent
  - CHECK

Default

- Valeur par défaut
  - Utiliser le mot clé DEFAULT <valeur>
  - Garantit que le champ n’est pas vide
  - Valeur :
    - constante numérique
    - constante alphanumérique (chaîne de caractères)
    - le mot clé USER (nom de l'utilisateur)
    - le mot clé NULL
    - le mot clé CURRENT_DATE (date de saisie)
    - le mot clé CURRENT_TIME (heure de saisie)
    - le mot clé CURRENT_TIMESTAMP (date et heure de saisie)

Contraintes

- Condition sur une colonne
  - Utiliser le mot clé CHECK(<condition logique>)
  - Si le valeur inséré est différent de NULL, le SGBD contrôle la condition et refuse de modifier la table si elle n'est pas satisfaite
  - La condition peut contenir des ordres SELECT
- Nommer une contrainte
  - Utiliser le mot clé CONSTRAINT <nom contrainte>
  - Lorsque une contrainte n'est pas respectée, le SGBD affiche son nom
  - Des noms (incomprehensibles) sont donnés par defaut aux contraintes par le SGBD

Contraintes

```
mysql> CREATE TABLE Clients(
    id INTEGER UNIQUE NOT NULL,
    nom CHAR(30) NOT NULL,
    prenom CHAR(30) NOT NULL,
    age INTEGER CONSTRAINT valid_age CHECK (age < 100),
    email CHAR(50) NOT NULL CHECK(email LIKE '@%@')
);
```

Contraintes

```
mysql> CREATE TABLE Clients(
    id INTEGER UNIQUE NOT NULL,
    nom CHAR(30) NOT NULL,
    prenom CHAR(30) NOT NULL,
    age INTEGER,
    email CHAR(50) NOT NULL,
    CHECK(age < 100 AND email LIKE '@%@')
);
```

Clés

- Clés primaires
  - PRIMARY KEY (colonne1, colonne2, ...)
  - Signifie que les valeurs ne peuvent pas être nulles et que deux lignes ne peuvent pas avoir simultanément la même valeur
- Clés étrangères
  - FOREIGN KEY (colonne1, colonne2, ...)
  - REFERENCES Nom_table_étrangère (colonne1, colonne2, ...)

Contraintes

```
mysql> CREATE TABLE Clients(
    id INTEGER UNIQUE NOT NULL,
    nom CHAR(30) NOT NULL,
    prenom CHAR(30) NOT NULL,
    age INTEGER,
    email CHAR(50) NOT NULL,
    CHECK(age < 100 AND email LIKE '@%@')
    PRIMARY KEY(id)
);
```

Vues

- Ce sont des tables virtuelles, dont les données ne sont pas stockés dans une table de la BD, et dans lesquelles il est possible de rassembler des informations provenant de plusieurs tables
- On parle de «vue» car il s'agit d'une représentations des données dans le but d'une exploitation visuelle
- Les données présentes dans une vue sont définies grâce à une clause SELECT

```
mysql> CREATE VIEW Gros_buveurs AS
       SELECT NB, Nom, Prenom FROM Buveurs, Abus
       WHERE Buveurs.NB=Abus.NB AND Abus.quantite>100;
```

Vues

- Intérêts d’une vue
  - Une vue représente une sorte d’intermédiaire entre la base des données et l’utilisateur
  - Une selection de données à afficher
  - Une restriction à l’accès aux tables pour l’utilisateur, c’est-a-dire une sécurité accrue des données
  - Un regroupement d’informations au sein d’une entité

Index

- C’est un objec complémentaire à la BD «indexant» certaines colonnes d’une table dans le but de reduire le temps d’acces aux données
- Index mis à jour à chaque modification de table
- C’est coûteux en termes de memoire
- Peut alourdir le temps de traitement d’un SGBD lors de la sasie des données
- Donc, la création d’un index doit être justifiée, et les colonnes sur lesquelles il porte doivent être judicieusement choisies
- Certains SGBD créent automatiquement un index lorsq’une clé primaire est définie

Création d’index

- La création d’un index se fait grâce à la clause INDEX précédée de la clause CREATE
- Elle permet de définir un index designé par son nom, portant sur certaines colonnes d’une table
- CREATE [UNIQUE] INDEX nom_index ON nom_table (nom_colonne [ASC|DESC], ...)
- L’option UNIQUE permet de définir la présence ou non des doublons pour les valeurs de la colonne
- Les options ASC/DESC permettent de définir un ordre de classement des valeurs présents dans la colonne

Suppression d’éléments: DROP

- Vue : DROP VIEW nom_vue
- Index : DROP INDEX nom_index
- Table (structure + données): DROP TABLE nom_table
- Table (données) : TRUNCATE TABLE nom_table
- Colonne : ALTER TABLE nom_table DROP nom_colonne
  - Si la colonne ne fait pas part d’une vue,
  - ni d’un index,
  - ni elle fait part d’une contrainte d’integrité
- Base : DROP DATABASE

Modifier la structure d'une table

- En ajoutant une colonne (donc modifiant le schéma de la relation)

```
mysql> ALTER TABLE Ecole ADD (age INTEGER);
```

- En ajoutant une contrainte d'integrité

```
mysql> ALTER TABLE Ecole ADD CONSTRAINT valid_age
CHECK (age < 100);
```

- En ajoutant une clé primaire ou étrangère

```
mysql> ALTER TABLE Ecole ADD PRIMARY KEY (id)
```

Modifier la structure d’une table

- Modifier une colonne

```
mysql> ALTER TABLE Ecole MODIFY age SMALLINT;
```

- Rénommer une table

```
mysql> RENAME Ecole AS Etudiants;
```

TD

- Création d’une base
- Création d’un usager administratif pour la base
- Création de tables avec contraintes et clés
- Création d’usagers avec differents privilèges
- Sasie des données
- Modification de la structure des tables
- Creation d’un index
- Modification des privilèges et du mot de passe
- Enlèvement de privilèges à un usager

TP

- Installation logiciel (MAMP)
- Création d’une base
- Création d’un usager administratif pour la base
- Création de tables avec contraintes et clés
- Création d’usagers avec differents privilèges
- Sasie des données
- Modification de la structure des tables
- Creation d’un index
- Modification des privilèges et du mot de passe
- Enlèvement de privilèges à un usager

TP : Mettre des droits sur la Base

- *Créer une base BASE et créer plusieurs utilisateurs. Affecter des droits différents à ces utilisateurs :*

  - *Un utilisateur qui aura tous les droits sur BASE*
  - *Un utilisateur qui n'aura que le droit de sélectionner*
  - *Un utilisateur qui aura le droit insert et update dans les tables de BASE*
  - *Modifier la password de l'un des utilisateurs*
  - *Enlever le droit de update à un utilisateur qui l'a*

Les jointures

- Quand on précise plusieures tables dans une clause FROM, on obtient leur produit cartesien
- Ce produit offre en général peu d’intérêt
- Il est bien plus intéressant de joindre les informations des diverses tables, recollant les lignes suivant les valeurs qu’elles ont dans certaines colonnes

Une jointure simple

Employes

Employé	Projet
Marc Vert	A
Julie Blanc	A
Cédric Bleu	B

Projets

Identifiant	Nom projet
A	Pages Web
B	SGBD

Employes ⋈ Projet=Identifiant **Projets**

Employé	Projet	Identifiant	Nom projet
Marc Vert	A	A	Pages Web
Julie Blanc	A	A	Pages Web
Cédric Bleu	B	B	SGBD

Jointure incomplète

- Certaines valeurs parmi les attributs commun ne coïncident pas
- Les n-uplets correspondants ne participent pas à former la relation jointe (*dangling tuples*)

Projets

Identifiant	Nom projet
A	Pages Web
B	SGBD
C	Résau

Employes ⋈ Projet=Identifiant **Projets**

Employé	Projet	Identifiant	Nom projet
Marc Vert	A	A	Pages Web
Julie Blanc	A	A	Pages Web
Cédric Bleu	B	B	SGBD

Jointure incomplète

- Certaines valeurs parmi les attributs commun ne coïncident pas
- Les n-uplets correspondants ne participent pas à former la relation jointe (*dangling tuples*)

Projets

Identifiant	Nom projet
X	Pages Web
Y	SGBD
Z	Résau

Employes ⋈ Projet=Identifiant **Projets**

Employé	Projet	Identifiant	Nom projet

Jointures

- Tables Emp(nome, dept) et Dept(nomd, dept)
- SELECT nome, nomd FROM Emp JOIN Dept ON Emp.dept=Dept.dept;
- La clause FROM indique de ne conserver dans le produit cartesien que les éléments pour lesquels le département est le même dans les deux tables
- On obtient donc une jointure entre les tables d’après le département
- Par opposition aux jointures extérnes que on va bientôt étudier, on peut ajouter le mot clé INNER
- SELECT nome, nomd FROM Emp INNER JOIN Dept ON Emp.dept=Dept.dept;

Les Jointures

Jointure interne	`SELECT ...`
	`FROM <table gauche>`
	`[INNER]JOIN <table droite>`
	`ON <condition de jointure>`

Jointure externe	`SELECT ...`
	`FROM <table gauche>`
	`LEFT | RIGHT | FULL OUTER JOIN <table droite>`
	`ON condition de jointure`

Jointure naturelle	`SELECT ...`
	`FROM <table gauche>`
	`NATURAL JOIN <table droite>`
	`[USING <noms de colonnes>]`

Jointure croisée	`SELECT ...`
	`FROM <table gauche>`
	`CROSS JOIN <table droite>`

Jointure d'union	`SELECT ...`
	`FROM <table gauche>`
	`UNION JOIN <table droite>`

Les Jointures

- Jointure Naturelle

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT NATURAL JOIN T_TELEPHONE [USING (CLI_ID)]
```

  - Avantage : évite de préciser le(s) attributs de jointures (le nom doit être identique !)
  - On peut préciser les attributs dans la clause using
- Jointure Interne (Par défaut)

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C [INNER] JOIN T_TELEPHONE T
    ON C.CLI_ID = T.CLI_ID
```

  - Le mot clé INNER est facultatif
  - Principe : fait correspondre 1 et 1 seul client à 1 et 1 seul Num_Tel

Les Jointures

- Problème : quel est la requête SQL qui permet
  - TOUS les clients avec leur Num_Tel s'ils en ont 1
  - TOUS les Num_Tel et les clients correspondant s'ils sont encore connus dans la BD ...
- Jointures Externes

```
SELECT ...
FROM    <table_gauche>
        LEFT | RIGHT | FULL OUTER JOIN <table_droite>
        ON <condition de jointure>
```

  - LEFT OUTER JOIN
    - recherche toutes les valeurs satisfaisant la condition de jointure, puis on rajoute toutes les lignes de la table *Gauche* qui n'ont pas été prises en compte
  - RIGHT OUTER JOIN
    - recherche toutes les valeurs satisfaisant la condition de jointure, puis on rajoute toutes les lignes de la table *Droite* qui n'ont pas été prises en compte
  - FULL OUTER JOIN
    - recherche toutes les valeurs satisfaisant la condition de jointure, puis on rajoute toutes les lignes de la table *Gauche* et *Droite* qui n'ont pas été prises en compte

Les Jointures

- Jointure Externe

```
SELECT CLI_NOM, TEL_NUMERO
FROM    T_CLIENT C
        LEFT OUTER JOIN T_TELEPHONE T
        ON C.CLI_ID = T.CLI_ID
```

CLI_NOM	TEL_NUMERO
DUPONT	01-44-28-52-50
DUPONT	05-59-45-72-42
MARTIN	01-47-66-29-55
BOUVIER	NULL
DUBOIS	04-66-62-95-64
DREYFUS	04-92-19-18-58
FAURE	NULL
LACOMBE	NULL
DUHAMEL	01-54-11-43-89
DUHAMEL	01-55-60-93-81
...	

- Jointure Interne

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C JOIN T_TELEPHONE T
        ON C.CLI_ID = T.CLI_ID
```

CLI_NOM	TEL_NUMERO
DUPONT	01-44-28-52-50
MARTIN	01-44-22-56-21
DUHAMEL	01-54-11-43-89
DUPONT	05-59-45-72-42
MARTIN	01-47-66-29-55
DUBOIS	04-66-62-95-64
DREYFUS	04-92-19-18-58
DUHAMEL	01-55-60-93-81
PHILIPPE	01-48-44-86-19
DAUMIER	01-48-28-17-95
...	

Les Jointures

- Jointure Croisée
  - Produit cartésien entre 2 tables

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT CROSS JOIN T_TELEPHONE
```

- Jointure d'Union
  - Fait l'union structurelle de 2 tables qui n'ont pas forcément quelque chose en commun

```
SELECT *
FROM   TORCHON UNION JOIN SERVIETTE
```

Domaines

- Soit la table Cours(NumC, Semestre, Nbr_inscrits)
- On veut que l'attribut semestre soit de la forme 'SEM1_ _' ou 'SEM2_ _' par exemple, le premier semestre de 2001 sera saisi 'SEM101'
- Instructions SQL

```
CREATE DOMAIN SEM CHECK ((VALUE LIKE 'SEM1_ _') OR
                         (VALUE LIKE 'SEM2_ _'))

CREATE TABLE Cours(
   NumC Integer,
   Semestre SEM,
   Nbr_inscrits Integer)
```

Les assertions(1)

- Les assertions sont des contraintes qui peuvent porter sur plusieurs tables
- Elles doivent être vérifiées par le SGBD à chaque fois qu'une des tables mentionnées est modifiée
- CREATE ASSERTION <nom> CHECK (<condition>)
- La condition doit être vraie lors de la création, sinon l'assertion n'est pas créée

Les assertions(2)

- Cours(NumC, Sem, Nb_inscrits)
  Inscription(NumEt, NumC, Sem)
- On veut que Nb_inscrits reflète exactement le nombre d'inscrits

```
CREATE ASSERTION NB_INSCR CHECK(
NOT EXISTS(select * from Cours C
      where C.Nb_inscrits <> (select COUNT(*)
                        FROM Inscription i
                        WHERE i.NumC=C.NumC AND   i.Sem=C.Sem
                        GROUP BY (i.NumC, i.Sem))
                ));
SET ASSERTION NB_INSCR DEFERRED;
```

- Il n'y pas beaucoup de SGBD qui utilisent les assertions

Les Triggers

- Assertions
  - Les assertions décrivent des contraintes qui doivent être satisfaites par la BD à tout moment.
  - Une assertion est vérifiée par le SGBD à chaque fois qu'une des tables qu'elle mentionne est modifiée
  - Si une assertion est violée, alors la modification est rejetée
- Triggers
  - Les triggers spécifient explicitement à quel moment doivent-ils être vérifiés (i.e. insert, delete, update)
  - Les triggers sont des règles ECA : Evénement, Condition, Action
  - Quand E a lieu, si C est vérifiée alors A est exécutée

Triggers – activation

- Un trigger peut être activé BEFORE/AFTER/INSTEAD OF un événement d'activation qui peut être une des commandes insert/delete/update dans une ou plusieurs tables

- L'action peut faire référence à l'ancienne et/ou à la nouvelle valeur des tuples insérés/supprimés/modifiés par l'action

  CREATE TRIGGER exemple1
  AFTER UPDATE OF solde ON compte

- La condition est exprimée par la clause WHEN

  WHEN (solde < 0)

- La clause WHEN peut être n'importe quelle condition booléenne

Trigger - Action

- Le concepteur a le choix entre spécifier l'action à exécuter soit
  - Une fois, pour chaque tuple modifié/inséré/supprimé, ou
  - Une fois pour tous les tuples modifiés lors d'une seule opération

```
FOR EACH ROW
UPDATE compte
    SET date_débit = SYSDATE
    WHERE Compte.IdC = OldTuple.IdC
```

- L'action peut être n'importe quelle code écrit dans le langage associé du SGBD (E.g sous Oracle, c'est PL/SQL, Java, C ; pour H2 c'est Java)

Triggers - row level

- Les insertions créent de nouveaux tuples
  - On ne peut faire référence à l'ancienne valeur
- Les modifications font passer de "old" à "new"
  - On peut faire référence à ces deux valeurs
- Avec les suppressions, on ne peut faire référence aux nouvelles valeurs.

- Les nouvelles/anciennes valeurs ne peuvent être accédées qu'en mode ligne (Row level).

Exemple

```
CREATE TRIGGER Trig_Inscrits
After INSERT ON INSCRPTION
FOR EACH ROW //pour chaque nouvel inscrit
BEGIN
   Update Cours SET Nb_Inscrits=Nb_Inscrits+1
   where Cours.Sem=new.Sem AND
   Cours.NumC=new.Numc;
END;
```

Exemple

- On veut garder la trace des opérations effectuées par les utilisateurs

```
CREATE TRIGGER TRIG_INSC
AFTER INSERT ON Inscrits
    BEGIN
    Insert into tab_logs(Utilisateur, operation, heure) values
    (USER, 'insertion', SYSDATE);
    END;
```

Triggers: Activation

- Activation
  - BEFORE – la condition WHEN est testée sur l'état avant l'arrivée de l'événement déclencheur
    - Si la condition est vraie, l'action est exécutée, ensuite
    - L'événement déclencheur est réalisé
  - INSTEAD OF – l'action est exécutée si la condition est vérifiée,
    - L'événement déclencheur n'est jamais effectué.
  - AFTER – l'action est exécutée si la condition WHEN est vérifiée après la réalisation de l'événement déclencheur.

Exemples

```
CREATE TRIGGER compter
BEFORE INSERT ON Emp
DECLARE nombre number;
BEGIN
   nombre := select COUNT(*) from EMP;
   dbms_output.put(‘On passera de' || :nombre);
   dbms_output.put(‘a ' || :nombre+1);
END;
```

Exemples

```
CREATE TRIGGER compte
AFTER INSERT ON Emp
DECLARE nombre number;
BEGIN
   nombre := select COUNT(*) from EMP;
   dbms_output.put(‘On est passé de'|| :nombre-1);
   dbms_output.put(‘a ' || :nombre);
END;
```

Exemples

```
CREATE TRIGGER compte
AFTER INSERT ON Emp
WHEN ((select COUNT(*) from EMP)=101)
BEGIN
   dbms_output.put('On vient de passer le cap');
   dbms_output.put('de 100 employés');
END;
```

Exemple

- Emp(Nom, Salaire)
  - Le salaire d'un employé ne peut baisser
  - Cette contrainte ne peut être prise en compte qu'en utilisant les triggers ou bien les procédure stockées

```
Create procedure mod_sal(in nomE, in salE)
a:=select Salaire from Emp where Nom=:nomE;
si Salaire < salE alors
 update Emp set Salaire = :salE where
                    Nom= :nomE;
```

Exemple

```
CREATE TRIGGER Mod_Sal
AFTER UPDATE OF Emp ON Salaire
FOR EACH ROW
BEGIN
   If new.Salaire < old.Salaire Rollback;
END;
```

Conception des triggers

- Ne pas utiliser les triggers pour gérer des contraintes qui sont facilement gérables par le système. Exemple : prise en compte des contraintes d'unicité de clé.
- Limiter la taille des triggers. S'il vous faut plus de 60 lignes, il est préférable de définir une procédure stockée et l'appeler par le trigger.
- Utiliser les triggers pour les opérations qui ne dépendent pas de l'utilisateur ni de l'application
- **Eviter les triggers récursifs.** E.g, un trigger déclenché lors d'une modification de la table employé et dont l'action va elle aussi modifier cette table

Fonction agrégate / procédure stockée

- Permet de définir une fonction utilisée dans les requêtes SQL
- Avec H2
  - Créer la procédure stockée “isPrime” en Java et l’enregistrer dans un fichier (Function2.java par exemple)
  - L’enregistrer ds le même répertoire que h2.jar (C:\Program Files\H2\bin)
  - Ouvrir une invite de commande
  - Se placer dans le répertoire où se trouve h2.jar
  - Compiler
    - javac –cp “h2.jar” Function2.java
  - Lancer H2 par la ligne de commande
    - java -cp "h2.jar;%H2DRIVERS%;%CLASSPATH%;./" org.h2.tools.Console
  - Créer un lien avec la méthode contenue dans la classe
    - CREATE ALIAS IS_PRIME FOR "Function2.isPrime"

```java
import java.math.BigInteger;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;

import org.h2.tools.SimpleResultSet;

public class Function {
    public static boolean isPrime(int value) {
        return new BigInteger(String.valueOf(value)).isProbablePrime(100);
    }
    /**
     * Execute a query.
     * @param conn the connection
     * @param sql the SQL statement
     * @return the result set
     */
    public static ResultSet query(Connection conn, String sql) throws SQLException {
        return conn.createStatement().executeQuery(sql);
    }
    /**
     * Creates a simple result set with one row.
     * @return the result set
     */
    public static ResultSet simpleResultSet() throws SQLException {
        SimpleResultSet rs = new SimpleResultSet();
        rs.addColumn("ID", Types.INTEGER, 10, 0);
        rs.addColumn("NAME", Types.VARCHAR, 255, 0);
        rs.addRow(new Object[] { new Integer(0), "Hello" });
        return rs;
    }
    public static ResultSet getMatrix(Connection conn, Integer id) throws SQLException {
        SimpleResultSet rs = new SimpleResultSet();
        rs.addColumn("X", Types.INTEGER, 10, 0);
        rs.addColumn("Y", Types.INTEGER, 10, 0);
        if (id == null) {
```

Optimisation

- Optimisation
  - De modélisation
  - De requêtes

Optimisation de modélisation

- Ce qui fait gagner du temps
  - Normalisez vos données, entités et relations
  - Standardisez le format et le type de vos colonnes
  - Utiliser des clefs composée d'une colonne unique d'un type 32 bits et purement informatique (un entier c'est parfait)
  - Évitez les colonnes nullables surtout si elle doivent être calculées
  - Préférez les types fixe (CHAR au lieu de VARCHAR) pour les colonnes fréquemment sollicitées en recherche et jointure
  - Utilisez des modèles performants pour vos relations complexes (héritage, arbres....)
  - Indexez l'essentiel, pas le superflu
  - Ajoutez une table des dates plutôt que d'utiliser des fonctions de calcul temporel
  - Dénormalisez vos relations lorsque tout le reste à échoué !

Optimisation de requêtes

- SQL : Langage déclaratif
  - n’indique ni les algorithmes à appliquer, ni les chemins d’accès aux données
  - Le SGBD fait un choix et les combine de manière à obtenir les meilleurs performances
  - L’optimiseur de requêtes joue un rôle extrêmement important

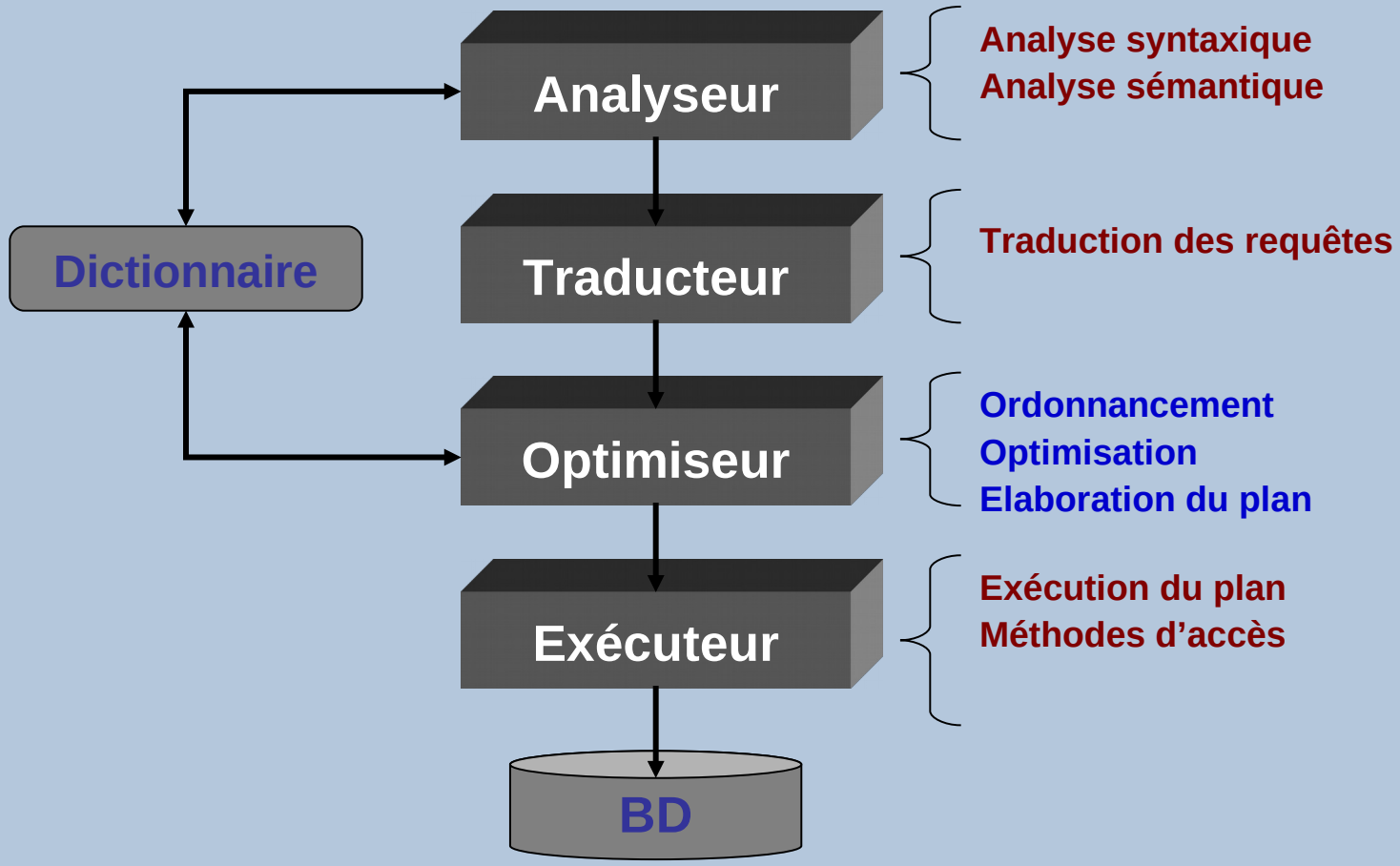
- Impossible de trouver en un temps raisonnable l’algorithme *optimal* pour exécuter une requête donnée

Optimisation de requêtes

- Objectif
  - optimiser, mais éviter de consacrer des ressources considérables à l'optimisation
- Démarche
  - comprendre les mécanismes d'exécution et d'optimisation pour distinguer les goulets d'étranglements
- Stratégies
  - Plans d'exécution : SQL → opérations sur tables
  - Algorithmes d'évaluation
  - Utilisations d'index qui conditionnent les temps de réponse

Optimisation de requêtes

- Étapes du traitement



Optimisation de requêtes

- Paramètres utilisés
  - Schéma de la BD
    - schémas des tables et des chemins d'accès
  - Statistiques
    - taille des tables, des index, distribution des valeurs
  - Caractéristiques des algorithmes mis en œuvre
- Optimisation statique vs dynamique
  - Statique
    - décomposition et optimisation n'ont lieu qu'une seule fois
    - sans tenir compte des modifications intervenues dans la BD
  - Dynamique
    - recherche de la meilleure stratégie à chaque exécution de la requête
    - au dépend des performances
  - Hybride
    - Mise à jour de l'optimisation si les statistiques indiquent des changements importants

Optimisation de requêtes

- Mesure du coût
  - Critère d'estimation : Nombre de lectures/écritures de pages sur disque
  - Amener en mémoire centrale des enregistrements coûte beaucoup plus cher que de les traiter en mémoire
  - Coût d'écriture du résultat d'une opération sur disque non pris en compte
    - indépendant de l'algorithme
    - résultats intermédiaires pas nécessairement réécrits sur disque
  - Paramètres pour l'estimation du coût d'une opération
    - Nombre de pages en mémoire destinées à contenir les entrées d'une opération et les résultats intermédiaires
    - Nombre de pages sur disque occupées par une relation en entrée d'une opération
    - Nombre d'enregistrements d'une table

Optimisation de requêtes

- Optimisation algébrique

1. **Commutativité des jointures** :
   $R \bowtie S \equiv S \bowtie R$
2. **Associativité des jointures** :
   $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
3. **Regroupement des sélections** :
   $\sigma_{A='a' \wedge B='b'}(R) \equiv \sigma_{A='a'}(\sigma_{B='b'}(R))$
4. **Commutativité de la sélection et de la projection**
   $\pi_{A_1,A_2,...A_p}(\sigma_{A_i='a}(R)) \equiv \sigma_{A_i='a}(\pi_{A_1,A_2,...A_p}(R)), i \in \{1,...,p\}$
5. **Commutativité de la sélection et de la jointure.**
   $\sigma_{A='a'}(R(...A...) \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$
6. **Distributivité de la sélection sur l'union.**
   $\sigma_{A='a'}(R \cup S) \equiv \sigma_{A='a'}(R) \cup \sigma_{A='a'}(S)$
   NB : valable aussi pour la différence.
7. **Commutativité de la projection et de la jointure**
   $\pi_{A_1...A_pB_1...B_q}(R \bowtie_{A_i=B_j} S) \equiv$
   $\pi_{A_1...A_p}(R) \bowtie_{A_i=B_j} \pi_{B_1...B_q}(S))\ i \in \{1,..,p\}, j \in \{1,...,q\}$
8. **Distributivité de la projection sur l'union** $\pi_{A_1A_2...A_p}(R \cup S) \equiv \pi_{A_1A_2...A_p}(R) \cup \pi_{A_1A_2...A_p}(S)$

Optimisation de requêtes

- Optimisation physique
  - Recherche d'un **plan d'exécution physique** combinant des opérateurs physiques (chemins d'accès et traitements de données)
  - Dépend de
    - des chemins d'accès qui restreignent le choix d'opérations physiques pour une opération logique donnée
    - des statistiques
    - du nombre de pages en mémoire centrale
  - SQL : instruction "Explain Plan"
- Algébrique/Physique
  - Une opération algébrique peut donner plusieurs opérations physiques
    - Exemple : jointure par boucles imbriquées
  - Plusieurs opérations algébriques peuvent être implantées par une seule opération physique
    - Exemple : sélection et projection peuvent être réalisées par un seul parcours séquentiel

Optimisation de requêtes

http://sqlpro.developpez.com/cours/optimiser/

N°	ÉVITEZ	PRÉFÉREZ
1	**évitez d'employer l'étoile dans la clause SELECT...** SELECT * FROM T_CLIENT	**...préférez nommer les colonnes une à une** SELECT CLI_ID, TIT_CODE, CLI_NOM, CLI_PRENOM, CLI_ENSEIGNE FROM T_CLIENT
2	**évitez d'employer DISTINCT dans la clause SELECT...** SELECT DISTINCT CHB_NUMERO, CHB_ETAGE FROM T_CHAMBRE	**...lorsque cela n'est pas nécessaire** SELECT CHB_NUMERO, CHB_ETAGE FROM T_CHAMBRE
3	**n'employez pas de colonne dans la clause SELECT... de la sous requête EXISTS...** SELECT CHB_ID FROM T_CHAMBRE T1 WHERE NOT EXISTS (SELECT CHB_ID FROM TJ_CHB_PLN_CLI T2 WHERE PLN_JOUR = '2000-11-11' AND T2.CHB_ID = T1.CHB_ID)	**...utilisez l'étoile ou une constante** SELECT CHB_ID FROM T_CHAMBRE T1 WHERE NOT EXISTS (SELECT * FROM TJ_CHB_PLN_CLI T2 WHERE PLN_JOUR = '2000-11-11' AND T2.CHB_ID = T1.CHB_ID)
4	**évitez de compter une colonne...** SELECT COUNT (CHB_ID) FROM T_CHAMBRE	**...quand-il suffit de compter les lignes** SELECT COUNT (*) FROM T_CHAMBRE

Optimisation de requêtes

5	**évitez d'utiliser le LIKE...** SELECT * FROM T_CLIENT WHERE CLI_NOM LIKE 'D%'	**...si une fourchette de recherche le permet** SELECT * FROM T_CLIENT WHERE CLI_NOM BETWEEN 'D' AND 'E '
6	**évitez les jointures dans le WHERE...** SELECT * FROM T_CLIENT C, T_FACTURE F WHERE EXTRACT(YEAR FROM F.FAC_DATE) = 2000 AND F.CLI_ID = C.CLI_ID	**...préférez l'opérateur normalisé JOIN** SELECT * FROM T_CLIENT C JOIN T_FACTURE F ON F.CLI_ID = C.CLI_ID WHERE EXTRACT(YEAR FROM F.FAC_DATE) = 2000
7	**évitez les fourchettes < et > pour des valeurs discrètes...** SELECT * FROM T_FACTURE WHERE FAC_DATE > '2000-06-18' AND FAC_DATE < '2000-07-15'	**...préférez le BETWEEN** SELECT * FROM T_FACTURE WHERE FAC_DATE BETWEEN '2000-06-18' AND '2000-07-14'
8	**évitez le IN avec des valeurs discrètes recouvrantes...** SELECT * FROM T_CHAMBRE WHERE CHB_NUMERO IN (11, 12, 13, 14)	**...préférez le BETWEEN** SELECT * FROM T_CHAMBRE WHERE CHB_NUMERO BETWEEN 11 AND 14

Optimisation de requêtes

9	**évitez d'employer le DISTINCT...** SELECT DISTINCT CLI_NOM, CLI_PRENOM FROM T_CLIENT C JOIN TJ_CHB_PLN_CLI J ON C.CLI_ID = J.CLI_ID WHERE PLN_JOUR = '2000-11-11'	**...si une sous requête EXISTS vous offre le dédoublonnage** SELECT CLI_NOM, CLI_PRENOM FROM T_CLIENT C WHERE EXISTS (SELECT * FROM TJ_CHB_PLN_CLI J WHERE C.CLI_ID = J.CLI_ID AND PLN_JOUR = '2000-11-11')
10	**évitez les sous requêtes...** SELECT CHB_ID FROM T_CHAMBRE WHERE CHB_ID NOT IN (SELECT CHB_ID FROM TJ_CHB_PLN_CLI WHERE PLN_JOUR = '2000-11-11')	**...quand vous pouvez utiliser les jointures** SELECT DISTINCT C.CHB_ID FROM T_CHAMBRE C LEFT OUTER JOIN TJ_CHB_PLN_CLI P ON C.CHB_ID = P.CHB_ID AND PLN_JOUR = '2000-11-11' WHERE P.CHB_ID IS NULL
11	**évitez les sous requêtes avec IN...** SELECT CHB_ID FROM T_CHAMBRE WHERE CHB_ID NOT IN (SELECT CHB_ID FROM TJ_CHB_PLN_CLI WHERE PLN_JOUR = '2000-11-11')	**...lorsque vous pouvez utiliser EXISTS** SELECT CHB_ID FROM T_CHAMBRE T1 WHERE NOT EXISTS (SELECT * FROM TJ_CHB_PLN_CLI T2 WHERE PLN_JOUR = '2000-11-11' AND T2.CHB_ID = T1.CHB_ID)

Optimisation de requêtes

12	**transformez les COALESCE...** SELECT LIF_ID, (LIF_QTE * LIF_MONTANT) * (1 - COALESCE(LIF_REMISE_POURCENT, 0)/100) - COALESCE(LIF_REMISE_MONTANT, 0) AS TOTAL_LIGNE FROM T_LIGNE_FACTURE	**...en UNION** SELECT LIF_ID, (LIF_QTE * LIF_MONTANT) FROM T_LIGNE_FACTURE WHERE LIF_REMISE_POURCENT IS NULL AND LIF_REMISE_MONTANT IS NULL UNION SELECT LIF_ID, (LIF_QTE * LIF_MONTANT) - LIF_REMISE_MONTANT FROM T_LIGNE_FACTURE WHERE LIF_REMISE_POURCENT IS NULL AND LIF_REMISE_MONTANT IS NOT NULL UNION SELECT LIF_ID, (LIF_QTE * LIF_MONTANT) * (1 - LIF_REMISE_POURCENT/100) FROM T_LIGNE_FACTURE WHERE LIF_REMISE_POURCENT IS NOT NULL AND LIF_REMISE_MONTANT IS NULL UNION SELECT LIF_ID, (LIF_QTE * LIF_MONTANT) * (1 - LIF_REMISE_POURCENT/100) - LIF_REMISE_MONTANT FROM T_LIGNE_FACTURE WHERE LIF_REMISE_POURCENT IS NOT NULL AND LIF_REMISE_MONTANT IS NOT NULL
13	**transformez les CASE...** ELECT CHB_NUMERO, CASE CHB_ETAGE WHEN 'RDC' THEN 0 WHEN '1er' THEN 1 WHEN '2e' THEN 2 END AS ETAGE, CHB_COUCHAGE FROM T_CHAMBRE ORDER BY ETAGE, CHB_COUCHAGE	**...en UNION** SELECT CHB_NUMERO, 0 AS ETAGE, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE = 'RDC' UNION SELECT CHB_NUMERO, 1 AS ETAGE, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE = '1er' UNION SELECT CHB_NUMERO, 2 AS ETAGE, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE = '2e' ORDER BY ETAGE, CHB_COUCHAGE

Optimisation de requêtes

14	**transformez les EXCEPT...** SELECT CHB_ID FROM T_CHAMBRE EXCEPT SELECT CHB_ID FROM TJ_CHB_PLN_CLI WHERE PLN_JOUR = '2000-11-11'	**...en jointures** SELECT DISTINCT C.CHB_ID FROM T_CHAMBRE C LEFT OUTER JOIN TJ_CHB_PLN_CLI P ON C.CHB_ID = P.CHB_ID AND PLN_JOUR = '2000-11-11' WHERE P.CHB_ID IS NULL
15	**transformez les INTERSECT...** SELECT CHB_ID FROM T_CHAMBRE INTERSECT SELECT CHB_ID FROM TJ_CHB_PLN_CLI WHERE PLN_JOUR = '2000-11-11'	**...en jointure** SELECT DISTINCT C.CHB_ID FROM T_CHAMBRE C INNER JOIN TJ_CHB_PLN_CLI P ON C.CHB_ID = P.CHB_ID WHERE PLN_JOUR = '2000-11-11'
16	**transformez les UNION...** SELECT OBJ_NOM AS NOM, OBJ_PRIX AS PRIX FROM T_OBJET UNION SELECT MAC_NOM AS NOM, MAC_PRIX AS PRIX FROM T_MACHINE ORDER BY NOM, PRIX	**...en jointure** SELECT COALESCE(OBJ_NOM, MAC_NOM) AS NOM, COALESCE(OBJ_PRIX, MAC_PRIX) AS PRIX FROM T_OBJET O FULL OUTER JOIN T_MACHINE M ON O.OBJ_NOM = M.MAC_NOM AND O.OBJ_PRIX = M.MAC_PRIX ORDER BY NOM, PRIX

Optimisation de requêtes

17	**transformez les sous requêtes <> ALL ...** SELECT CHB_ID, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_COUCHAGE <> ALL (SELECT CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE ='RDC')	**... en NOT IN** SELECT CHB_ID, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_COUCHAGE NOT IN (SELECT CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE ='RDC')
18	**transformez les sous requêtes = ANY ...** SELECT CHB_ID, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_COUCHAGE = ANY (SELECT CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE ='RDC')	**... en IN** SELECT CHB_ID, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_COUCHAGE IN (SELECT CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE ='RDC')
19	**transformez les sous requêtes ANY / ALL ...** SELECT CHB_ID, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_COUCHAGE > ALL (SELECT CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_ETAGE ='RDC')	**...en combinant sous requêtes et aggrégat** SELECT CHB_ID, CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_COUCHAGE > (SELECT MAX(CHB_COUCHAGE) FROM T_CHAMBRE WHERE CHB_ETAGE ='RDC')

Optimisation de requêtes

20	**évitez les sous requêtes corrélées...** SELECT DISTINCT VILLE_ETP FROM T_ENTREPOT AS ETP1 WHERE NOT EXISTS (SELECT * FROM T_RAYON RYN WHERE NOT EXISTS (SELECT * FROM T_ENTREPOT AS ETP2 WHERE ETP1.VILLE_ETP = ETP2.VILLE_ETP AND (ETP2.RAYON_RYN = RYN.RAYON_RYN)))	**...préférez des sous requêtes sans corrélation** SELECT DISTINCT VILLE_ETP FROM T_ENTREPOT WHERE RAYON_RYN IN (SELECT RAYON_RYN FROM T_ENTREPOT WHERE RAYON_RYN NOT IN (SELECT RAYON_RYN FROM T_ENTREPOT WHERE RAYON_RYN NOT IN (SELECT RAYON_RYN FROM T_RAYON))) GROUP BY VILLE_ETP HAVING COUNT (*) = (SELECT COUNT(DISTINCT RAYON_RYN) FROM T_RAYON)
21	**évitez les sous requêtes corrélées...** SELECT FAC_ID, (SELECT MAX(LIF_QTE * LIF_MONTANT) FROM T_LIGNE_FACTURE L WHERE F.FAC_ID = L.FAC_ID) FROM T_FACTURE F ORDER BY FAC_ID	**...préférez des jointures** SELECT F.FAC_ID, MAX(LIF_QTE * LIF_MONTANT) FROM T_FACTURE F JOIN T_LIGNE_FACTURE L ON F.FAC_ID = L.FAC_ID GROUP BY F.FAC_ID ORDER BY F.FAC_ID
22	**n'utilisez pas de nombre dans la clause ORDER BY...** SELECT LIF_ID, (LIF_QTE * LIF_MONTANT) FROM T_LIGNE_FACTURE ORDER BY 1, 2	**...spécifiez de préférence les noms des colonnes, y compris dans la clause SELECT** SELECT LIF_ID, (LIF_QTE * LIF_MONTANT) AS LIF_MONTANT FROM T_LIGNE_FACTURE ORDER BY LIF_ID, LIF_MONTANT

Optimisation de requêtes : Explain Plan

http://www.toutenligne.com/index.php?contenu=sql_explain

- SGBD propose souvent une commande
  - EXPLAIN PLAN qui donne le plan d'exécution de l'optimiseur
- Utilisation de la commande
  1. Créer une table destinée à contenir toutes les informations relatives à un plan d'exécution (Plan_table)
  2. Exécuter une requête en demandant le stockage des explications relatives à cette requête dans la table crée
  3. Interroger la table remplie pour connaître le plan d'exécution

Optimisation de requêtes : Explain Plan

1. Création de la table

```
CREATE TABLE PLAN_TABLE (
COM_ID VARCHAR (30),   #la valeur provient de la commande SQL
TS DATE,               #date d'exécution de EXPLAIN PLAN
COMMMENT VARCHAR (80),#commentaires mis dans la commande
OP VARCHAR(30),        #nom de l'opération interne effectuée (cf tab.)
OPTIONS VARCHAR(30),   #option de la commande (cf tab.)
NODE VARCHAR(30),      #nom du lien de base de données utilisé pour référencer l'objet
OWNER VARCHAR(30),     #propriétaire de la table ou de l'index
NAME VARCHAR(30),      #nom de la table ou de l'index
INSTANCE Integer,      #position de l'objet dans la commande SQL (gauche à droite)
TYPE VARCHAR(30),      #type de l'objet
SEARCH_COLUMNS Integer,# non utilisé
ID Integer,            #numéro affecté à chaque étape dans le plan d'exécution
PARENT_ID Integer,     #numéro de l'étape suivante qui utilisera comme flux d'entrée les sorties de l'étape numéro ID
POSITION Integer,      #ordre de traitement des étapes qui ont le même PARENT_ID
OTHER LONG);
```

Optimisation de requêtes : Explain Plan

OPERATIONS	OPTIONS	SIGNIFICATION
AGGREGATE	GROUP BY	Une recherche d'une seule ligne qui est le résultat de l'application d'une fonction de group à un groupe de lignes sélectionnées.
AND-EQUAL		Une opération qui a en entrée des ensembles de rowids et retourne l'intersection de ces ensembles en éliminant les doublants. Cette opération est utilisée par le chemin d'accès par index.
CONNECT BY		Recherche de ligne dans un ordre hiérarchique
COUTING		Opération qui compte le nombre de lignes sélectionnées.
FILTER		Accepte un ensemble de ligne, appliqué un filter pour en éliminer quelque unes et retourne le reste.
FIRST ROW		Recherché de le première ligne seulement.
FOR UPDATE		Opération qui recherche et verrouille les lignes pour une mise à jour
INDEX	UNIQUE SCAN	Recherche d'une seule valeur ROWID d'un index.
INDEX	RANGE SCAN	Recherche d'une ou plusieurs valeurs ROWID d'un index. L'index est parcouru dans un ordre croissant.
INDEX	RANGE SCAN DESCENDING	Recherche d'un ou plusieurs ROWID d'un index. L'index est parcouru dans un ordre décroissant.
INTERSECTION		Opération qui accepte deux ensembles et retourne l'intersection en éliminant les doublons.
MARGE JOIN+		Accepte deux ensembles de lignes (chacun est trié selon un critère), combine chaque ligne du premier ensemble avec ses correspondants du deuxième et retourne le résultat.

Optimisation de requêtes : Explain Plan

OPERATIONS	OPTIONS	SIGNIFICATION
MARGE JOIN+	OUTER	MARGE JOIN pour effectuer une jointure externe
MINIUS		Différence de deux ensembles de lignes.
NESTED LOOPS		Opération qui accepte deux ensembles, l'un externe et l'autre interne. Oracle compare chaque ligne de l'ensemble externe avec chaque ligne de l'ensemble interne et retourne celle qui satisfont une condition.
NESTED LOOPS	OUTER	Une boucle imbriquée pour effectuer une jointure externe.
PROJECTION		Opération interne
REMOTE		Recherche de données d'une base distante.
SEQUENCE		Opération nécessitant l'accès à des valeurs du séquenceur
SORT	UNIQUE	Tri d'un ensemble de lignes pour éliminer les doublons.
SORT	GROUP BY	Opération qui fait le tri à l'intérieur de groupes
SORT	JOIN	Tri avant la jointure (MERGE-JOIN).
SORT	ORDER BY	Tri pour un ORDER BY.
TABLE ACCESS	FULL	Obtention de toutes lignes d'une table.
TABLE ACCESS	CLUSTER	Obtention des lignes selon la valeur de la clé d'un cluster indexé.
TABLE ACCESS	HASH	Obtention des lignes selon la valeur de la clé d'un hash cluster
TABLE ACCESS	BY ROW ID	Obtention des lignes on se basant sur les valeurs ROWID.
UNION		Union de deux ensembles avec élimination des doublons.

Optimisation de requêtes : Explain Plan

2. Utilisation de la table en SQL
   a. EXPLAIN PLAN
      SET STATEMENT_ID=' Identifiant_choisi '
      FOR requête ;
      ex :
        EXPLAIN PLAN
        SET STATEMENT_ID='sel1'
        FOR SELECT NOM from ACTEURS order by NOM;
   b. SELECT COM_ID, OP, OPTIONS, ID, PARENT_ID, POSITION
      FROM PLAN_TABLE
      WHERE STATEMENT_ID='sel1';
      ex :

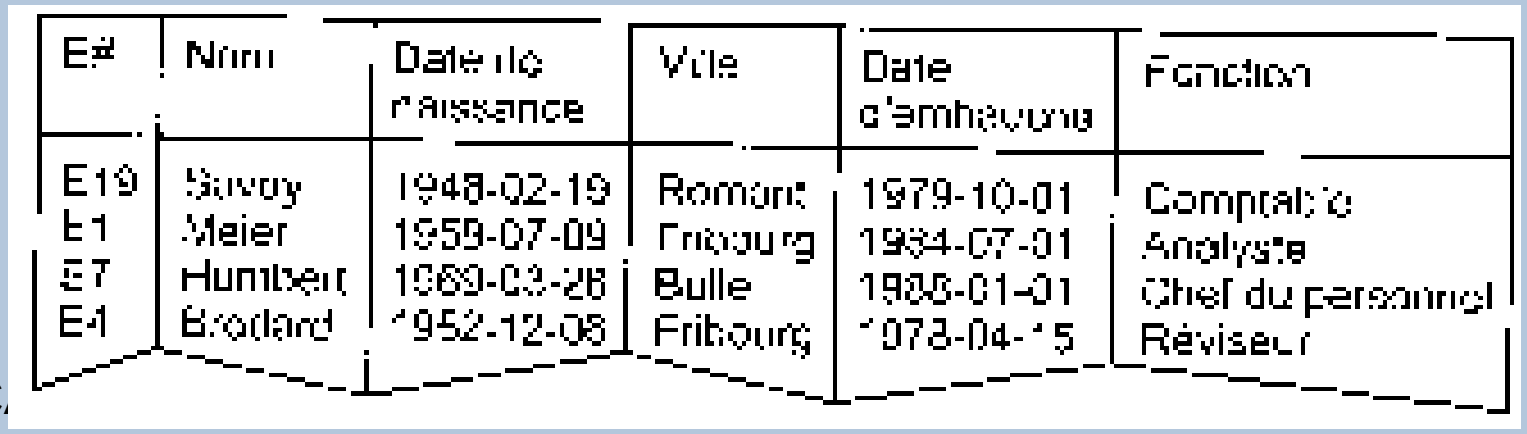
STATEMENT_ ID	OPERATION	OPTIONS	ID	PARENT_ID	POSITION
sel2	SELECT STATEMENT	GROUP BY	0	0	1
sel2	SORT	FULL	1	1	1
sel2	TABLE ACCESS		2		

BD Temporelles

- Constat
  - Les BD actuelles stockent les informations du présent et pas les informations du passé qui ont été supprimées
  - Si utilisateur souhaite interroger & analyser des données en considérant le facteur temps, il est responsable de la gestion & maintenance des données passées et futures
- Objectif
  - Introduire un facteur temps dans les BD actuelles
  - Permet de comparer des valeurs sur l'axe du temps
- Implémentation
  - Toute information d'une BD temporelle (valeur, table, tuples…) sont définies par rapport à l'axe du temps
  - En introduisant la validité de l'information stockée

BD Temporelles

- Contre-exemple
  - Extraire les employés engagés par l'entreprise avant l'âge de 20 ans?
    - SELECT E#, NOM
      FROM EMPLOYE
      WHERE DATE_EMBAUCHE — DATE_NAISSANCE <=20
  - Aucune info sur le passé !
    - Si Humbert change de fonction, on écrase sa précédente fonction



E#	Nom	Date de naissance	Ville	Date d'embauche	Fonction
E19	Savoy	1948-02-19	Romont	1979-10-01	Comptable
E1	Meier	1959-07-09	Fribourg	1984-07-01	Analyste
E7	Humbert	1969-03-26	Bulle	1988-01-01	Chef du personnel
E4	Brodard	1952-12-08	Fribourg	1978-04-15	Réviseur

BD Temporelles

- Solution
  - Introduire un champ valid_from dans la table
  - SGBD Temporel
    - Prend en charge l'ensemble des valeurs de l'axe de temps comme temps valide
    - Dispose dans le langage d'éléments temporels

TEMP_EMPLOYÉ (Extrait)

E#	*VALID_FROM*	Nom	Ville	Date d'embauche	Fonction
E1	01.07.1984	Meier	Bulle	1984-07-01	Programmeur
E1	13.09.1986	Meier	Fribourg	1984-07-01	Programmeur
E1	04.05.1987	Meier	Fribourg	1984-07-01	Analyste-progr.
E1	01.04.1989	Meier	Bulle	1984-07-01	Analyste

BD Temporelles

- Exemple
  - Déterminer la fonction de Meier au 01/01/1988?
    - SQL classique

```
SELECT FONCTION
FROM TEMP_EMPLOYE A
WHERE NOM=« MEIER » AND A.VALID_FROM =
  SELECT( MAX(VALID_FROM)
  FROM TEMP_EMPLOYE B
  WHERE B.NOM=A.NOM AND B.VALID_FROM
<=’01/01/1988’)
```

    - SQL Temporel

```
SELECT FONCTION
FROM TEMP_EMPLOYE
WHERE NOM=‘MEIER’ AND VALID_AT(01/01/1988)
```

BD Réparties

- Principe
  - BD stockées sur différents sites
  - Transparent pour l'utilisateur (⇒ 1 seule BD pour lui)
  - Traitement d'une requête peut se faire sur un site différent de celui où la requête a été posée
  - Problèmes particuliers
    - Gestion des transactions sur le SGBD
    - Optimisation des requêtes

BD Réparties

- Duplication
  - Le système maintient différentes copies des données sur tous les sites (exécution rapide des requêtes et résistance aux pannes)
- Fragmentation
  - Les relations (tables) sont partitionnées en différents fragments chacun sur un site distinct
- Fragmentation et Duplication
  - Combinaison des 2

BD Réparties

- Disponibilité
  - Si 1 site est indisponible, on peut utiliser un autre pour extraire l'information recherchée
- Parallélisme
  - Les requêtes sur R peuvent être traitée en exécutant plusieurs sous-requêtes en parallèle sur différents sites
- Mise À Jour
  - Le coût des MAJ se voit augmenter (maintenir la cohérence des différentes copies)
- Contrôle de concurrence
  - Plus complexe car verrouiller un tuple sur le site si implique son verrouillage sur sj

BD Réparties

- Division de R en différents fragments R1, …, Rn (contenant assez d'infos pour reconstruire R au besoin)
  - Horizontale : chaque tuple de R est affecté à un ou plusieurs fragments
  - Verticale : le schéma R est subdivisé en sous-schémas
- Le même clé pour tous les sous-schémas (afin de garantir une décomposition sans perte)
- On peut rajouter un attribut identifiant le tuple dans la sous-relation (l'union des Ri peut être différente de R)

BD Réparties

- Exemple de décomposition horizontale
  - Compte(Agence,Numcompte,Solde)

$Compte_1$

Agence	**NumCompte**	**Solde**
Talence	123	1200
Talence	321	-1220

$Compte_2$

Agence	**NumCompte**	**Solde**
Pessac	124	100
Pessac	421	-122

BD Réparties

- Exemple de décomposition verticale
  - Compte(Agence,Numcompte,NomTit,Solde)

Compte$_1$

Agence	**NomTit**	**IdTuple**
Talence	David	1
Pessac	David	3
Talence	Martin	2
Pessac	Dupont	4

Compte$_2$

Numcompte	**Solde**	**IdTuple**
123	1200	1
321	-1200	2
124	100	3
421	-122	4

BD Réparties

- Horizontale
  - Possibilité de parallélisation
  - Les tuples sont situés au niveau des sites où ils sont le plus souvent accédés
- Verticale
  - Chaque partie d'un tuple peut être localisée là où elle est le plus accédée
  - Parallélisme
  - L'attribut IdTuple permet de reconstituer le tuple

BD Réparties

- L’utilisateur n’a pas à connaître la localisation des données. Il ne faut pas non plus qu’un même nom désigne deux choses différentes sur des sites différents
- Chaque donnée doit avoir un nom unique connu dans tous le réseau
- Localiser une donnée doit se faire efficacement
- On doit pouvoir changer l’emplacement d’une donnée sans qu’il faille revoir les transactions
- Chaque site doit pouvoir créer de nouvelles données

BD Réparties

- Lors MAJ, SGBDD doit garantir que tous les duplicata et les fragments associés à une info soient modifiées
- Horizontale
  - Considérons l'insertion du tuple (Talence, 555,1000) dans Compte
    Compte1=σAgence=Talence(Compte)
    Compte2=σAgence=Pessac(Compte)
    $\Rightarrow$ tuple doit être insérer dans Compte1
- Verticale
  - Tuple (Talence,555,Nathalie,1000) doit être décomposé en 2 tuples : (Talence,Nathalie,5) et (555,1000,5)
- Problème : si la relation globale est accédée en concurrence, une copie peut être modifiée avant une autre

BD Réparties

- Ds 1 BD centralisée, le coût = le nombre d'accès disque
- Ds 1 BD distribuée, existe d'autres paramètres
  - Transmissions des données via le réseau
  - Le gain potentiel a faire exécuter la requête en // sur plusieurs sites
- Exemples de parallélisation
  - $\sigma_{Solde>0}(Compte)$ décomposée en $\sigma_{Solde>0}(Compte_1) \cup \sigma_{Solde>0}(Compte_2)$

Entrepôt de données

Caractéristique	Base de données	Entrepôt de données
Opération	gestion courante, production	analyse, support à la décision
Modèle de données	entité/relation	étoile, flocon de neige
Normalisation	fréquente	plus rare
Données	actuelles, brutes	historisées, parfois agrégées
Mise à jour	immédiate, temps réel	souvent différée
Niveau de consolidation	faible	élevé
Perception	bidimensionnelle	multidimensionnelle
Opérations	lectures, mises à jour, suppressions	lectures, analyses croisées, rafraîchissements
Taille	en gigaoctets	en téraoctets

Entrepôt de données

- Ces différences tiennent au fait que les entrepôts permettent des requêtes qui peuvent être complexes et qui ne reposent pas nécessairement sur une unique table.
- Exemples de requêtes OLAP :
- *Quel est le nombre de paires de chaussures vendues par le magasin "OnVendDesChaussuresIci" en mai 2003 ET Comparer les ventes avec le même mois de 2001 et 2002*
- *Quelles sont les composantes des machines de production ayant eu le plus grand nombre d'incidents imprévisibles au cours de la période 1992-97 ?*
- Les réponses aux requêtes OLAP peuvent prendre de quelques secondes à plusieurs minutes.
- **Architecture d'un entrepôt de données** [modifier]
- Un entrepôt de données est généralement construit selon une architecture en 3 strates :
- d'un serveur d'entrepôt (serveur de données)
- d'un serveur OLAP (de type HOLAP/MOLAP ou ROLAP)
- d'un client
  - outil pour l'exécution des requêtes
  - outil pour l'analyse des données

Fouille de données

- Analyse exploratoire des données
  - Par l'application de méthodes statistiques & IA sur une grande quantité de données
- Objectifs
  - Trouver des configurations intéressantes ds les données qui représentent des comportements inhabituels
    - Exemple
      - Soudaine augmentation d'activité d'une carte de crédit peut signifier qu'elle a été volée

Fouille de données

- Identifier des règles
  - D'association
    - une personne achetant des chaussures, achète souvent des chaussettes
  - De corrélation de séquence
    - Si une personne achète des chaussures *aujourd'hui*, il achètera des chaussettes dans les *5 jours*
  - De classification
    - Si revenu > 50000€/an Alors faible risque pour le crédit d'emprunts
  - ...

$\forall \Rightarrow$ Cours de Data Mining

BD hétérogènes & Web

- XML
  - eXtensible Markup Language
  - ≠ HTML mais voisin dans le concept de langage à balises
- Balise
  - Ouvrante : <nom_balise>
  - Fermante : <\nom_balise>
- Élément : Balise ouvrante + contenu + balise fermante
  - <hamlet> Être ou ne pas être <\hamlet>
- Attributs : ensemble de propriétés données à une balise
  - <BODY BGCOLOR= "#0000" ...>
- XML
  - Plus souple que HTML car on peut définir ses propres balises & attributs
  - Pas uniquement utilisé pour être affiché sur le web…
  - C'est une façon de baliser des données afin de les rendre auto descriptives

XML
Exemple

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Facture
    nomClient="Kevin Williams"
    adresseFacturation="742 Evergeen Terrace"
    villeFacturation="SpringField"
    ...
    <LigneDeCommande
     codeArticle="124AB3A"
     descriptionArticle="Truc(7.5cm)"
     quantité="1"  ...
    </LigneDeCommande>
    <LigneDeCommande
     codeArticle="124C56A"
     descriptionArticle="Machin"
     quantité="1"  ...
    </LigneDeCommande>
</Facture>
```

XML
Intérêts & DTD

- XML = texte brut
  - Accessibles à tout langage de programmation, à toute plate-forme
  - Facilement transmissible par HTTP
  - Échange de données très simplifié
- Vocabulaire
  - Ensemble des balises & attributs pour un domaine donné
  - Définitions de type de documents (*Definition Type Document* = DTD) = Moyen de définir un vocabulaire
    - Éléments pouvant être présent dans un document
    - Le nombre d'instances de chaque éléments
    - Ordre des éléments
    - Attributs acceptés par les balises (obligatoire, facultatif, valeur par défaut...)

XML
Traitements

- Parseur
  - Composant logiciel lisant le document XML comme texte brut
- API (Application Programming Interface)
  - Offre aux programmeurs un ensemble de fonctionnalités pouvant être appelées depuis 1 prog pour réclamer des infos au parseur alors qu'il traite le doc. XML
- Validateurs
  - Certains parseurs ont la possibilité de vérifier qu'un document XML est au format d'une DTD mais cela nécessite plus de ressources (temps, mémoire)

XML
Blocs fondamentaux

- Un fichier XML peut contenir
  - Déclaration XML
  - Éléments
  - Attributs
  - Données textuelles (CDATA)
  - Instructions de traitement
  - Commentaires
  - Références d'entités

XML
Déclaration XML

- Elle est facultative mais fortement recommandée pour que les applications utilisatrices de ce document sachent que c'est un document XML
- Indique le format XML utilisé
  - <?xml version="1.0">
  - Rien avant, même un espace
  - Possibilité de définir le langage utilisé (encoding="ISO-8859-1")

XML
Éléments

- Correspondent aux composants les plus importants des documents XML
- Un document XML doit posséder au moins un élément , dans lequel sont imbriqués tous les autres balisages
- L'élément de plus haut niveau porte le nom d'élément document (élément racine)
- Syntaxe
  - L'écriture <Facture /> correspond à un élément vide (ayant aucun contenu)
  - Distinction majuscule/minuscule
  - Imbrication correcte des balises. Aucun chevauchement de balises est autorisé

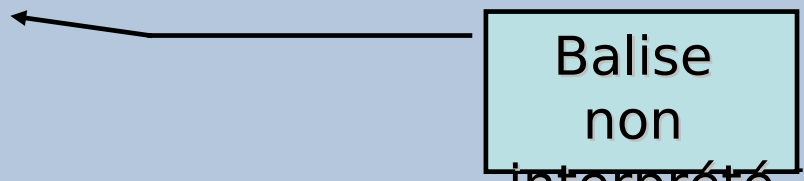
XML
Attributs & Données & Commentaires

- Attributs
  - La valeur d'un attribut est toujours entre guillemets ("")
  - Pas d'ordre dans les attributs d'une balise
- Données textuelles
  - Section non interprétée par le parseur
  - Ne doit pas contenir les caractères & et < sinon ils sont interprétés par le parseur
- Commentaires
  - <!– mes commentaires -->
  - Un commentaire ne peut donc contenir --

XML
Section CDATA

- Pour éviter que le parseur interprète des caractères spéciaux, il faut les mettre dans la section CDATA qui débute par <![CDATA[texte à ne pas interpréter]]>
- Désactive la détection de balisage
- Exemple

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<BalisageExemple>
    <![CDATA[Pour déclarer un élément, utilisez une balise d'ouverture
    d'élément : <MaBalise>]]>
</BalisageExemple>
```



XML
Données textuelles ≠ CDATA

- 1ére version

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<BalisageExemple>
    <![CDATA[Pour déclarer un élément, utilisez une balise d'ouverture d'élément :
    <MaBalise>]]>
</BalisageExemple>
```

- 2ème version

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<BalisageExemple>
    Pour déclarer un élément, utilisez une balise d'ouverture d'élément : <!
    CDATA[[<MaBalise>]]>
</BalisageExemple>
```

- Dans la 2ème version, le parseur va considérer que l'élément BalisageExemple est composé de deux choses :une donnée textuelle ("Pour ...élément:") et une section CDATA

XML
Références d'entités

- Pour éviter le problème précédent, on introduit les références d'entités qui permettent d'inclure des caractères spéciaux dans un document XML
- Les différentes références d'entités standard

&lt;	<		(espace insécable)
&gt;	>	&#chiffre;	(caractère de code ascii chiffre)
&amp;	&		
&apos;	'		
&quot;	"		

- Exemple

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<BalisageExemple>
    Pour déclarer un élément, utilisez une balise d'ouverture d'élément : &lt;MaBalise&gt;
</BalisageExemple>
```

XML
Instructions de traitement

- Permet de préciser qu'à un endroit précis du document, le parseur doit déclencher une certaine action
- <? Instructions de traitement ?>
- Exemple
  - <?xml-stylesheet type="text/html" href="#style1"?>
  - La cible de l'instruction de traitement : xml-stylesheet
  - Chaîne sur laquelle opérer : type="text/html" href="#style1"
  - Il faudra analyser manuellement la chaine si on veut accéder aux différentes informations

XML
DTD

- Définition de Type de Documents
  - Constitué d'un ensemble de règles devant être respectés par tout document XML auquel la DTD s'applique
  - Se caractérise par
    - Un fichier externe
      <!DOCTYPE MonDocXML SYSTEM "http://www.monsite. Com/Xml/MonDocXML.dtd">
    - Inclus dans le document XML
      <!DOCTYPE MonDocXML **[**
      <!ELEMENT MonDocXML (#PCDATA)>]
      >
    - Une définition mixte (externe + interne)

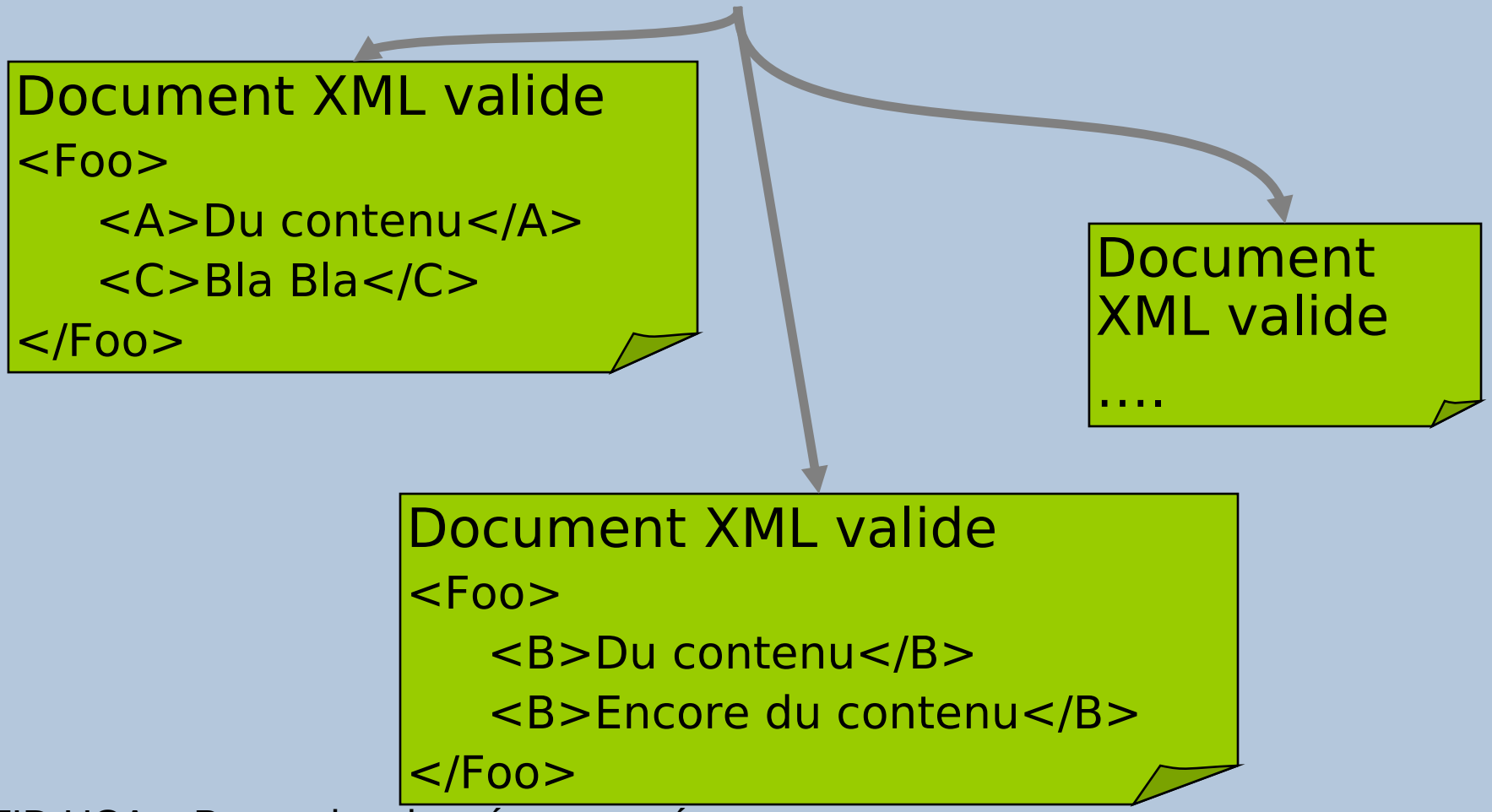
XML
DTD – Déclaration d'éléments

- C'est le plus important lors de la déclaration d'une DTD. Toute DTD doit en posséder au moins une
- Déclaration
  <!ELEMENT Nom_Element (ModèleContenu)>
  ModèleContenu définit ce que peut contenir l'élément
- Syntaxe
  - | permet de choisir parmi un ensemble d'éléments
    <!ELEMENT Foo (A | B | C)> soit l'un ou l'autre mais pas tous
  - Cardinalité des éléments
    - ? Facultatif (présent 0 ou 1 fois)
    - * Facultatif multiple (présent 0 fois ou plus)
    - (rien) Obligatoire (présent 1 et 1 seule fois)
    - + Obligatoire multiple (présent au moins 1 fois)

XML

DTD – Documents valides

DTD : <!ELEMENT Foo((A|B+), C?)>



XML
DTD – Types de données

- Chaîne de caractères : #PCDATA
- Vide : EMPTY
- Quelconque de structuré : ANY
- Exemple
  - DTD : <!ELEMENT Foo (#PCDATA|A|B|C)*>
    Voici du <A>texte</A> avec<B> des éléments</B>
  - DTD : <!ELEMENT Foo (EMPTY)>
    <Foo /> ou bien <Foo></Foo>
  - DTD : <!ELEMENT Foo ANY>
    <A>Ceci</A><B> est</B><C> bien </C><D>balisé</D>

XML
DTD – Attributs

- Déclaration d'attribut
  - <!ATTLIST nom_élément définition_attrib*>
  - définition_attrib ⇔ nom_attrib type_attrib valeur_défaut
  - Exemple
    <!ELEMENT Foo EMPTY>
    <!ATTLIST Foo Texture CDATA #REQUIRED>
- Type d'attribut
  - Chaîne de caractères : CDATA
    <!ELEMENT Foo EMPTY>
    <ATTLIST Foo Texture CDATA #REQUIRED
    Couleur CDATA #REQUIRED >
    XML:<Foo Texture="bumpy" Couleur="rouge&bleu" />

XML
DTD – Attributs

- Identifiant unique : ID
  <!ELEMENT Foo EMPTY>
  <ATTLIST Foo FooID ID #REQUIRED>
  XML : <Foo FooID="foo1" />
  <Foo FooID="foo1" /> <!—2 éléments Foo ne peuvent pas porter le même nom-->
  <Foo FooID="17" /><!—17 n'est pas un nom XML valide-->
- Référence à un autre élément XML : IDREF
  <!ELEMENT Auteur EMPTY>
  <!ATTLIST Auteur AuteurID ID #REQUIRED>
  <!ELEMENT Livre EMPTY>
  <!ATTLIST Livre LivreID #REQUIRED AuteurIDREF IDREF>
  XML : <Auteur AuteurID="auteur1" />
  <Livre LivreID="livre1" AuteurIDREF="auteur1" />
  <Livre LivreID="livre2" AuteurIDREF="livre1" />
  la dernière ligne est malheureusement valide

XML
DTD – Attributs

- Référence à plusieurs autres éléments XML : IDREFS
  <!ELEMENT Foo EMPTY>
  <ATTLIST Foo FooID ID #REQUIRED>
  <!ELEMENT Bar EMPTY>
  <ATTLIST Bar BarID ID #REQUIRED FooIDREF IDREFS>
  XML : <Foo FooID="foo1" />
  <Foo FooID="foo2" />
  <Bar BarID="bar1" FooIDREF="foo1 foo2"/>
- Référence à 1 ou +sieurs entités non-XML : ENTITY ou ENTITIES (≡ IDREF & IDREFS)
- Valeur respectant la règle des noms de variable XML (chiffre, lettre et autres caractères spécifiques) : NMTOKEN
- Ensemble de valeurs respectant les règles : NMTOKENS

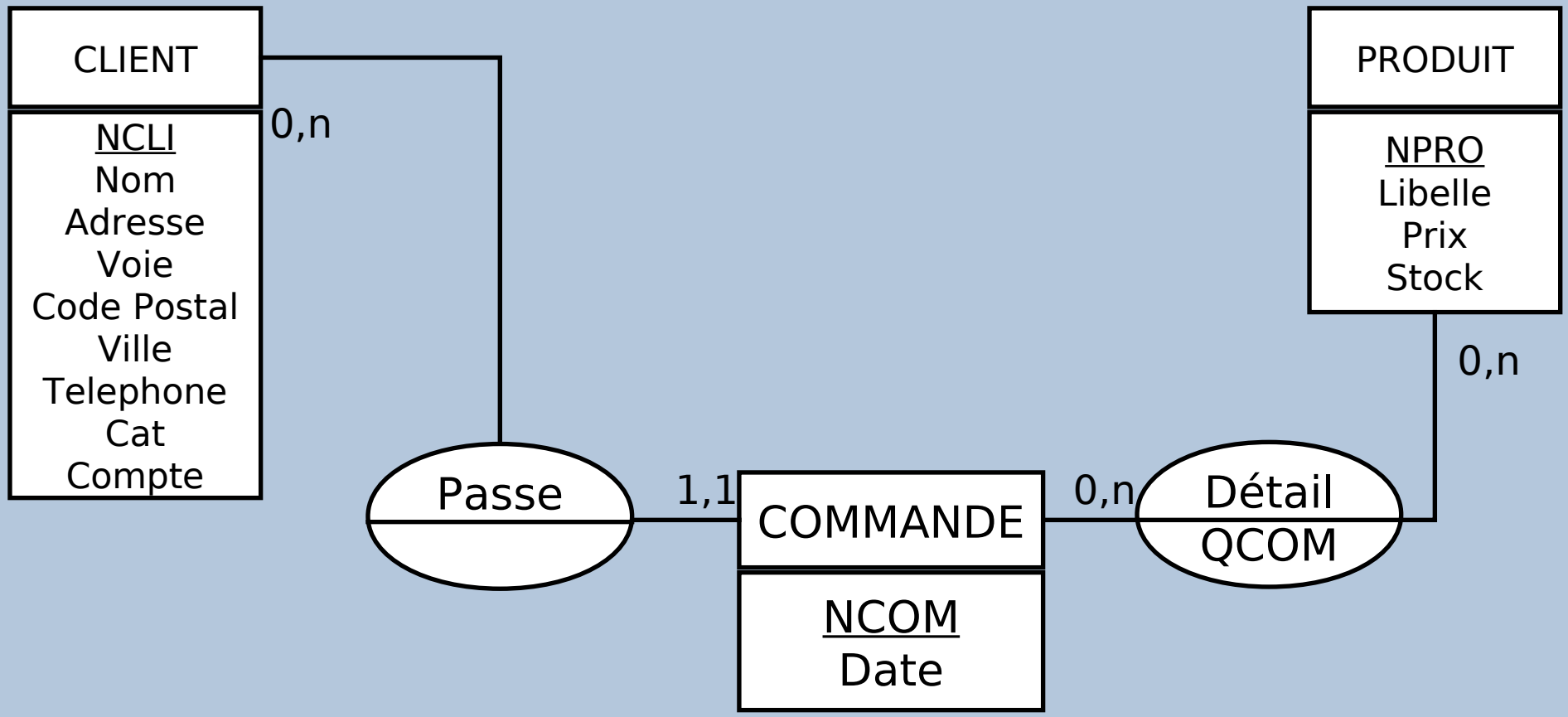
MCD → DTD + XML

- Proposé par Christine Delcroix
- Vocabulaire
  - <!ELEMENT Client (Nom, Prénom?, Adresse, Téléphone+, Commandes*)>
  - Nom est un **sous-types** de Client
  - Client est un **sur-types** de Adresse
- Plan de transformation en 9 étapes

  Appliquer sur le schéma conceptuel suivant

MCD → DTD + XML

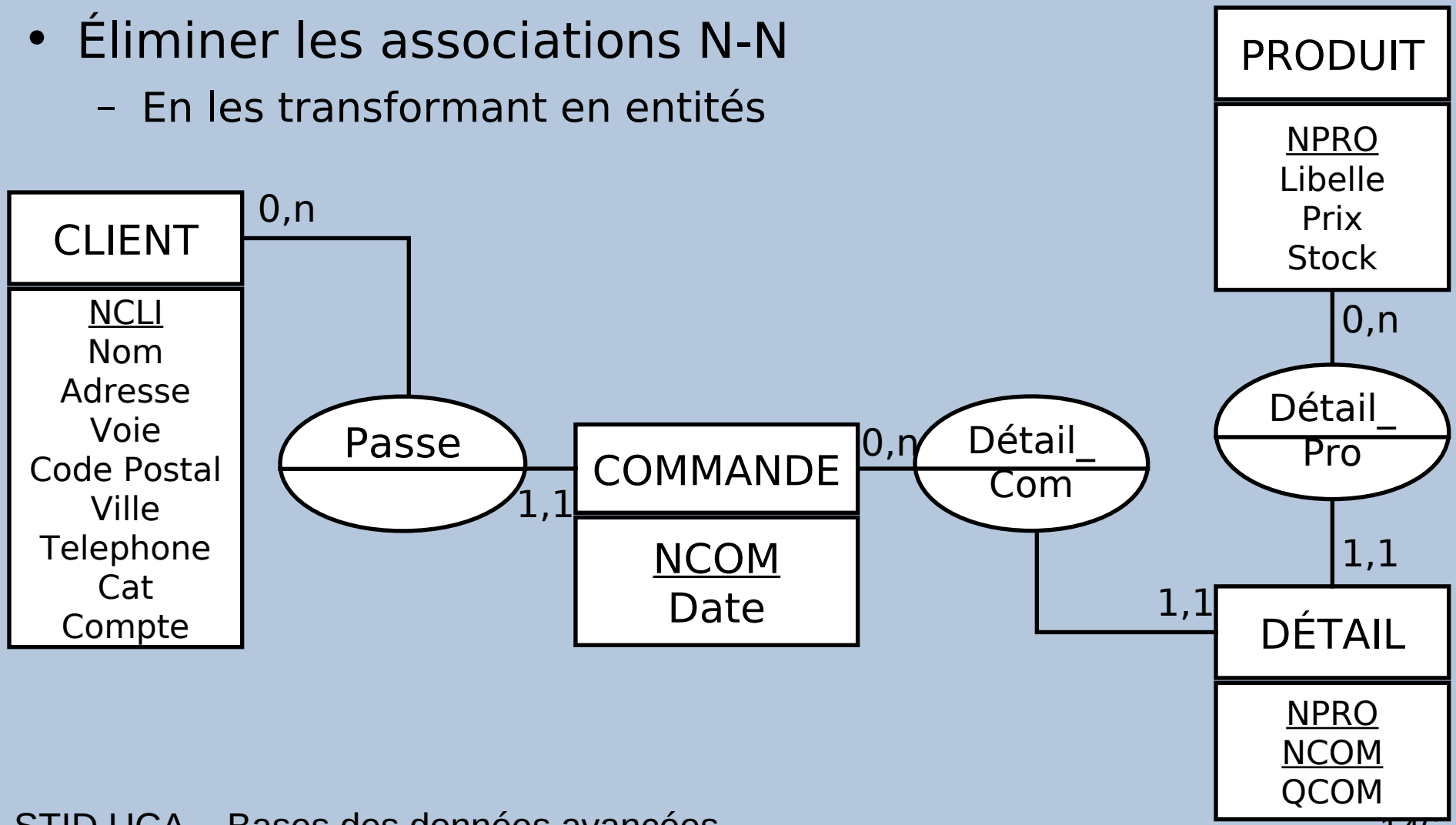
Étude de cas



MCD → DTD + XML

Étape 1

- Éliminer les associations N-N
  - En les transformant en entités



MCD → DTD + XML

Étape 2

- Construire la structure hiérarchique du document XML à partir du graphe du MCD
  - (1) Déterminer les entités racines = correspondent aux entités qui ne jouent aucun rôle de cardinalité égale à 1-1
    D'autres entités racines peuvent être choisies par l'utilisateur de la méthode de traduction

    Exemple : CLIENT et PRODUIT ne font référence à aucune autre entités. Ils sont donc racines

MCD → DTD + XML

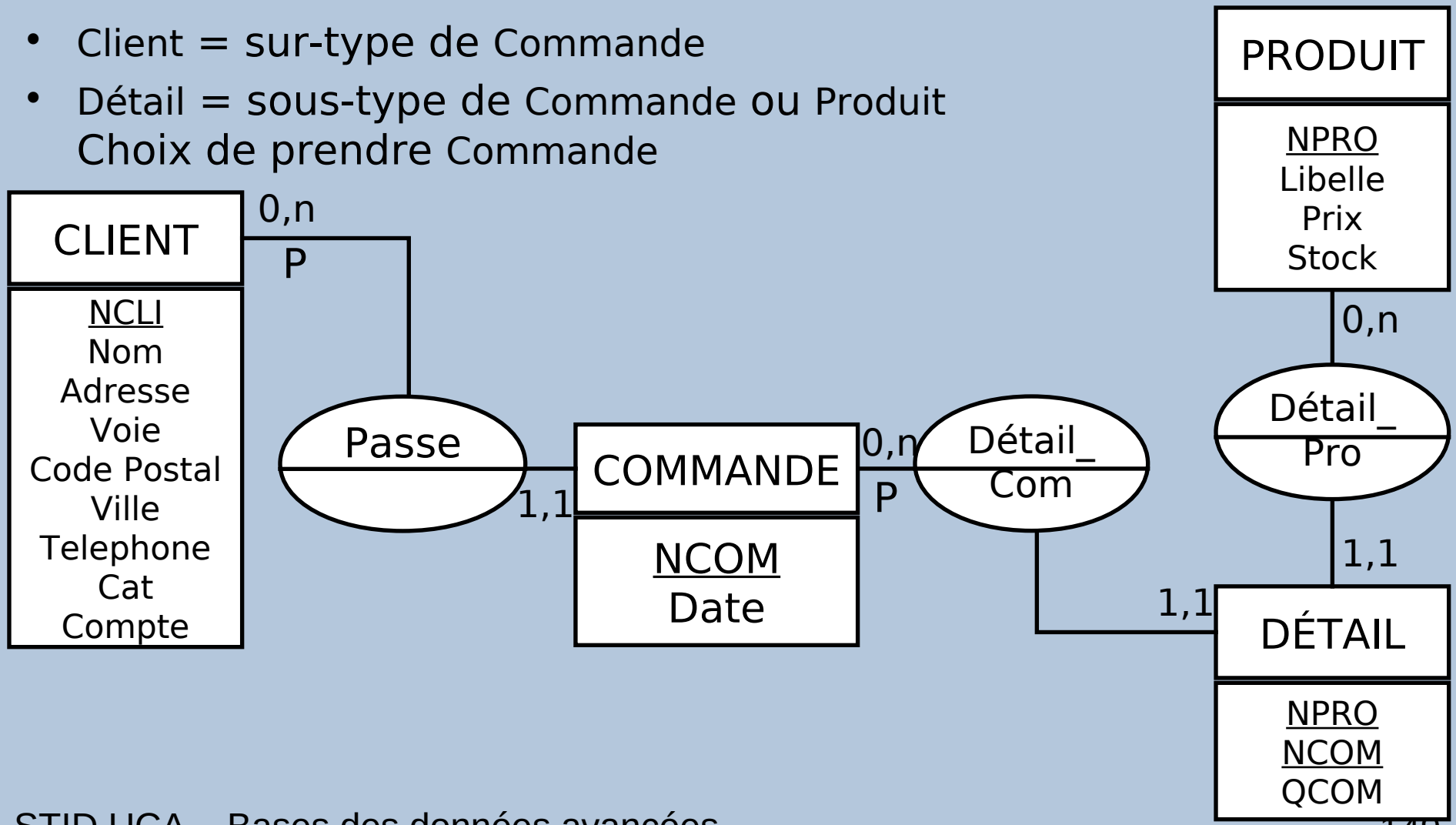
Étape 2

- (2) Déterminer les descendants des racines
  - Graphiquement, une association représente lien entre un sur-type d'entités et un ensemble de sous-types donc les sous-types sont les descendants du sur-type
- Construire la hiérarchie revient à nommer un certain nombre de rôles d'associations par « p » pour père en respectant les règles suivantes :
  - Le rôle joué par 1 sous-type doit avoir une cardinalité égale à 1-1
  - Dans la plupart des cas, une entité n'à qu'un seul sur-type
  - Une entité marqué racine ne peut avoir de sur-type
- Remarque
  - Toutes les associations du MCD ne seront pas toutes conservées Les associations qui ne possèdent pas de « p » seront supprimées

MCD → DTD + XML

Étape 2

- Client = sur-type de Commande
- Détail = sous-type de Commande ou Produit
  Choix de prendre Commande



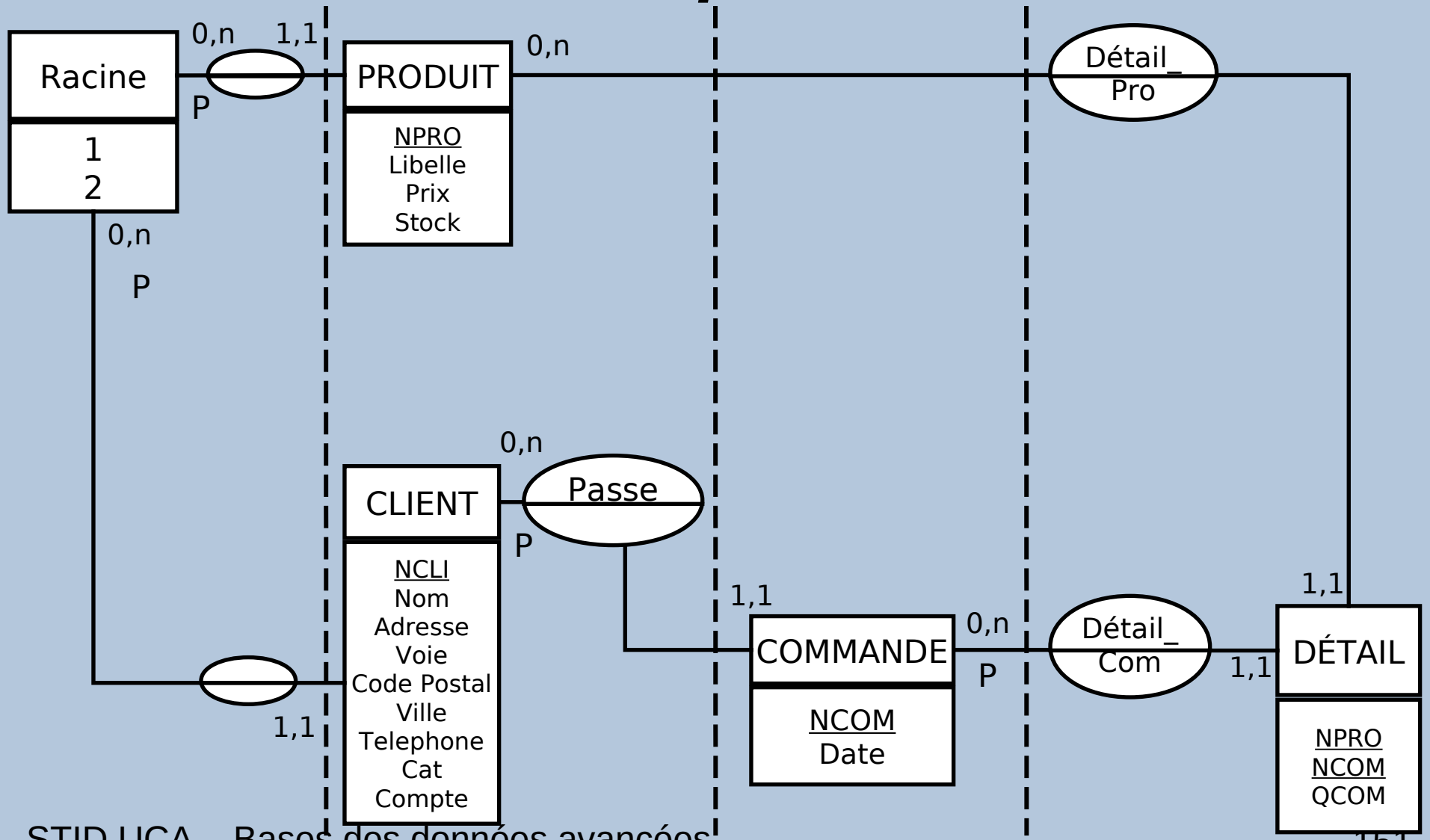
MCD → DTD + XML

Étape 3

- Raffinement de la structure hiérarchique
  - Objectif : obtenir un schéma à racine unique et la représentation du MCD doit avoir une structure d'arbre
  - Création d'une racine unique qui sera sur-type des anciens éléments identifiés comme racines dans étape 2

MCD → DTD + XML

Étape 3



MCD → DTD + XML

Étape 4

- Spécifier le type de contenu des entités
  - Pour chaque entité, il faut préciser l'organisation (le nombre d'occurrence, et l'ordre d'apparitions) de l'ensemble de ses entités sous-types
  - Exemple
    Seul l'élément Racine à plusieurs entités sous-types (Produit et Client)
    et la définition DTD correspondante est
    <!ELEMENT Racine(PRODUIT *, CLIENT *)>

MCD → DTD + XML
Étape 5 et Étape 6

- Éliminer les types d'associations non marqués
  - Faire disparaître les rôles qui n'apparaissent pas comme « P »
  - Les associations non marqués sont transformées en clef étrangères; ce qui se traduit en XML par l'utilisation des mots-clés ID et IDREF
- Modifier la cardinalités des rôles des associations
  - Transformer chaque cardinalité du MCD en cardinalité valide XML (0-1, 1-1, 0-N, 1-N)
    - Si la cardinalité minimale est supérieure à 1, la ramener à 1
    - Si la cardinalité maximale est supérieure à 1, la mettre à N
  - Exemple : pas de modifs

MCD → DTD + XML

Étape 7, 8, 9

- Étape 7 : Gestion des références d'entités
  - Traitements des groupes référentiels
  - Traitements des identifiants
  - Traitements des autres groupes
  - Correspond à tous transformer sous forme de ID, IDREF, IDREFS
- Étape 8 : traitements des attributs
  - Transformer les attributs au format XML donc #PCDATA ou #ANY
- Étape 9 : Renomination
  - Nom de certains objets est superflu ou explicite
- Relativement simple sous le schéma est en 3FN