Cours bases de données

Modèle conceptuel de données, Modèle relationnel, SQL, PL/SQL, C/SQL, Concurrence d'accès, Optimisation de requêtes

Spécialisation informatique 2005/06

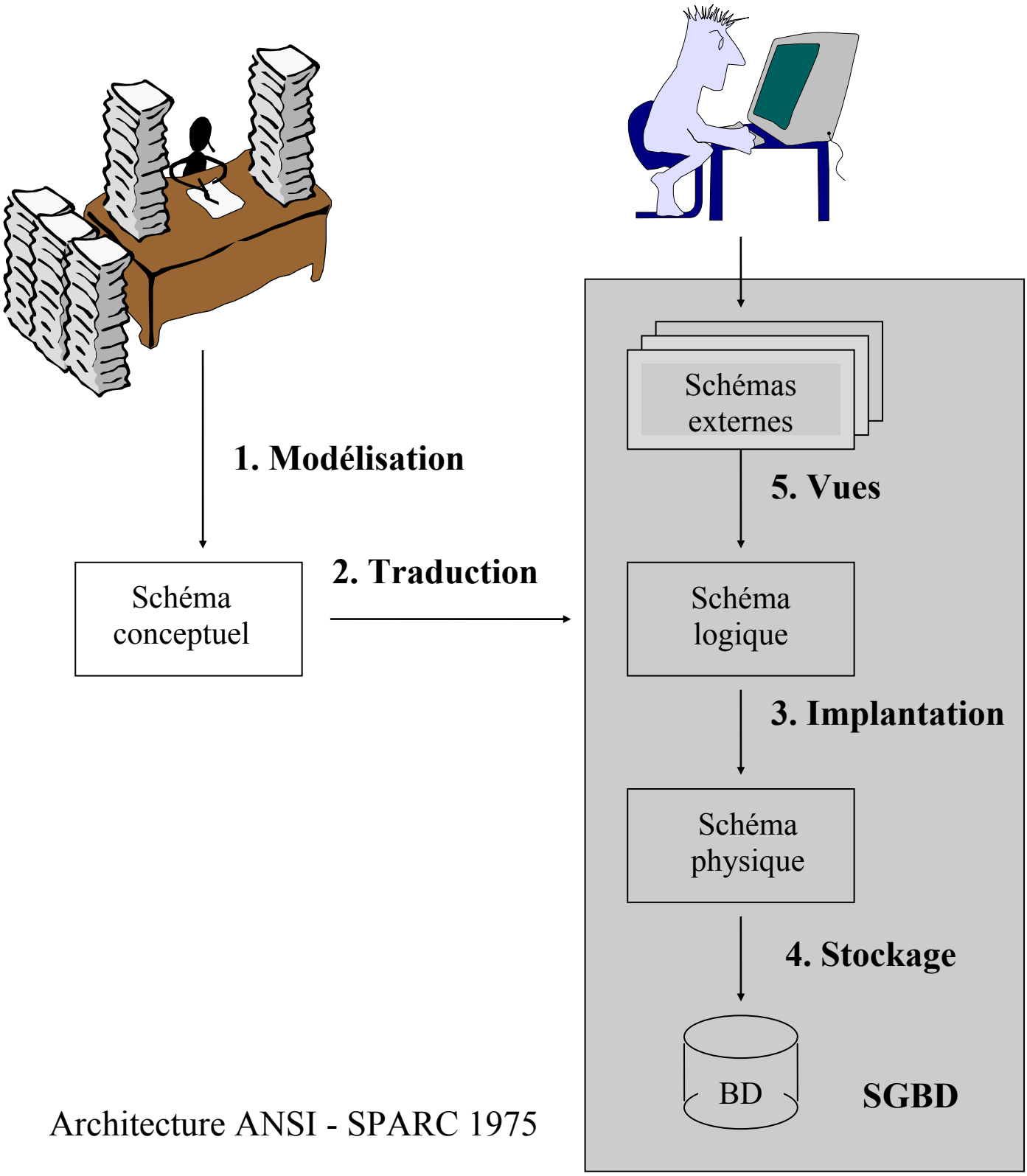
Le présent support de cours a été réalisé par Patrice Buche
UER d'informatique de l'INA PG
16 rue Claude Bernard, 75231 Paris Cedex 05

1. Introduction

1.1 Base de données, une définition

Une base de données est un ensemble **structuré** de données **persistantes** accessibles **aisément** par **plusieurs** programmes qui les utilisent **simultanément** avec des objectifs **différents**.

1.2 Les étapes dans la réalisation d'une base de données



Architecture ANSI - SPARC 1975

1.3 Les acteurs

Acteur	Rôle	Outils	Vie du projet	Réalisation
Maître d'ouvrage	Le demandeur du projet		Du début à la fin	Cahier des charges
Maître d'œuvre	Le réalisateur du projet		Du début à la fin	Documents contractuels
Chef de projet	Coordonne les acteurs, suit la réalisation du projet, contrôle coûts et délais, met en œuvre une démarche qualité.	Méthode de conduite de projet	Du début à la fin	Documents de suivi de projet
Concepteur	Définit le MCD et le dictionnaire de données en interaction avec les utilisateurs futurs	Cahier des charges Méthode de conception	Au début	Schéma conceptuel. Dictionnaire de données initial.
Administrateur de données	Suit l'évolution du dictionnaire de données, participe aux spécifications du produit, interface entre BD et utilisateurs d'une part et BD et SGBD d'autre part.	Outil de conception	Du début à la fin	Dictionnaire de données, dossier de spécifications, modèle physique de données
Architecte	Définit l'architecture du système et le déploiement des applications.		Au début	Schéma d'architecture
Administrateur système	Gère l'impantation physique de la base, met en œuvre des procédures d'optimisation, de sauvegarde, ...	Le SGBD	Du début à la fin	
Développeur d'applications	Réalise les applicatifs pour l'utilisateur.	Dossiers de spécifications	Au début	Maquettes d'interfaces, prototypes, produit final.

1.4 Système de gestion de base de données

Aujourd'hui, il existe quatre catégories principales de SGBD :

- relationnels,
- objet-relationnels,
- objets,
- XML.

Un SGBD est un logiciel offrant 7 services :

- Persistance
- Gestion du disque
- Partage des données
- Fiabilité des données
- Sécurité des données
- Indépendance logique/physique
- Langage de description, d'interrogation et de traitement des données

1.4.1 Persistance des données

Des données sont persistantes si elles survivent à l'exécution des programmes.

La persistance est obtenue en stockant les données sur disque.

1.4.2 Gestion du disque

Toutes les données de la base ne peuvent pas en général tenir en mémoire centrale.

Un simple mécanisme de pagination virtuelle n'est pas suffisant pour obtenir des temps d'accès acceptables aux informations.

Une entrée/sortie disque est coûteuse.

Les SGBD intègrent des techniques spécifiques pour avoir de bonnes performances :

- Index, hash-coding
- Regroupement des données sur disque
- Optimisation des requêtes
- Cache mémoire

1.4.3 Partage des données

Les données sont partagées par un ensemble d'utilisateurs. Chacun d'entre eux doit avoir l'illusion d'être seul.

Notion de transaction (begin, abort, commit), unité de cohérence des mises à jour effectuées par un utilisateur.

Notion de cohérence collective : sérialisabilité.

Support pour la sérialisabilité des transactions :

- stratégie pessimiste : le verrouillage à deux phases (pose et libération des verrous).
- stratégie optimiste : l'estampillage.

Le contrôle de la concurrence porte sur :

- la base de données
- le schéma
- les index

1.4.4 Fiabilité des données

Les données sont chères et stratégiques : elles doivent être fiables.

Les données vérifient des contraintes d'intégrité (intégrité référentielle, réflexes).

Atomicité des transactions : la transaction doit être complètement effectuée ou pas du tout.

Résistance aux pannes :

- en cas de panne mémoire : restauration automatique de la base intégrant les dernières transactions validées avant la panne.
- en cas de panne disque : restauration d'une sauvegarde et déroulement du journal archivé.
- mise en place d'un mécanisme de réplication synchrone de la base dans une base miroir : mirroring.

1.4.5 Sécurité des données

Tous les utilisateurs ne peuvent pas tout faire sur toutes les données.

- Notion de groupes d'utilisateurs.
- Notion d'autorisation (lecture, écriture, exécution).
- Granularité des autorisations : base de donnée, table, colonne, nuplet, ...
- Possibilité d'accorder ou de supprimer des droits.

1.4.6 Indépendance logique/physique

L'organisation physique de la base de données est transparente pour le développeur d'application.

On doit pouvoir changer la répartition des données sur le disque (exemple : un regroupement ou la pose d'un index) sans changer le code de l'application manipulant les données.

1.4.7 Langage de requête

Les requêtes doivent être :

- simples,
- déclaratives,
- optimisées avant leur exécution

Les langages de requêtes ne sont pas complets.

L'état de l'art : SQL et QBE, OQL, XQUERY

1.5 L'architecture client-serveur et les bases de données

Depuis le début des années 90, les réseaux LAN (Local Area Network) et WAN (Wide Area Network) sont devenus l'épine dorsale du système informatique des entreprises en s'appuyant sur la technologie Internet.

Depuis le milieu des années 90, cette technologie est mise à la disposition du grand public (WEB).

L'accès client-serveur aux données est donc devenue une activité essentielle pour un grand nombre de personnes partout dans le monde.

Les types d'accès sont :

- accès informationnel (requête d'interrogation et réponse),
- accès transactionnel (mises à jour rapides en temps réel),
- en accès décisionnel (exécution de requêtes complexes sur de gros volumes de données).

Mise en place de serveurs offrant :

- des systèmes de gestion de bases de données gérant le partage des données,
- des interfaces standards permettant la connectivité des applications exécutées en général sur ordinateurs personnels.

Les architectures client-serveur sont un modèle d'architecture où les programmes sont répartis entre processus clients et serveurs communiquant par des requêtes avec réponse.

1.5.1 Définitions de bases

Client

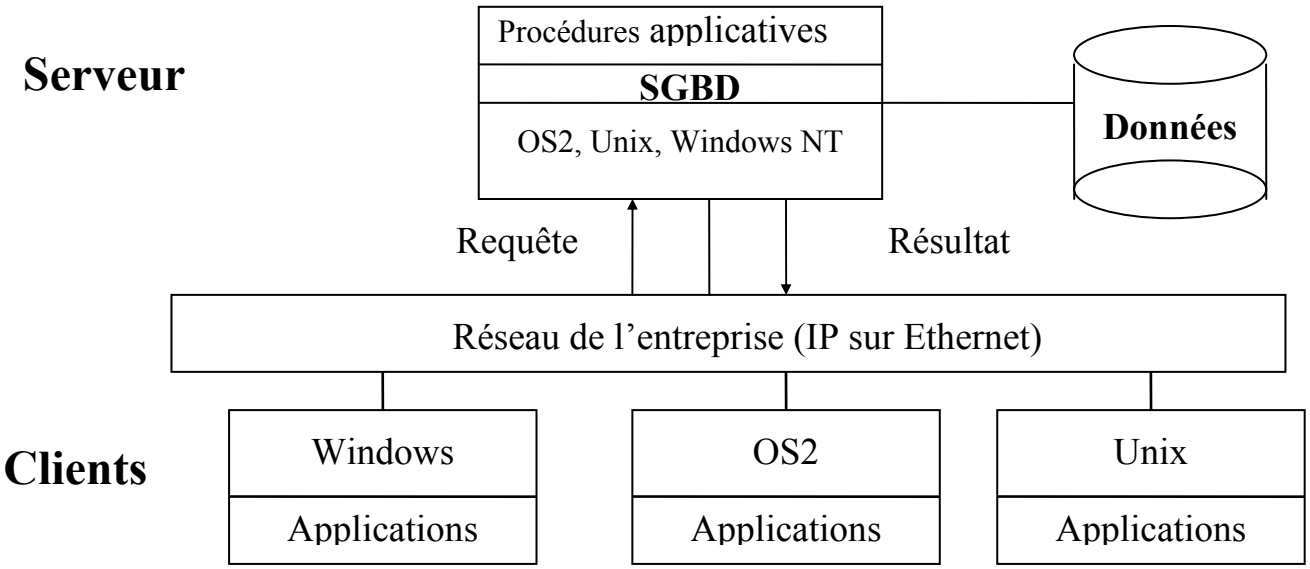
- Processus demandant l'exécution d'une opération à un autre processus par envoi d'un message (requête) contenant le descriptif de l'opération à exécuter et attendant la réponse à cette opération par un message en retour.
- Il supporte une partie du code de l'application. Le code implémente au minimum les dialogues avec les utilisateurs et l'affichage des résultats (GUI : Graphical User Interface).

Serveur

- Processus accomplissant une opération sur demande d'un client et transmettant la réponse à ce client.
- Il assure le stockage, la distribution, la gestion de la disponibilité et de la sécurité des données. Ces fonctionnalités sont offertes par le SGBD résidant sur le serveur ou accessibles sur un autre serveur connecté au réseau. Il supporte une partie du code de l'application : un certain nombre de traitements sur les données peuvent être regroupés et exécutés sur le serveur.

1.5.2 Deux générations de clients/serveurs : du gros client ventru au petit client chétif

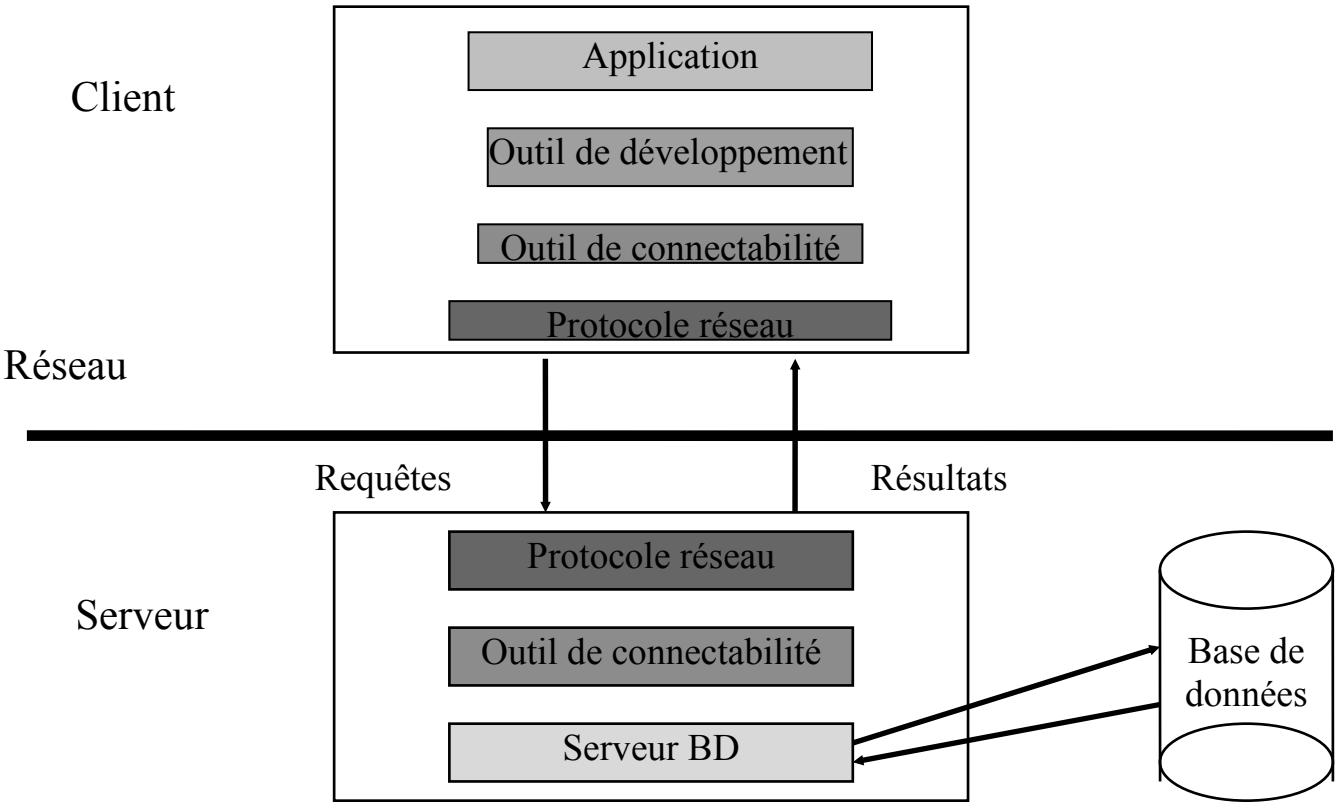
1.5.2.1 Première génération : Client-Serveur sur le réseau d'entreprise



Avantages du C/S de première génération :

- Cette architecture supporte des machines hétérogènes (Windows 3.11, Windows 95 ou 98, Windows NT, Mac OS, Unix, ...)
- Répartition des traitements entre clients et serveurs
- Relative indépendance vis à vis du choix des protocoles réseau et des systèmes d'exploitation sur les serveurs et les clients. Les vendeurs de logiciel doivent aujourd'hui avoir cette flexibilité.

Les composants de l'architecture C/S



Outil de connectabilité, généralement appelé le middleware, est le logiciel qui implémente le protocole qui permet au processus client de dialoguer avec le processus serveur (ODBC, SQL Net, ..).

Outils de développement :

- langages de 3ème génération : programmes C, Cobol appelant des bibliothèques du SGBD.
- langages de 4ème génération : ils offrent des composants logiciels qui permettent de développer très rapidement une application (Access, PowerBuilder, Uniface, ...) et intègrent un langage de programmation en général interprété.

Inconvénients du C/S de première génération :

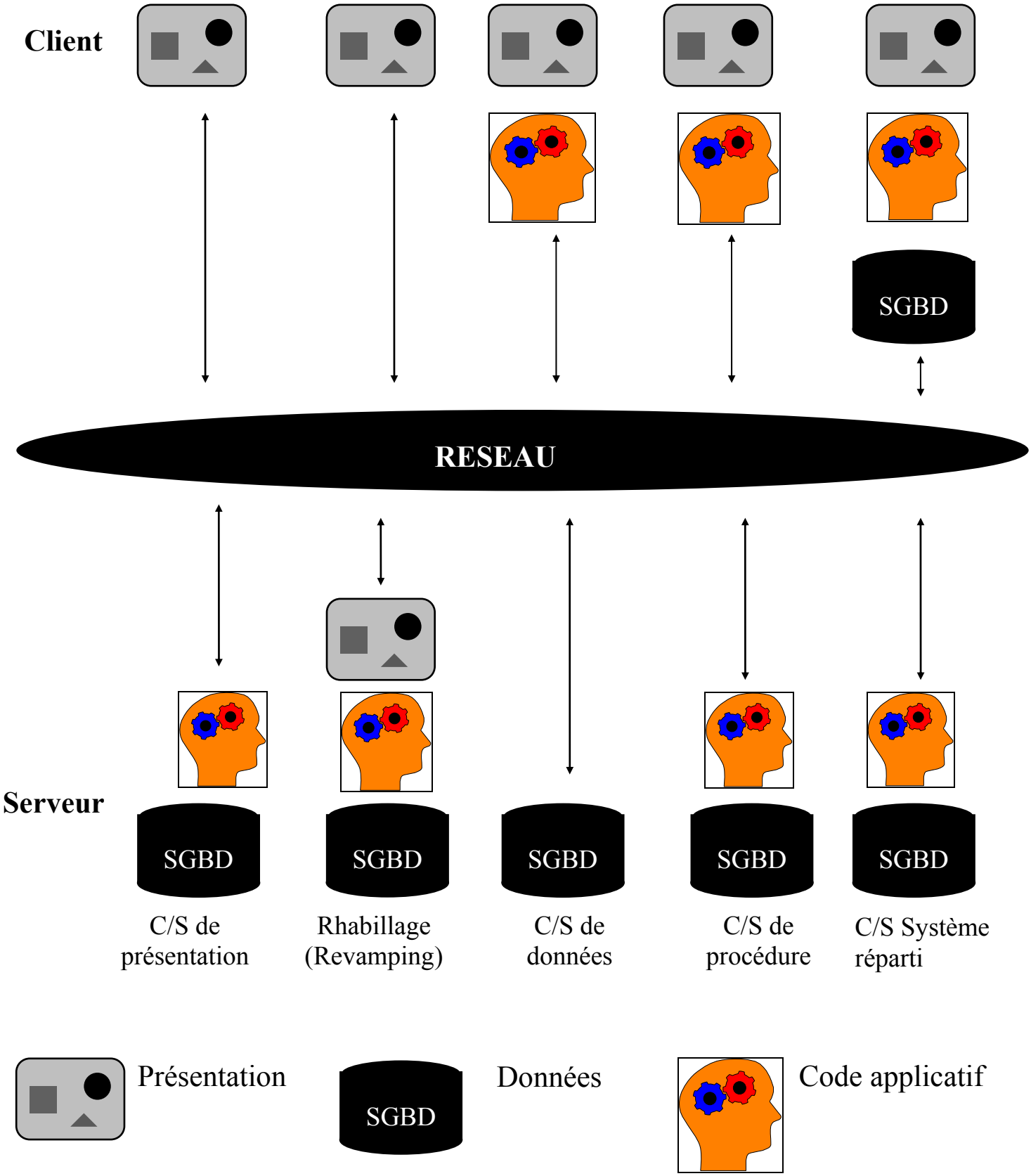
La technologie C/S est diffusée depuis 10 ans dans les entreprises. Le retour d'expérience est : l'administration du C/S coûte cher ! !

- Il est parfois nécessaire de fabriquer une version d'application par type de plateforme
- En cas de mise à jour, il faut déployer la mise à jour sur tous les clients
- Sur chaque client, lorsque l'on installe un logiciel, il faut installer le ou les middlewares qui lui permet(tent) de communiquer avec des serveurs.
- Peu de développeurs maitrisent réellement l'ensemble des technologies mise en œuvre dans une application C/S.

Etude du Gartner Group en 1996 :

administrer un PC coûte 12000 US$ par an.

Les différents types de C/S de première génération

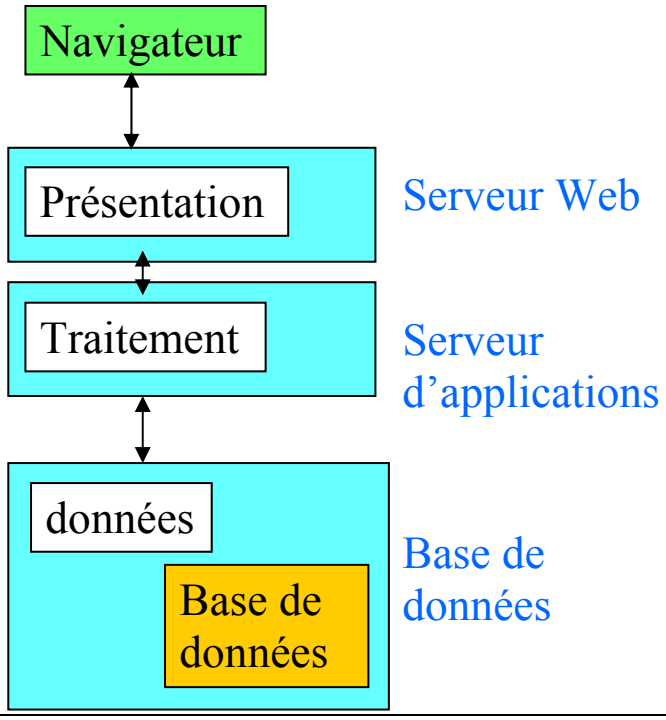


1.5.2.2 Deuxième génération : Client-serveur sur le WEB

Internet et Java sont les solutions largement utilisées pour réduire ces coûts .

Le petit client chétif est un PC équipé d'un navigateur Internet. Les applications sont des pages HTML téléchargées sur le PC.

On parle dans ce cas d'architecture three-tiers car le serveur WEB et le serveur d'application jouent le rôle d'intermédiaires entre le client et le serveur de la base de données.



On peut distinguer deux types de client-serveur sur Internet :

- le C/S HTML/WEB (CGI : Common Gateway Interface, servlets Java),
- le C/S à composants distribués (applets Java, Active X).

Principe du C/S HTML/WEB :

L'utilisateur saisit des informations dans un formulaire HTML. Le navigateur envoie une requête HTTP au serveur WEB. Celle-ci contient le nom d'un programme exécutable depuis le serveur ainsi que les informations saisies dans le formulaire. Le programme est exécuté sur le serveur WEB (ou un autre serveur du réseau): il interroge la base de données et renvoie au navigateur la réponse formatée en HTML.

Avantages :

- Une seule version des applications (sur le serveur)
- Mise à jour de l'application uniquement sur le serveur
- Compatible avec les modèles de sécurité (firewall, ...)

Inconvénients :

- La gestion du contexte de session est plus lourde à gérer que dans la solution composants distribués.

Principe du C/S à composants distribués :

Le navigateur WEB télécharge un composant logiciel (Active X ou applet Java) stocké sur le serveur WEB. Le composant s'exécute sur le navigateur. On peut à nouveau distinguer deux types d'architectures :

- Fat client : l'applet gère l'ensemble des traitements (interface graphique, connexion BD, traitements spéficiques de l'application). Il établit la connexion avec la base de données en utilisant le serveur WEB comme serveur de connexion (JDBC, SQLJ).
- Thin client : l'applet gère l'interface graphique. Les traitements sont répartis entre client et serveur. L'application est distribuée sur le réseau, mais le programmeur peut accéder aux objets gérés par le serveur en manipulant des pointeurs comme s'il s'agissait d'objets locaux. Trois protocoles actuellement permettent d'utiliser ce type d'architecture : RMI (Remote method invocation), DCOM (Distributed Component Object Model), et CORBA (Common Object Request Broker Architecture)

Avantages :

- Une seule version des applications (machine virtuelle Java sur chaque plateforme).
- Les mises à jour sont faites uniquement sur le serveur.

Inconvénients :

- Incompatibilité avec les modèles de sécurité lors du déploiement
- Installation de la Java Runtime sur les postes clients.

- Java est un langage interprété, problèmes de performance possibles.

1.6 Plan du cours

Le modèle conceptuel de données

Le modèle relationnel

Les langages pour le développeur

- Le langage SQL
- Le langage PL/SQL
- L'interface C/SQL

Fonctionnement du SGBD

- Gestion des accès concurrents
- Optimisation des requêtes
- Architecture du SGBD

2. Le modèle conceptuel des données (MCD)

2.1 Concepts de base

Les concepts de base du modèle entité-association (encore appelé modèle conceptuel des données) sont la **propriété**, **l'entité**, **l'association** et les **cardinalités**.

2.1.1 Propriétés

La propriété est définie comme étant une donnée élémentaire :

▲ ayant un sens,

▲ pouvant être utilisée de manière autonome.

Exemples : NomAssuré, AnnéeSouscription, NbContrats

Les propriétés servent à décrire les entités et les associations. Ce sont donc des particules d'information. On les appelle également attributs ou colonnes (dans le modèle relationnel).

Les propriétés prennent des valeurs appelées **occurrences** de la propriété.

Propriété	**Occurrences**
NomAssuré	Dupont
	Dupond
	Martin
AnnéeSouscription	1988
	1989

2.1.2 Entités

Une entité est définie comme étant un objet concret ou abstrait :

▲ ayant une existence propre,

▲ présentant un intérêt pour l'entreprise,

▲ traduisant une préoccupation de gestion.

Exemples : Assuré, Contrat.

Les entités possèdent un ensemble de propriétés. Une occurrence de l'entité est composée d'une occurrence de chacune de ses propriétés.

Entité	**Propriétés**	**Occurrences des entités**	
Assuré	NomAssuré	Dupont	Martin
	AgeAssuré	32	58
Contrat	NumContrat	IN15	CO81
	AnnéeSouscr	1988	1991

Dans une entité, une propriété joue un rôle particulier. Il s'agit de **l'identifiant** (encore appelée la clé). Ses valeurs sont discriminantes. Deux occurrences distinctes de l'entité ne peuvent avoir même valeur pour la propriété identifiante.

Exemple : NumContrat

2.1.3 Associations

Une association est définie comme un lien sémantique :

▲ reliant un ensemble d'entités,

▲ présentant un intérêt pour l'entreprise,

▲ traduisant une préoccupation de gestion.

Exemples : **Assurer** entre les entités Contrat et Assuré.

Une association présente les caractéristiques suivantes :

▲ elle n'a d'existence qu'au travers des entités qu'elle relie,

▲ elle n'a pas d'identifiant propre,

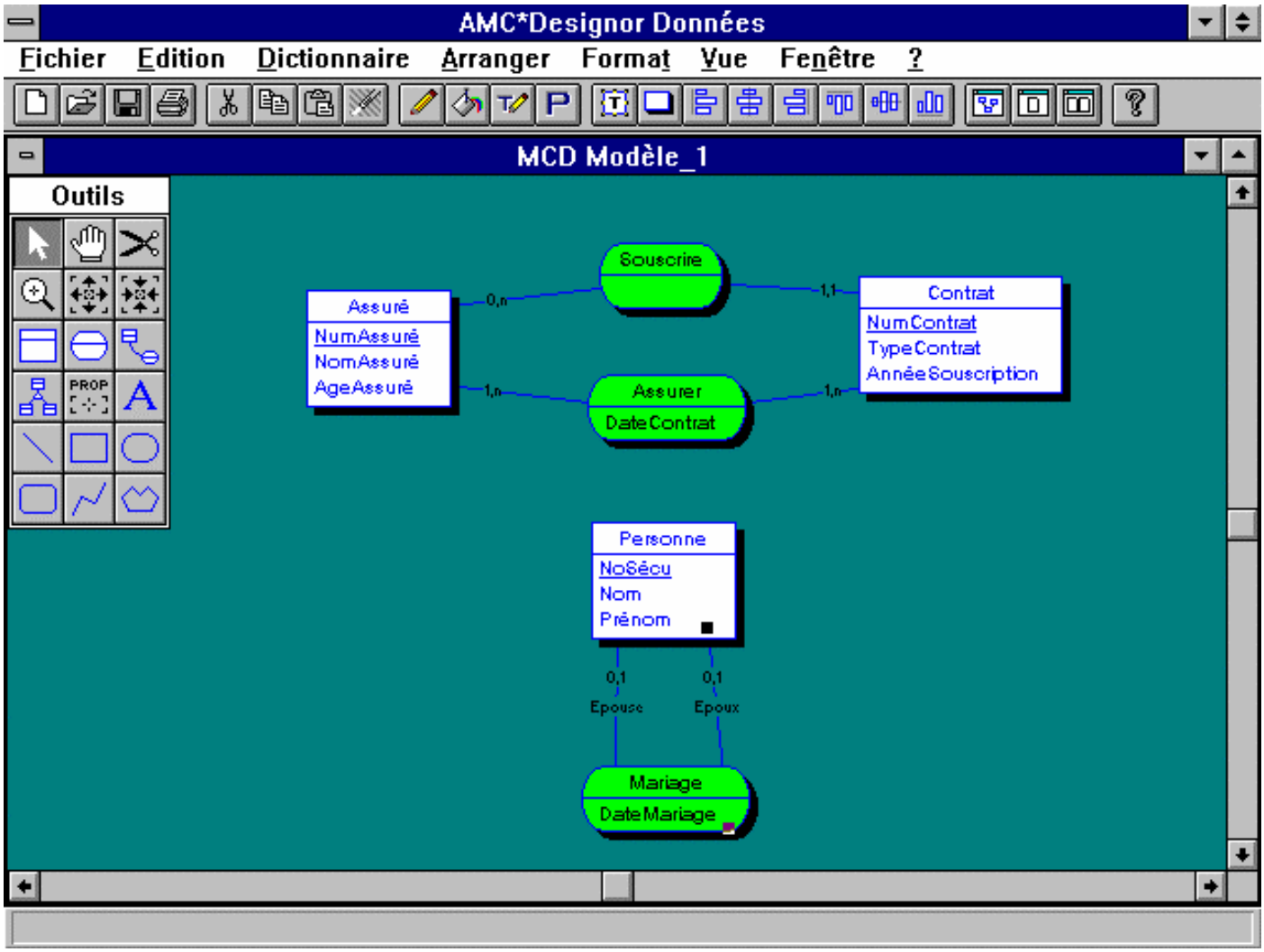
▲ elle est identifiée par la concaténation des identifiants des entités reliées,

▲ elle peut posséder des propriétés.

Les occurrences de l'association Assurer peuvent être les suivantes :

#NumContrat	#NumAssuré	Date
IN15	127	01/06/96
IN15	356	15/08/96
CO81	127	05/02/95

2.1.4 Représentation graphique



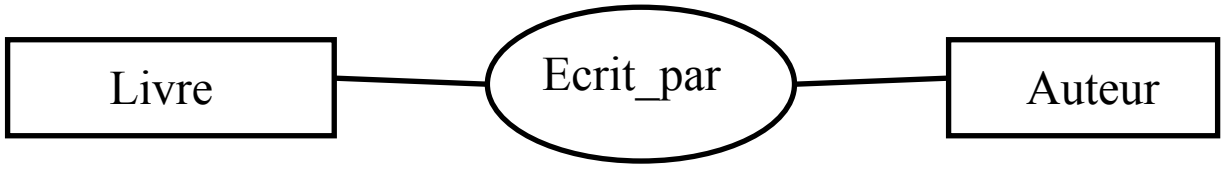
Il peut exister plusieurs associations entre les mêmes entités.

Une association peut être réflexive, c'est-à-dire qu'elle relie une entité à elle-même.

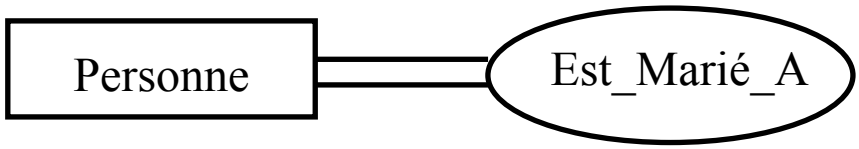
2.2 Caractéristiques d'une association

Dimension : La dimension d'une association est le nombre d'entités concernées par celle-ci.

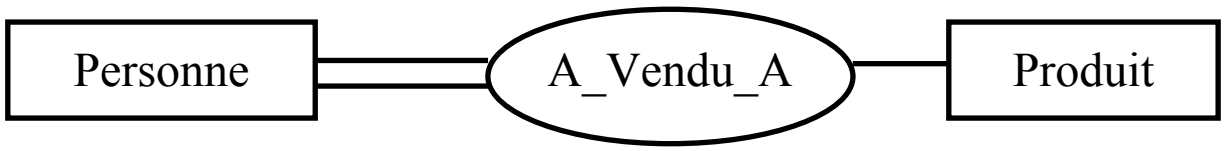
- Exemple : L'association Ecrit_Par est de dimension deux (binaire) :



- Exemple : L'association Est_Marié_A est de dimension deux (binaire réflexive) :

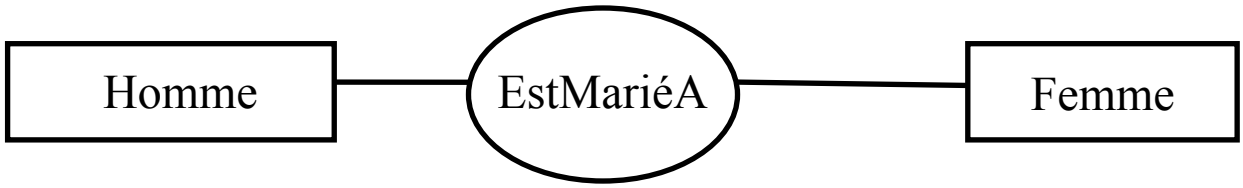


- Exemple : L'association A_Vendu_A est de dimension trois (ternaire) :



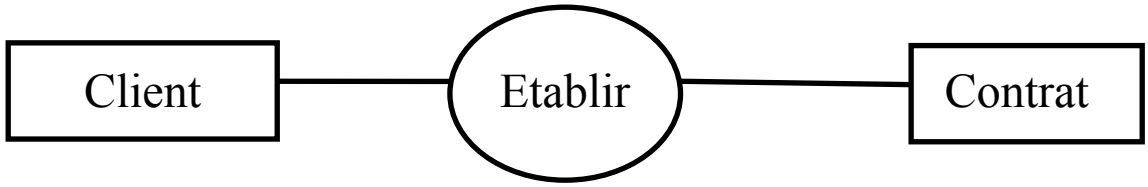
Type de liaison inter-entités : On distingue trois types de liaisons entre deux entités X et Y participant à l'association.

- **Liaison de type 1 à 1 :** A toute occurrence de X correspond une et une seule occurrence de Y et réciproquement.



Dans le mariage judéo-chrétien, un homme est marié à une seule femme et réciproquement

- **Liaison de type 1 à plusieurs (1 à n) :** A toute occurrence de X correspond une ou plusieurs occurrences de Y et à toute occurrence de Y une seule de X.



Un client a établi un ou plusieurs contrat(s) mais un contrat a été établi avec un seul client.

- **Liaison de type plusieurs à plusieurs (n à m) :** A toute occurrence de X correspond une ou plusieurs occurrences de Y et réciproquement.

Un client peut commander plusieurs produits et un produit peut être commandé par plusieurs clients.

2.3 Cardinalités

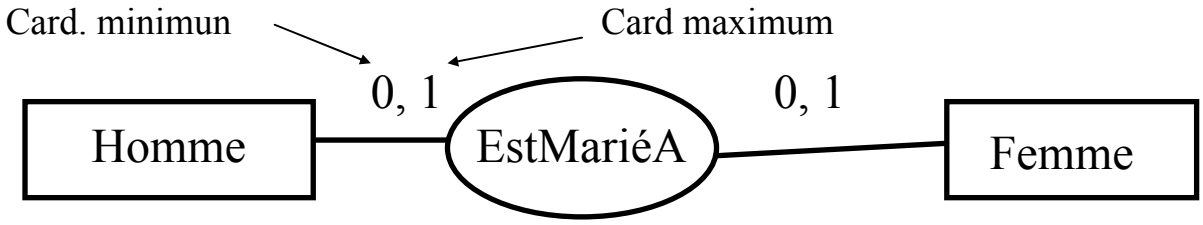
La notion de cardinalité minimum/maximum est liée aux types de liaison inter-entités.

- La cardinalité minimum est le nombre minimum d'occurrences d'une entité X dans l'association considérée.

- La cardinalité maximum est le nombre maximum d'occurrences d'une entité dans l'association

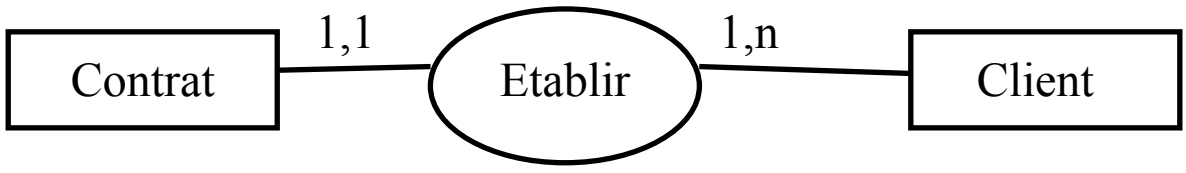
- Exemples :

Un lien inter-entités 1 à 1 (card. max à 1 pour les deux entités).



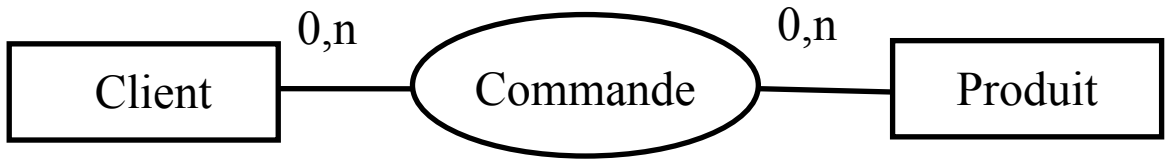
Un homme (resp. une femme) est marié ou célibataire

Un lien inter-entités 1 à n (card. max à 1 pour l'entité Contrat et card. max à n pour l'entité Client).



Un client a établi un ou plusieurs contrat, un contrat a été établi par un seul client

Un lien inter-entités n à m (card. max à n pour les deux entités).



Un client peut avoir fait de 0 à n commandes, un produit est commandé de 0 à n fois.

Les valeurs de cardinalités sont en général 0, 1 ou n:

- La cardinalité minimum à 0 veut dire que certaines occurrences de l'entité X ne sont pas impliquées dans une occurrence de l'association.

- La cardinalité minimum à 1 veut dire qu'une occurrence de l'entité X ne peut exister sans participer à une occurrence de l'association.

- La cardinalité maximum à 1 veut dire que toute occurrence de l'entité X ne peut participiter au plus qu'à une occurrence de l'association.

- La cardinalité maximum à n veut dire qu'une occurrence de l'entité X peut être impliquée dans un maximum de n occurrences de l'association.

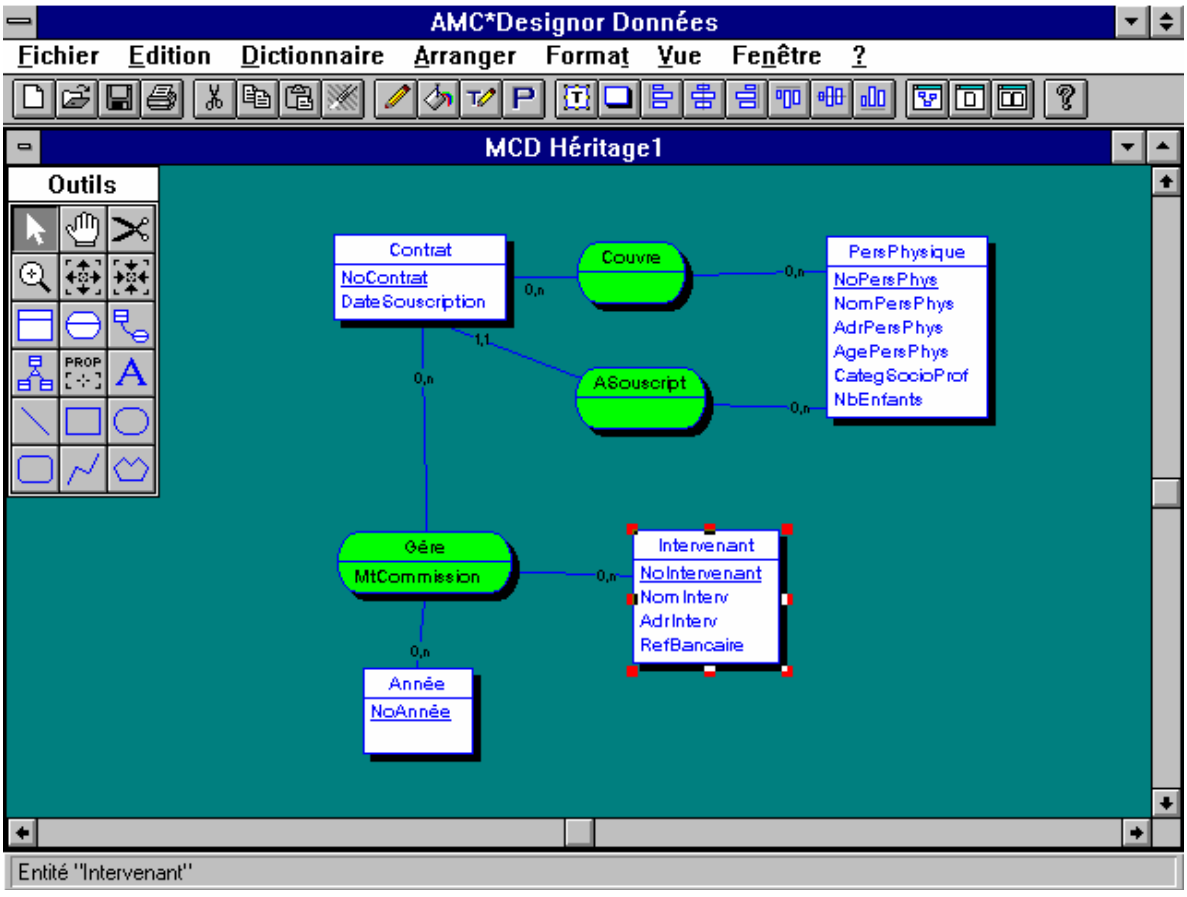
2.4 Généralisation-spécialisation

On introduit des spécialisations dans une entité lorsque :

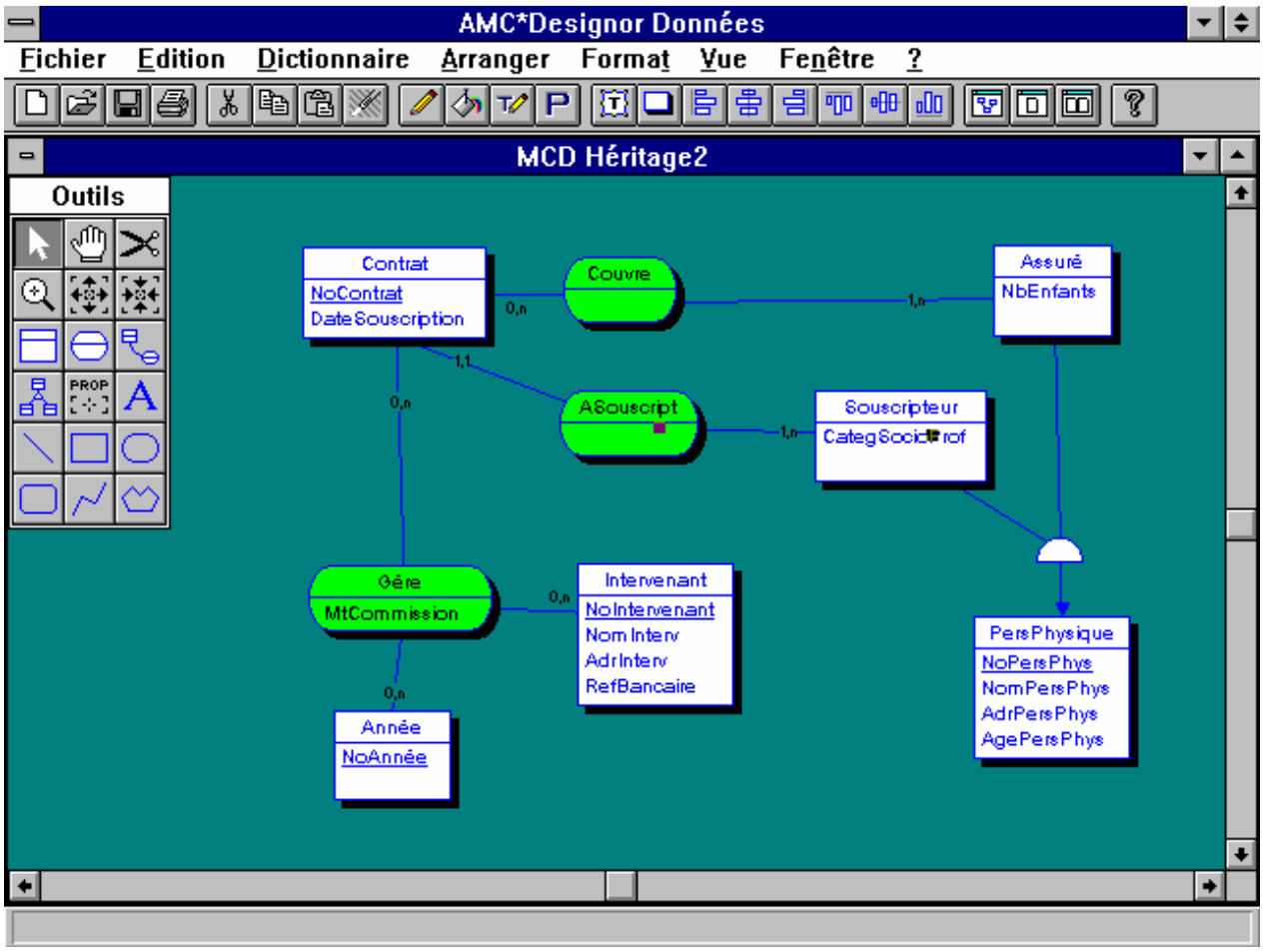
▲ certaines propriétés n'ont pas de sens pour l'ensemble des occurrences de cette entité.

▲ le lien (association) entre deux entités n'a de sens que pour certaines occurrences de ces entités.

Exemple : La gestion d'un contrat d'assurance du point de vue de son souscripteur, de ses assurés et de ses intervenants (courtiers, agents).



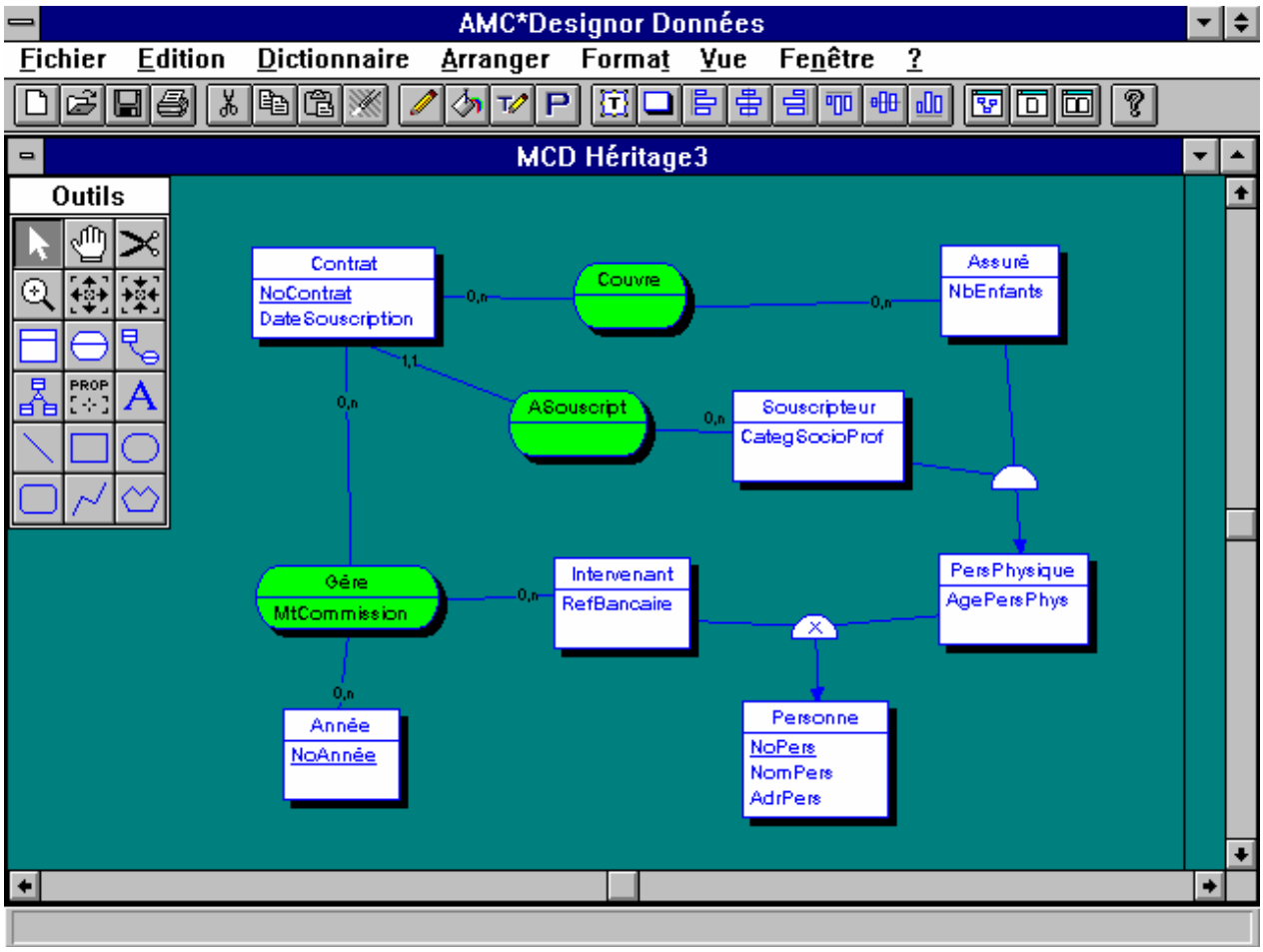
A partir de l'entité PersPhysique, on a envie de distinguer dans deux entités spécialisées deux sous-populations : les souscripteurs de contrat et les assurés. L'héritage est non exclusif : certaines personnes peuvent être à la fois souscripteurs et assurés.



Dans ce modèle avec entités spécialisées, toutes les occurrences d'assurés et de souscripteurs sont concernés par les associations (cf cardinalités).

Par généralisation des entités Intervenant et PersPhysique, on peut identifier une entité générique Personne.

Cette fois-ci, l'héritage est **exclusif** (symbolisé par une croix) : les intervenants sont considérés en tant que personnes morales et les autres (souscripteurs et assurés) sont considérés en tant que personnes physiques.



2.5 Identification relative

Une entité est définie comme étant un objet concret ou abstrait :

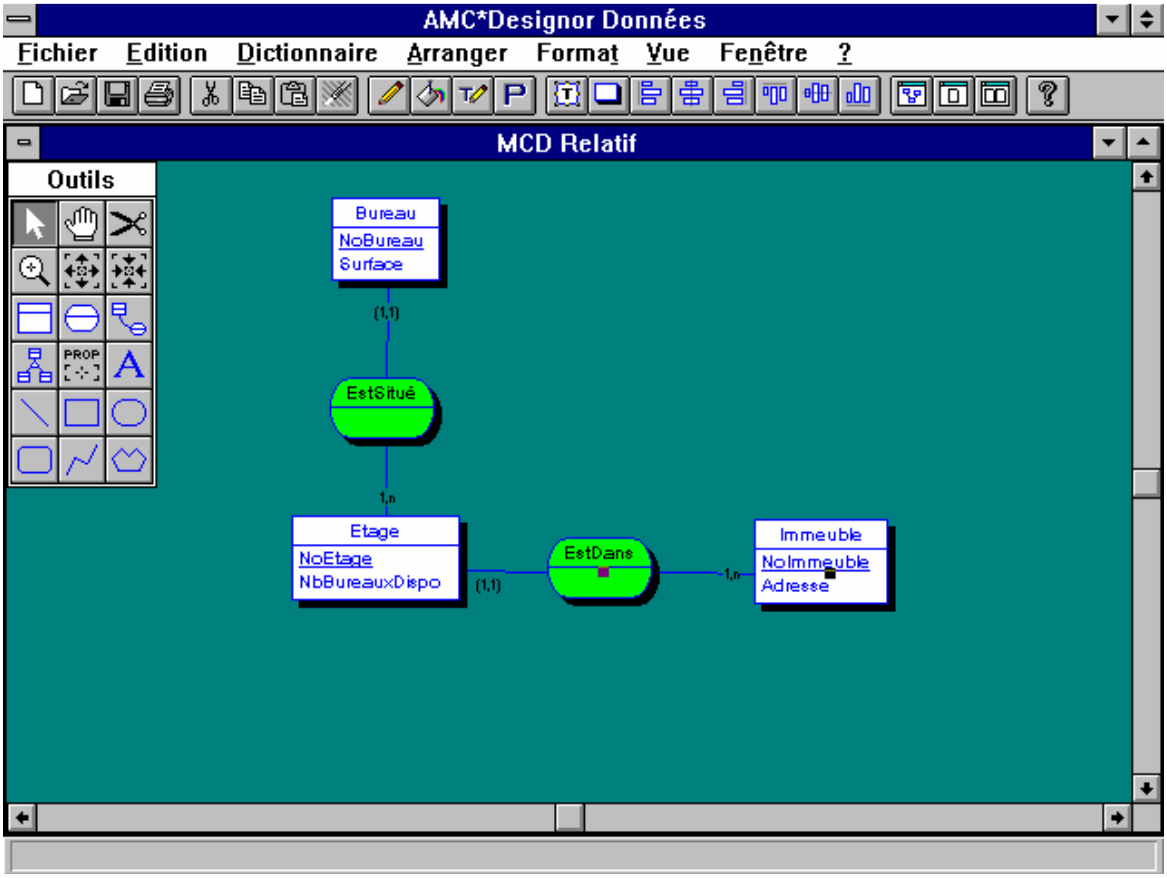
▲ ayant une existence propre qui peut être soit :

- o ***indépendante*** *de l'existence des autres entités (identifiant absolu),*
- o ***dépendante*** *de l'existence d'une autre entité (identifiant relatif),*

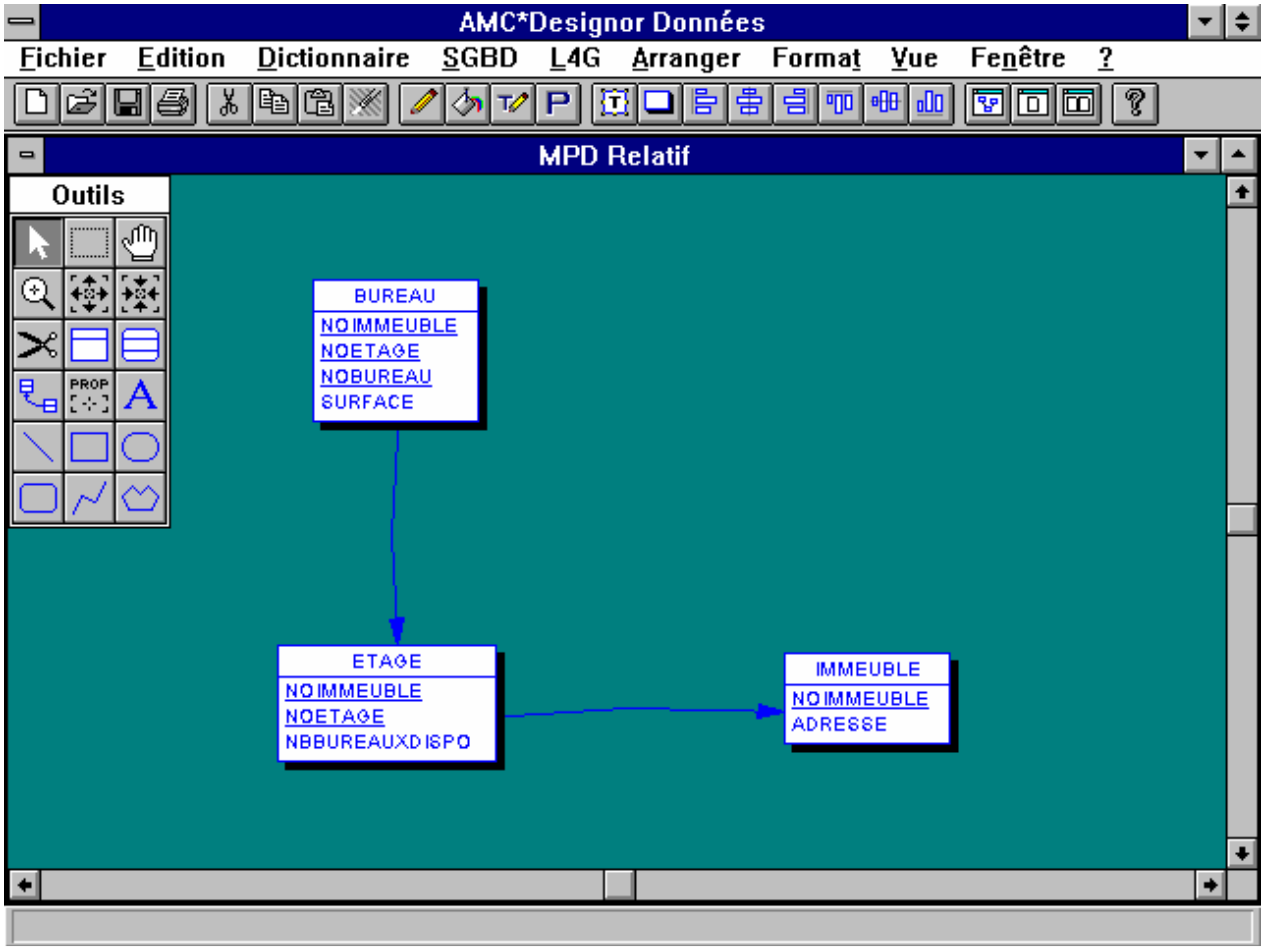
▲ présentant un intérêt pour l'entreprise,

▲ traduisant une préoccupation de gestion.

Exemples : Immeuble, Etage, Bureau.



Le modèle relationnel dérivé est :



2.6 Esquisse d'une méthode d'élaboration d'un MCD

 Déterminer les entités puis les associations

 Déterminer les cardinalités entre entités et associations

 Recenser les attributs des entités et associations (constitution du dictionnaire de données)

Polysémie : Deux ou plusieurs propriétés portent le même nom alors qu'elles représentent sans ambiguïté des notions sémantiques différentes.

▲ Exemples : taux ou date

▲ Dans le dictionnaire de données, on lévera l'ambiguïté en précisant TauxTva, TauxRemise, DateContrat, DateMariage.

Synonymie : Des propriétés sous des noms différents désignent la même notion sémantique.

▲ Exemple : base de données, table, entité

▲ On choisit un nom pour le modèle de données et on conserve dans le dictionnaire de données l'ensemble des synonymes.

 Les sources sont :

- l'interview des utilisateurs
- des fichiers manuels
- des structures de fichiers informatiques existants
- ...

2.7 Un exemple : La gestion d'une bibliothèque

Soit le dictionnaire des données :

- numéro de livre : n° attribué à chaque livre présent dans la bibliothèque

- numéro ISBN : n° attribué à chaque ouvrage par une commission nationale. Un ouvrage peut exister en plusieurs exemplaires dans la bibliothèque. Chaque exemplaire ou livre a un n° de livre, interne à la bibliothèque

- état du livre : (3 valeurs : bon, moyen, mauvais)

- titre de l'ouvrage

- nombre de pages de l'ouvrage

- nom de ou des auteur(s)

- mot-clé : un ouvrage (et un livre) peut être caractérisé par plusieurs mots-clés. Un mot-clé est un libellé, défini en vue de recherches. Exemples : informatique, compilation, musique, rock

- numéro d'adhérent
- nom d'adhérent (en fait, nom et prénom dans une seule propriété)
- commune de l'adhérent
- date de l'emprunt.

Question : Déterminez le modèle conceptuel de données selon le formalisme de MERISE.

Mise en évidence des entités

LIVRE
No Livre Etat Livre

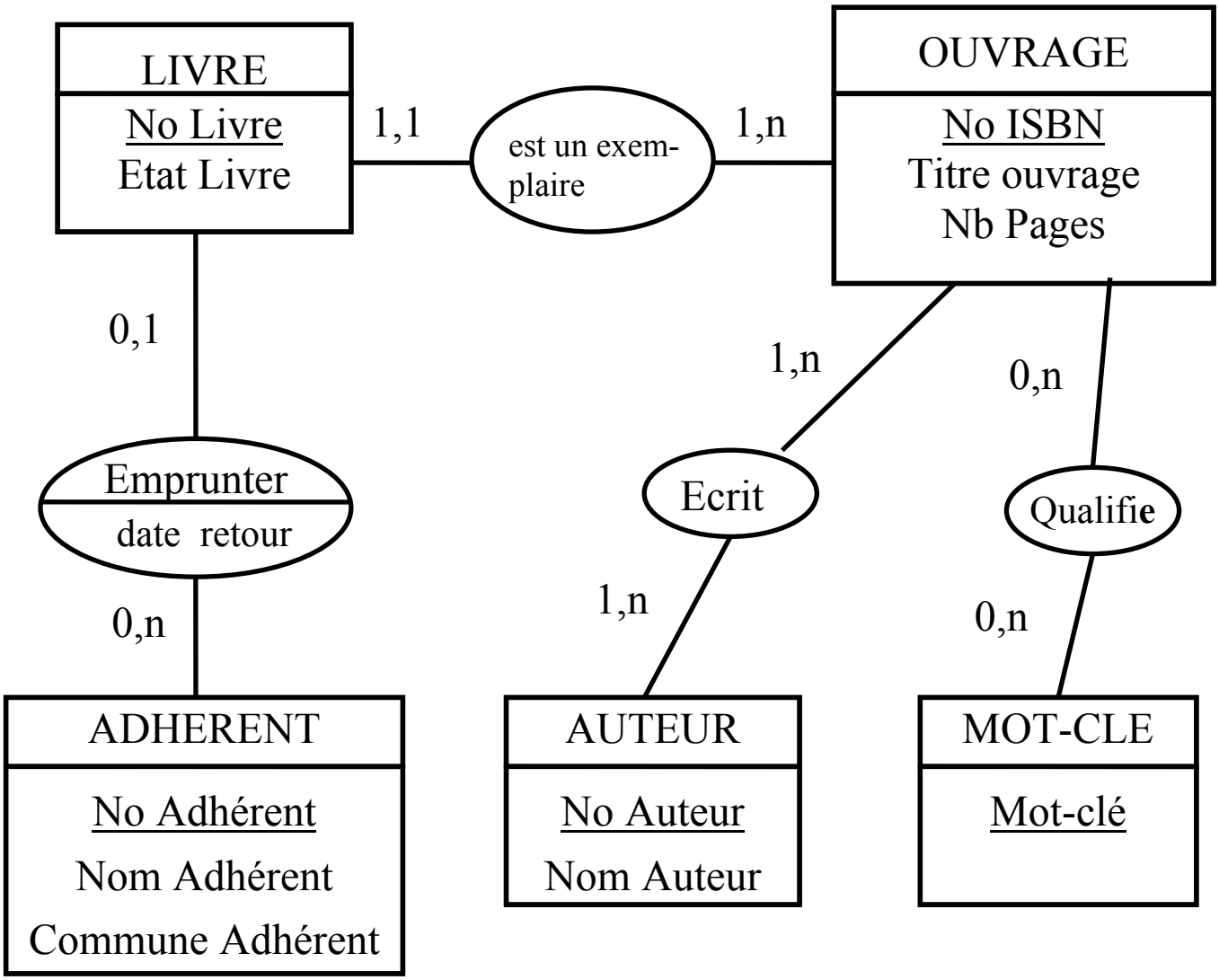
OUVRAGE
No ISBN Titre ouvrage Nb Pages

ADHERENT
No Adhérent Nom Adhérent Commune Adhérent

AUTEUR
No Auteur Nom Auteur

MOT-CLE
Mot-clé

Mise en évidence des associations



2.8 Réalisez le MCD du cas location de voitures

Une société de location de voiture désire automatiser son service commercial et le suivi des véhicules. Actuellement, les supports utilisés par l'entreprise sont :

- Un catalogue donnant le tarif de chaque modèle ; l'accès à ce catalogue se fait par marque et modèle.
- Un bac de fiches clients dont l'accès se fait par numéro de client.
- Un classeur où sont rangées les fiches des véhicules, l'accès se faisant par numéro minéralogique du véhicule.
- Les contrats, rangés par numéros dans des classeurs.
- Un planning mural indiquant pour chaque véhicule s'il est en location ou en révision.
- Le carnet d'entretien de chaque véhicule, rangé dans un meuble à casiers, dans la case correspondant au modèle.

Notations :

- X(10) signifie une chaine de 10 caractères
- 9(2) signifie un nombre sur deux positions
- 9(2) occ 3 signifie un tableau de 3 nombres sur deux positions.

Document 1

Désignation : Modèle et tarifs

Classement : Marque
Modèle
Numéro minéralogique des véhicules

Description des informations :

Marque	X (10)
Modèle	X (13)
Puissance	9 (2)
Tarifs de location	9 (8) occ 3 (liste de 3 tarifs)
Montant de la caution	9 (4)
Numéro minéralogique	X (8) occ 10 (liste de 10 numéros)

Observations :

* Il y a trois tarifs par modèle :
  un tarif au km, un forfait week-end, un forfait semaine

* La société gère 50 véhicules répartis en 10 modèles et 4 marques

Document 2

Désignation : Clients

Classement : Numéro de client

Description des informations :

Numéro du client	9 (3)
Nom du client	X (30)
Adresse du client	X (42)
Code postal	9 (5)

Observations : rien

Document 3

Désignation : Véhicules

Classement : Numéro minéralogique

Description des informations :

Numéro minéralogique	X (8)
Marque	X (10)
Couleur	X (10)
Kilométrage en cours	9 (5)
Modèle	X (13)
Puissance	9 (2)

Document 4

Désignation : Contrats

Classement : Numéro de contrat

Description des informations :

Numéro de contrat	9 (5)
Date d'établissement	9 (6)
Date prévue de prise en charge	9 (6)
Date prévue de restitution	9 (6)
Numéro du client	9 (3)
Nom du client	X (30)
Adresse du client	X (42)
Code postal	9 (5)
Marque du véhicule	X (10)
Modèle du véhicule	X (13)
Numéro minéralogique	X (8)
Tarif	9 (8)
Montant caution	9 (4)

Observations :

Pour un même client, le nombre maximum de contrats rencontrés est de 10.

Document 5

Désignation : Planning

Classement :
- Marque
- Modèle
- Numéro minéralogique

(Ligne)

- Jour

(Colonne)

Description des informations :

Marque	X (10)
Modèle	X (13)
Numéro minéralogique	X (8)
Date de début du contrat	9 (6)
Date de fin du contrat	9 (6)
Numéro du contrat	9 (5)
Date du début de révision	9 (6)
Date de fin de révision	9 (6)

Observations :

Sur le planning, un véhicule est lié en général à 3 contrats.

Document 6

Désignation : Carnet d'entretien

Classement : Modèle
Numéro minéralogique

Description des informations :

Numéro minéralogique	X (8)
Marque	X (10)
Modèle	X (13)
Date de début de révision	9 (6)
Date de fin de révision	9 (6)
Coût révision	9 (6)
Nature de la révision	9 (6)
Kilométrage de la révision	9 (5)

Observations :

* Actuellement, il y a au maximum 30 révisions pour un véhicule.

* Une révision est déclenchée par le kilométrage du véhicule (révision des 5000, 10000, 20000, ...)

2.9 TD Edition du MCD location de voiture sous AMC Designor

2.9.1 Objectif du TD :

On se propose d'éditer avec le progiciel AMC Designor **MCD**, le MCD de l'étude de cas Location de voitures.

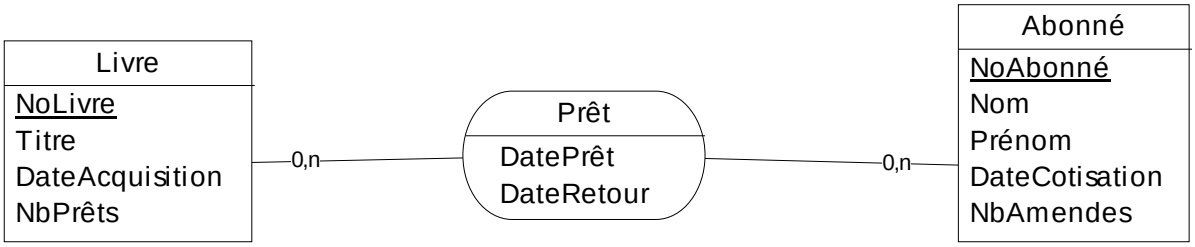
2.9.2 Utilisation d'AMC Designor pour éditer le MCD :

Sous Windows, sélectionnez l'application AMC Designor **MCD**. L'édition des entités, associations et liens d'héritage se réalise en utilisant la palette d'outils. Lorsque vous aurez terminé l'édition du MCD, vous pourrez lancer l'option **Vérifier Modèle** du menu **Dictionnaire**

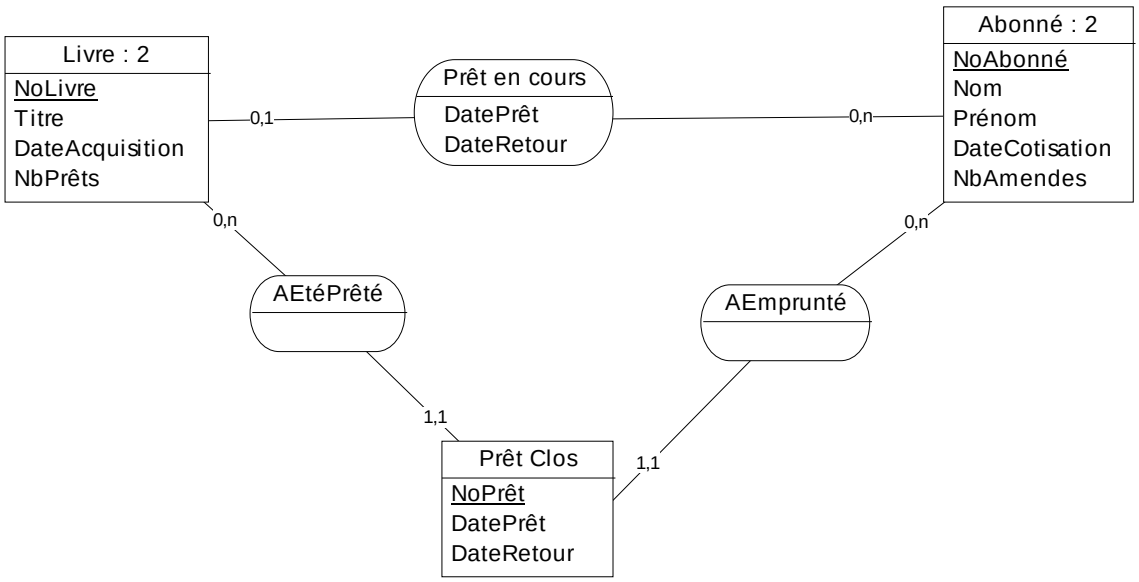
2.9.3 Edition papier du graphique associée au MCD :

Lorsque vous avez terminé l'édition du modèle, choisissez dans le menu **Fichier**, l'option **Imprimer Graphique.**

2.10 Représentation des historiques



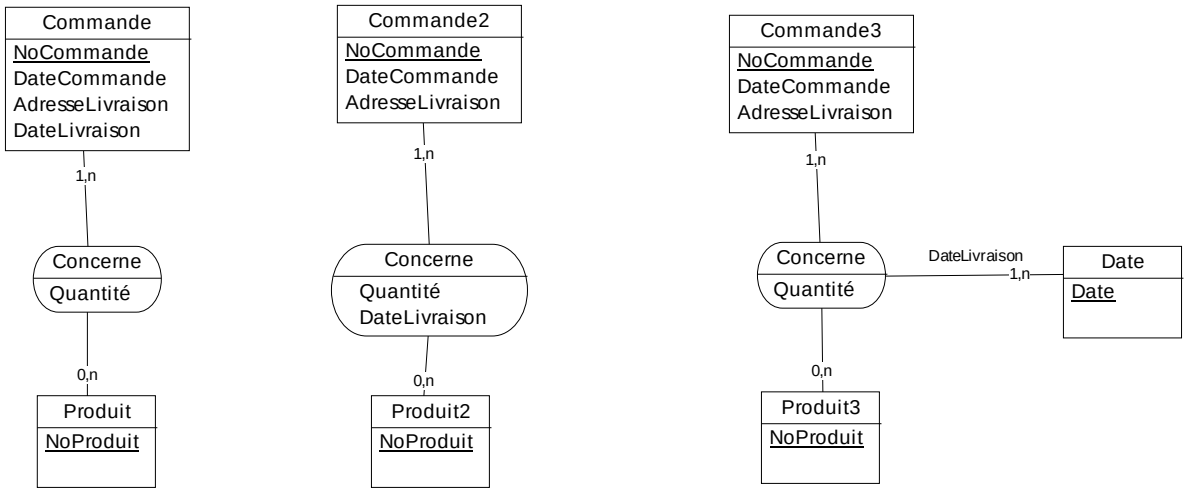
Solution 1 : Pour un abonné donné, s'il a emprunté plusieurs fois le même ouvrage, on n'a que les informations sur le dernier prêt.



Solution 2 : Représentation séparée de l'état actuel du prêt et de l'historique.

2.11 Exercice : Où placer les propriétés ?

Quelles différences y-a-t'il entre les trois modèles ci-dessous (propriété DateLivraison)?



3. Le modèle relationnel

3.1 Conception du modèle logique

3.1.1 Concepts de base

Un domaine D est un ensemble de valeurs

Exemples :

- booléen = {0, 1}
- couleur = {rouge, vert,bleu}

Le produit cartésien d'un ensemble de domaines D_1, D_2, ..., D_n est l'ensemble des n-uplets (ou tuples) $<v_1, v_2, ..., v_n>$ tels que v_i appartient à D_i;

Exemple : le produit cartésien de
$D_1 = \{0, 1\}$ et D_2 = {rouge, vert, bleu} est

rouge	0
rouge	1
vert	0
vert	1
bleu	0
bleu	1

Une relation est un sous-ensemble du produit cartésien d'une liste de domaines.

Exemple : à partir de D1 et D2, on construit la relation R

A1	A2
rouge	0
vert	1
bleu	0
bleu	1

Les colonnes sont les attributs de la table.

L'ensemble des tuples est une extension possible de R.

Le schéma de la table est composé du nom de la relation et de la liste des couples (attribut, domaine).

Exemple : R(A1 : couleur, A2 : booléen)

Une base de données relationnelle est un ensemble de schémas relationnels.

3.1.2 La démarche de conception

3.1.2.1 Problèmes soulevés par une mauvaise perception du réel

Soit la relation Propriétaire dont voici une extension :

NV	MARQUE	TYPE	PUIS	COUL	NSS	NOM	PRENOM	DATE	PRIX
1	RENAULT	R20	9	ROUGE	100	MARTIN	JACQUES	85	50000
2	PEUGEOT	604	9	VERTE	100	MARTIN	JACQUES	90	70000
3	CITROEN	2CV	2	BLEUE	200	DUPOND	PIERRE	86	25000
4	CITROEN	2CV	2	VERTE	200	DUPOND	PIERRE	90	25000
5	RENAULT	CLIO	6	VERTE	300	FANTAS	YVES	91	60000

Deux types d'anomalie :

- Données redondantes
- Risques d'incohérence

Il faut tolérer les valeurs nulles si on veut représenter des voitures sans propriétaire ou des personnes sans véhicule

3.1.2.2 Approche par décomposition

Principe : On part d'une relation (dite universelle) qui regroupe tous les attributs du modèle et on la décompose en sous-relations affranchies des anomalies mentionnées.

On définit deux opérations sur les relations

1) **la projection** : on ne garde que certains attributs en supprimant les tuples en double

Exemple : Propriétaire[Nom, Prénom]

Nom	Prénom
Martin	Jacques
~~Martin~~	~~Jacques~~
Dupont	Pierre
Fantas	Yves

2) **la jointure** (opération inverse) :

soient R(A_1 , A_2 , ..., A_n) et S(B_1 , B_2 , ..., B_p), leur jointure a pour attribut {A_1 , A_2 , ..., A_n } U {B_1 , B_2 , ..., B_p } et pour tuples la concaténation des tuples de R et de S ayant même valeur pour les attributs de même nom.

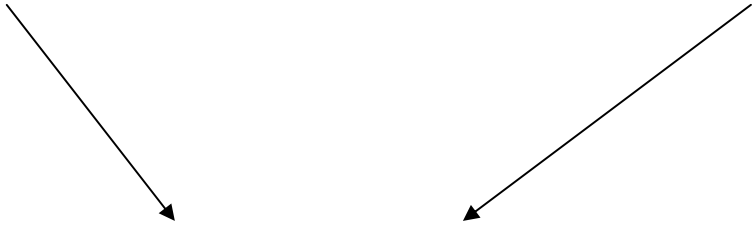
Exemple :

R

Marque	Couleur
Renault	Rouge
Peugeot	Rouge
Peugeot	Verte

S

Couleur	Puissance
Rouge	6
Verte	5
Rouge	4



R ⋈ S

Marque	Couleur	Puissance
Renault	Rouge	6
Peugeot	Rouge	6
Peugeot	Verte	5
Renault	Rouge	4
Peugeot	Rouge	4

On décompose R en la remplaçant par un ensemble de relations R_1 , R_2 , ..., R_n obtenus par des projections de R et tels que la jointure de R_1 , R_2 , ..., R_n ait même schéma que R.

Une décomposition est sans perte si pour toute extension de R, on a :

R = jointure de R_1 , R_2 , ..., R_n

Objectif : Décomposer la relation universelle sans perte pour déterminer et isoler les entités et associations du monde modélisé

3.1.3 Dépendances fonctionnelles

Soit R(A_1 , A_2 , ..., A_n) un schéma de relation, X et Y deux sous-ensembles de (A_1 , A_2 , ..., A_n) :

Définition 1 : Y dépend fonctionnellement de X ssi à toute valeur de X correspond une valeur unique de Y

Définition 2 : On dit que X → Y (X détermine Y) si pour toute extension r de R, pour tout tuple t1 et t2 de r, on a :

R[X](t1) = R[X](t2) => R[Y](t1) = R[Y](t2)

Exemples : dans la relation Voiture

NV → COULEUR

TYPE → MARQUE

TYPE → PUISSANCE

TYPE, MARQUE → PUISSANCE

3.1.4 Règles d'inférence sur les DF

Réflexivité : Y inclus dans X => X → Y

Augmentation : X → Y => XZ → YZ

Transitivité : X → Y et Y → Z => X → Z

On en déduit :

Union : X → Y et X → Z => X → YZ

Pseudo-transitivité : X → Y et WY → Z => XW → Z

Décomposition : X → Y et Z inclus dans Y => X → Z

Une dépendance fonctionnelle élémentaire est une DF de la forme X → A où A est un attribut unique non inclus dans X et où il n'existe pas de X' inclus dans X tel que X' → A

Exemples : dans la relation Voiture

NV, TYPE → MARQUE, PUISSANCE

3.1.5 Fermeture et couverture minimale

Définition 1 : La fermeture transitive F^+ d'un ensemble F de DF est constituée d'elle-même et des DF déduites par transitivité

Définition 2 : F est équivalente à F' ssi $F^+(F) = F^+(F')$

Définition 3 : La couverture minimale de F notée Min(F) est telle que :

1) Minimalité : Aucune DF de Min(F) n'est redondante, ie $F^+(Min(F) - f) <> F^+(Min(F))$

2) Exhaustivité : $(Min(F))^+ = F^+$

Exemple :

F = {NV → TYPE, TYPE → MARQUE, TYPE → PUISS, NV → COULEUR}

F^+ = {NV → MARQUE, NV → PUISS} U F

Min(F) = F

3.1.6 Clé et formes normales

3.1.6.1 Clé de la relation

Définition 1 : La clé primaire est un ensemble minimum d'attributs qui détermine tous les autres attributs de la relation.

Définition 2 : Soit X inclus dans (A_1 , A_2 , ..., A_n), X est clé de la relation R(A_1 , A_2 , ..., A_n) si :

1) X → A_1 , A_2 , ..., A_n

2) il n'existe pas de Y inclus dans X tel que Y → A_1, A_2, ..., A_n

Exemple : Dans la relation Voiture(NV, TYPE, MARQUE, PUISS, COULEUR) avec :

NV → COULEUR

TYPE → MARQUE

TYPE → PUISSANCE

NV → TYPE

(NV, TYPE) vérifie 1) et pas 2)

NV vérifie 1) et 2)

3.1.6.2 Processus de décomposition en formes normales

Première forme normale

Définition 1 : Chaque attribut de la relation doit être atomique

Exemple : Soit la relation Vol(n°-vol, Av-Nom, Capa) avec Capa(capa1, capa2)

N°-VOL	AV-NOM	CAPA
100	AIRBUS	(250, 275)
101	B 727	(150, 180)
102	B 747	(350, 425)

N°-VOL	AV-NOM	CAPA 1	CAPA 2
100	AIRBUS	250	275
101	B 727	150	180
102	B 747	350	425

Deuxième forme normale

Définition : Une relation est en 2 FN ssi :

1) Elle est en 1 FN

2) Tout attribut non clé ne dépend pas fonctionnellement d'une partie de la clé

Exemple : Fournisseur(nom, adresse, article, prix) avec :

nom, article → prix

nom → adresse

nom	adresse	article	prix
f1	adr1	p1	1000
f2	adr2	p1	800
f3	adr3	p1	500
f1	adr1	p2	10000
f1	adr1	p3	20000

Redondance sur attribut adresse

Décomposition en isolant la DF non conforme en 2 FN dans une relation indépendante et en supprimant de la relation initiale les attributs en partie droite de cette DF.

Fournisseur(nom, adresse)

Produit(nom, article, prix)

Troisième forme normale

Définition : une relation est en 3 FN ssi :

1) Elle est en 2 FN

2) Tout attribut non clé ne dépend pas d'un attribut non clé

Exemple : Voiture(NV, Marque, Type, Puiss, Couleur)

NV	Marque	Type	Puissance	Couleur
1	Renault	R21	7	Rouge
2	Renault	R20	7	Vert
3	Renault	R21	7	Jaune
4	Peugeot	505	7	Bleu
5	Renault	R21	7	Violet

Redondance sur Marque et Puissance :

car Type → Marque, Puissance

Décomposition en isolant la DF non conforme en 3 NF dans une relation indépendante et en supprimant de la relation initiale les attributs en partie droite de cette DF.

Voiture (NV, Type, Couleur)

Modèle (Type, Marque, Puissance)

Elimination des redondances dues aux DF transitives :

NV → TYPE → PUISSANCE

NV → TYPE → MARQUE

Toute relation a une décomposition en 3 FN telle que :

1) La décomposition préserve les DF

2) La décomposition est sans perte d'information

3.1.7 Algorithme de décomposition en 3 FN

Entrées de l'algorithme :

- L'ensemble des DFE (garantie la 2FN)
- L'ensemble des attributs

Les sorties :

L'ensemble des clés et des relations décomposées en 3 FN

Les étapes :

1) Construire Min(F) en se demandant si chaque DF peut être déduite de F

2) Partitionner Min(F) en F_1 , F_2 , ..., F_n tels que les DF de F_i aient même source. H est l'ensemble des sources différentes (ie les clés du schéma).

3) On construit n relations R_i (n = card (H)) telles que les attributs de R_i sont les attributs figurant dans F_i. La clé de R_i est la source de F_i . R_i est alors en 3 FN.

Exemple : La relation proprietaire

- Liste des attributs : NV, MARQUE, TYPE, PUISSANCE, COULEUR, NSS, NOM, PRENOM, DATE, PRIX

- Liste des DF :

1) NV → MARQUE

2) NV → COULEUR

3) NV → PUISSANCE

4) TYPE → MARQUE

5) TYPE → PUISSANCE

6) NV → TYPE

7) NSS → NOM

8) NSS → PRENOM

9) NSS, NV → DATE

10) NSS, NV, DATE → PRIX

F = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Etape 1 :

1 est-elle déductible de F - {1} ?

Itération 1 : I_1 = {COULEUR, PUISSANCE, TYPE}

Itération 2 : I_2 = {MARQUE} U I_1

=> 1 est déductible de F - {1}

2 est-elle déductible de F - {1, 2} ?

Itération 1 : I_1 = {PUISSANCE, TYPE}

Itération 2 : I_2 = {MARQUE} U I_1

=> 2 n'est pas déductible de F - {1, 2}

3 est-elle déductible de F - {1, 3} ?

Itération 1 : I_1 = {COULEUR, TYPE}

Itération 2 : I_2 = {MARQUE, PUISSANCE} U I_1

=> 3 est déductible de F - {1, 3}

Les autres DF ne sont pas déductibles

Remarque : Dans la DF 10, on peut réduire la partie gauche à {NSS, NV} car la DF 9 donne DATE.

=> 10') NSS, NV → PRIX

MIN(F) = {2, 4, 5, 6, 7, 8, 9, 10'}

Etape 2 :

Liste des DF de MIN(F) :

2) NV → COULEUR

4) TYPE → MARQUE

5) TYPE → PUISSANCE

6) NV → TYPE

7) NSS → NOM

8) NSS → PRENOM

9) NSS, NV → DATE

10') NSS, NV → PRIX

$F_1 = \{2, 6\}$

$F_2 = \{4, 5\}$

$F_3 = \{7, 8\}$

$F_4 = \{9, 10'\}$

Etape 3 :

F_1 : VOITURE(<u>NV</u>, COULEUR, TYPE)

F_2 : MODELE(<u>TYPE</u>, MARQUE, PUISSANCE)

F_3 : PERSONNE(<u>NSS</u>, NOM, PRENOM)

F_4 : PROPRIETAIRE(<u>NSS</u>, <u>NV</u>, DATE, PRIX)

3.2 Autres formes normales

3.2.1 Forme normale de Boyce-Codd (BCNF)

Définition : Une relation est en BCNF ssi :

1) Elle est en 3 FN

2) Tout attribut non clé ne doit pas déterminer une partie de la clé

Exemple : la vente de types de produits par des réseaux de distribution.

TYPEPROD, ANNEE $\rightarrow$ RESEAU

Règle 1 : Un type de produit, pour une année donnée est distribué par un seul réseau

RESEAU $\rightarrow$ TYPEPROD

Règle 2 : Un réseau ne peut distribuer qu'un seul type de produit.

Ce schéma est en 3FN mais pas en BCNF. Il peut engendrer des incohérences.

#TypeProd	#Année	Réseau
TP1	**1990**	**R15**
TP1	**1991**	**R15**
TP1	**1992**	**R41**
TP2	**1990**	**R2**
TP2	**1991**	**R87**
TP2	**1992**	**R26**

#Réseau	TypeProd
R15	**TP1**
R26	**TP2**
R33	**TP3**
R41	**TP1**
R57	**TP1**
R87	**TP2**

Avec cette modélisation en 3FN, on peut très bien ajouter dans la première table un tuple (TP1, 1993, R26). Cette information est incohérente avec le contenu de la deuxième table.

Une décomposition en BCNF donne :

#Réseau	#Année
R15	**1990**
R15	**1991**
R41	**1992**
R2	**1990**
R87	**1991**
R26	**1992**

#Réseau	TypeProd
R15	**TP1**
R26	**TP2**
R33	**TP3**
R41	**TP1**
R57	**TP1**
R87	**TP2**

3.2.2 Quatrième Forme normale

Basée sur la notion de dépendance multivaluée (notée DM) entre trois colonnes, notée P→→ Q / R

Exemple : la vente de types de produits par des réseaux de distribution.

#Réseau	#TypeProd	#Année
R15	TP1	**1998**
R15	TP1	**1999**
R41	TP1	**1997**
R41	TP1	**1998**
R41	TP2	**1997**
R41	TP2	**1998**

Il existe une DM #Réseau →→ #TypeProd, #Année si lorsqu'un réseau de distribution vend un ensemble de produits sur un ensemble d'années, il vend chaque produit chaque année.

Si l'on veut ajouter le tuple (R41, TP3, 1997), alors pour respecter la DM, il faut également ajouter le tuple (R41, TP3, 1998).

Définition : Une relation est en 4FN si elle est BCNF et :

- cas 1 : s'il n'existe pas de DM
- ou cas 2 : s'il existe une DM et une DF portant sur les mêmes colonnes.

Exemple pour le cas 1 :

#Réseau	**#Année**
R15	**1998**
R15	**1999**
R41	**1997**
R41	**1998**

#Réseau	#TypeProd
R15	**TP1**
R41	**TP1**
R41	**TP2**

Exemple pour le cas 2 : Il existe une DF

#Réseau, #Année, #TypeProd → MontantCom

#Réseau	**#TypeProd**	**#Année**	**MontantCom**
R15	TP1	**1998**	100000
R15	TP1	**1999**	125000
R41	TP1	**1997**	80000
R41	TP1	**1998**	82000
R41	TP2	**1997**	10000
R41	TP2	**1998**	25000

Lorsqu'il y a une DF de ce type en plus de la DM, on ne décompose pas la relation car on ne pourrait plus représenter la DF.

Remarque : il existe une 5ème forme normale basée sur les dépendances de jointure (cf Morejon 1995 et Ullman 1988 pour une définition de cette forme).

3.3 Exercice

Soit la table **Contamination** dans laquelle on mémorise les charges microbiennes en micro-organismes pathogènes de produits alimentaires:

La clé de cette table est (PRODUIT, MICRO-ORGANISME).

FILIERE	PRODUIT GENERIQUE	#PRODUIT	PAYS ORIGINE	#MICRO-ORGANISME	NIVEAU CONTAMINATION MOY	UNITE MESURE NIVEAU
Produits végétaux	Carottes	Carottes tranchées	Italie	Flore aérobie mésophile	5,8. 10^5	CFU/Gr
Produits végétaux	Végétaux crus	Céleri	USA	Flore aérobie mésophile	4,5	log CFU/g
Produits végétaux	Végétaux crus	Chou	USA	Flore aérobie mésophile	6,1	log CFU/g
Produits végétaux	Végétaux crus	Chou rouge	USA	Flore aérobie mésophile	5,6	log CFU/g
Produits végétaux	Végétaux crus	Choux fleur	USA	Flore aérobie mésophile	4,8	log CFU/g
Produits laitiers	Fromage	Colby	USA	Escherichia coli	0	log CFU/g
Produits végétaux	Végétaux crus	Courge	USA	Flore aérobie mésophile	7,1	log CFU/g

Proposez une organisation de la base de données qui vérifie les trois premières règles de normalisation.

3.4 Traduction d'un MCD Merise en Modèle Logique Relationnel

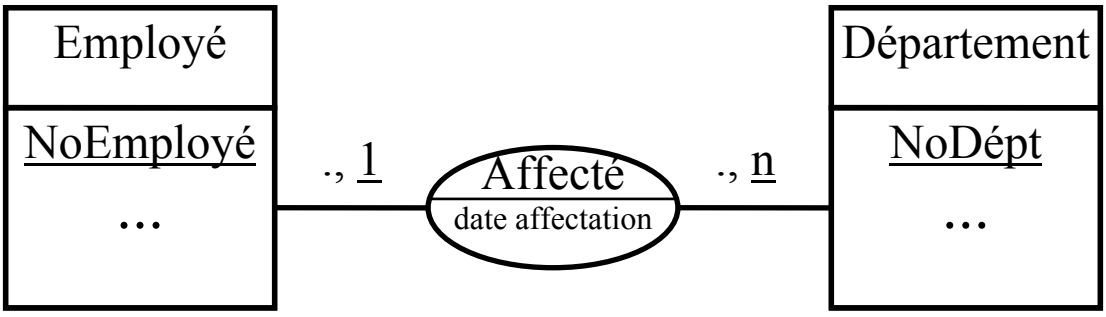
3.4.1 Les règles de traduction

R1 : Une propriété du MCD devient une colonne

R2 : Une entité devient une table

R3 : Un identifiant devient une clé primaire

R4 : Association inter-entités de type N:1 (ou 1:N)



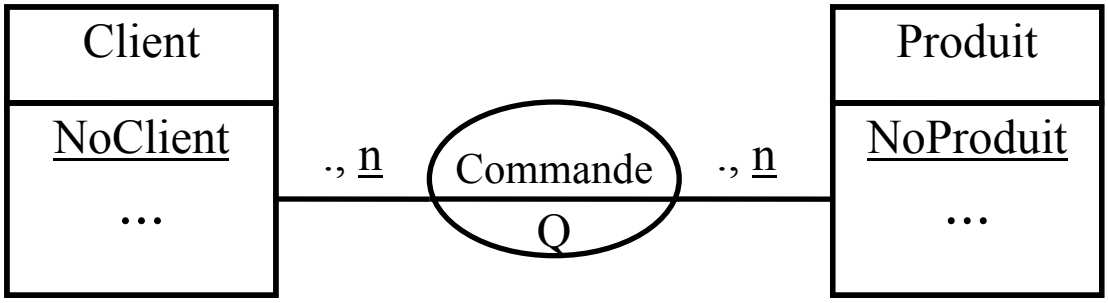
Association **Affecté** non traduite

NoDept devient clé étrangère dans Employé

Employé(NoEmployé, ..., NoDept, DateAffectation...)

Département(NoDept, ...)

R5 : Association inter-entités de type N : M

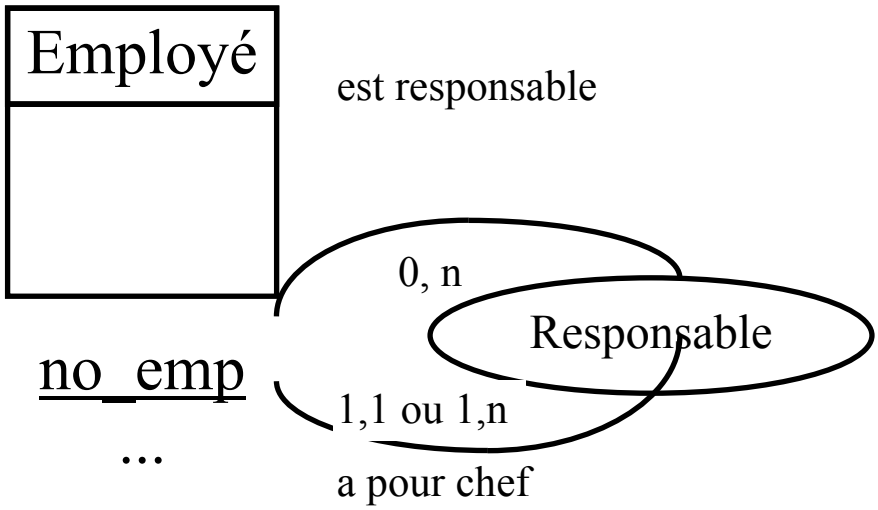


L'association Commande devient une table ayant pour clé la concaténation des identifiants de Client et Produit :

Client(NoClient, ...)
Produit(NoProduit, ...)
Commande(NoClient, NoProduit, Q)

Remarque : si on a n pattes, on a n clés concaténées dans la table traduisant l'association

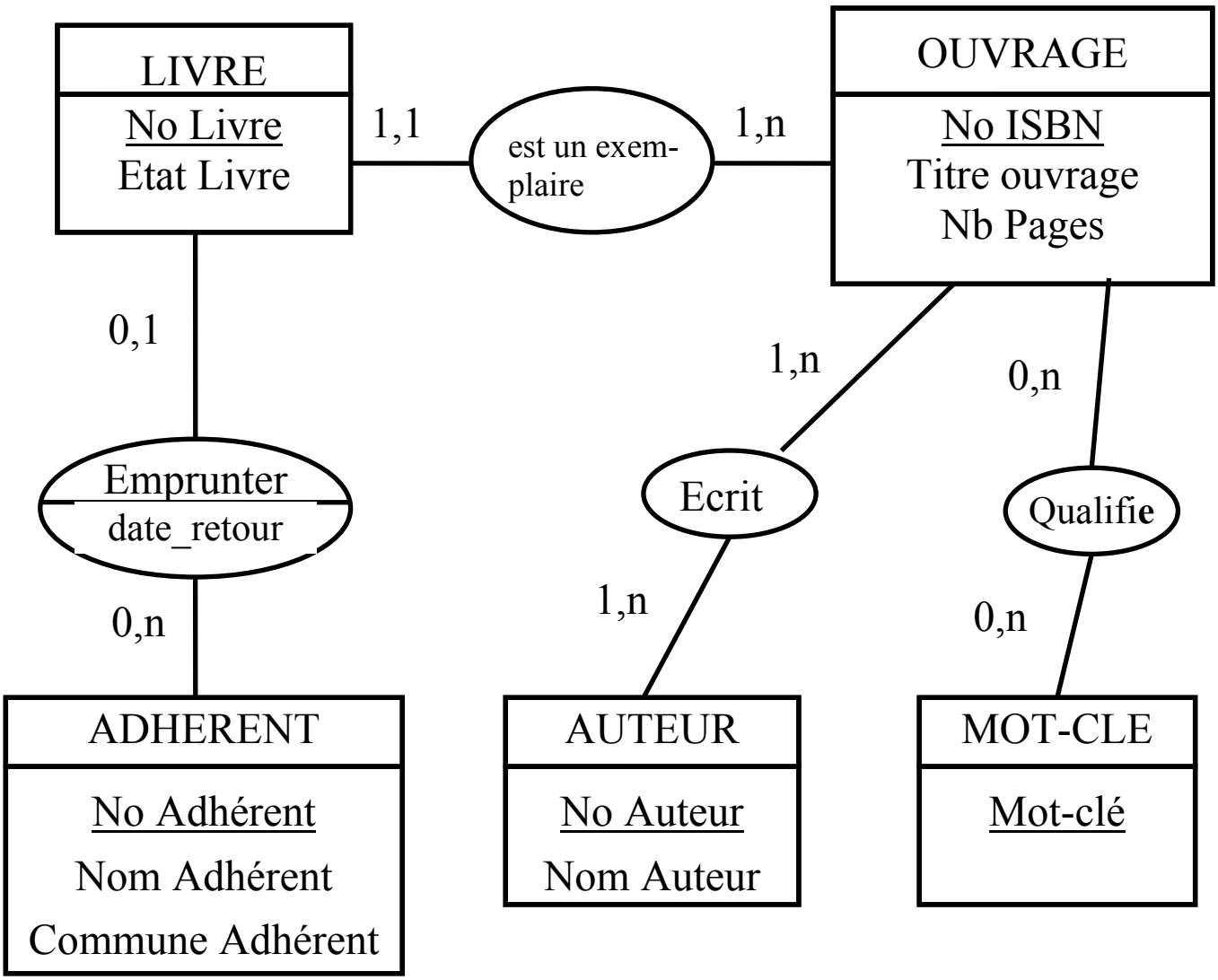
R6 : Lien inter-entité récursif (N : 1 ou N : M)



si cardinalité 1,1 (lien N : 1) : Employé(no-emp, no-resp)

si cardinalité 1,N (lien N : M) : Employé(no-emp, ...)
Responsable(no-emp, no-resp)

3.4.2 Exemple : traduction du MCD de gestion d'une bibliothèque



Par R1 , R2 , R3 :

Livre(no-livre, état-livre)

Ouvrage(no-isbn, titre-ouvrage, nb-pages)

Motclé (motclé)

Adhérent(no-adh, nom-adh, commune-adh)

Auteur (n° auteur, nom-auteur)

Par R4 :

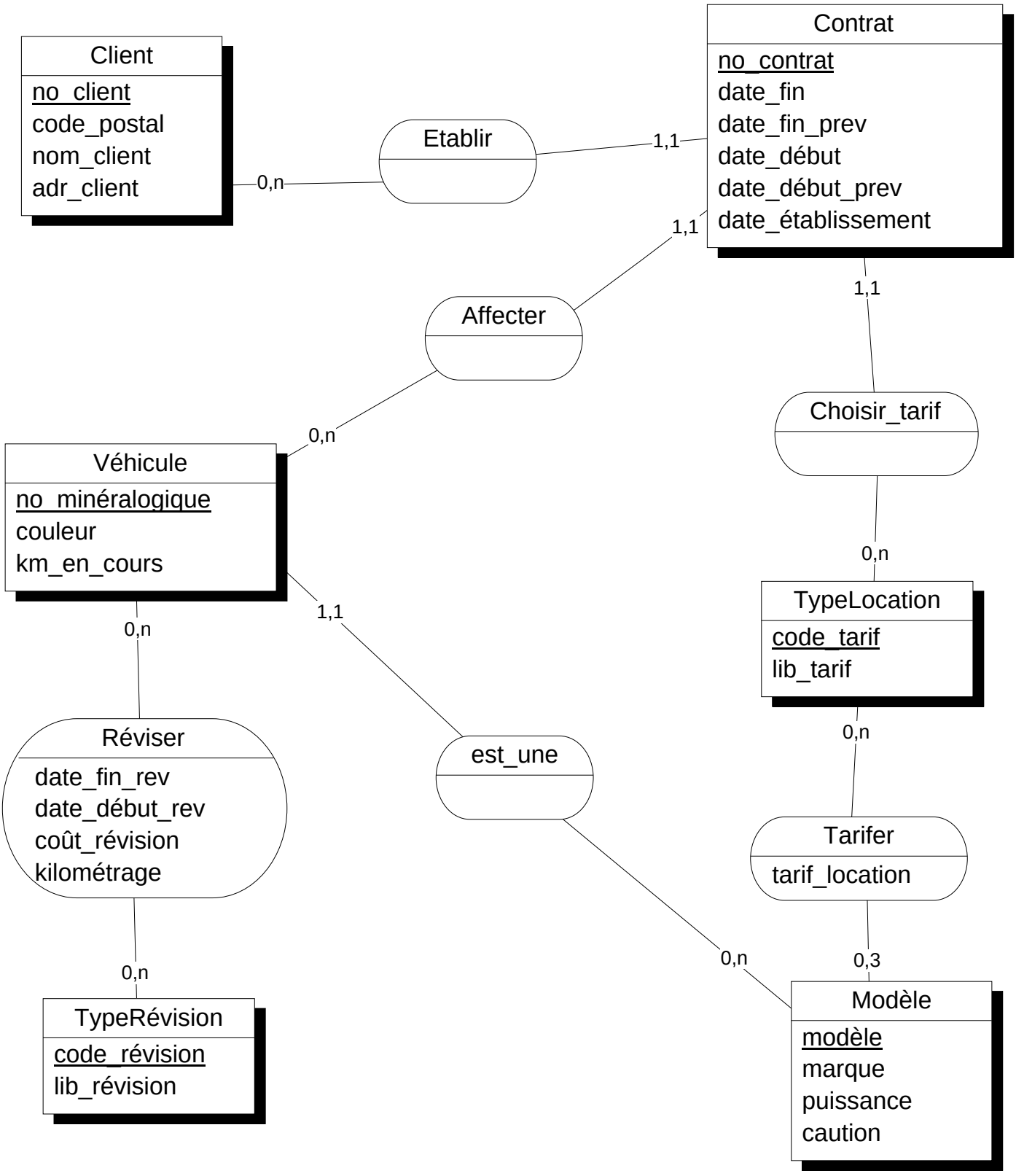
Livre(no-livre, état-livre) ⇨
Livre(no-livre, état-livre, no-isbn, no-adh, date-retour)

Par R5 :

Ecrit(no-isbn, no-auteur)

Qualifie(no-isbn, mot-clé)

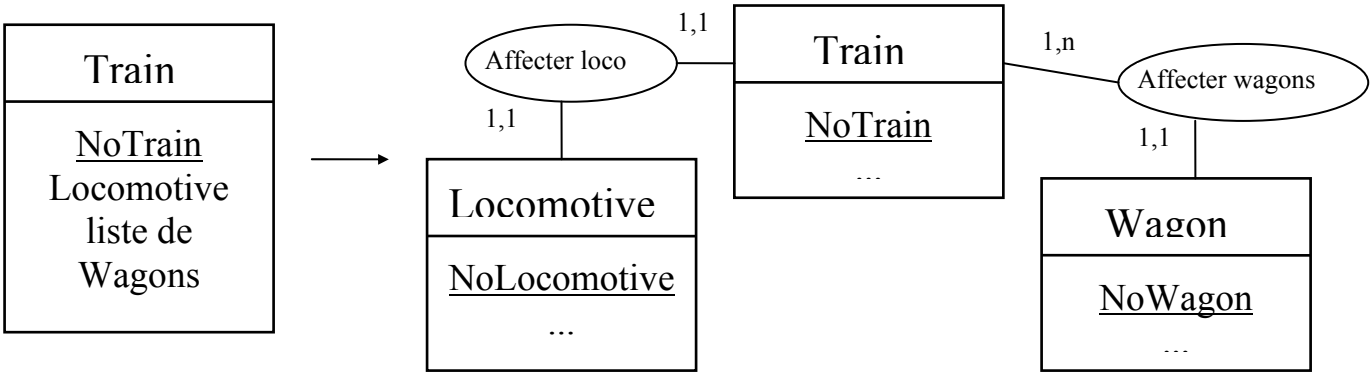
3.4.3 Traduisez le MCD du cas location de voiture en ML relationnel.



3.5 Expression des règles de normalisation dans la terminologie d'un MCD

Règle 1 : Toute entité doit posséder un identifiant. Cela permet d'identifier de manière unique toute occurrence d'entité (lève les ambiguïtés).

Règle 2 : Toutes les propriétés d'une entité ou d'une association doivent être de type élémentaires, i.e. non décomposables (1ère forme normale).



Règle 3 : Chaque occurrence d'entité ou d'association ne peut prendre qu'une seule valeur, i.e. on ne peut pas avoir de listes de valeurs pour une même propriété. Cette règle est à rapprocher de la troisième règle de normalisation du modèle relationnel (élimination des groupes répétitifs).

Règle 4 : Toutes les propriétés autres que l'identifiant doivent dépendre pleinement de l'identifiant et non d'une partie de celui-ci. Cette règle est à rapprocher de la deuxième règle de normalisation du modèle relationnel (attributs dépendant d'une partie de la clé).

Règle 5 : Chaque propriété doit dépendre directement de l'identifiant et non par l'intermédiaire d'une ou plusieurs propriétés. Cette règle est à rapprocher de la troisième règle de normalisation du modèle relationnel (attributs dépendant d'autres attributs non clé).

3.6 TD Traduction automatique du MCD en modèle relationnel

3.6.1 Objectif du TD

On se propose de reprendre sous AMC Designor MCD, le MCD de l'étude de cas **Gestion de location de voitures** afin de générer automatiquement le modèle relationnel qui lui est associé. On utilisera le protocole ODBC (Object DataBase Connectivity) pour créer automatiquement la structure des tables de la base de données relationnelle.

3.6.2 Utilisation d'AMC Designor pour générer le MPD à partir du MCD

Lancez Windows et sélectionnez l'application AMC Designor **MCD**. Vous partirez du MCD que vous avez validé dans le TD précédent. Vous pouvez lancer l'option **Vérifier Modèle** du menu **Dictionnaire**. Traitez les éventuels messages d'anomalie. Puis vous passerez à la génération du MPD (modèle physique de données, c'est à dire le modèle relationnel) par l'option **Générer Modèle Physique** du même menu. Sélectionnez comme base de données cible **Access 95**.

3.6.3 Vérification du MPD

Vérifiez le résultat de la génération automatique en le comparant à ce que vous avez obtenu en appliquant les règles de traduction présentées dans le cours. Les flèches entre les tables indiquent les références à des clés étrangères. Si le MPD vous parait satisfaisant, vous allez procéder à la création de la base Access.

3.6.4 Création de la base Access

Exécutez les étapes suivantes :

Créez une base Access vide.

Créez un DSN (DataBase Server Name) ODBC pointant sur cette base.

Générez depuis la fenêtre du Modèle Physique de Données par l'option **Générer Base de Données** du menu **SGBD**. (validez les options de génération avec l'enseignant).

3.7 Langage de manipulation des données

3.7.1 Introduction

Il se compose d'un ensemble de commandes permettant :

1)d'interroger la base de données

2)de la modifier (insertion, mise à jour, suppression)

Les langages relationnels (SQL, QBE, ...) sont assertionnels (ie permettent de définir les données que l'on souhaite sans dire comment y accéder).

Ils sont basés sur **l'algèbre relationnel** défini par CODD.

3.7.2 L'exemple support

La base Epicerie comprend :

- les attributs : f, nom, remise, ville, p, qte, nom-p, couleur.
- les relations :

fournisseur (f, nom, remise, ville)

mafourniture (f, p, qte)

produit (p, nom-p, couleur, origine)

Une réalisation possible de ces 3 relations :

Fournisseur

f	nom	remise	ville
f1	Bornidus	5	Paris
f2	Mercier	7	Paris
f3	Colbert	3	Reims
f4	Bossuet	6	Dijon
f5	Tanguy	10	Riec
f6	Dupont	0	Paris

Produit

p	nom-p	couleur	origine
p1	cassis	rouge	Dijon
p2	champagne	blanc	Reims
p3	huitre	vert	Riec
p4	moutarde	jaune	Dijon
p5	salade	vert	Nice
p6	cornichon	vert	Dijon
p7	muscadet	blanc	Nantes

Mafourniture

f	p	qté
f1	p1	1
f1	p4	1
f1	p6	2
f2	p2	1
f3	p2	5
f4	p6	3
f4	p5	7
f1	p5	8
f4	p4	2
f3	p4	1
f2	p4	1
f5	p3	10

3.7.3 Les opérateurs de l'algèbre relationnel

3.7.3.1 La projection

Déjà introduit dans le chapitre consacré à la normalisation.

Définition : On supprime les colonnes non retenues dans la projection et on élimine les doubles.

Exemple : mafourniture [f]

f
f1
f2
f3
f4
f5

3.7.3.2 La jointure naturelle

Déjà introduit dans le chapitre consacré à la normalisation.

Définition : On fait l'union des colonnes des deux relations et on concatène les tuples qui ont même valeur pour le(s) attribut(s) portant le même nom dans les deux relations.

Exemple : Produit * Mafourniture

p	**f**	**nom-p**	**qté**	couleur	origine
p1	f1	Cassis	1	rouge	Dijon
p2	f2	Champagne	1	blanc	Reims
p2	f3	Champagne	5	blanc	Reims
p3	f5	Huitre	10	vert	Riec
p4	f1	moutarde	1	jaune	Dijon
p4	f4	moutarde	2	jaune	Dijon
p4	f3	moutarde	1	jaune	Dijon
p4	f2	moutarde	1	jaune	Dijon
p5	f4	salade	7	vert	Nice
p5	f1	salade	8	vert	Nice
p6	f1	cornichon	2	vert	Dijon
p6	f4	cornichon	3	vert	Dijon

3.7.3.3 La θ-jointure

Définition : Deux attributs A et B sont dits θ-compatibles s'ils appartiennent à des domaines sur lesquels l'opérateur θ est défini. θ est choisi dans l'ensemble $\{<, =, >, \leq, \geq, \neq\}$. On forme le produit cartésien des deux relations S et R, on élimine tous les éléments ne satisfaisant pas à la relation θ.

Exemple : Fournisseur [ville = origine] produit

f	nom	remise	ville	p	nom-p	couleur	origine
f3	Colbert	3	Reims	p2	champagne	blanc	Reims
f4	Bossuet	6	Dijon	p1	cassis	rouge	Dijon
f4	Bossuet	6	Dijon	p4	moutarde	jaune	Dijon
f4	Bossuet	6	Dijon	p6	cornichon	vert	Dijon
f5	Tanguy	10	Riec	p3	huitre	vert	Riec

3.7.3.4 La sélection

Définition : on balaye la relation R en ne conservant que les lignes qui vérifient le prédicat P portant sur les attributs de R.

Exemple : fournisseur {remise > 5}

f	nom	remise	ville
f2	Mercier	7	Paris
f4	Bossuet	6	Dijon
f5	Tanguy	10	Riec

3.7.3.5 La division

Définition : soient R (X, A) et S (B, Y) avec A et B compatibles (ie ont même domaine), la division de R par S suivant A et B, notée R [A/B] S, est une relation X définie par :

<x> appartient à R [A/B] S **si** pour tout a appartenant à S [B], < x, a > appartient à R

Exemple : la relation P

p	**nom-p**
p4	moutarde
p5	salade
p6	cornichon

(mafourniture [f, p]) [p/p] P

Résultat

f
f1
f4

3.7.3.6 Les opérateurs ensemblistes

Définition : Soient R (x) et S (x), deux relations définies sur le même ensemble d'attributs :

- S (x) $\cup$ R (x) représente l'union des deux tables
- S (x) $\cap$ R (x) représente l'intersection
- S (x) - R (x) représente la différence ensembliste

Définition : Soient S (X, Z) et R (Z, Y) avec X $\cap$ Y vide et Z éventuellement vide, on note S x R, le produit cartésien de S et de R, défini par :

< x, z_1, z_2, y > appartient à S x R

si < x, z_1 > appartient à S **et** < z_2, y > appartient à R

3.7.4 Exemples sur la base épicerie

Q1 : Trouver le numéro des fournisseurs qui me fournissent au moins un article :

mafourniture [f]

Q2 : Trouver le numéro des fournisseurs qui ne me fournissent aucun article :

fournisseur [f] - mafourniture [f]

Q3 : Trouver le numéro des fournisseurs qui me fournissent p6 :

(mafourniture {p = "p6"}) [f]

Q4 : Trouver le numéro des fournisseurs qui me fournissent quelque chose d'autre que p6 :

(mafourniture {p ≠ "p6"}) [f]

Q4' : Trouver le numéro des fournisseurs qui me fournissent quelque chose, mais pas "p6" :

Q4 - Q3

Q5 : Trouver le numéro des fournisseurs qui ne fournissent que "p6" :

Q3 - Q4

Q6 : Trouver le numéro des fournisseurs qui me fournissent chacun au moins p4, p5 et p6 :

(mafourniture {p = "p4"}) [f]

∩ (mafourniture {p = "p5"}) [f]

∩ (mafourniture {p = "p6"}) [f]

Q7 :Trouver le nom et la ville des fournisseurs qui me fournissent tous les produits originaires de Dijon :

((mafourniture [p/p] (produit {origine = "Dijon"}))

* fournisseur) [nom, ville]

Q8 : Trouver les numéros des fournisseurs qui me fournissent au moins deux produits :

On renomme mafourniture en M1 et M2

(M1 x M2 {M_1.f = M_2.f **et** M_1.p ≠ M_2.p})[M_1.f]

3.7.5 Propriétés des opérateurs relationnels

Il existe un ensemble minimum d'opérateurs défini par :

- le produit cartésien
- la différence ensembliste
- l'union ensembliste
- la sélection
- la projection

Les autres en sont dérivées.

- l'intersection : $R \cap S = R - (R - S)$
- la θ-jointure : $S [A \theta B] R = S \times R \{S.A \theta R.B\}$
- la jointure naturelle : $S * R = S \times R \{S.Z = R.Z\} [X, S.Z, Y]$
- la division : $R [A/B] S = R [X] - (R[X] \times S[B] - R) [X]$

Exemple : la division

R

X	A
$\underline{x_1}$	a_1
$\underline{x_1}$	a_2
x_2	a_1
$\underline{x_3}$	a_1
$\underline{x_3}$	a_2
x_3	a_3
x_4	a_3

S

B
a_1
a_2

R [X] x S correspond aux tuples qui devraient être présents dans R pour garder tous les x_i

X	A
x_1	a_1
x_1	a_2
x_2	a_1
x_2	a_2
x_3	a_1
x_3	a_2
x_4	a_1
x_4	a_2

R [X] x S - R correspond à ce qu'il n'y a justement pas dans R (pour garder tous les x_i)

(R [X] x S - R) [X] : les x_i de R qui ne marchent pas

R [X] - (R [X] x S - R) [X] : la réponse attendue

L'ensemble minimum sert de référence pour tout langage d'interrogation.

Etant donné un langage I, on s'interroge sur sa complétude en le comparant à l'algèbre relationnel.

3.7.6 Modification de la base

Insertion

fournisseur : = fournisseur $\cup$ < f7, Durand, 5, Lyon >

Destruction

mafourniture : = mafourniture - < f_2, p_2, 1 >

Mise à jour

mafourniture : = (mafourniture - < f_2, p_2, 1 >) $\cup$ < f_2, p_2, 4 >

3.7.7 Exercice

Soit la base buveurs de bière :

bar (nbar, nom, adresse)

amateur (buveur, bière)

pilier (buveur, nbar)

sert (nbar, bière)

Exemple :

La table Amateur

buveur	bière
Dupont	Tuborg
Martin	Valstar
Meteyer	Tuborg
Meteyer	Kronenbourg
Meteyer	Valstar

Pilier

buveur	nbar
Dupont	1
Dupont	2
Dupont	3
Martin	2
Meteyer	2
Meteyer	3

Sert

nbar	bière
1	Tuborg
1	Kronenbourg
1	Valstar
2	Tuborg
3	Valstar
3	La Meuse

Bar

nbar	nom	adresse
1	Aux Amis	5, rue de Paris
2	Le Bilboquet	8, rue de Vern
3	Aux Amis	13, rue de Picardie

Exprimer les questions suivantes en algèbre relationnel :

Q1 : Trouver les noms de bars qui servent au moins une bière que Dupont aime.

Q2 : Trouver les buveurs qui fréquentent au moins un bar qui sert une bière qu'ils aiment.

Q3 : Trouver les buveurs qui ne fréquentent pas de bar qui sert une bière qu'ils aiment.

Q4 : Trouver les bars qui portent le même nom.

Q5 : Trouver les bars qui servent au moins toutes les bières que Dupont aime.

4. Le langage SQL

4.1 Présentation du langage

SQL est un standard :

- SQL 1 : normalisation ISO/CEI 9075 en 1989
- SQL 2 : normalisation ISO/CEI 9075 N5739 en 1991
- SQL 3 : normalisation ISO/CEI 9075 : 1999
- SQL 2003 : normalisation ISO/CEI 9075 : 2003.

SQL (Structured Query Language/Langage de Requête Structuré) est l'interface de communication avec les SGBD relationnels.

SQL est un langage non procédural. On exprime ce que l'on veut obtenir et non pas comment on veut l'obtenir.

SQL n'est pas un langage de programmation complet. il permet :

- de définir le schéma de la base de données (DDL),
- de charger les tables relationnelles (DML),
- de manipuler les données stockées dans les tables (DML),
- de gérer la base de données (DDL) : sécurité, organisation physique.

Verbes du DDL (Data Description Language)

create : création d'un objet

alter : modification d'un objet

drop : suppression d'un objet

grant : accorder une autorisation sur un objet

revoke : supprimer une autorisation sur un objet

Verbes du DML (Data Manipulation Language)

select : rechercher une donnée

insert : insérer une ligne (ou plusieurs) dans une table

update : modifier une ligne (ou plusieurs) dans une table

delete : supprimer une ligne (ou plusieurs) dans une table

4.2 Identificateurs

Longueur maximum : 30 caractères

Une lettre suivie de lettres, chiffres ou caractères $, # et _ (souligné) :

Exemples : NOM, NOM_DE_TABLE, CLI#ADRESSE

Identificateur qualifié : nom_de_schéma.nom_de_table.nom_de_colonne

Exemple : SPEINFO1.FILM.TITRE

Oracle ne fait pas la différence entre minuscules et majuscules.

4.3 Types de données élémentaires

4.3.1 Les données numériques

entiers longs sur 38 bits (INTEGER ou SMALLINT)

nombres décimaux avec virgule fixe sur 38 bits :

1. DECIMAL (11, 2) représente un nombre signé de 11 chiffres dont 2 après la virgule. Assure la compatibilité DB2.
2. Sous Oracle, équivalent à NUMBER(11,2)

nombres en virgule flottante :

1. REAL sur 63 bits,
2. FLOAT(b) avec 1 < b < 126). Assure la compatibilité avec le type ANSI float

4.3.2 Les chaînes de caractères

Chaîne de caractères de longueur fixe : CHAR(n) avec n compris entre 1 et 255.

☞ A n'utiliser que pour les colonnes contenant un seul car.

Chaîne de caractères de longueur variable : VARCHAR2(n) avec n compris entre 1 et 2000.

☞ Type à utiliser de préférence pour stocker les chaînes de caractères.

Chaîne de caractères de grande taille LONG (jusqu'à 2 giga car)

☞ A utiliser pour les textes longs, il n'occupe que l'espace nécessaire à la chaîne de grande taille.

✋ Ce type est soumis à plusieurs restrictions :

- on ne peut pas lui appliquer les fonctions de manipulation de chaînes,
- on ne peut pas interroger sur une colonne de type LONG.

4.3.3 Le type Date

Oracle alloue 7 caractères pour une colonne de type DATE.

Oracle stocke à la fois la date et l'heure dans ce type.

Le format par défaut est : jj-mmm/aa (DATE)

exemple : '01-JAN-94'

4.3.4 BLOBs

BLOB veut dire Binary Long Object

Avec le développement du multimédia, la plupart des SGBD propose un type de données qui permet de stocker des objets binaires de grande taille (documents, image, son, vidéo, ...). Actuellement deux stratégies s'opposent :

1. on stocke dans la BD la référence du fichier qui contient le BLOB.
2. on stocke l'objet directement dans une colonne de la BD.

La solution 1 fournit en principe un temps d'accès plus rapide au BLOB.

La solution 2 a l'avantage de la portabilité en cas de déplacement des fichiers et/ou de changement de plateforme.

Les types de données binaires (image, son) sont :

- RAW (jusqu'à 2000 bytes),
- LONG RAW (jusqu'à 2 giga bytes)

4.4 Création du schéma relationnel

C'est la description de l'ensemble des tables qui composent la base de données.

4.4.1 Création d'une table

On utilise l'ordre SQL CREATE TABLE :

```
CREATE TABLE LIVRE
   (AUTEUR     VARCHAR2(8),
   TITRE       VARCHAR2(100),
   ANNEE       SMALLINT,
   GENRE       VARCHAR2(8),
   PRIX        DECIMAL(7,2)) ;

CREATE TABLE ECRIVAIN
   (AUTEUR     VARCHAR2(8),
   NE-EN       SMALLINT,
   LIEU        VARCHAR2(20),
   SALLE       SMALLINT,
   RAYON       SMALLINT) ;
```

4.4.2 Modification de la structure d'une table

On utilise l'ordre SQL ALTER TABLE :

Différents types d'altération sont possibles :

- ajout d'une colonne (ADD COLUMN),
- modification de la définition d'une colonne (MODIFY COLUMN),
- suppression d'une colonne (DROP COLUMN),
- modification du nom de la table ou d'une colonne (RENAME TO, RENAME COLUMN).

Exemples

```
ALTER TABLE LIVRE RENAME TO LIVRE2;

ALTER TABLE LIVRE RENAME COLUMN TITRE TO TITRE2;

ALTER TABLE          LIVRE
ADD COLUMN           DATE_ACHAT DATE,
MODIFY               AUTEUR VARCHAR2(30) NOT NULL;
```

Limitations :

- on peut augmenter la précision d'une colonne numérique (par exemple float(60)->float(100)), on peut uniquement diminuer la précision lorsque la colonne ne contient que des NULL.

- On peut passer de CHAR à VARCHAR2 et réciproquement. On peut diminuer la précision tant que l'on ne dépasse pas la taille de la plus longue chaîne stockée en base.

4.4.3 Suppression d'une table

On utilise l'ordre SQL DROP TABLE :

Exemple

```
DROP TABLE    LIVRE [CASCADE CONSTRAINT] ;
```

Remarque : L'option *cascade constraint* permet de supprimer la table même si des contraintes d'intégrité référentielle porte sur des colonnes de cette table.

4.4.4 Définition de contraintes d'intégrité

Les **contraintes de colonne** portent sur un attribut particulier.

Les **contraintes de relation** peuvent porter sur un ensemble d'attributs.

4.4.4.1 Contraintes de colonne

L'attribut doit être obligatoirement renseigné :

```
TITRE      VARCHAR2(100)   NOT NULL
```

Donner une valeur par défaut à l'attribut:

```
GENRE      VARCHAR2(8)     DEFAULT 'BD'
```

La valeur de l'attribut doit être unique :

```
NO_SS      VARCHAR2(13)    UNIQUE
```

L'attribut est la clé primaire :

```
NO_LIVRE   SMALLINT        PRIMARY KEY
```

Remarque : les différences entre clé primaire et valeur unique sont :

1. Il peut y avoir plusieurs attributs à valeur unique mais un seul attribut (ou un seul groupe d'attributs) peut être déclaré comme clé primaire
2. Un attribut clé primaire doit forcément être renseigné (NOT NULL implicite) alors qu'un attribut à valeur unique ne l'est pas obligatoirement.

La valeur de l'attribut doit être présente dans la clé étrangère spécifiée, ie la colonne AUTEUR de la table ECRIVAIN :

```
AUTEUR    VARCHAR2(8)  REFERENCES ECRIVAIN(AUTEUR)
```

La valeur de l'attribut doit vérifier une condition (plage ou listes de valeurs possibles) :

```
AUTEUR VARCHAR2(8) CHECK (AUTEUR=UPPER(AUTEUR))
```

```
AUTEUR VARCHAR2(8) CHECK (AUTEUR IN ('Dupont', 'Durant'))
```

```
ANNEE SMALLINT CHECK(ANNEE BETWEEN 1990 AND 2000)
```

4.4.4.2 Contraintes de tables

Elles permettent d'exprimer des contraintes portant sur plusieurs colonnes de la table.

```
CREATE TABLE LIVRE
   (    AUTEUR          VARCHAR2(30),
        TITRE           VARCHAR2(100),
        ANNEE           SMALLINT,
        PRIXHT          DECIMAL(7,2)
        PRIXTTC         DECIMAL(7,2),
        GENRE           VARCHAR2(8),

        CONSTRAINT C_LIVRE_PRIX CHECK (PRIXHT <= PRIXTTC)
   )
```

4.4.4.3 Déclaration de la clé primaire et des contraintes d'intégrité référentielle

Pour pouvoir retrouver facilement les informations stockées dans le dictionnaire de données à propos des contraintes posées sur les colonnes, il est conseillé de baptiser chaque contrainte. On fabrique le nom de la contrainte à partir du nom de la table et de la colonne sur laquelle porte la contrainte.

```
CREATE TABLE LIVRE
   (NOLIVRE    SMALLINT
   CONSTRAINT PK_LIVRE PRIMARY KEY,
   AUTEUR VARCHAR2(8),
   TITRE  VARCHAR2(100),
   ANNEE  SMALLINT,
   GENRE  VARCHAR2(8),
   PRIX   DECIMAL(7,2)
      CONSTRAINT C_LIVRE_PRIX CHECK (PRIX<10000),
   CONSTRAINT FK_LIVRE_AUTEUR FOREIGN KEY (AUTEUR)
      REFERENCES ECRIVAIN(AUTEUR)
   )
```

Pour la contrainte d'intégrité référentielle vers la table parente (ici Ecrivain), il est possible de préciser l'option ON DELETE CASCADE. Si cette option est validée, lorsqu'une ligne est supprimée de la table parente, toutes les lignes qui la référencent dans la table fille sont détruites automatiquement pour maintenir l'intégrité référentielle.

4.4.4.4 Visualiser les contraintes

On dispose de 2 vues sur le dictionnaire de données :

- user_constraint
- user_cons_columns

Quelles sont les contraintes définies sur la table Livre (exemple ci-dessus) :

```
select constraint_name, constraint_type, search_condition
from user_constraints
where table_name='LIVRE' ;
```

→ Réponse : C_LIVRE_PRIX C PRIXHT <= PRIXTTC

constraint_type : prend comme valeurs possibles

C : contrainte de type Check,

P : clé primaire

U : clé unique

R : intégrité référentielle

V : contrainte de type check sur une vue

Sur quelle colonne porte la contrainte ?

```
select column_name

from user_cons_columns

where
   table_name='LIVRE'
   and
   constraint_name=' C_LIVRE_PRIX';
```

→ Réponse : PRIXHT et PRIXTTC

4.4.4.5 Activer/désactiver les contraintes

On utilise le verbe **alter table**

```
alter table livre enable constraint C_LIVRE_PRIX ;
alter table livre disable constraint C_LIVRE_PRIX ;
```

La colonne *status* de la table *user_constraints* indique si la contrainte est active ou pas (enabled, disabled).

4.4.4.6 Supprimer les contraintes

On utilise le verbe alter table

```
alter table livre drop constraint C_LIVRE_PRIX ;
alter table livre drop constraint FK_LIVRE_AUTEUR
alter table livre drop primary key ;
```

Remarque : Oracle refuse de supprimer une clé primaire si elle est référencée par une clé étrangère.

4.5 Les index

Définition : Un index est une structure de la base de données qui associe à chaque valeur distincte d'une ou plusieurs colonnes d'une table la ou les lignes contenant cette valeur dans la table.

Quand les utiliser ? : pour accélérer le temps d'exécution des jointures, donc notamment sur les colonnes pour lesquelles on a spécifiées des contraintes d'intégrité référentielle.

Exemple :

```
CREATE INDEX        I1-LIVRE
                    ON LIVRE
                    (AUTEUR) ;
```

crée un index sur la table LIVRE par ordre croissant d'auteur.

Attention *:* Il ne faut pas créer d'index sur les colonnes dans lesquelles on a une distribution de valeurs faible (le sexe par exemple).

Oracle génère automatiquement un index unique pour la clé primaire.

4.6 Enregistrement des données dans les tables

4.6.1 Insertion d'un tuple dans une table

On insère l'ensemble des valeurs associées à un tuple de la table.

```
INSERT INTO        LIVRE
VALUES ('HUGO', 'HERNANI', 1830, 'THEATRE', 120.00) ;
```

On insère un nouveau tuple en utilisant un ordre différent de l'ordre de définition.

```
INSERT INTO     LIVRE
(AUTEUR, TITRE, ANNEE, PRIX, GENRE)
VALUES ('HUGO', 'Les Misérables', 1862, 148.5, 'ROMAN') ;
```

On insère un nouveau tuple sans initialiser le prix.

```
INSERT INTO LIVRE
(AUTEUR, TITRE, ANNEE, GENRE)
VALUES ('BALZAC', 'Le Père Goriot', 1834, 'ROMAN') ;
```

Remarque : Il existe également des utilitaires de chargement non normalisés (exemple : sqlload d'Oracle).

4.6.2 Mise à jour de tuples dans une table

UPDATE modifie la valeur des données des colonnes spécifiées d'une ou plusieurs lignes

```
UPDATE      ECRIVAIN                    (Q1)
   SET      NE-EN = 1802,
            SALLE = 2,
            RAYON = 3
   WHERE    AUTEUR = 'HUGO' ;
```

Q1 met à jour la table ECRIVAIN en valorisant l'année de naissance de Hugo par 1802, la salle avec 2 et le rayon avec 3.

4.6.3 Suppression de tuples

Suppression directe

```
DELETE FROM      ECRIVAIN
WHERE            NE_EN = 1800 ;
```

supprime de la table ECRIVAIN les lignes correspondant à des auteurs nés en 1800

4.7 Les droits d'accès aux données

Le propriétaire d'une table ou d'une vue est celui qui l'a créée. Par défaut, lui seul peut l'utiliser. Il peut aussi en accorder l'accès sélectivement à d'autres utilisateurs ou à tout le monde (PUBLIC).

Droits d'accès en lecture, insertion, mise à jour :

```
GRANT SELECT, INSERT, UPDATE ON MATABLE TO SPEINFO1;
```

Droit de modification de la structure de la table, de suppression de tuples, de création d'index, de référencer une colonne de la table en tant que clé étrangère :

```
GRANT ALTER, DELETE, INDEX, REFERENCES ON MATABLE TO
SPEINFO1;
```

Remarque : un droit peut être passé avec le droit de le transmettre

```
GRANT SELECT, INSERT, UPDATE ON
  MATABLE TO SPEINFO1 WITH GRANT OPTION;
```

Remarque : On retire un droit par l'ordre REVOKE :

```
REVOKE SELECT, UPDATE ON MATABLE FROM SPEINFO1;

REVOKE ALL ON MATABLE FROM SPEINFO1;
```

Remarque : On ne peut pas retirer un droit que l'on n'a pas donné directement.

4.8 TD Création d'un schéma de tables

4.8.1 Mise en place de l'environnement pour utiliser Oracle

Le SGBD relationnel Oracle est disponible sur un serveur Unix. Connectez-vous sur le compte Unix qui vous a été attribué sur cette machine.

Pour être surs de pouvoir utiliser Oracle depuis votre compte Unix, vérifiez les variables d'environnement suivantes :

```
ORACLE_BASE=/usr/local/oracle
ORACLE_HOME=/usr/local/oracle/product/10.1.0/Db_1
ORACLE_SID=speinfo
NLS_LANG=AMERICAN_AMERICA.WE8DEC
```

Vérifiez également que la règle de recherche suivante se trouve dans votre path :

```
/usr/local/oracle/product/10.1.0/Db_1/bin
```

Nota : Vous pouvez lister l'environnement du processus Shell par la commande :

```
env
```

4.8.2 Spécifications de la base Stage de danse

Créez une base relationnelle sous Oracle qui permette de gérer un stage de danse d'une semaine en résidence.

Dans la semaine, les participants peuvent s 'inscrire à trois cours de danse (salsa, tango argentin, lindy hop). Dans chaque cours, il y a deux niveaux (débutant, intermédiaire/avancé). Chaque atelier (une danse, un niveau) est assuré par un intervenant. Le schéma relationnel de cette base est le suivant (les colonnes clé sont précédées d'un dièse) :

#Id_Intervenant	Nom_Intervenant	Prénom_Intervenant
1		Rey
2		Gilles
3		Denis
4		Amandine

Table 1 Intervenant

#Id_ Participant	Nom_ Participant	Prénom_Participant	Age
1		Rogelio	35
2		Iskander	32
3		Susana	43
4		Ada	20

Table 2 Participant

#Id_Atelier	Id_Intervenant
Salsa1	4
Salsa2	1
Lindy1	5
Lindy2	3
Tango1	4
Tango2	2

Table 3 Atelier

#Id_Participant	#Id_Atelier
4	Salsa1
1	Salsa2
4	Lindy1
3	Lindy1
4	Tango1
2	Tango2

Table 4 Est_Inscrit_En

#Id_Atelier	#Jour	#Heure_Début	Heure_Fin
Salsa1	Lundi	10	12
Salsa1	Mercredi	14	16
Salsa2	Mardi	10	12

Table 5 Emploi_du_temps

Ecrivez dans un fichier stage.sql, sous un éditeur de texte, les requêtes SQL de création de ce schéma relationnel en tenant compte des contraintes suivantes :

- Le jour est un élément de la liste de valeurs (Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche)
- Heure_Début et Heure_Fin doivent prendre des valeurs entre 0 et 24. On suppose que l'on travaille sur des heures entières sans utiliser le type Date
- L'âge doit être inférieur à 100.

Pour appeler l'éditeur, tapez sous le shell la commande :

```
emacs stage.sql&
```

Exécutez les requêtes de création sous sqlplus. Insérez dans les tables créées les valeurs indiquées dans les tables ci-dessus. Pour exécuter les requêtes lancez l'interpréteur SQL d'Oracle en vous connectant au compte Oracle qui vous a été attribué (noté speinfoI ci-dessous). La commande à exécuter est la suivante :

```
sqlplus speinfoI (I variant de 1 à 8 selon le compte)
```

Afin de pouvoir analyser sous éditeur de texte la trace d'exécution du chargement du schéma, passez en mode trace sur fichier avec la commande :

```
SQL> spool cre_stage.lst (cre_stage.lst est le nom du
fichier trace)
```

Lancez la création de la base par la commande :

```
SQL> start stage ou @stage (si votre script se trouve
dans le fichier stage.sql).
```

Le fichier de trace sera lisible lorsque vous aurez tapé la commande :

```
SQL> spool off
```

Remarque : Vous pouvez vérifier que vos tables sont bien créées en utilisant une vue sur le dictionnaire de données :

```
SQL> select table_name from user_tables;
```

Vérifiez que leur structure est conforme à ce que vous avez spécifié. L'instruction suivante donne la structure de la table ATELIER :

```
SQL> desc ATELIER
```

Pour insérer les valeurs, vous avez le choix entre :

(i) utiliser des ordres SQL d'insertion que vous exécuterez sous l'interpréteur Oracle présenté ci-dessus.

(ii) attacher vos tables Oracle sous Access et saisir les valeurs dans les tables à partir de l'interface graphique d'Access. L'attachement des tables se fait sous Access à partir du menu Fichier, Données externes, fichier ODBC. Il faut avoir préalablement créer un DSN (Data Source Name) avec l'administrateur ODBC de Windows référençant la base de données Oracle tournant sur la machine Unix.

4.9 TD Du MCD au SQL, liaison avec la conception

4.9.1 Objectif du TD

On se propose de créer une base de données sous Oracle à partir d'un script SQL généré par l'outil de conception AMC Designor. Puis, on fige la version de la base de données créée sous forme de modèle archivé sous AMC. On effectue ensuite une mise à jour du schéma sous AMC. Cette mise à jour est ensuite répercutée sur la base Oracle. Enfin, on réalise une opération de reverse engineering pour reconstituer un MCD (Modèle Conceptuel de Données) à partir du schéma enregistré sous Oracle.

4.9.2 Utilisation d'AMC Designor pour générer le MPD à partir du MCD

Vous partirez du MCD fourni par l'enseignant. Recopiez le fichier **voiture.mcd** se trouvant sur le réseau sous votre répertoire de travail. Lancez Windows et sélectionnez l'application AMC Designor.

Vous pouvez lancer l'option **Vérifier Modèle** du menu **Dictionnaire** puis vous passerez à la génération du MPD (modèle relationnel) par l'option **Générer Modèle Physique** du même menu. Sélectionnez comme base : Oracle Version 7.x.

4.9.3 Vérification et modification du MPD

Vérifiez le résultat de la génération automatique. Les flèches entre les tables indiquent les références à des clés étrangères. Lancez la génération du script SQL pour Oracle V7 depuis la fenêtre du Modèle Physique de Données par l'option **Générer Base de Données** du menu **SGBD**.

Sous l'onglet Génération de la base de données, cochez les options suivantes :

- Suppression de tables
- Création de tables
- Index de clé étrangère
- Autres index

Sous l'onglet Options, cochez les options :

- Utiliser le Code
- Intégrité référentielle déclarative

Suffixez le nom du fichier texte qui contiendra le script SQL par sql (exemple : voiture.sql).

Analysez le contenu du fichier. Comment est-il structuré ?

4.9.4 Génération de la base de données sous Oracle

Transférez votre fichier script *voiture.sql* sur le serveur de base de données par scp depuis une session putty.

Afin de pouvoir analyser sous emacs la trace d'exécution du chargement du schéma, passez en mode trace sur fichier avec la commande :

```
SQL> spool cre_voiture.lst (cre_voiture.lst est le nom
du fichier trace)
```

Lancez la création de la base par la commande :

***SQL>* start voiture** ou **@voiture** (si votre script se trouve dans le fichier voiture.sql).

Le fichier de trace sera lisible lorsque vous aurez tapé la commande :

***SQL>* spool off**

Regardez le fichier de trace sous emacs pour vérifier qu'il n'y a pas d'erreur injustifiée dans la trace de chargement du schéma. Si vous en trouvez, corrigez le script SQL sous emacs et relancez le chargement du schéma jusqu'à ce qu'il n'y ait plus d'erreur au chargement.

4.9.5 Mise à jour du MCD

Lorsque vous aurez terminé le chargement du schéma sous Oracle, figez la version sous AMC Designor sous forme de modèle archivé. Pour cela, ouvrez le modèle physique de données (voiture.mpd) et sélectionnez l'option **Archiver modèle** du menu **SGBD**.

Sauvez le modèle archivé sous le nom **voiture.mpa**.

Ouvrez le MCD voiture et réalisez une modification. Ajouter une entité et une association. Générez le MPD comme vu précédemment. Puis vous allez générer le script SQL qui va permettre de modifier la base Oracle. Pour cela:

- Sélectionnez l'option **Modifier base de données** du menu **SGBD**.
- Vérifiez que le fichier voiture.mpa est bien sélectionné en tant que modèle archivé.
- Décochez l'option Modification des triggers.
- Cliquez sur le bouton **Générer script**.

Visualisez le fichier généré et constatez qu'il contient bien tous les ordres SQL de mise à jour du schéma. Transférez-le sur le serveur de base de données et modifiez le schéma sous Oracle.

4.9.6 Reconstituer un MCD à partir d'un schéma Oracle

Vous allez maintenant considérer qu'il n'y a pas de dossier de conception MCD pour la base Voiture gérée sous Oracle. Votre objectif est de constituer un dossier de conception MCD en récupérant l schéma relationnel stocké dans la base Oracle.

Sous AMC Designor, fermez tous les modèles ouverts. Dans le menu Fichier, Modèle physique, sélectionnez l'option **Reverse Engineering**.

- Sélectionnez **Oracle V7** (Nom de la base) et **A partir d'un pilote ODBC**.
- Sélectionnez le nom de la source de données (DSN) créée dans le TD. Si vous n'avez pas créé le DSN dans le TD, faites le maintenant. Tapez votre nom d'utilisateur Oracle et votre mot de passe Oracle.
- Dans la fenêtre **Options de reverse engineering par ODBC**, vous devez voir les tables de l'application Voiture. Sélectionnez-les toutes et cliquez sur OK.

Lorsque toutes les tables sont importées sous AMC, sélectionnez l'option **Générer Modèle Conceptuel** du menu **Dictionnaire**. Analysez le MCD reconstitué.

4.10 La recherche de données

Syntaxe générale d'une recherche en SQL :

```
select nom_table.nom_colonne*
from nom_table*
[where conditions_de_sélection_sur_lignes*]
[group by nom_colonne_de_regroupement*]
[having conditions_de_sélection_sur_groupe*]
[order by nom_colonne_tri*]
```

Notes : *= plusieurs occurrences possibles, [] = optionnel

4.10.1 Expression des projections

Une projection effectue l'extraction des colonnes spécifiées d'une relation, puis élimine les tuples en double.

L'ordre SQL Select n'élimine pas les doubles, à moins que ce ne soit explicitement demandé par le mot-clé DISTINCT.

- Sélection de l'ensemble des attributs de la table :

```
SELECT     *      FROM   LIVRE ;
```

- Sélection des titres des ouvrages de la table LIVRE:

```
SELECT     TITRE  FROM   LIVRE ;
```

- Sélection de la date de parution et du titre des ouvrages de la table LIVRE:

```
SELECT     ANNEE, TITRE  FROM  LIVRE ;
```

- Sélection des titres et du prix exprimé en centimes des ouvrages de la table LIVRE (calcul arithmétique sur les colonnes) :

```
SELECT TITRE, PRIX * 100  'centimes' AS VALEUR FROM
  LIVRE ;
```

- Sélection des noms d'auteur (avec élimination des doubles) des ouvrages de la table LIVRE:

```
SELECT     DISTINCT AUTEUR      FROM     LIVRE ;
```

4.10.2 Expression des sélections

Une sélection est une restriction suivie d'une projection. Une restriction est une combinaison booléenne (or, and, not) de conditions élémentaires portant sur les colonnes d'une table. Les prédicats de restriction permettent la comparaison d'une valeur portée par une colonne à une valeur constante. Ils peuvent s'exprimer de différentes manières :

- à l'aide des opérateurs =, <>, <, >, <=, >= (cf Q1, Q2)
- à l'aide du prédicat d'intervalle BETWEEN qui teste si la valeur de l'attribut est compris entre deux valeurs constantes. L'attribut doit porter une valeur numérique ou de type date (cf Q3).
- à l'aide du prédicat de comparaison de texte LIKE qui teste si un attribut de type chaîne contient une ou plusieurs sous-chaînes. Le caractère souligné remplace un caractère quelconque. Le caractère % remplace une séquence de caractères (cf Q4 et Q5).
- à l'aide du prédicat de test de nullité (attribut non renseigné) NULL (cf Q7).
- à l'aide du prédicat d'appartenance IN qui teste si la valeur de l'attribut appartient à une liste de valeurs (cf Q6).

Exemples

```
(Q1) SELECT  TITRE, GENRE   FROM LIVRE
      WHERE AUTEUR = 'DUMAS' ;
```

On recherche dans la table LIVRE le titre et le genre des ouvrages écrits par DUMAS

```
(Q2) SELECT * FROM LIVRE
      WHERE AUTEUR <> 'DUMAS' AND ANNEE > 1835 ;
```

On recherche les ouvrages non écrits par Dumas et postérieurs à 1835

```
(Q3) SELECT AUTEUR, LIEU FROM ECRIVAIN
      WHERE NE_EN BETWEEN 1802 AND 1850 ;
```

On recherche le nom des auteurs nés entre 1802 et 1850 (ainsi que leur lieu de naissance).

```
(Q4) SELECT * FROM ECRIVAIN WHERE AUTEUR LIKE 'B %'
```

On filtre les auteurs dont le nom commence par un B

```
(Q5) SELECT * FROM ECRIVAIN WHERE AUTEUR LIKE '_A%'
```

On filtre les auteurs dont le nom contient un A en deuxième position

```
(Q6) SELECT AUTEUR, TITRE, ANNEE
     FROM LIVRE
     WHERE ANNEE IN (1839, 1866, 1857) ;
```

On sélectionne les ouvrages sortis en 1839, 1866 ou 1857.

```
(Q7) SELECT AUTEUR, TITRE, ANNEE
     FROM LIVRE
     WHERE ANNEE NOT NULL;
```

On sélectionne les ouvrages pour lesquels on dispose de l'année de parution.

4.10.3 Expression des jointures

Une jointure sans qualification (sans clause WHERE) est le produit cartésien (cf Q1). Dans une jointure avec qualification, le produit cartésien est restreint par une combinaison de prédicats de comparaison (cf Q2 à Q5). Enfin, dans une jointure, il est possible de privilégier une table (cf Q6).

Exemple support :

```
CREATE TABLE LIVRE
   (AUTEUR      CHAR (8),
    TITRE       CHAR (24),
    ANNEE       SMALLINT,
    GENRE       CHAR (8),
    PRIX        DECIMAL (5,2),
   CONSTRAINT LIVRE_PK
   PRIMARY KEY(AUTEUR, TITRE)) ;

CREATE TABLE ECRIVAIN
   (AUTEUR      CHAR (8)
   CONSTRAINT ECRIVAIN_PK
   PRIMARY KEY,
    NE-EN       SMALLINT,
    LIEU        CHAR (20),
    RAYON       SMALLINT ) ;

CREATE TABLE RAYON
   (RAYON       SMALLINT
   CONSTRAINT RAYON_PK
```

```
PRIMARY KEY,
 SALLE        SMALLINT ) ;
```

Exemples

```
(Q1) SELECT A.AUTEUR, B.LIEU
     FROM LIVRE A, ECRIVAIN B ;
```

forme le produit cartésien des tables LIVRE et ECRIVAIN sur les attributs AUTEUR de LIVRE et LIEU de ECRIVAIN. On peut associer un nom abrégé (alias) aux noms de table pour alléger l'écriture des requêtes.

```
(Q2) SELECT A.AUTEUR, A.TITRE, B.RAYON
     FROM LIVRE A, ECRIVAIN B
     WHERE A.AUTEUR = B.AUTEUR ;
```

Equi-jointure : liste le titre, l'auteur et le rayon de rangement des ouvrages de la bibliothèque.

Soit la table OUVRAGE définie comme :

```
CREATE TABLE OUVRAGE
   (  AUTEUR        CHAR (8),
      TITRE         CHAR (24),
      NE-EN         SMALLINT,
      SALLE         SMALLINT,
      RAYON         SMALLINT ) ;
```

```
(Q3) SELECT X.AUTEUR, X.TITRE, Y.NE_EN,
              Z.SALLE, Z.RAYON
     FROM       LIVRE X, ECRIVAIN Y, RAYON Z
     WHERE      X.AUTEUR = Y.AUTEUR
     AND       Y.RAYON = Z.RAYON ;
```

reconstitue la table OUVRAGE à partir des tables LIVRE, ECRIVAIN et RAYON

```
(Q4) SELECT A.AUTEUR, A.TITRE, A.ANNEE,
           A.GENRE, A.PRIX, B.NE_EN, B.LIEU, B.RAYON
           FROM LIVRE A, ECRIVAIN B
           WHERE A.AUTEUR = B.AUTEUR ;
```

réalise une jointure naturelle

```
(Q5) SELECT A.AUTEUR, A.NE_EN, B.AUTEUR
          FROM ECRIVAIN A, ECRIVAIN B
          WHERE A.NE_EN = B.NE_EN
          AND A.AUTEUR > B.AUTEUR;
```

Une jointure d'une table sur elle-même : liste le nom, la date de naissance des écrivains nés la même année.

```
(Q6) SELECT A.RAYON, B.AUTEUR
     FROM       RAYON A, ECRIVAIN B
     WHERE      A.RAYON = B.RAYON  (+) ;
```

réalise une jointure dans laquelle on privilégie la table Rayon. Cette jointure affiche la liste de tous les rayons de la bibliothèque et les noms d'auteurs qui y sont rangés si les rayons ne sont pas vides.

4.10.4 Sous-questions

On peut imbriquer des sous-questions au niveau de la clause WHERE.

Le résultat d'une sous-question peut être soit une valeur simple, soit un ensemble de valeurs.

Le résultat d'une sous-question est exploité pour :

- tester l'appartenance d'une valeur à une liste de valeurs élaborée dans la sous-question,
- vérifier un prédicat de comparaison exprimé à l'aide des quantificateurs de la logique du premier ordre et d'opérateurs de comparaison (=, >, <, >=, <=).

4.10.5 Sous-question, liste de valeurs

```
SELECT        AUTEUR, LIEU  (Q1)
FROM          ECRIVAIN
WHERE         AUTEUR IN
              (SELECT       AUTEUR
               FROM         LIVRE);
```

Q1 liste le nom et le lieu de naissance des auteurs de la table AUTEUR figurant dans la table LIVRE. Q1 est équivalent à la requête Q2 utilisant une jointure :

```
SELECT        AUTEUR, LIEU  (Q2)
FROM          ECRIVAIN, LIVRE
WHERE         AUTEUR.AUTEUR = LIVRE.AUTEUR;
```

Il est possible d'utiliser des variables définies dans un bloc interne au niveau d'un bloc externe. On parle alors de variable de corrélation. Q3 liste le nom, le lieu de naissance et le genre des livres écrits par des auteurs de la table AUTEUR figurant dans la table LIVRE.

```
SELECT        AUTEUR, LIEU, A.GENRE(Q3)
FROM          ECRIVAIN
WHERE         AUTEUR IN
              (SELECT       AUTEUR
               FROM         LIVRE A);
```

4.10.6 Questions quantifiées

On peut comparer le (ou les) attribut(s) se trouvant dans la clause WHERE avec l'ensemble de valeurs résultat d'une sous-question à l'aide de quantificateurs. Le prédicat de comparaison est vrai :

- s'il est vérifié pour tous les éléments de l'ensemble avec le quantificateur ALL,
- s'il est vérifié pour au moins un élément de l'ensemble avec le quantificateur SOME (ou ANY, synonyme)

Exemples :

```
SELECT          TITRE                (Q1)
FROM            LIVRE
WHERE           ANNEE > ALL
                (SELECT              NE-EN
                 FROM                AUTEUR) ;
```

Q1 liste le titre des ouvrages de la table LIVRE écrits après la naissance de tous les auteurs de la table ECRIVAIN

```
SELECT      AUTEUR                (Q2)
FROM        ECRIVAIN
WHERE       AUTEUR = SOME
            (SELECT               AUTEUR
             FROM                 LIVRE);
```

Q2 liste les auteurs de la table ECRIVAIN ayant un livre dans la table LIVRE.

Remarque : = SOME est équivalent à IN

On peut tester si le résultat d'une sous-question est vide ou non. Pour cela, on utilise le quantificateur EXISTS. EXISTS <sous-question> est vrai si le résultat de la sous-question est non vide.

```
SELECT          AUTEUR, TITRE  (Q3)
FROM            LIVRE X
WHERE           EXISTS
                (SELECT        *
                 FROM          ECRIVAIN
                 WHERE         NE_EN = X.ANNEE);
```

Q3 liste le nom des auteurs et le titre des ouvrages de la table LIVRE écrits la même année que l'année de naissance d'un des auteurs de la table ECRIVAIN.

4.10.7 Expression des unions

SQL permet de représenter l'opération d'union.

Exemple :

```
SELECT      ANNEE, 'PARUTION ', AUTEUR (Q1)
FROM        LIVRE
WHERE       ANNEE > = 1825
            AND ANNEE < = 1850
UNION
SELECT      NE_EN, 'NAISSANCE', AUTEUR
FROM        ECRIVAIN
WHERE       NE_EN > = 1825
            AND NE-EN < = 1850
ORDER       BY 1;
```

Q1 liste par ordre chronologique les événements intervenus entre 1825 et 1850 figurant dans la table LIVRE (PARUTION) et dans la table ECRIVAIN.

Remarque : les tables résultantes des selects doivent avoir le même nombre de colonnes et deux colonnes de même rang doivent être de même nature (chaînes de caractères, données numériques ou temporelles).

4.10.8 Expression d'une recherche hiérarchique

On peut stocker des informations hiérarchisées dans une table. SQL permet de faire l'exploration d'un sous-arbre.

Exemple : la table Employé

```
create table Employe (
   id_employe int primary key,
   nom_employe varchar2(30),
   id_chef int ,
      constraint fk_employe foreign key (id_chef)
      references employe(id_employe))
```

La liste des employés sous la responsabilité de Dupont (pas forcément en tant que chef direct) :

```
select level, id_chef, id_employe, nom_employe
   from employe
   start with nom_employe = 'Dupont'
   connect by prior id_employe = id_chef;
```

level est une pseudo-colonne indiquant le niveau hiérarchique depuis la racine considérée.

4.10.9 Agrégats, partitionnement

Partitionnement d'une table en sous-ensembles en fonction des valeurs d'un ou de plusieurs attributs de partitionnement.

Intérêt : pouvoir dénombrer ou pouvoir effectuer un calcul (somme, moyenne, ...) sur le résultat d'une requête découpé en sous-ensembles de tuples.

Le partitionnement s'exprime avec l'option GROUP BY. Exemple :

```
SELECT          AUTEUR, GENRE, COUNT(TITRE)  (Q1)
FROM            LIVRE
GROUP BY        AUTEUR, GENRE;
```

Q1 liste en les regroupant par auteur et genre, les attributs auteur et genre de la table LIVRE.

Sur le résultat suivant :

NO PARTITION	AUTEUR	GENRE	COUNT
1	Balzac	Roman	1
2	Dumas	Roman	1
3	Dumas	Théâtre	1
4	Hugo	Poésie	2
5	Hugo	Roman	1

il y a autant de groupes (ici 5) que de couples (auteur, genre) distincts.

Il est possible d'effectuer des restrictions sur le résultat du partitionnement, i.e écarter des sous-ensembles. Pour cela, on utilise l'option HAVING.
Exemples :

```
SELECT          AUTEUR, GENRE              (Q1)
FROM            LIVRE
GROUP BY        AUTEUR, GENRE
HAVING          GENRE <> 'POESIE';
```

Q1 liste, en les regroupant par auteur et genre, les attributs auteur et genre en éliminant les groupes dont le genre est "poésie". Dans l'exemple ci-dessus, élimination du sous-ensemble 4.

```
SELECT          AUTEUR, 'GENRE :', GENRE    (Q2)
FROM            LIVRE
WHERE           ANNEE<>1834
GROUP BY        GENRE, AUTEUR
HAVING          GENRE <> 'Théâtre'
ORDER BY        GENRE ASC, AUTEUR DESC ;
```

Q2 liste l'auteur, le libellé "GENRE" et le genre des ouvrages non publiés en 1834, en les regroupant par genre et par auteur, en éliminant les pièces de théâtre et en triant le résultat par genre croissant et par auteur décroissant.

Remarque : L'option WHERE permet de sélectionner les tuples avant la formation des partitions. L'option HAVING est évaluée ensuite. Elle permet d'effectuer une restriction sur les partitions.

4.11 Fonctions prédéfinies

On peut appliquer des fonctions sur les colonnes résultats d'un select. On dispose de fonctions pour :

- manipuler des valeurs numériques,
- manipuler des chaînes de caractères,
- manipuler des dates.

4.11.1 Fonctions de calcul

Il est possible de réaliser un calcul sur un ensemble de valeurs, résultat d'une question avec ou sans partitionnement.

- utilisées dans un select sans GROUP BY, le résultat ne contient qu'une ligne de valeurs,
- utilisées dans un select avec GROUP BY, le résultat contient une ligne de valeurs par partition.

Les fonctions implantées sont :

- **AVG, STDDEV, VARIANCE** : calculent respectivement la moyenne, l'écart-type et la variance des valeurs d'une collection de nombres figurant dans une colonne

```
SELECT      AVG (PRIX)
FROM        LIVRE ;
```

donne le prix moyen des ouvrages de la table LIVRE

- **SUM** calcule la somme des valeurs d'une collection de nombres

```
SELECT      SUM (PRIX)
FROM        LIVRE
WHERE       GENRE = 'ROMAN' ;
```

donne la somme des prix des romans de la table LIVRE

- **MIN** et **MAX** permettent d'obtenir la valeur minimum (resp. maximum) d'une collection de valeurs

```
SELECT      MAX (PRIX), GENRE
FROM        LIVRE
GROUP BY    GENRE;
```

  donne le prix maximum des ouvrages de la table LIVRE pour chaque genre

- **COUNT** permet de dénombrer une collection de lignes

```
SELECT      COUNT (*)
FROM        AUTEUR
WHERE       LIEU = 'PARIS';
```

  compte le nombre d'auteurs de la table AUTEUR nés à Paris

```
SELECT      COUNT (NE_EN)
FROM        AUTEUR
WHERE       LIEU = 'PARIS';
```

  compte le nombre d'auteurs de la table AUTEUR nés à Paris dont on connaît la date de naissance (colonne not null).

- **ROUND, TRUNC, FLOOR, CEIL** pour arrondir et tronquer

```
ROUND(123.27, 1) → 123.3

ROUND(123.22, 1) → 123.2

ROUND(101.8) → 102

ROUND(123.27, -2) → 100

TRUNC(123.33) → 123

TRUNC(123.567, 2) → 123.56

TRUNC(123.567, -2) → 100

FLOOR(128.7) → 128 /* entier inf */

FLOOR(129.1) → 129

CEIL (128.7) → 129 /* entier sup */

CEIL(129.1) → 130
```

4.11.2 Fonctions sur les chaînes de caractères

- LENGTH renvoie le nombre de caractères d'une colonne

- SUBSTR extrait une sous-chaîne dans une chaîne de caractères :

```
SUBSTR(NOM, 1, 15)
```

extrait les 15 premiers caractères de la chaîne NOM.

```
SUBSTR(NOM, LENGTH(NOM)-2, 3)
```

extrait les 3 derniers caractères de la chaîne NOM.

- LIKE permet de rechercher un profil-type dans une chaîne

```
NOM LIKE '%DE%'
```

recherche tous les noms contenant la sous-chaîne 'DE'

- REPLACE permet de remplacer une chaîne par une autre chaîne dans une colonne

```
UPDATE ECRIVAIN
   SET NOM=REPLACE(NOM, 'DUPONT', 'DUPOND') ;
```

remplace dans la colonne NOM de la table ECRIVAIN les occurrences de la chaîne « Dupont » par la chaîne « Dupond ».

- LTRIM et RTRIM permettent de supprimer les blancs parasites en début et en fin de chaîne.

```
UPDATE ECRIVAIN
   SET NOM=LTRIM(RTRIM(NOM));

UPDATE ECRIVAIN
   SET NOM=LTRIM(NOM, '1234567890') ;
```

supprime toutes les occurrences de ces caractères en début de chaîne.

- LPAD et RPAD permettent de formatter une chaîne en justifiant à droite ou à gauche. La chaîne est complétée par des blancs, par une chaîne constante ou le contenu d'une colonne.

```
LPAD(NOM, 20)
```

justifie à droite sur 20 colonnes en complétant par des blancs

```
LPAD(NOM, LENGTH(NOM) + 5, 'NOM : ')
```

justifie à droite sur la longueur du nom + 5 caractères en complétant par la chaîne 'Nom : '

'Dupont' → 'Nom : Dupont'

```
LPAD(NOM, LENGTH(NOM)+LENGTH(PRENOM)+2, PRENOM || ', ')
```

justifie à droite sur la longueur des 2 colonnes + 2 caractères

'Dupont', 'Jules' → 'Jules, Dupont'

- LOWER, UPPER, INITCAP permettent respectivement de mettre une chaîne en minuscules, majuscules et le premier caractère de chaque mot en majuscule

- DECODE permet de définir le transcodage d'une valeur numérique en chaîne de caractères.

```
SELECT NO_JOUR,
    DECODE(NO_JOUR, 1, 'LUNDI', 2, 'MARDI', 3, 'MERCREDI',
   4, 'JEUDI', 5, 'VENDREDI', 6, 'SAMEDI', 7, 'DIMANCHE')
   FROM MA_TABLE ;
```

→ 6 SAMEDI

si MA_TABLE ne contient qu'une seule ligne avec 6 dans la colonne NO_JOUR.

- ASCII renvoie le code Ascii du premier caractère de la chaîne passée en paramètre.

4.11.3 Fonctions sur les dates

- SYSDATE retourne la date et l'heure courante. SYSDATE ne peut pas être utilisé dans une contrainte CHECK.

```
INSERT INTO COMMANDE VALUES ('COM01', SYSDATE) ;
```

- TO_CHAR permet de convertir une date en chaîne de caractères.

```
SELECT TO_CHAR(SYSDATE, 'HH24, MI') FROM DUAL ;
```

- TO_DATE permet de convertir une chaîne de caractères en date.

```
INSERT INTO PLANNING
   VALUES (TO_DATE('15 :30', 'HH24, MI')) ;
```

Elément	Signification
D	jour de la semaine (1-7)
DAY	nom du jour
DD	jour du mois (1-31)
HH	heure du jour (1-12)
HH24	heure du jour (1-24)
MI	minute (0-59)
MM	mois (1-12)
MONTH	nom du mois
SS	seconde (0-59)
YYYY	année sur 4 chiffres
YY	année (2 derniers chiffres)

Table 6 Quelques éléments de format de date

4.11.4 Autres fonctions de conversion entre types

- TO_CHAR permet de convertir des valeurs numériques en chaîne de caractères.

```
SELECT TO_CHAR(SALAIRE) FROM EMPLOYE ;

SELECT TO_CHAR(SALAIRE, '999999.99') FROM EMPLOYE ;
   55  →   55.00
   456 →  456.00

SELECT TO_CHAR(SURFACE, '9.9999EEEE') FROM POLYGONE ;
   123 → 1.2300E+02
   12345 → 1.2345E+04
```

- TO_NUMBER permet de convertir des chaînes de caractères en valeurs numériques. Par exemple, si ANNEE_DEB et ANNEE_FIN sont des chaînes :

```
SELECT
   SUM(1000* TO_NUMBER(ANNEE_FIN) - TO_NUMBER(ANNEE_DEB))
   FROM EMPLOYE_RECOMPENSE ;
```

4.12 Tri de la table résultat

Il est possible de trier les valeurs de l'ensemble résultat d'une question avec l'option ORDER BY

Exemples :

```
SELECT        AUTEUR, TITRE, ANNEE          (Q1)
FROM          LIVRE
ORDER BY      ANNEE ;
```

Q1 liste l'auteur, le titre et l'année triés par ordre chronologique.

```
SELECT        PRIX * 100, AUTEUR, TITRE     (Q2)
FROM          LIVRE
ORDER BY      1 DESC, TITRE ;
```

Q2 liste par ordre de prix (exprimé en centimes) décroissant et par titre croissant, le prix en centimes, l'auteur et le titre. Le premier attribut de tri est identifié par sa position relative.

4.13 Mise à jour des tables (compléments)

4.13.1 Insertion de données dans une table

Insertion avec sous-question

```
CREATE TABLE OUVRAGE
   ( AUTEUR   CHAR (8),
     TITRE    CHAR (24),
     NE-EN    SMALLINT,
     SALLE    SMALLINT,
     RAYON    SMALLINT ) ;

INSERT INTO   OUVRAGE  (AUTEUR, TITRE)
            SELECT     AUTEUR, TITRE
            FROM       LIVRE ;
```

On initialise les attributs AUTEUR et TITRE de la table OUVRAGE avec ceux de la table LIVRE.

4.13.2 Mise à jour de tuples

Sélection avec intention de mise à jour

```
SELECT SALLE, RAYON
FROM OUVRAGE
WHERE AUTEUR IN ('HUGO', 'MALOUF')
FOR UPDATE ;
...
UPDATE  OUVRAGE
SET     SALLE = 2,
        RAYON = 3
WHERE   AUTEUR IN ('HUGO', 'MALOUF') ;
COMMIT ;
```

Mise à jour avec sous-question

```
UPDATE      OUVRAGE                          (Q2)
SET         SALLE = 2,
            RAYON = 3

WHERE       AUTEUR IN
            (SELECT     AUTEUR
             FROM       ECRIVAIN
             WHERE      LIEU = 'PARIS');
```

met à jour la table OUVRAGE en affectant les valeurs 2 pour la salle et 3 pour le rayon pour les auteurs nés à Paris.

4.13.3 Suppression de tuples

Suppression avec sous-question :

```
DELETE FROM          OUVRAGE
WHERE                AUTEUR IN
                     (SELECT          AUTEUR
                      FROM            ECRIVAIN
                      WHERE           LIEU = 'PARIS');
```

supprime de la table OUVRAGE les lignes correspondant à des auteurs pour les auteurs nés à Paris.

4.14 Gestion des transactions

Une transaction est une unité logique de traitement des données vis à vis des accès multiples et des pannes éventuelles (logicielles ou matérielles).

Trois instructions permettent de contrôler l'exécution des transactions :

- **COMMIT** : confirme toutes les modifications effectuées depuis le dernier COMMIT. Tant qu'un COMMIT n'est pas fait, seul l'utilisateur qui a fait les modifications peut les voir, pas les autres. Un verrou en écriture est posé sur les tuples modifiés. Les autres sont bloqués en attente s'ils veulent aussi mettre à jour le tuple jusqu'au prochain COMMIT ou ROLLBACK. Remarque : un COMMIT automatique est effectué en fin de session SQLPLUS.

- **ROLLBACK** : invalide toutes les modifications effectuées depuis le dernier COMMIT. En cas de panne (logicielle ou matérielle), Oracle effectue un ROLLBACK automatique des transactions non validées (par un COMMIT)

- **SAVEPOINT** : permet de définir un point de validation intermédiaire pour ne défaire qu'une partie de la transaction.

Exemple :

```
savepoint val1
...
rollback to val1
```

Remarque : Tous les ordres SQL du DDL (create, drop, alter, ...) génèrent un COMMIT automatique.

4.15 Représentation des opérateurs de l'algèbre relationnel

- Projection
- Jointure naturelle
- θ-jointure
- Sélection
- Division : Sur l'exemple de la base aérienne, quels sont les pilotes qui conduisent tous les AIRBUS ?

```
AVION (#AV_N°, …)
PILOTE (#PLN°, …)
VOL (#VOLN°, AVN°, PLN°)

SELECT DISTINCT          PLN°
FROM           VOL
WHERE          AVN° IN
               (SELECT   AVN°
                FROM     AVION
                WHERE    AVNOM = 'AIRBUS')
GROUP BY       PLN°
HAVING         COUNT (DISTINCT AVN°) =
               (SELECT   COUNT(AV_N°)
                FROM     AVION
                WHERE    AV_NOM = 'AIRBUS');
```

- Union

- Intersection :

```
INSERT INTO LIVRE1
            SELECT *
            FROM LIVRE
            WHERE ANNEE < 1850 ;

INSERT INTO LIVRE2
            SELECT *
            FROM LIVRE
            WHERE ANNEE >= 1850 ;
```

On crée deux sous-tables LIVRE1 et LIVRE2 en affectant les tuples en fonction de l'année 1850.

```
SELECT     A.AUTEUR
FROM       LIVRE1 A
WHERE      EXISTS
           (SELECT     B.AUTEUR
            FROM       LIVRE2 B
            WHERE      A.AUTEUR = B.AUTEUR);
```

On liste les auteurs qui ont écrit un livre avant 1850 et un livre après 1850.

Remarque : Sous Oracle, on dispose de l'opérateur INTERSECT :

```
SELECT      AUTEUR
FROM        LIVRE
WHERE ANNEE < 1850

INTERSECT

SELECT      AUTEUR
FROM        LIVRE
WHERE ANNEE > 1850 ;
```

- Différence ensembliste :

```
SELECT      A.AUTEUR
FROM        LIVRE1 A
WHERE       NOT EXISTS
            (SELECT   B.AUTEUR
             FROM     LIVRE2 B
             WHERE    A.AUTEUR = B.AUTEUR);
```

On liste les auteurs qui ont écrit un livre avant 1850 et qui n'en ont pas écrit après.

Remarque : Sous Oracle, on dispose de l'opérateur MINUS (EXCEPT en norme SQL2)

```
SELECT     AUTEUR
FROM       LIVRE
WHERE ANNEE < 1850
MINUS
SELECT     AUTEUR
FROM       LIVRE
WHERE ANNEE > 1850 ;
```

4.16 Les vues

Les vues sont des tables virtuelles qui permettent de définir des filtres sur des tables réelles. Elles se manipulent comme des tables.

Elles permettent de limiter l'accès à certaines colonnes d'une table pour un groupe d'utilisateurs et le type d'accès :

```
CREATE VIEW INFOLIVRE AS
         SELECT          AUTEUR, TITRE
         FROM LIVRE;

GRANT SELECT ON INFOLIVRE TO PUBLIC;

SELECT * FROM INFOLIVRE;
```

Elles permettent de simplifier l'accès aux données :

```
CREATE VIEW LOCLIVRE AS
    SELECT        C.RAYON, SALLE, TITRE, A.AUTEUR
    FROM          LIVRE A, ECRIVAIN B, RAYON C
    WHERE         A.AUTEUR = B.AUTEUR AND
                  B.RAYON = C.RAYON;

GRANT SELECT ON LOCLIVRE TO PUBLIC;

SELECT *
FROM LOCLIVRE
WHERE TITRE = 'LES CHOUANS';
```

Elles contribuent à l'indépendance des données vis à vis des programmes. On peut se servir des vues pour éviter d'avoir à modifier les programmes quand on modifie la structure de la base (ajout, suppression de tables).

On veut retrouver le rayon pour un couple (livre, écrivain)

```
SELECT     RAYON
FROM       LOCLIVRE
WHERE      TITRE = 'SMALLTALKV' AND AUTEUR='CLAVEL';
```

On veut pouvoir indiquer plusieurs auteurs pour les ouvrages communs :

```
CREATE TABLE OUVRAGE_COMMUN (
TITRE VARCHAR2(24),
AUTEUR VARCHAR2(8),
CONSTRAINT PK_OUVRAGE_COMMUN PRIMARY KEY(TITRE,
AUTEUR));


INSERT INTO OUVRAGE_COMMUN(TITRE, AUTEUR)
SELECT TITRE, AUTEUR
FROM LIVRE;


ALTER TABLE LIVRE DROP COLUMN AUTEUR ;


DROP VIEW LOCLIVRE;
```

```
CREATE VIEW LOCLIVRE AS
 SELECT A.TITRE, B.AUTEUR, C.RAYON, C.SALLE
 FROM LIVRE A, OUVRAGE_COMMUN B, RAYON C, ECRIVAIN D
 WHERE A.TITRE = B.TITRE
 AND
 D.RAYON = C.RAYON
AND
B.AUTEUR = D.AUTEUR;


SELECT     RAYON
FROM       LOCLIVRE
WHERE      AUTEUR = 'VEILLON'
 AND TITRE='Smalltalk V') ;
```

4.17 Les synomymes

L'identificateur d'une table ou d'une vue est nom_propriétaire.nom_table.

Il est pratique de définir des synonymes qui permettent d'éviter de connaitre le propriétaire de la table :

```
CREATE PUBLIC SYNONYM LOCLIVRE
        FOR SPEINFO.LOCLIVRE;
```

4.18 Conclusion sur SQL

Langage hybride mi-ensembliste mi-prédicatif

La syntaxe du langage est lourde

SQL est la référence de nombreux systèmes commercialisés.

SQL2 est complet par rapport à l'algèbre relationnel par émulation des opérateurs

SQL3 : toutes les fonctionnalités ne sont pas implémentées.

- Fonctionnalités orientées objet : constructeurs de types abstraits de données permettant la définition d'objets complexes
- Réflexes : Règles déclenchées par des événements base de données

Les requêtes SQL peuvent être intégrées dans les langages de programmation classique (C, Pascal, Fortran, Cobol, L4G, C++, Java...)

4.19 TD Manipulation d'une base en SQL

4.19.1 Objectif du TD

On se propose de réaliser un cycle complet de manipulation d'une base de données relationnelle en langage SQL (création du schéma, chargement des données, interrogation, mise à jour). L'étude de cas support du TD est la base Cinéma.

La première partie du TD consiste à charger la définition des tables et à insérer des tuples exemples pour chacune de ces tables. Elle doit être réalisée sur le serveur de bases de données. Dans la deuxième partie du TD, il s'agit de réaliser une série d'interrogations. Dans la troisième partie, on effectuera une liste d'opérations de mise à jour. Les deux dernières parties peuvent être réalisées indifféremment sur le serveur de bases de données ou depuis Windows (avec une application cliente de la base de données).

Les fichiers nécessaires à la réalisation du TD se trouvent sur le serveur de bases de données:

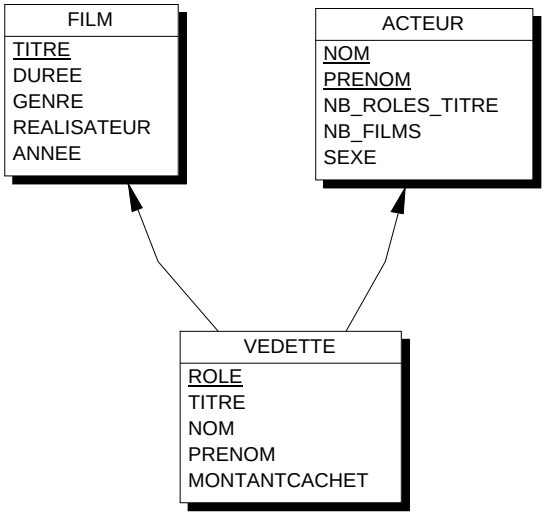
```
/home/test/td_oracle/td5
```

Recopiez ces fichiers dans vos répertoires sur le serveur de bases de données.

4.19.2 Chargement du schéma de la base Cinéma et insertion de tuples dans les tables

Le Modèle de Données de la base Cinéma est très simple. Il est composé de trois tables. Une occurrence de la table **FILM** donne des informations générales sur un film donné. Une occurrence de la table **ACTEUR** donne des informations élémentaires sur un acteur donné. Une occurrence de la

table **VEDETTE** représente une association de type lien N-M entre les deux entités précédentes. Le schéma de la base est donné ci-dessous.



Dans un premier temps, vous allez créer les trois tables sous votre compte Oracle en exécutant le fichier SQL **crebas.sql** :

```
@crebas ou start crebas.sql
```

Puis, vous allez insérer des tuples dans ces trois tables avec l'utilitaire **sqlload** d'Oracle.

Cet utilitaire requière deux fichiers en entrée. Un premier fichier (de suffixe dat) qui contient les données *à plat* au format ASCII . Dans ce fichier, les valeurs associées à un tuple se trouvent sur une même ligne.

Un deuxième fichier (de suffixe sod) contient l'ordre de chargement des tuples.

Vérifiez que vous avez bien recopié les six fichiers nécessaires au chargement des tables, à savoir :

acteur.dat et acteur.sod

film.dat et film.sod

vedette.dat et vedette.sod

Le lancement du chargement d'une table se fait sous l'interpréteur Shell pour le compte Oracle *speinfoXX* (*XX* correspondant au numéro de compte qui vous a été attribué) par la commande suivante qui se trouve dans le fichier lance_sqlload :

```
sqlload speinfoI/speinfoI, CONTROL=ma_table.sod,
LOG=ma_table.txt
```

Vous chargerez dans l'ordre : acteur, film puis vedette (puisque vedette requière l'existence d'acteurs et de films). Consultez les fichiers acteur.txt, film.txt et vedette.txt. Ils contiennent un rapport d'exécution du chargement des trois tables. Lorsque les trois tables sont chargées, passez sous l'interpréteur SQL.

4.19.3 Interrogation de la base Cinéma

Vous allez dans cette partie écrire des requêtes SQL que vous allez exécuter pour interroger la base. Dans cette partie, et la suivante, vous avez le choix de travailler (édition des requêtes et soumission à Oracle) sur le serveur de la base de données ou sur un client Windows.

Si vous travaillez sous Unix, utilisez deux sessions putty :

- La première pour éditer vos requêtes sous emacs.
- La deuxième pour exécuter les requêtes sous l'interpréteur SQL d'Oracle lancé depuis un interpréteur de commandes SHELL.

Si vous travaillez sous Windows, utilisez deux fenêtres Windows :

- La première pour éditer vos requêtes sous un éditeur de texte (NotePad par exemple).
- La deuxième pour exécuter les requêtes à partir d'un interpréteur SQL d'Oracle fonctionnant en mode Client/Serveur. Dans ce cas,

afin d'établir la connexion, vous allez devoir spécifier en plus du compte Oracle sur lequel vous travaillez, le nom de la base de données sur laquelle vous désirez travailler (paramètre chaîne hôte), à savoir : **speinfo**.

Le schéma de la base est (le caractère # préfixe les colonnes participant à la clé primaire):

```
Acteur(#nom, #prenom, nb_roles_titre, nb_films, sexe)

Film(#titre, duree, realisateur, genre, annee)

Vedette(#role, titre, nom, prenom)
```

Ecrivez les requêtes suivantes en SQL :

Q1 : Editer titre, année, réalisateur, genre, durée de chaque film.

Q2 : Editer nom, prénom, sexe et nombre de films de chaque acteur.

Q3 : Editer le contenu de la table vedette.

Q4 : Editer nom et prénom des actrices.

Q5 : Editer nom et prénom des vedettes du film POLICE.

Q6 : Editer nom et prénom des vedettes, accompagnés du nom des réalisateurs avec lesquels elles ont fait au moins un film.

Q7 : Editer le titre des films où joue au moins une vedette.

Q8 : Editer le titre des films où ne joue aucune vedette.

Q9 : Editer nom et prénom des acteurs qui ont participé à un nombre de films supérieur à la moyenne du nombre de films par acteur.

Q10 : Editer le nom du réalisateur qui a fait le film le plus long.

Q11 : Pour tout film où le réalisateur était en même temps vedette, éditer le titre du film, le nom du réalisateur et son prénom (les noms sont discriminants).

Q12 : Editer le nom des films qui ont à la fois BOURVIL et DE FUNES comme vedette.

Q13 : Editer le nom des vedettes qui ont joué uniquement dans les films du réalisateur VEBER.

Q14 : Editer le nom des réalisateurs qui ont fait au moins deux films.

Q15 : Editer le nom des réalisateurs qui ont fait au moins trois films.

Q16 : Editer le nom des vedettes qui ont joué dans tous les films du réalisateur OURY.

4.19.4 Mise à jour de la base

Utilisez les instructions **update**, **insert** et **delete** pour mettre à jour la base. Procédez de la même manière que pour les interrogations.

Q17 : Depardieu a tourné 53 films et non 50.

Q18 : Ajouter un film : Les fugitifs, 1986, Veber, Comédie, 89

Q19 : Dans ce film, ajouter deux rôles joués par des vedettes : le chômeur par P. Richard, l'ancien gangster par Depardieu

Q20 : Supprimer l'actrice R. Schneider et son (ou ses) rôle(s).

A la fin de chaque modification, listez la ou les tables concernée(s) pour contrôler les mises à jour.

4.20 TD Manipulation d'une base Postgresql

4.20.1 Objectif du TD

On se propose de réaliser un cycle complet de manipulation d'une base de données relationnelle avec le SGBD Postgresql (création du schéma, chargement des données, interrogation, mise à jour). L'étude de cas support du TD est la base Cinéma. L'objectif est de se rendre compte sur un cas concret du travail à réaliser pour porter une base SQL d'un logiciel à un autre.

4.20.2 Chargement du schéma de la base Cinéma et insertion de tuples dans les tables

Une fois connecté sous l'interpréteur de requêtes Postgresql, on cherchera à exécuter le script de création de la base Cinéma fourni pour une base Oracle dans le TD précédent. Ce script sera adapté pour tenir compte des différences syntaxiques entre les deux implémentations de SQL (celle d'Oracle et celle de Postgresql).

Les données de la base Oracle seront ensuite importées dans la base Postgresql en utilisant Access comme application cliente pour récupérer les données dans la base Oracle, puis pour les stocker dans la base Postgresql.

4.20.3 Interrogation de la base Cinéma

Vous allez dans cette partie reprendre les requêtes SQL que vous avez écrites pour la base Oracle. Vous les adapterez si nécessaire pour pouvoir les exécuter dans la base Postgresql.

5. Programmation PL/SQL

SQL n'est pas dans sa version actuelle un langage de programmation complet. On ne dispose pas en particulier des structures de contrôle classiques des langages de programmation.

PL/SQL est une extension de SQL. C'est un langage de programmation de type L3G adapté au contexte des bases de données.

On peut s'en servir pour :

- étendre l'ensemble des fonctions disponibles dans SQL,
- programmer des contrôles réalisés automatiquement au cours du cycle de vie d'une donnée (création, mise à jour, suppression) par l'intermédiaire de réflexes (triggers),
- développer des types abstraits (données + comportement) utilisables dans des applications externes à la base de données.
- optimiser une requête SQL.

5.1 Un premier programme

Un programme PL/SQL peut être une fonction, une procédure, un trigger, un package ou un bloc anonyme.

Un bloc PL/SQL est constitué de 3 sections

- Section de déclaration des variables
- Section des traitements
- Section de gestion des exceptions

Un premier exemple :

```
declare
   nb_nombres constant int := 10;
   nombre int := 100;
   i int := 1;

begin
   dbms_output.enable;
   /* création d'une suite de 10 éléments */
   for i in 1..nb_nombres loop
      insert into suite values (nombre);
      nombre := nombre + 10;
   end loop;
   dbms_output.put_line('La suite est créée');

end;

/* fin du bloc PL/SQL  */

/
```

5.2 Déclaration des variables

On manipule les mêmes types de données qu'en SQL, on dispose en plus du type booléen.

BOOLEAN peut prendre les valeurs TRUE, FALSE ou NULL.

%TYPE permet de déclarer une variable du même type qu'une colonne de table.

```
create table employe (
   no_emp int primary key,
   nom_emp varchar2(40)) ;
```

Dans le bloc PL/SQL :

```
declare
   nom_emp_courant employe.nom_emp%TYPE ;

begin
   select nom_emp
      into nom_emp_courant
   from employe
   where no_emp = 100 ;

end ;
```

%ROWTYPE permet de déclarer une variable composite ayant la même structure que la table spécifiée.

Dans le bloc PL/SQL :

```
declare
   emp_courant employe%ROWTYPE ;
   emp_nouveau employe%ROWTYPE ;
   ...

begin
   emp_courant.nom_emp := 'Dupont' ;
   ...
   select *
   into emp_courant
   from employe
   where no_emp = 100 ;

   emp_nouveau := emp_courant ;
   dbms_output.put_line('Nom : ' || emp_nouveau.nom_emp) ;

end ;

/
```

Remarque 1 : Il faut exécuter la commande *set serveroutput on* pour visualiser les impressions *dbms_output*.

Remarque 2 : || est l'opérateur de concaténation de chaînes de caractères.

5.3 Structures de contrôle

L'instruction IF :

```
if nb_employes < 10 then
   montant := 100 ;

elsif nb_employes < 100 then
   montant := 1000 ;

else /* supérieur à 1000 */
   montant := 10000 ;

end if ;
```

L'instruction LOOP

```
declare
   i int := 1 ;
   max_i int constant := 100 ;

begin

loop
   i := i + 1 ;
   exit when i > max_i ;

end loop ;

end ;

/
```

L'instruction WHILE :

```
while i < 100 loop
   i := i + 1 ;

end loop ;
```

L'instruction GOTO :

```
if taux > 0.98 then
   goto trait_particulier ;

endif ;

montant := taux*nb_commandes ;

...

<<trait_particulier>>

...
```

L'instruction NULL :

```
exception
   when VALUE_ERROR then
      dbms_output.put_line('Débordement de capacité dans
      une affectation') ;
   when OTHERS then
      NULL ;
```

5.4 Procédures et fonctions

Le passage de paramètres est proche de celui de Pascal (paramètres sortie et entrée/sortie sont différenciés) :

```
declare
   arg1 number ;
   arg2 number ;
   arg3 number ;

   procedure test_arg(p_arg1 in number,
    p_arg2 out number, p_arg3 in out number) is
      begin /* proc */
      p_arg2 := p_arg1 ;
      p_arg3 := p_arg3 + 1 ;
      end ; /* proc */
   -- Debut du bloc PL/SQL
   begin
   dbms_output.enable;
   arg1 := 2.23 ; arg3 := 4.1 ;
   test_arg(arg1, arg2, arg3) ;
   dbms_output.put_line(to_char(arg1, '99.99') ||
   to_char(arg2, '99.99') || to_char(arg3, '99.99'));

   end ;
   /
```

Un exemple de fonction :

```
declare
   n integer := 10 ;
   function Factorielle(i in integer)
      return integer is
      begin
      if    i = 0 then return 1 ;
      elsif i = 1 then return 1 ;
      else return i*Factorielle(i-1) ;
      end if ;
      end ; /* Factorielle */
   begin
   dbms_output.enable;
   dbms_output.put_line(to_char(Factorielle(n))) ;
   end ;
   /
```

5.5 Procédures et fonctions stockées

Pour donner aux procédures un caractère permanent, il faut les stocker dans la base de données.

```
CREATE OR REPLACE procedure test_arg(p_arg1 in number,
   p_arg2 out number, p_arg3 in out number) is
   begin /* proc */
   p_arg2 := p_arg1 ;
   p_arg3 := p_arg3 + 1 ;

end ; /* proc */


CREATE OR REPLACE function Factorielle(i in integer)


   return integer is
   begin
   if i = 0 then return 1 ;
   elsif i = 1 then return 1 ;
   else return i*Factorielle(i-1) ;
   end if ;

end ; /* Factorielle */
```

Remarque : s'il y a une ou plusieurs erreurs de compilation, il faut utiliser la commande *show error* pour obtenir les messages d'erreur.

5.6 Comment retrouver le code d'une procédure

Vous disposez de deux vues sur le dictionnaire de données :

- user_objects
- user_source

Dans user_objects, on peut lire la liste des procédures stockées :

```
select object_name

from user_objects

where object_type in ('FUNCTION', 'PROCEDURE') ;
```

Dans user_source, on peut accéder au source d'une procédure :

```
select line, text

from user_source

where name='FACTORIELLE'

order by line ;
```

5.7 Utilisation de fonctions stockées dans une requête SQL

SQL est un langage extensible. Toute fonction stockée est accessible dans une requête SQL.

Exemple : Le calcul du périmètre d'un cercle.

```
create table cercle(id_cercle varchar2(3), rayon
decimal) ;

create or replace function perimetre(rayon in decimal)
   return number is
   begin
   return 2*3.14159*rayon ;
   end ; /* perimetre */

select id_cercle, perimetre(rayon) from cercle ;
```

5.8 Types composite tableau et structure

Le type *table* de PL/SQL correspond au type tableau des langages de programmation classiques. La particularité du type table est que le dimensionnement est dynamique.

Exemple :

```
declare
   /* déclaration des types */
   type Tab_Rayon_Type is table of cercle.rayon%TYPE
        index by binary_integer;
   type Tab_Car_Type is table of varchar2(3)
        index by binary_integer;

   /* déclaration des tableaux */
   tab_rayon Tab_Rayon_Type;
   tab_car Tab_Car_Type;

   i binary_integer := 0;
   nb_tuples binary_integer := 0;
   somme number := 0;
   cercle_rec cercle%rowtype;
```

```
begin
   dbms_output.enable;
   for cercle_rec in (select id_cercle, rayon from cercle)
   loop
     i := i + 1;
     tab_rayon(i) := cercle_rec.rayon;
     tab_car(i) := cercle_rec.id_cercle;
   end loop;

   nb_tuples := i;

   for i in 1..nb_tuples loop
    somme := somme + tab_rayon(i);
   end loop;

   dbms_output.put_line(to_char(somme));
   dbms_output.put_line(tab_car(1));

end;

/
```

Le type *record* correspond au type *struct* du langage C.

Exemple :

```
declare
   type Rec_Cercle_Type is record
        (id_cercle cercle.id_cercle%TYPE,
         rayon cercle.rayon%TYPE,
         info varchar2(3));

   rec_rayon Rec_Cercle_Type;

begin
   dbms_output.enable;
   for cercle_rec in (select id_cercle, rayon from cercle)
   loop
     rec_rayon.rayon := cercle_rec.rayon;
     rec_rayon.id_cercle := cercle_rec.id_cercle;
     rec_rayon.info := cercle_rec.id_cercle;

     rec_rayon.rayon := rec_rayon.rayon + 1;

     update cercle
     set rayon = rec_rayon.rayon
     where id_cercle = rec_rayon.id_cercle;
   end loop;

   dbms_output.put_line('Fin maj');

end;

/
```

5.9 Création d'un package

La notion de package permet de construire des types abstraits. On y regroupe l'ensemble des fonctions et des procédures qui décrivent le comportement de l'objet.

La structure de données de l'objet est implémentée :

- sous la forme de table(s) s'il s'agit d'un objet persistant,
- sous la forme de variables s'il s'agit d'un objet temporaire.

Lorsque l'on crée un package, on déclare :

- un package de spécification, interface avec l'application. On y déclare tous les objets (fonctions, procédures et variables) visibles par l'application.
- un package d'implémentation

Exemple : spécification d'un package pour gérer des cercles

```
create or replace package P_Cercle is
   nombre_visible number := 10;
   function Perimetre(rayon in number) return number;
   function Surface(rayon in number) return number;

end; /* package P_Cercle */

/
```

Exemple : implémentation du package pour gérer des cercles

```
create or replace package body P_Cercle is
   /* Implementation du package */

   nombre_cache number := 1;

   function Perimetre(rayon in number) return number is
   begin
      return 2*3.14159*rayon;
   end; /* Perimetre */

   function Surface(rayon in number) return number is
   begin
      return 3.14159*rayon*rayon;
   end; /* Surface */

end P_Cercle;

/
```

Exemple : Utilisation du package pour gérer des cercles

```
begin
   dbms_output.enable;
   dbms_output.put_line(to_char(p_cercle.perimetre(10)));
   dbms_output.put_line(to_char(p_cercle.nombre_visible));

end;

/
```

5.10 Création de triggers

Les triggers sont des traitements déclenchés automatiquement lorsqu'interviennent des opérations de mise à jour dans la base de données.

- Ils permettent d'implémenter des règles de gestion portant sur les données.

  Dans une application de réapprovisionnement de stock, on désire lancer automatiquement une commande lorsque le stock d'un produit passe en dessous d'un certain seuil.

- Ils permettent de renforcer les contraintes d'intégrité sur les données.

  Avec Oracle, tous les contrôles de saisie mettant en jeu la date du jour sont réalisés sous la forme de trigger.

- Ils permettent d'auditer le fonctionnement de la base de données.

  Statistiques sur le nombre de mises à jour par table par poste de travail, par unité de temps, ...

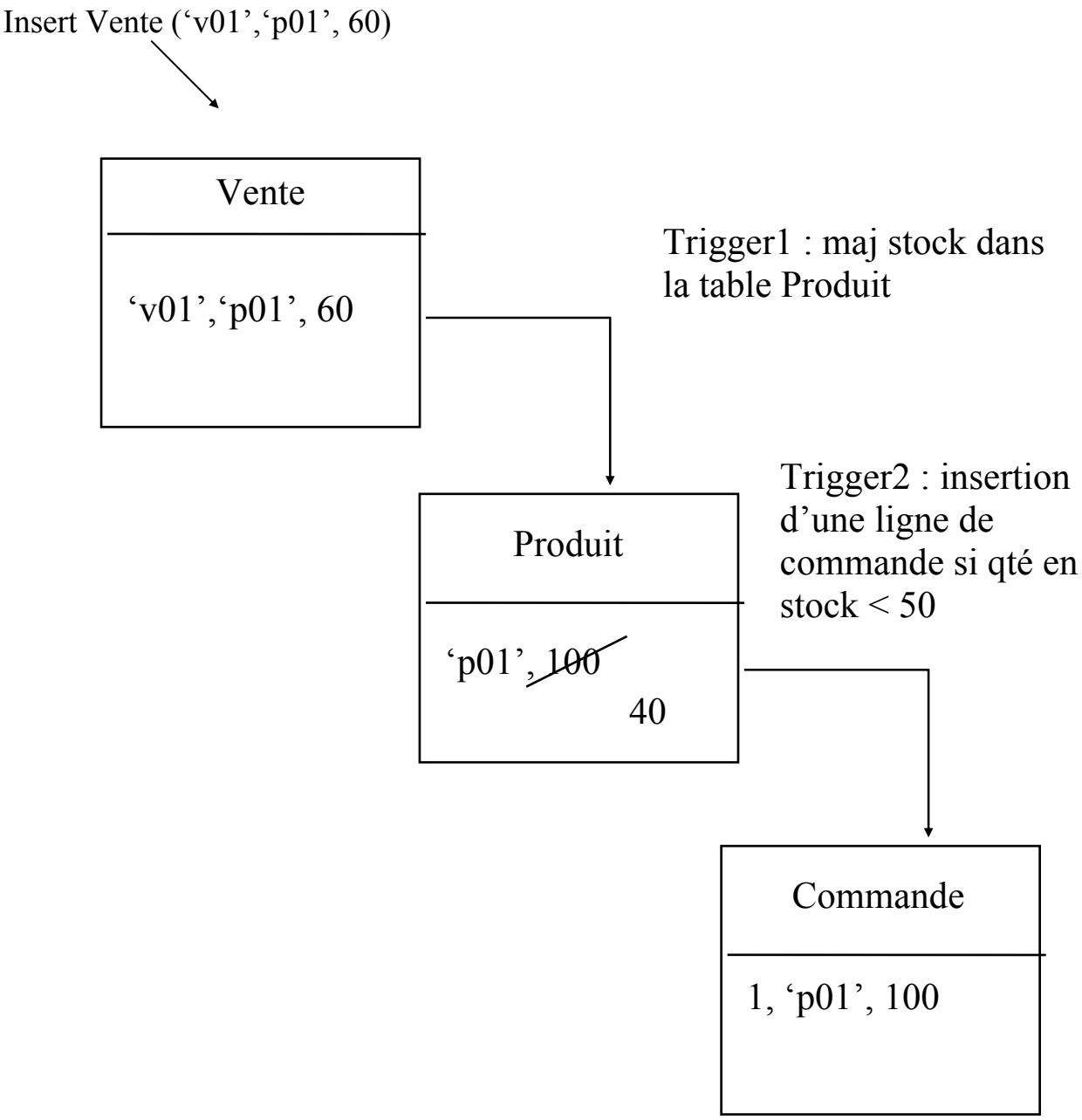
Exemple : On a trois tables

```
create table produit (
   ref_produit varchar2(3)
      constraint pk_produit primary key,
   qte_stock number);

create table vente (
   ref_vente number
      constraint pk_vente primary key,
   ref_produit varchar2(3),
      constraint fk_vente foreign key(ref_produit)
      references produit,
   qte_vendue number);

create table commande (
   ref_commande varchar2(3)
      constraint pk_commande primary key,
   ref_produit varchar2(3),
      constraint fk_commande foreign key(ref_produit)
   references produit,
   qte_commandee number);
```

Scénario d'enchaînement des triggers :



Remarque : un select implicite est réalisé lors de l'exécution du 2ème trigger pour vérifier l'intégrité référentielle (identifiant produit).

Trigger 1 : calcul de la variation de stock

```
create or replace trigger Calcul_variation_stock
   after insert on vente
   for each row

begin
   update produit
          set qte_stock = qte_stock - :new.qte_vendue
          where ref_produit = :new.ref_produit;

end;
   /
```

Il existe deux types de triggers : trigger ligne (option *for each row*) et trigger instruction (option par défaut)

L'ordre SQL provoquant le déclenchement du trigger peut modifier plusieurs lignes.

Définitions : Un trigger instruction est exécuté une seule fois quelque soit le nombre de lignes modifiées. Un trigger ligne est exécuté pour chaque ligne modifiée.

Dans un trigger ligne, on peut référencer les colonnes de la ligne modifiée par l'ordre SQL.

- suite à un insert : on accède aux valeurs insérées par **:new.nom_colonne,**
- suite à un update, on accède aux anciennes valeurs par **:old.nom_colonne** et aux nouvelles par **:new.nom_colonne,**
- suite à un delete, on accède aux anciennes valeurs par **:old.nom_colonne**.

Trigger 2 : Déclenchement d'une commande sur rupture de stock

Une première version du trigger :

```
create sequence seq_id_commande increment by 1 start with
1 order;

create or replace trigger Lancer_commande_produit
   after insert or update of qte_stock on produit
   for each row

begin
   if :new.qte_stock < 50 then
      insert into commande values
   (seq_id_commande.nextval, :new.ref_produit, 100);
   end if;

end;
   /
```

Gestion des triggers : supprimer, activer, désactiver

```
drop trigger Lancer_commande_produit ;

alter trigger Lancer_commande_produit enable ;

alter trigger Lancer_commande_produit disable ;
```

Accès aux triggers :

```
select trigger_name from user_triggers ;

select trigger_body from user_triggers
    where trigger_name = 'LANCER_COMMANDE_PRODUIT'
```

5.11 TD Création de triggers pour la base Cinéma

1) Ecrivez un trigger qui met à jour l'attribut nb_roles_titre de la table Acteur à chaque mise à jour d'une ligne de la table Vedette.

2) Ecrivez un trigger qui stocke dans une table de statistiques les accès à la table Acteur en insertion, suppression, modification. La structure de la table de statistiques est :

id_stat, user, date, nom_table, type_acces, nom_acteur

- avec user = nom du compte Oracle ayant réalisé la modification
- nom_table = 'acteur'
- type_acces = 'INSERT' ou 'DELETE' ou 'UPDATE'

Remarque 2 : L'identifiant de la table de statistique est géré avec un objet de type séquence

Remarque 3 : on obtient la date du jour avec le mot-clé SYSDATE et le nom de l'usager ORACLE avec le mot-clé USER.

5.12 TD Création d'un package de fonctions SIG

Pour une application de gestion du cadastre, on désire stocker dans la base de données le parcellaire. Dans ce TD, vous allez créer l'ensemble des tables permettant de gérer ces informations. Puis, vous allez écrire deux fonctions. La première calcule le rectangle englobant d'une parcelle. La seconde renvoie pour un point (x,y) donné (sélectionné sur une carte par un clic souris par exemple) le numéro du polygone auquel il appartient ou la valeur 0 sinon.

Les fonctions seront regroupées dans un package. Leurs signatures sont les suivantes :

```
procedure Rectangle_englobant (no_parcelle in smallint) ;

function Point_dans_Poly (
   coordx in Point.x%TYPE, coordy in Point.y%TYPE)
   return smallint ;
```

Le schéma de la base est le suivant :

Parcelle(#id_parcelle, adresse, x_hg, y_hg, x_bd, y_bd)

Proprietaire (#id_proprietaire, nom_proprietaire)

Point(#id_point, x, y)

Segment(#id_segment, id_point1, id_point2)

Delimite(#id_parcelle, #id_segment)

Possede(#id_proprietaire, #id_parcelle, date_achat)

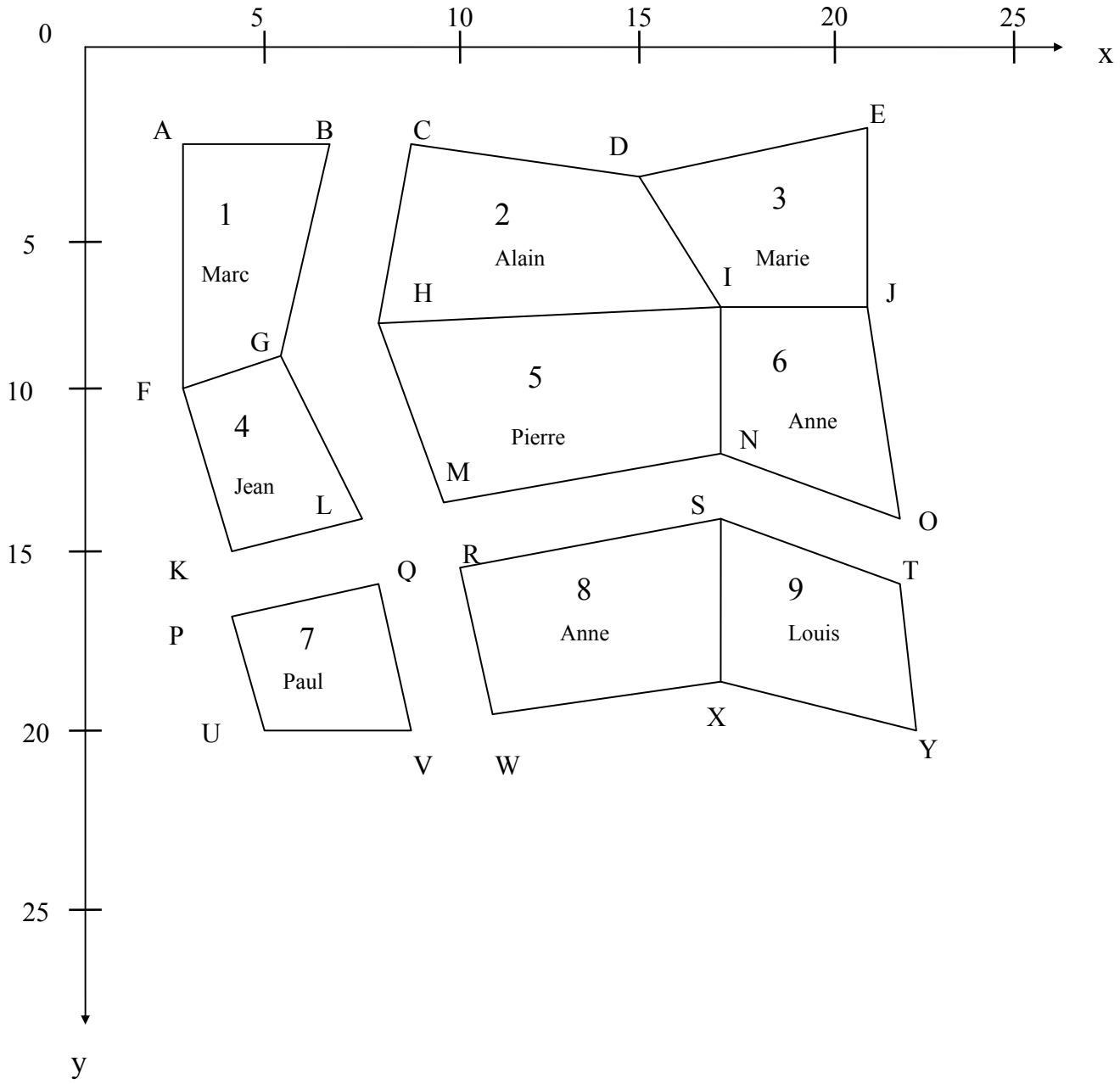
/Bloc PL/SQL pour tester le package :

```
declare
   resultat smallint:=0;
   begin
   dbms_output.enable;
   p_sig.rectangle(3);
   resultat := p_sig.point_dans_poly(25,10);
   if resultat = 0 then
      dbms_output.put_line('le point n''est inclus dans
   aucun polygone');
   else
      dbms_output.put_line('le point est inclus dans le
   polygone '|| resultat );
   end if;
   end;

/
```

Remarque : Il faut exécuter la commande *set serveroutput on* pour visualiser les impressions *dbms_output*.

Exemple de parcellaire :



Théorème de Jordan : Lorsqu'un point se trouve à l'intérieur d'un polygone, toute demi-droite partant de ce point a un nombre d'intersections impair avec les segments délimitant ce polygone. S'il est à l'extérieur de tout polygone, le nombre d'intersections est pair.

Optimisation algorithmique :
On ne fait le test de l'intersection que si le point (x, y) se situe à l'intérieur du rectangle englobant le polygone considéré. Puis, on considère la demi-droite horizontale partant du point vers la droite. De cette manière, le test de l'intersection entre la demi-droite et un segment du polygone revient à comparer l'ordonnée du point aux ordonnées des extrémités du segment. Si l'ordonnée du point est comprise entre les deux ordonnées des extrémités du segment, il reste à vérifier que l'origine de la demi-droite se trouve à gauche du segment pour qu'il y ait intersection.

6. L'interface C/SQL

6.1 Objectif

Pouvoir accéder aux données gérées par le SGBD relationnel depuis un programme écrit en C.

6.2 Syntaxe des ordres SQL inclus dans un programme

Chaque élément de communication avec le SGBD (déclaration de variables de communication, requêtes, gestion des erreurs, ...) doit être précédé par le mot-clé EXEC SQL.

Le pré-compilateur traduit les instructions EXEC SQL en appels de fonctions C que l'éditeur de liens saura résoudre en utilisant les bibliothèques spécifiées dans le Makefile.

Le cycle de développement d'un programme C incluant des accès SGBD est donc :

1. Pré-compilation du programme (*prog.pc*) incluant des instructions EXEC SQL et génération d'un source *prog.c*.
2. Compilation du programme source (*prog.c*)
3. Edition de liens de l'objet (prog.o) avec les bibliothèques d'appel du SGBD.

6.3 Zone de communication avec le SGBD : la SQLCA

La SQLCA est une structure de données copiée dans le programme dans laquelle le SGBD écrit des informations sur le déroulement de la requête (code erreur éventuel, nombre de tuples retournés par la requête, ...).

On inclut cette structure de données par :

```
EXEC SQL INCLUDE sqlca.h
```

6.4 Communication de données et variable hôte

Les variables hôtes sont les zones de communication entre le SGBD et le programme hôte. Une variable hôte peut être de type scalaire (un entier, une chaine de caractères, ...) ou tableau.

Elles sont utilisées en entrée pour transmettre des données au SGBD et en sortie pour récupérer des données résultat ou des informations sur le déroulement des requêtes.

Les variables hôtes doivent être déclarées dans une DECLARE SECTION au début du programme et doivent être précédées par le caractère **:** dans les requêtes SQL.

6.5 Un premier exemple : la connection au SGBD

```
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR username[20];
    VARCHAR password[20];
    EXEC SQL END DECLARE SECTION;
    /* VARCHAR est un pseudo-type qui permet de gérer les
    chaines de caractères de taille variable */
    EXEC SQL INCLUDE sqlca.h;

void Connection()
    {/* on renseigne les deux champs de la structure VARCHAR
    */
    strcpy(username.arr, "speinfo");
    username.len = strlen(username.arr);
    strcpy(password.arr, "speinfo");
    password.len = strlen(password.arr);
    EXEC SQL WHENEVER SQLERROR GOTO sqlerror;
    EXEC SQL CONNECT :username IDENTIFIED BY :password;
    printf("Connexion a Oracle sous le compte : %s \n",
    username.arr);
    return;
    sqlerror:
    printf("\nMessage d'erreur : \n% .70s \n",
    sqlca.sqlerrm.sqlerrmc);
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(1);
    }/* Connection */
```

6.6 Gestion des erreurs

Pour effectuer des contrôles sur le bon déroulement des requêtes, on utilise l'instruction WHENEVER.

```
EXEC SQL WHENEVER <condition> <action>
```

L'instruction WHENEVER est valide jusqu'à la fin du déroulement du programme. Sauf si une autre instruction WHENEVER est rencontrée.

Les conditions peuvent être du type :

- Erreur d'exécution de la requête : SQLERROR
- Aucun tuple sélectionné par un SELECT : NOT FOUND

Les actions peuvent être du type :

- Branchement inconditionnel à une étiquette pour traiter l'erreur : GOTO label
- Eliminer les risques de bouclage dans une section de traitement d'erreur qui inclut une instruction SQL : CONTINUE

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
sql_error :
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL ROLLBACK WORK RELEASE;
...
```

- Sans le EXEC SQL WHENEVER SQLERROR CONTINUE, une erreur sur le ROLLBACK aurait invoqué la section sql_error de manière infinie.

6.7 Récupération d'un seul tuple

On veut réaliser deux types d'opération transactionnelles sur la table des acteurs de la base Cinéma : une interrogation sur le nom de l'acteur et une modification des attributs.

```
#include <stdio.h>
#include <ctype.h>
/* prog4 illustre la connection a Oracle et la
manipulation d'une table en insertion, modification,
suppression et recherche */
EXEC SQL BEGIN DECLARE SECTION;
char nom[21];
char prenom[21];
char sexe[2];
int  nb_films;
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca.h;
void main()
{char action[10];

/* connexion sur le compte oracle */
Connection();
while(1)
{
printf("\nI)nsertion, M)odification, S)uppression,
R)echerche, F)in ?\n");

gets(action);
```

```
switch (action[0])
{
   case 'I' : case 'i' : insertion(); break;
   case 'M' : case 'm' : modification(); break;
   case 'S' : case 's' : suppression(); break;
   case 'R' : case 'r' : recherche(); break;
   case 'F' : case 'f' : break;
   default : printf("\nTouche fonction invalide\n");
   break;
}/* switch */

if (action[0] == 'F' || action[0] == 'f') break;
}/* while */

/* Deconnexion de la base */
EXEC SQL COMMIT WORK RELEASE;
printf("fin de prog4\n");
exit(0);
} /* main */
```

L'interrogation :

```
void recherche ()
{
printf("Entrez le nom de l'acteur a rechercher :\n");
scanf("%s", nom);
if (!strcmp(nom, "")) return;

EXEC SQL WHENEVER SQLERROR GOTO sqlerror;
EXEC SQL WHENEVER NOT FOUND GOTO pastrouve;
EXEC SQL SELECT NOM, PRENOM, SEXE, NB_FILMS
INTO :nom, :prenom, :sexe, :nb_films
FROM ACTEUR
WHERE NOM = :nom;

printf("Acteur recherche :\n");
printf("%s %s %s %d", nom, prenom, sexe, nb_films);
return;

pastrouve:
printf("L'acteur %s n'existe pas \n", nom);
return;

sqlerror:
printf("\nMessage d'erreur : \n% .70s \n",
sqlca.sqlerrm.sqlerrmc);
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
} /* recherche */
```

La modification :

```
void modification ()
{
char e_nom[21];
char e_prenom[21];
char e_sexe[2];
char e_nb_films[5];

printf("Entrez le nom de l'acteur a modifier :\n");
scanf("%s", nom);
if (!strcmp(nom, "")) return;

EXEC SQL WHENEVER SQLERROR GOTO sqlerror;

EXEC SQL WHENEVER NOT FOUND GOTO pastrouve;

EXEC SQL SELECT NOM, PRENOM, SEXE, NB_FILMS
INTO :nom, :prenom, :sexe, :nb_films
FROM ACTEUR
WHERE NOM = :nom FOR UPDATE;

printf("Acteur a modifier :\n");
printf("%s %s %s %d\n", nom, prenom, sexe, nb_films);
printf("Entrez une nouvelle valeur pour le prenom (%s) :
", prenom);
gets(e_prenom);
if (strcmp(e_prenom, "")) strcpy(prenom, e_prenom);
printf("Entrez une nouvelle valeur pour le sexe (%s) : ",
sexe);
gets(e_sexe);
if (strcmp(e_sexe, "")) strcpy(sexe, e_sexe);
```

```
printf("Entrez une nouvelle valeur pour le nombre de
films (%d) : ", nb_films);
gets(e_nb_films);
if (strcmp(e_nb_film, "")) nb_films = atoi(e_nb_films);

EXEC SQL UPDATE ACTEUR
SET
PRENOM = :prenom, SEXE = :sexe, NB_FILMS = :nb_films
WHERE NOM = :nom;

EXEC SQL COMMIT;

printf("\n\nLa modification pour l'acteur %s est
enregistre\n", nom);
return;

pastrouve:
printf("L'acteur %s n'existe pas \n", nom); return;

sqlerror:
printf("\nMessage d'erreur : \n% .70s \n",
sqlca.sqlerrm.sqlerrmc);
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}/* modification */
```

6.8 Accès à un ensemble de tuples et notion de curseur

Remarque : Dans l'exemple d'interrogation du paragraphe précédent, une recherche sur le sexe de l'acteur aurait renvoyé plusieurs tuples. Lorsque l'on veut récupérer un ensemble de tuples, il faut déclarer un curseur (i.e un pointeur) qui permet de se déplacer d'un tuple au suivant.

On réalise la gestion de l'ensemble de tuples retourné par le SGBD avec 4 ordres : DECLARE, OPEN, FETCH et CLOSE.

- DECLARE permet de déclarer le curseur et de lui associer une requête.
- OPEN exécute la requête et identifie tous les tuples associés. Le curseur est placé avant le premier tuple.
- FETCH ramène les tuples un par un dans les variables hôte pour exploitation par le programme. Après chaque FETCH, le curseur se déplace du tuple lu au tuple suivant.
- CLOSE libère les zones de mémoire réservées pour traiter la requête et l'ensemble de tuples associé

Exemple: Recherche dans la table Acteur en interrogeant sur le sexe :

```
EXEC SQL DECLARE liste_acteurs CURSOR FOR
SELECT NOM, PRENOM, SEXE, NB_FILMS
FROM ACTEUR
WHERE SEXE = :sexe;

printf("Quel sexe (M/F) ? :\n"); scanf("%s", sexe);

EXEC SQL OPEN liste_acteurs;

EXEC SQL WHENEVER NOT FOUND GOTO fin_du_fetch;

for ( ; ; ) {
EXEC SQL FETCH liste_acteurs
INTO :nom, :prenom, :sexe, :nb_films;

printf("%s %s %s %d\n", nom, prenom, sexe, nb_films);
}/* for */

fin_du_fetch: /* on ferme le curseur */
EXEC SQL CLOSE liste_acteurs;
```

Le curseur ne peut se déplacer qu'en marche-avant dans l'ensemble des tuples retournés. Pour revenir à un tuple déjà traité (par FETCH), il faut réouvrir le curseur (par OPEN) et redémarrer depuis le premier tuple.

6.9 Accès à un ensemble de tuples stockés dans un tableau local

Intérêt : Dans la gestion du curseur montrée ci-dessus, la communication se fait tuple par tuple. Il est possible de communiquer tableau de tuples par tableau de tuples. L'intérêt essentiel est l'amélioration des performances obtenues.

Exemple: Recherche dans la table Acteur en interrogeant sur le sexe :

```
EXEC SQL BEGIN DECLARE SECTION;
char sexe_entree[2];
char nom[4][11];
char prenom[4][21];
char sexe[4][2];
int  nb_films[4];
EXEC SQL END DECLARE SECTION;

void affiche_tuples(int nb_tuples_a_afficher)
{int i;
   for (i=0; i < nb_tuples_a_afficher; i++)
   {nom[i][10]='\0'; prenom[i][20]='\0'; sexe[i][1]='\0';
   printf("%s %s %s %d", nom[i], prenom[i], sexe[i],
   nb_films[i]);
   }/* for i */
}/* affiche_tuples */
main()
{int nb_tuples; /* nombre de tuples retourne apres n
boucles */
/* connexion sur le compte oracle */ Connection();
```

```
EXEC SQL DECLARE liste_acteurs CURSOR FOR
SELECT NOM, PRENOM, SEXE, NB_FILM
FROM ACTEUR
WHERE SEXE= :sexe_entree;
printf("Quel sexe (M/F) :\n");
scanf("%s", sexe_entree);
EXEC SQL OPEN liste_acteurs;
EXEC SQL WHENEVER NOT FOUND GOTO fin_de_boucle;
nb_tuples = 0;

while(1)
{EXEC SQL FETCH liste_acteurs
INTO :nom, :prenom, :sexe, :nb_film;
affiche_tuples(sqlca.sqlerrd[2] - nb_tuples);
nb_tuples = sqlca.sqlerrd[2];
/* nb tuples deja retournes */
printf("nb_tuples = %d\n", nb_tuples);
}/* while */

fin_de_boucle:
/* NOT FOUND est vrai quand le fetch ne remplit pas le
tableau */
if ((sqlca.sqlerrd[2] - nb_tuples) > 0)
affiche_tuples(sqlca.sqlerrd[2] - nb_tuples);

EXEC SQL CLOSE liste_acteurs;
EXEC SQL COMMIT RELEASE;
/* Deconnexion de la base */
printf("fin de prog3\n"); exit(0);
}/* main */
```

6.10 Appel de procédures stockées depuis le programme C

On insère un bloc PL/SQL dans le programme C.

```
EXEC SQL BEGIN DECLARE SECTION;
int arg, fact, x, y, no_poly, nb_rect;
EXEC SQL END DECLARE SECTION;

void main()
{Connection();
arg = 4 ; nb_rect = 3 ; x=5 ; y=5 ;
EXEC SQL EXECUTE
DECLARE
i INTEGER ;
BEGIN
/* Debut du bloc PL/SQL */
:fact := Factorielle(:arg) ;
for i in 1.. :nb_rect loop
  /* calcul du rectangle englobant du polygone i */
  p_sig.rectangle(i);
end loop ;
/* détermination du polygone pointé par (x, y) */
:no_poly := p_sig.point_dans_poly(:x, :y);
/* Fin du bloc PL/SQL */
END;
END-EXEC;
printf(" Factorielle de %d = %d\n ", arg, fact) ;
printf(" no_poly = %d\n ", no_poly) ;
}/* fin du main */
```

6.11 TD Manipulation d'une base en C/SQL

6.11.1 Objectif du TD :

On se propose de réaliser une mini application client/serveur en langage C/SQL pour se familiariser à l'intégration de requêtes SQL dans du code L3G (C en l'occurrence). L'étude de cas support du TD est la base Cinéma déjà utilisée pour le TD de manipulation SQL.

Vous commencerez par recopier dans votre répertoire de travail les fichiers et sous-répertoires se trouvant dans le répertoire :

```
Sur le lecteur td, dans le répertoire \td_BDSPE\Td9
```

Vous y trouverez un répertoire ProC qui contient

- les fichiers sources en C/SQL. Ces fichiers sont suffixés par *.pc* : util.pc, main.pc, acteur.pc, film.pc et vedette.pc.
- les fichiers signatures associés (d'extension *.h*).
- le fichier de configuration pour le préprocesseur (Td6Eleve.pre)

Le fichier **util.pc** contient la fonction pour se connecter à Oracle. Vous éditerez ce fichier pour y indiquer dans la fonction **Connection** votre compte Oracle et le password associé. Le fichier **main.pc** contient les fonctions de dialogue avec l'utilisateur de l'application **Cinéma**.

Dans les fichiers acteur.pc, film.pc et vedette.pc, sont décrites les fonctions pour gérer les données stockées dans les tables acteur, film et vedette **(recherche, suppression, insertion, modification).** On y trouvera également des fonctions d'édition.

Le sous-répertoire Td6Eleve contient les fichiers projet pour Microsoft Visual C++. Les fichiers main.pc, util.pc et acteur.pc contiennent des fonctions déjà implémentées. Votre travail va consister à implémenter les fonctions du fichier vedette.pc. On ne s'intéressera pas au fichier film.pc.

6.11.2 Vérification du fonctionnement de l'application cine sur votre base Oracle :

Vous éditerez le fichier **util.pc** pour y indiquer dans la fonction **Connection** votre compte Oracle et le password associé (par défaut, la connexion se fait sur le compte test.

Modifiez dans le fichier *TdEleve.pre* sous l'application cliente Pro*C l'emplacement des fichiers sources C/SQL (.pc) correspondant à votre répertoire de travail. Demandez un contrôle SQL de type sémantique lors du préprocessing par le menu Edit/option (vérification que les objets référencés dans le programme C/SQL existent bien dans la base). Lancez la traduction des sources C/SQL en programme C (extension .c).

Depuis le répertoire TDEleve, démarrez Visual C++ avec le fichier projet *TdEleve.dsw*. Vérifiez que le fichier bibliothèque des fonctions Oracle (oraSQL8.lib) est bien inclus dans le projet.

Ensuite, lancez la fabrication de l'exécutable.

En vous connectant à la base Cinéma, testez l'application **cine** sur la table **acteur** pour vérifier que tout fonctionne bien.

6.11.3 Implémentation des fonctions d'accès à la table vedette :

Dans ce TD, vous allez implémenter les fonctions associées à la table **vedette**. L'implémentation de ces fonctions s'inspirera de celles de la table **acteur.** Cette partie du TD consiste donc à écrire les fonctions **insertion_vedette**, **modification_vedette**, **recherche_vedette** et **suppression_vedette**.

Dans **insertion_vedette**, il faut vérifier que l'acteur ne joue pas déjà un autre rôle-titre dans le film. On vérifiera également que le rôle n'a pas déjà été affecté à un autre comédien. Dans ces deux cas, on rejettera la transaction.

Dans **modification_vedette**, on saisit le nom de l'acteur et le titre du film et on recherche le tuple. S'il existe, on propose la modification du nom du rôle.

On vérifiera que le nouveau rôle saisi n'est pas déjà affecté à un autre comédien.

Dans **recherche_vedette**, on saisit le nom de l'acteur et le titre du film et on recherche le tuple. S'il existe, on affiche le nom du rôle.

Dans **suppression_vedette**, on saisit le nom de l'acteur et le titre du film. On recherche le tuple pour le supprimer.

6.11.4 Implémentation des fonctions d'édition

Dans cette partie, on écrira les fonctions **Edition_vedettes** et **Edition_films**.

La fonction **Edition_vedettes** édite à l'écran le titre du film, le nom du rôle et le nom de l'acteur pour chaque tuple de la table Vedette. Cette édition correspond à la requête Q3 du TD3. La requête est simple. Elle est exécutée depuis le programme C en utilisant un curseur.

La fonction **Edition_film** édite les noms, prénoms et le nombre de films des vedettes d'un film donné. Cette édition correspond à la requête Q5 du TD3. Elle fait intervenir une jointure entre la table **acteur** et la table **vedette** pour pouvoir récupérer le nombre de films des acteurs.

De manière générale, lorsque l'on écrit une requête SQL, il est conseillé de commencer par la mettre au point sous l'interpréteur SQL (sqlplus). Puis, on l'intègre dans le code C.

7. Le contrôle des accès concurrents

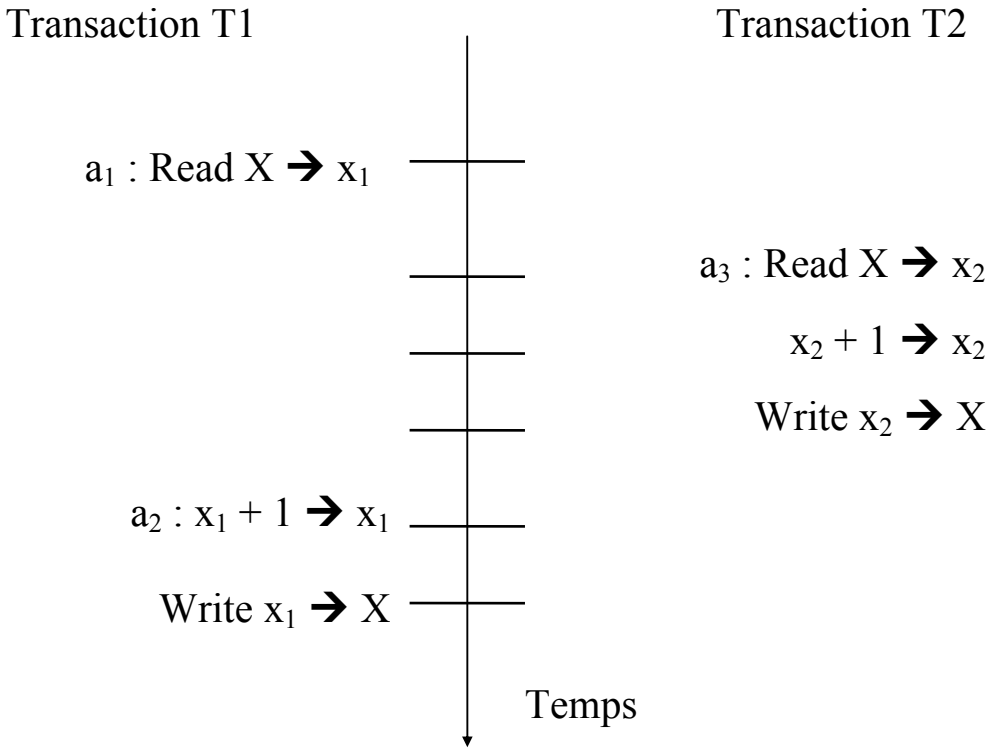
7.1 Problèmes de concurrence

Le contrôle de concurrence rend le partage des données transparent pour l'usager.

Quels types de problème peuvent survenir lors de l'exécution simultanée de transactions ?

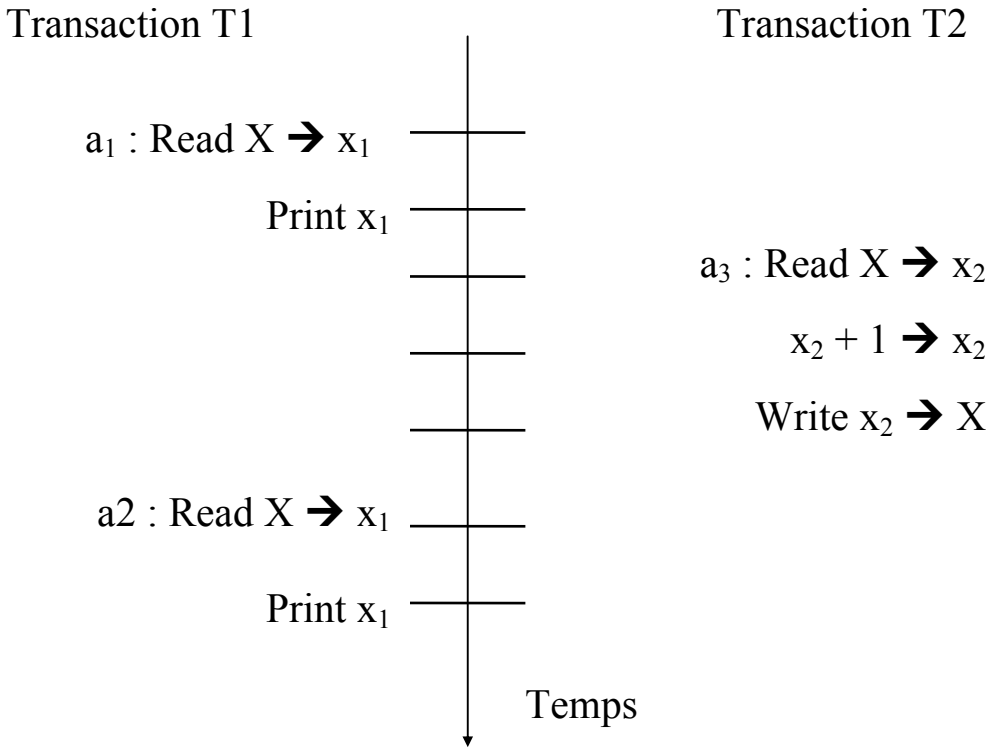
7.1.1 Perte d'opération de mise à jour

Une transaction T1 lit dans un tampon un objet (action a_1), le modifie à partir de la valeur stockée dans le tampon (action a_2), alors qu'entre a_1 et a_2 une autre transaction T2 a modifié la valeur de l'objet (action a_3) :



7.1.2 Non reproductibilité des lectures

Une transaction lit deux fois une même donnée et trouve deux valeurs différentes du fait que la donnée a été modifiée par une transaction concurrente entre les deux lectures.



7.2 Exécution sérialisable, transaction ACID

Une exécution successive des transactions (sans simultanéité entre les transactions) garantie une exécution sans perte d'opérations.

Objectif : Mettre en place un système de contrôle qui garantisse une exécution sérialisée des transactions concurrentes. On parle de transactions ACID. ACID est un acronyme pour Atomic, Consistent, Isolation, et Durable. Les transactions suivent un modèle simple de succès ou d'échec. Soit une transaction "commits" (i.e. toutes ses actions sont exécutées), soit elle "aborts" (i.e. toutes ses actions sont défaites). Ce fonctionnement en tout ou rien rend le modèle de programmation simple. Les caractéristiques d'une transaction ACID sont:

Atomicité : Une transaction autorise le regroupement d'une ou plusieurs modification(s) dans les tables de la base de données pour former une opération atomique (indivisible). Soit toutes les modifications sont enregistrées, soit aucune. Si, pour une raison quelconque, la transaction ne peut être terminée, tout ce qui a été modifié par la transaction doit être restauré dans l'état antérieur à l'exécution de la transaction par une opération de rollback.

Consistence : Les transactions opèrent toujours sur une vue consistante des données et quand elles se terminent, elles laissent les données dans un état consistant. Les données sont dites consistantes, tant qu'elles vérifient un certain nombre d'invariants (unicité de la clé primaire, contrôle d'intégrité référentielle). Tant qu'une transaction s'exécute, ces invariants peuvent être violés, mais aucune autre transaction n'est autorisée à les voir et toutes ces inconsistences doivent être éliminées avant la fin de la transaction.

Isolation : Pour une transaction donnée, tout se passe comme-ci la transaction est seule à opérer sur la base. Les effets des transactions concurrentes sont invisibles pour cette transaction et les effets de cette transaction sont invisibles pour les autres tant qu'elle n'est pas committée.

Durabilité : Lorsque la transaction est committée, ses effets sont guarantis, même si une panne système intervient. Par contre, en cas de panne, les effets d'une transaction en cours seront perdus.

7.3 Stratégie basée sur l'ordonnancement initial des transactions

Stratégie optimiste : Les conflits n'arrivent pas souvent. En cas de problème, l'une des deux transactions qui entrent en conflit est défaite et doit être reprise (stratégie curative). Le contrôle s'appuie sur un mécanisme d'estampillage.

- **Estampille de transaction :** on associe une référence unique (valeur numérique) à chaque transaction permettant d'ordonner cette transaction par rapport aux autres.
- **Estampille de granule :** on associe à chaque granule d'information (par exemple chaque valeur stockée dans une colonne de table) l'estampille de la dernière transaction ayant opéré sur cette granule.

7.3.1 Algorithme d'ordonnancement total

Cet algorithme vérifie que les accès aux granules par les transactions s'effectuent bien dans l'ordre affecté au lancement des transactions (repéré par les estampilles). Si l'ordre n'est pas respecté, le contrôleur interrompt et provoque la reprise de la transaction.

Notations : l'estampille du granule g est notée E(g), la transaction T_i a pour estampille i.

```
Procédure Read(Ti, g);
   si (E(g) ≤ i) alors
      exécuter la lecture; E(g) := i;
   sinon Abort; finsi

FinProc Read
```

```
Procédure Write(Ti, g);
   si (E(g) ≤ i) alors
      exécuter l'écriture; E(g) := i;
   sinon Abort; finsi

FinProc Write
```

Remarque : Cet algorithme ordonne toutes les opérations (lecture ou écriture) sur les granules. Or, on peut permuter sans introduire d'anomalie deux opérations de lecture.

7.3.2 Algorithme d'ordonnancement partiel

Ce deuxième algorithme ordonne uniquement les opérations non permutables, ie Read/Write, Write/Read, Write/Write.

Deux estampilles sont associées aux granules :

- EL, l'estampille en lecture (numéro de la dernière transaction ayant opérée une lecture sur la granule),
- EE, l'estampille en écriture (numéro de la dernière transaction ayant opérée une écriture sur la granule).

Lorsqu'une transaction exécute Read, le contrôleur vérifie le séquencement (uniquement) par rapport au dernier Write.

```
Procédure Read(Ti, g);
   si (EE(g) ≤ i) alors
      exécuter la lecture;
      EL(g) := Max(EL(g), i);
      /* Max car, une autre transaction
       peut avoir fait un accès en lecture */
   sinon Abort; finsi

FinProc Read
```

Lorsqu'une transaction exécute Write, le contrôleur vérifie le séquencement par rapport au dernier Write et au dernier Read.

```
Procédure Write(T_i, g);
   si (EL(g) ≤ i) et (EE(g) ≤ i) alors
      exécuter l'écriture;
      EE(g) := i;
   sinon Abort; finsi

FinProc Write
```

7.4 Stratégie basée sur le verrouillage des granules

On met en place une stratégie de prévention des conflits. La stratégie de type verrouillage évite les conflits en faisant attendre les transactions voulant exécuter des opérations conflictuelles sur un même objet.

L'algorithme de verrouillage ne laisse s'exécuter que des opérations compatibles.

7.4.1 Modes d'opération

On définit une liste d'opérations possibles, par exemple :

- M1 = consultation (Read) non protégée (avec risque de lecture inconsistante),
- M2 = consultation (Read) protégée (sans risque de lecture inconsistante),
- M3 = mise à jour non protégée (autres mises à jour autorisées),
- M4 = mise à jour protégée,
- M5 = consultation exclusive,
- M6 = mise à jour exclusive.

On définit les compatibilités entre ces modes opératoires dans une matrice C_{ij} où $C_{ij}=1$ si les modes M_i et M_j sont compatibles et 0 sinon.

Exemple : Matrice du groupe de normalisation DBTG Codasyl

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

7.4.2 Protocole de verrouillage

LOCK(g, M) permet à une transaction d'indiquer au contrôleur le type d'opération (défini par le vecteur de bits M) qu'il désire réaliser sur le granule g.

$$\mathrm{M} = \begin{bmatrix} m_1 \\ \cdots \\ m_j \\ \cdots \\ m_6 \end{bmatrix}$$

où $m_j = 1$ si le mode M_j est demandé par la transaction T_i sur le granule g et 0 sinon.

UNLOCK(g) permet à une transaction d'indiquer au contrôleur qu'elle a terminé l'opération sur le granule g.

7.4.3 Algorithme de verrouillage

On mémorise la liste des opérations en cours sur le granule g dans une liste de vecteurs $A_i(g)$ tels que :

$$A_i(g) = \begin{bmatrix} a_1 \\ \dots \\ a_j \\ \dots \\ a_6 \end{bmatrix}$$

où $a_j = 1$ si le mode M_j est utilisé par la transaction T_i sur le granule g et 0 sinon.

Les modes d'opérations demandés lors d'un appel à la primitive Lock(g, M) exécutée par une transaction T_p sont compatibles avec les modes en cours d'exécution sur g par les autres transactions si :

$$M \subset \neg(\neg C * \bigcup_{i \neq p} A_i(g))$$

Notations employées :

$\bigcup$ union logique de vecteurs booléens

$\subset$ inclusion logique de vecteurs booléens

$*$ produit matriciel booléen (l'élément (i,j) de la matrice produit est l'union des intersections des éléments de même rang de la $i^{ème}$ ligne de la matrice gauche et de la $j^{ème}$ colonne de la matrice droite).

$\bigcup_{i \neq p} A_i(g)$ est un vecteur de bits dont le bit j est à 1 si le mode M_j est en cours d'exécution sur le granule g par une transaction en cours. Cette expression définit l'état actuel de verrouillage d'une granule par l'ensemble des transactions ayant obtenu un verrou sur elle.

Exemple : Soient 3 transactions en cours T_1, T_2, T_3

$$\bigcup_{i=1,..,3} A_i(g) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cup \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cup \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$\neg C$ est par définition la matrice des incompatibilités entre les modes d'opération.

$$\neg C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$\neg C * \bigcup_{i \neq p} A_i(g)$ est un vecteur de bits dont le bit j est à 1 si le mode M_j est incompatible avec un mode en cours d'exécution sur le granule g.

Rappel : Le produit matriciel booléen est l'union (ou logique) des intersections (et logique) des éléments de même rang de la $i^{ème}$ ligne de la matrice gauche et de la $j^{ème}$ colonne de la matrice droite.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

⇐ *Il existe un mode opératoire en cours incompatible avec le mode opératoire 2*

⇑ *Vecteur des modes opératoires incompatibles avec le verrouillage en cours*

$\neg(\neg C * \bigcup_{i \neq p} A_i(g))$ est donc un vecteur de bits dont le bit j est à 1 si le mode M_j est **compatible** avec un mode en cours d'exécution sur le granule g par une transaction autre que $T_{p.}$

```
Procédure LOCK(g, M)
   si  M  ⊂  ¬ ( ¬ C * ⋃ A_i ( g ))
                       i ≠ p
   alors   AjouterDansListeA(M, g);
      /* on ajoute M dans la liste des verrous posés sur g
      */
   sinon   Insérer (p, M) dans la file d'attente Q[g];
           Bloquer la transaction T_p
   finsi

finproc LOCK
```

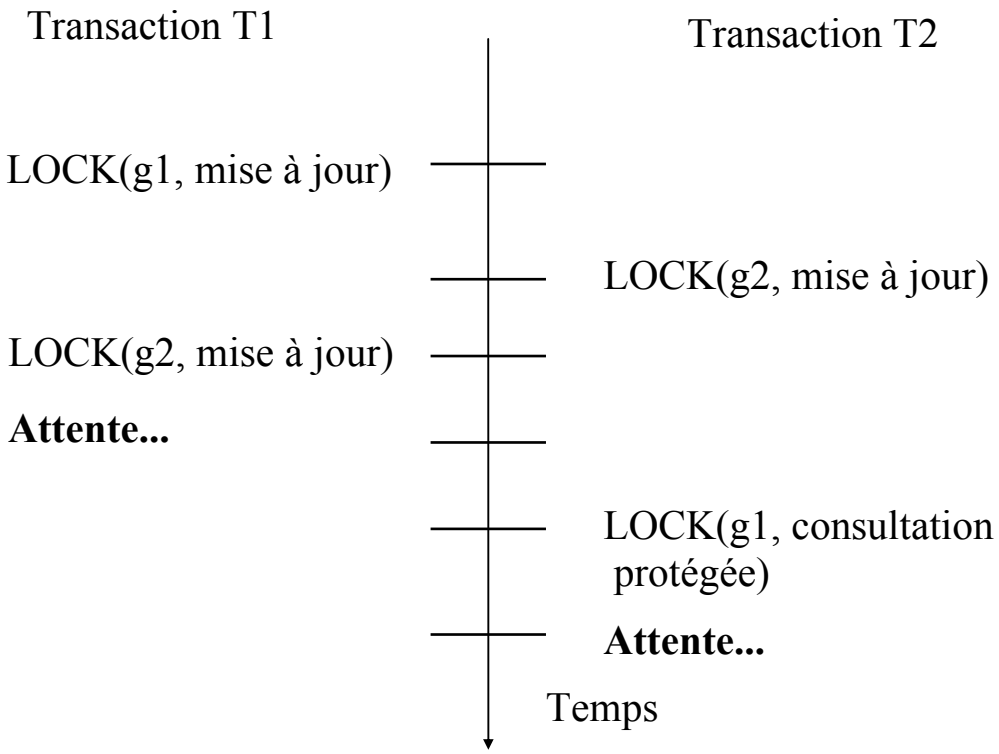
```
Procédure UNLOCK(g, M)
   /* on retranche M de la liste des verrous posés sur g */
   RetirerDansListeA(g, M);
   /* On essaye de débloquer toutes les transactions
    en attente de la granule g */
   Pour chaque (q, M') de Q[g] faire
      si  M '⊂ ¬ (¬ C * ⋃ A_i ( g ))
                        i ≠ p
      alors  AjouterDansListeA(M',g)
             Extraire (q, M') de Q[g];
             Débloquer la transaction T_q;
      finsi;
   finPour;

finproc UNLOCK
```

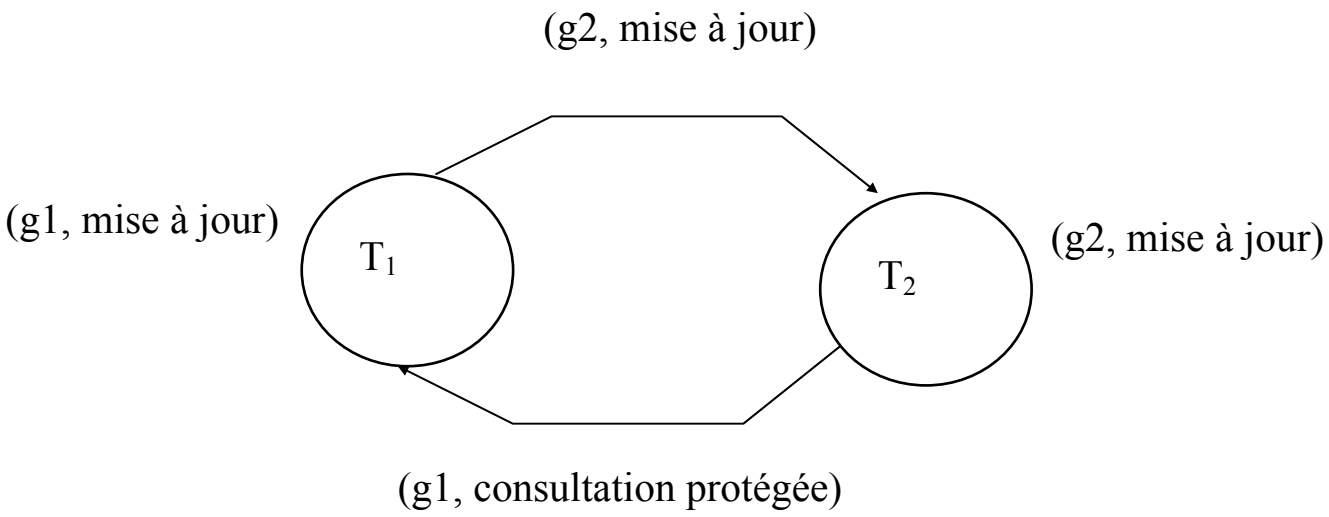
7.4.4 Problème du verrou mortel (deadlock)

Exemple de verrou mortel : La transaction T_1 a obtenu le verrouillage du granule g_1 et attend le verrouillage du granule g_2. La transaction T_2 a obtenu le verrouillage du granule g_2 et attend le verrouillage du granule g_1.

L'attente est infinie : il y a interblocage des transactions.



Le système peut détecter ces situations par analyse des graphes d'attente entre transactions (détection de circuit). Dans ce cas, l'une des transactions est défaite.



Afin de limiter les risques de verrou mortel, il est conseillé aux programmeurs de procéder au verrouillage des tables (explicite ou implicite) dans toutes les transactions en respectant toujours le même ordre.

7.5 Le contrôle de la concurrence avec Oracle

7.5.1 Niveaux d'isolation d'une transaction

Le niveau Read Commited est le niveau de transaction par défaut. Une requête voit les valeurs committées par les autres transactions avant le démarrage de l'exécution de la requête.

- Une requête ne lit jamais de données salies et non committées par une autre transaction.
- Une donnée peut avoir été modifiée par une autre transaction entre deux exécutions de la même requête. Il est donc possible qu'une lecture soit non répétable et que des données fantômes apparaissent.

Le niveau Read Only voit **seulement** les modifications committées avant le démarrage de la transaction. Ce niveau garantie donc les lectures consistantes.

- On ne peut pas faire d'Insert, Update et Delete dans une transaction en mode Read Only.

Le paramétrage du mode d'isolation se fait au niveau transaction :

```
set transaction read write; (par défaut)
set transaction read only;
```

7.5.2 Mécanisme de verrouillage

Pour gérer les conflits d'accès concurrents aux données, Oracle a mis en place un système **automatique** de contrôle par verrouillage. Le verrouillage est réalisé au **niveau ligne**.

Un verrou exclusif est automatiquement posé sur une donnée lorsque l'on exécute un ordre SQL SELECT ... FOR UPDATE, INSERT, UPDATE, DELETE.

- De cette manière, aucune autre transaction ne peut modifier cette donnée tant que le verrou est posé.
- La durée du verrou est celle de la transaction.
- Le verrou est relâché sur l'exécution d'un Commit ou d'un Rollback.

Oracle détecte automatiquement les situations de verrou mortel (deadlock). L'une des transactions participant au deadlock est défaite par Oracle.

7.5.3 Les différents types de verrous utilisés par Oracle

Les ordres SQL de manipulation de données peuvent acquérir des verrous à deux niveaux :

- niveau ligne (TX)
- niveau table (TM)

Lorsqu'une transaction acquière un verrou sur la ligne qu'elle veut modifier, elle obtient également un verrou sur la table toute entière.

- Ceci empêche toute autre transaction d'exécuter un ordre DDL (data description language) tant que le verrou n'est pas relâché.
- Par exemple, une autre transaction peut vouloir supprimer (DROP) la table ou en modifier sa structure (ALTER) alors que la modification apportée par la transaction n'est pas committée.

Les verrous sur tables sont de type :

- RS (Row Share) : il est posé sur la table par une transaction qui a posé des verrous sur des lignes dans l'intention de les modifier. Ce verrou empêche toute autre transaction de verrouiller la table en mode exclusif.

- S (Share) : ce verrou est posé manuellement par l'instruction LOCK TABLE table IN SHARE MODE;. Il empêche toute autre transaction de modifier le contenu de la table verrouillée. Seule la transaction qui a posé le verrou Share peut modifier le contenu de la table dans le cas où elle est la seule à détenir ce type de verrou. Dans le cas contraire, aucune des transactions détenant le verrou Share ne peut modifier le contenu de la table tant que les autres n'auront pas libérer leur verrou. Ce verrou ne garantit donc pas un accès exclusif en écriture.

- SRX (Share Mode Exclusif) : ce verrou est posé manuellement. Même effet que les verrous Share mais il ne peut être posé que par une seule transaction.

- RX (Row Exclusive) : ce verrou est en général posé sur la table par une transaction qui a réalisé une ou plusieurs mises à jour sur des lignes de la table. Il est plus restrictif que le verrou Row Share pour les autres transactions. Il empêche les verrouillages en mode Share (S) et Exclusive (X)

- X (Exclusive) : ce verrou ne peut être obtenu que par une seule transaction. Il lui confère un droit d'écriture exclusif dans la table. Il n'autorise aux autres transactions qu'un accès en lecture. Les ordres DDL de définition du schéma **drop table** et **alter table** posent implicitement un verrou X sur la table.

Les ordres DDL sont automatiquement commités.

Tableau 1 : récapitulatif des verrous table posés automatiquement à l'exécution d'un ordre SQL

Requête SQL	Type de verrou table posé	Modes de verrouillage compatibles				
		RS	RX	S	SRX	X
SELECT ... FROM table	aucun	oui	oui	oui	oui	oui
INSERT INTO table ...	RX	oui	oui	non	non	non
UPDATE table ...	RX	oui*	oui*	non	non	non
DELETE FROM table ...	RX	oui*	oui*	non	non	non
SELECT ... FROM table FOR UPDATE	RS	oui*	oui*	oui*	oui*	non
LOCK TABLE table IN ROW SHARE MODE	RS	oui	oui	oui	oui	non
LOCK TABLE table IN ROW EXCLUSIVE MODE	RX	oui	oui	non	non	non
LOCK TABLE table IN SHARE MODE	S	oui	non	oui	non	non
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX	oui	non	non	non	non
LOCK TABLE table IN EXCLUSIVE MODE	X	non	non	non	non	non

Remarque : les oui* représentent des verrouillages table compatibles. Mais, s'il y a conflit sur le verrouillage ligne, la transaction qui veut poser un verrou doit quand même attendre.

7.6 TD concurrence d'accès sous Access et Oracle

7.6.1 Stratégie par estampillage sous Access

Recopiez la base Access **concurre.mdb** (ainsi que **concurre.ldb**) dans votre répertoire de travail. Ouvrez cette base et ouvrez la table **Adhérent**. Depuis un autre poste de travail, créez une nouvelle base Access (Menu Fichier, Créer une nouvelle base). Attachez la table **Adhérent** de la base **concurre.mdb** dans cette nouvelle base (Menu Fichier, Attacher une table). Ouvrez également la table **Adhérent** depuis cette nouvelle base. Les deux postes sont en concurrence d'accès sur la table.

Modifiez la première ligne depuis l'un des deux postes sans valider la modification (ie sans changer de ligne). Modifiez cette ligne depuis le deuxième poste en validant cette fois-ci la modification. Depuis le premier poste, tentez maintenant de valider la modification. Constatez la réaction d'Access et testez les différentes alternatives proposées.

7.6.2 Stratégie par verrouillage sous Oracle

Comme le but de ce TD est d'étudier le comportement d'Oracle en cas d'accès concurrents aux mêmes données, vous allez ouvrir deux sessions sqlplus sous votre compte Oracle sur la base **Cinéma** que vous avez créée lors d'un précédent TD.

7.6.2.1 Consistance de la lecture, blocage

Exécutez la requête SQL suivante depuis la première session :

```
update acteur
set nb_films = 53 where nom='Depardieu' ;
```

Etudiez les conséquences du verrouillage réalisé par Oracle pour la session concurrente.

- L'autre session doit pouvoir **lire** les mêmes données. Elles contiennent les valeurs avant modification par la première session tant que ces modifications ne sont pas commitées. Refaites la lecture après avoir committé, vous devez voir cette fois-ci les modifications (mode de fonctionnement par défaut **Read Commited**).

- Si la deuxième session entreprend de modifier les mêmes données, la transaction sera bloquée en wait tant que la première session n'aura pas terminé ses modifications par un COMMIT ou un ROLLBACK.

Testez le mode de transaction en lecture consistante. Une session débute une transaction par l'instruction :

```
set transaction read only;
```

Puis, elle exécute une lecture. L'autre session modifie l'une des valeurs lues et valide (commit) sa modification. Si maintenant la première session exécute à nouveau la lecture, elle doit toujours voir l'ancienne valeur. Puis, après avoir exécuté un commit (changement de transaction), lorsqu'elle relance la lecture, elle voit la nouvelle valeur (mode par défaut : read commited).

7.6.2.2 Situation d'interblocage

Essayez de créer une situation de verrou mortel entre les deux sessions. Constatez qu'Oracle la détecte et tue la requête qui a démarré en dernier.

7.6.3 Verrouillages ligne et table sous Oracle

Indiquez dans la colonne **Commentaire** le comportement constaté (instruction bloquée, défaite, exécutée, ...) et la raison associée (compatibilité entre les verrous).

Transaction 1	No Ordre SQL	Transaction 2	Commentaire
lock table acteur in row share mode;	1		
	2	drop table acteur;	
	3	lock table acteur in exclusive mode nowait;	
	4	select nb_films from acteur where nom= 'Depardieu' for update;	
update acteur set nb_films=60 where nom= 'Depardieu';	5		
	6	rollback;	
rollback;	7		
lock table acteur in row exclusive mode;	8		
	9	lock table acteur in exclusive mode nowait;	
	10	alter table acteur add age smallint;	
	11	lock table acteur in share row exclusive mode nowait;	
	12	lock table acteur in share mode nowait;	
	13	update acteur set nb_films=60 where nom= 'Depardieu';	
rollback;	14		
	15	rollback;	
lock table acteur in share mode;	16		

	17	lock table acteur in exclusive mode nowait;	
	18	lock table acteur in share row exclusive mode nowait;	
	19	lock table acteur in row exclusive mode nowait;	
	20	lock table acteur in share mode;	
	21	select nb_films from acteur where nom= ‘Depardieu’ for update;	
	22	update acteur set nb_films=60 where nom= ‘Depardieu’;	
rollback;	23		
	24	rollback;	
lock table acteur in share row exclusive mode;	25		
	26	lock table acteur in exclusive mode nowait;	
	27	lock table acteur in share row exclusive mode nowait;	
	28	lock table acteur in share mode nowait;	
	29	lock table acteur in row exclusive mode nowait;	
	30	lock table acteur in row share mode;	
	31	select nb_films from acteur where nom= ‘Depardieu’ for update;	
	32	update acteur set nb_films=60 where nom= ‘Depardieu’;	
update acteur set nb_films=60 where nom= ‘Depardieu’;	33		
	34	commit;	
rollback;	35		

lock table acteur in exclusive mode;	36		
	37	lock table acteur in exclusive mode nowait;	
	38	lock table acteur in share row exclusive mode nowait;	
	39	lock table acteur in share mode nowait;	
	40	lock table acteur in row exclusive mode nowait;	
	41	lock table acteur in row share mode nowait;	
	42	select nb_films from acteur where nom= ‘Depardieu’;	
	43	select nb_films from acteur where nom= ‘Depardieu’ for update;	
update acteur set nb_films=60 where nom= ‘Depardieu’;	44		
commit;	45		

8. L'évaluation et l'optimisation des requêtes

L'objectif de ce chapitre est d'étudier le mécanisme d'évaluation d'une requête SQL et de chercher à optimiser son temps d'exécution (pour améliorer les performances de la requête et aussi les performances globales du SGBD).

8.1 Les structures de données pour optimiser les accès

On commence par présenter les structures de données disponibles sous Oracle qui permettent en principe de rendre plus rapide les accès aux données.

- Index de type arbre B
- Organisation de tables en cluster index
- Organisation d'une table en hash cluster
- Index bitmap

8.1.1 Les index de type arbre B

8.1.1.1 Principe

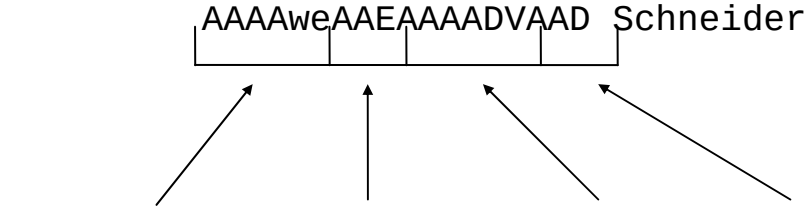
Lorsque l'on construit un index sur une colonne d'une table, on associe à chaque valeur de cette colonne la liste des **adresses** des lignes de la table qui contiennent cette valeur. L'adresse d'une ligne s'appelle le **rowid**.

8.1.1.2 Définition du ROWID

C'est une pseudo-colonne qui permet d'identifier la position d'une ligne dans l'un des fichiers qui constituent la base de données.

```
SQL> select rowid, nom from acteur;

ROWID              NOM
------------------ --------------------
AAAAweAAEAAAADVAAG Allen
AAAAweAAEAAAADVAAF Bourvil
AAAAweAAEAAAADVAAA Coluche
AAAAweAAEAAAADVAAE De Funes
AAAAweAAEAAAADVAAB Depardieu
AAAAweAAEAAAADVAAH Marceau
AAAAweAAEAAAADVAAC Richard
AAAAweAAEAAAADVAAD Schneider
```



No segment dans le tablespace (6 positions)

No de fichier associé au tablespace (3 positions)

No de bloc dans le fichier (6 positions)

No de ligne dans le bloc (3 positions)

Le ROWID est codé en base 64. Les caractères de codage sont : A-Z, a-z, 0-9, + et /.

Un tablespace est un espace logique de stockage. Il est représenté physiquement par un ou plusieurs fichiers du système d'exploitation. Un fichier est décomposé en blocs.

Un tablespace est découpé logiquement en segments. Chaque segment correspond à un objet logique (une table, un index, …).

La notion de ROWID fait donc référence à la fois à des notions logiques et physiques.

8.1.1.3 Recherche par index

L'accès à une ligne se fait en :

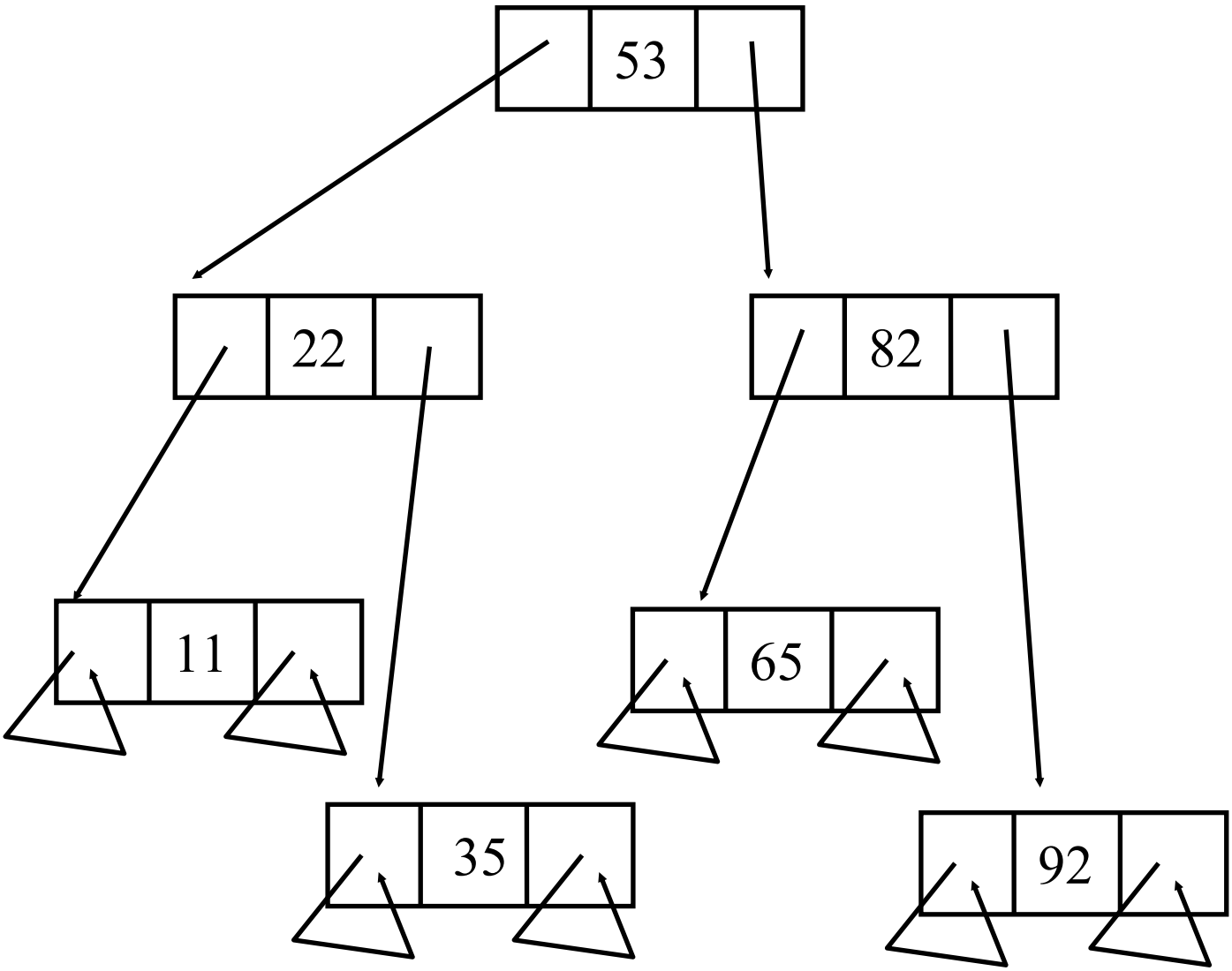
1. recherchant la valeur dans l'index (index lookup)
2. accèdant à la ou les lignes dans la table par le rowid

L'index est stocké dans une structure de données indépendante de la table.

L'index est trié : il autorise la recherche dichotomique.

8.1.1.4 Organisation de l'index en arbre balancé

- Organisation sous forme de tableau en mémoire centrale pas possible si l'index est de taille importante.
- Utilisation d'une structure arborescente, l'index hiérarchisé ou index multi-niveaux
- Exemple : un arbre binaire contenant des valeurs de clé primaire



Après insertion ou suppression de noeud, la recherche reste efficace si l'arbre est équilibré, ie tout chemin de la racine vers n'importe quelle feuille a la même longueur.

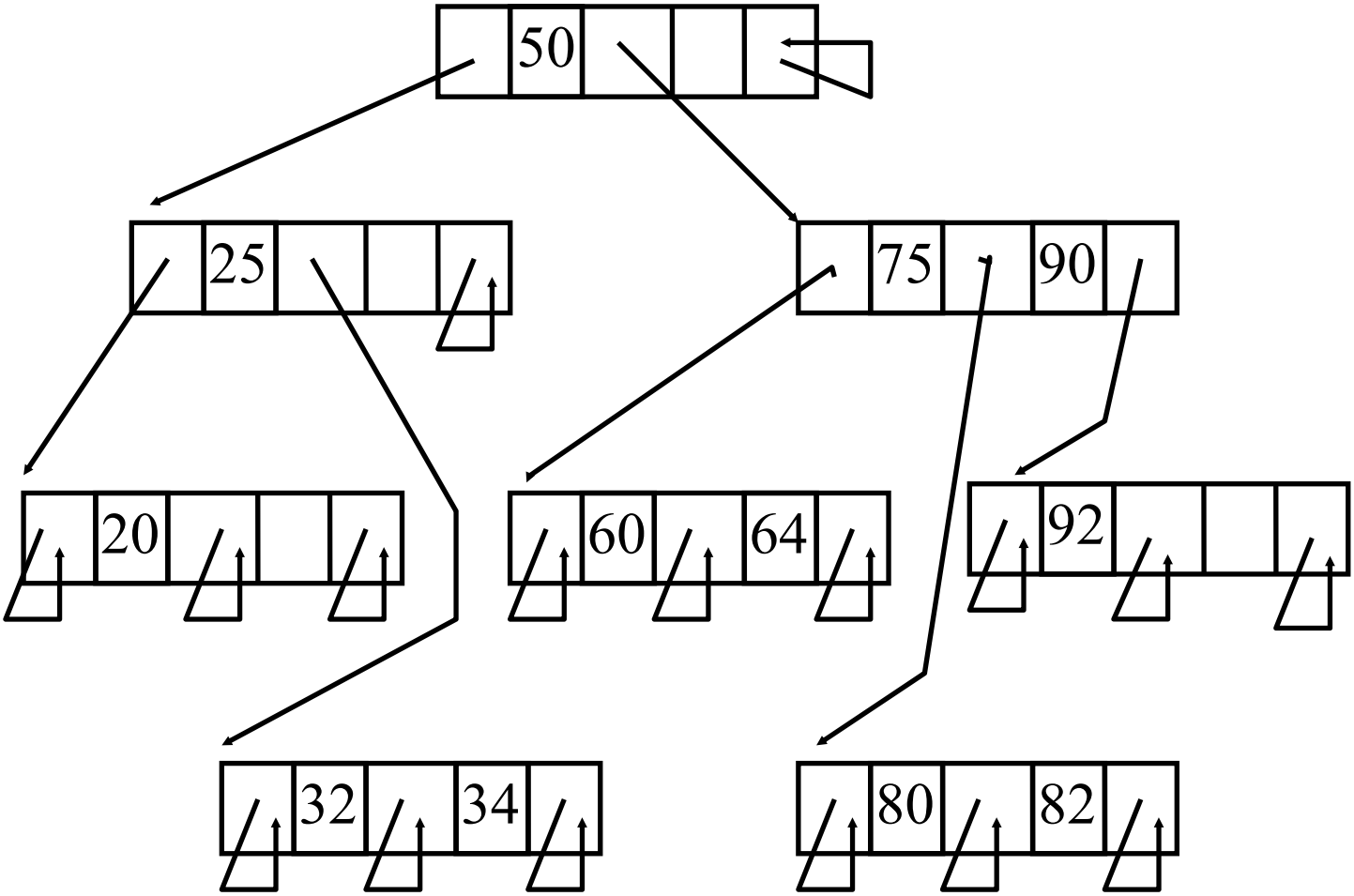
Définition : un B-arbre d'ordre d est un arbre équilibré avec pour chaque noeud :

- un nombre de clés compris entre d et 2d
- un nombre de pointeurs compris entre d+1 et 2d+1
- la taille d'un noeud correspond à la taille maximale de l'unité de transfert entre mémoire secondaire et mémoire centrale.

La hauteur h d'un B-arbre d'ordre d indexant une table de N lignes est égale à $\log_d N$.

- A chaque niveau, dichotomie de l'espace de recherche en d+1 sous-espaces égaux : $N = d^h$
- Le nombre d'accès disque est logarithmique par rapport à la taille de la table.
- Exemple : si d = 4 et nombre de tuples = 500000 lignes, la recherche se fait sur les 500000 lignes, puis 100000 lignes, 20000 lignes, etc …

Exemple : un B-arbre d'ordre d = 1



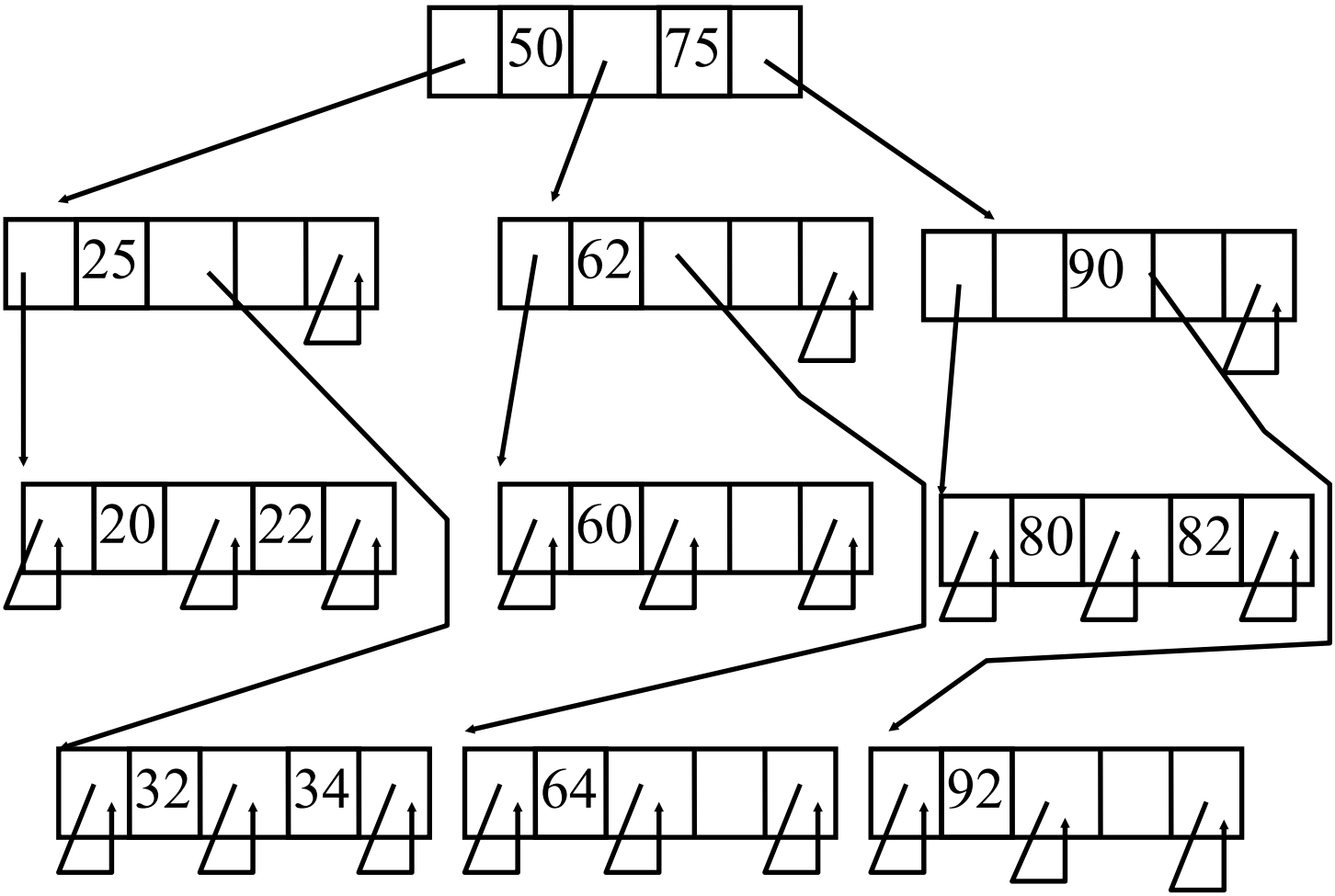
Manipulation d'un B-arbre

Recherche :

- Elle débute toujours à partir de la racine
- On compare la valeur de clé recherchée (CR) aux valeurs dans le noeud courant n (C_{n_j}) considérées de gauche à droite. Si CR < C_{nj}, on descend dans le sous-arbre gauche associé à C_{nj}, si $C_{nj} \leq CR < C_{n,j+1}$ alors on descend dans le sous-arbre droit de C_{nj} (ou le sous-arbre gauche de $C_{n,j+1}$), sinon on compare à $C_{n,j+2}$ ...

Insertion : 2 cas possibles

- Si noeud non saturé : insertion de la clé 22
- Si noeud saturé : insertion de la clé 62, éclatement du noeud (60, 64) en 2 noeuds, insertion de 62 dans le noeud père, éclatement de (75, 90), insertion de 75 dans le noeud père.



remarque : la structure reste équilibrée (3 niveaux)

Suppression : il y a deux problèmes à considérer

- **Problème 1** : l'emplacement de la clé dans le B-arbre

  Si le noeud est une feuille : on effectue la suppression (exemple : suppression de 34)

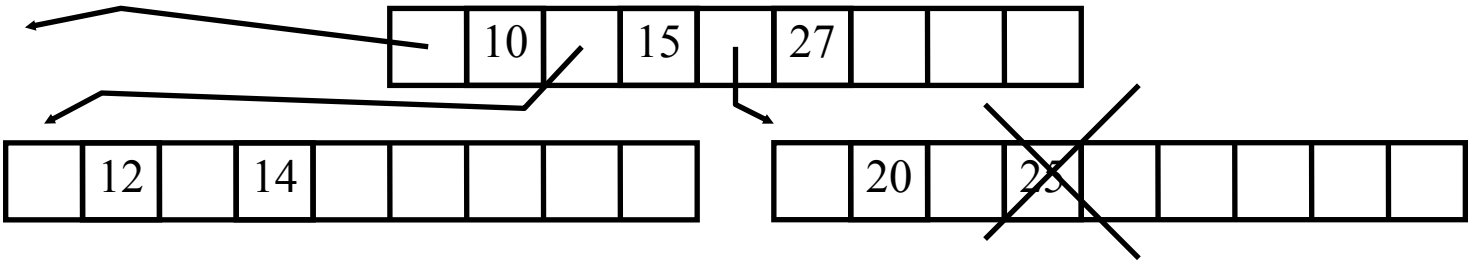
  Sinon on remplace la clé supprimée par celle située dans la feuille la plus à gauche dans le sous-arbre droit (exemple : suppression de 75, remplacé par 80)

- **Problème 2** : la suppression provoque un sous-remplissage (ie nombre de clés < d)

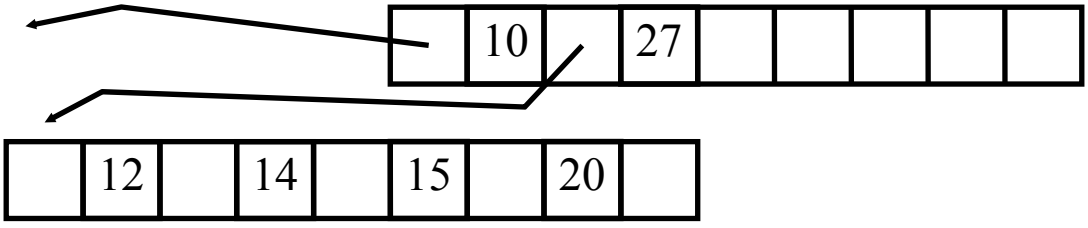
  Si le taux de remplissage du noeud voisin inférieur à 50 %, on effectue une concaténation

  Sinon on effectue une redistribution égale entre les 2 voisins

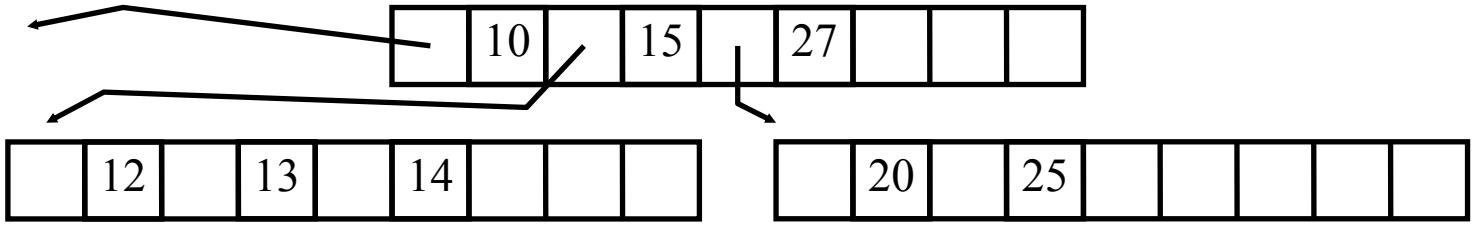
Exemple 1 : soit un B-arbre d'ordre 2



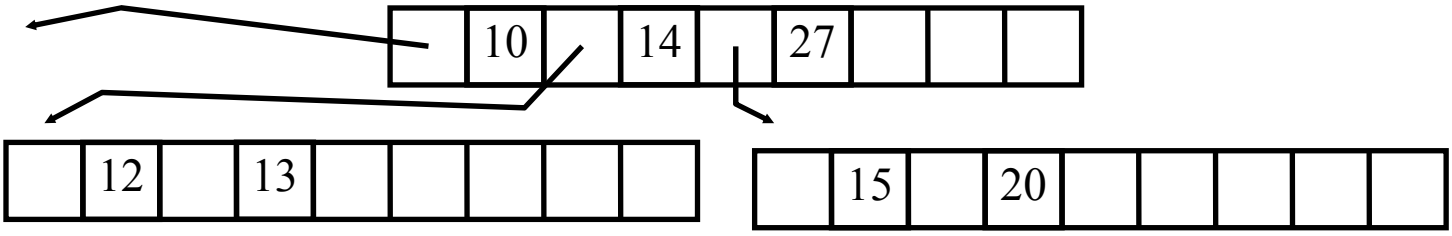
on supprime 25 et on applique le **si**



Exemple 2 :



on supprime 25 et on applique le sinon



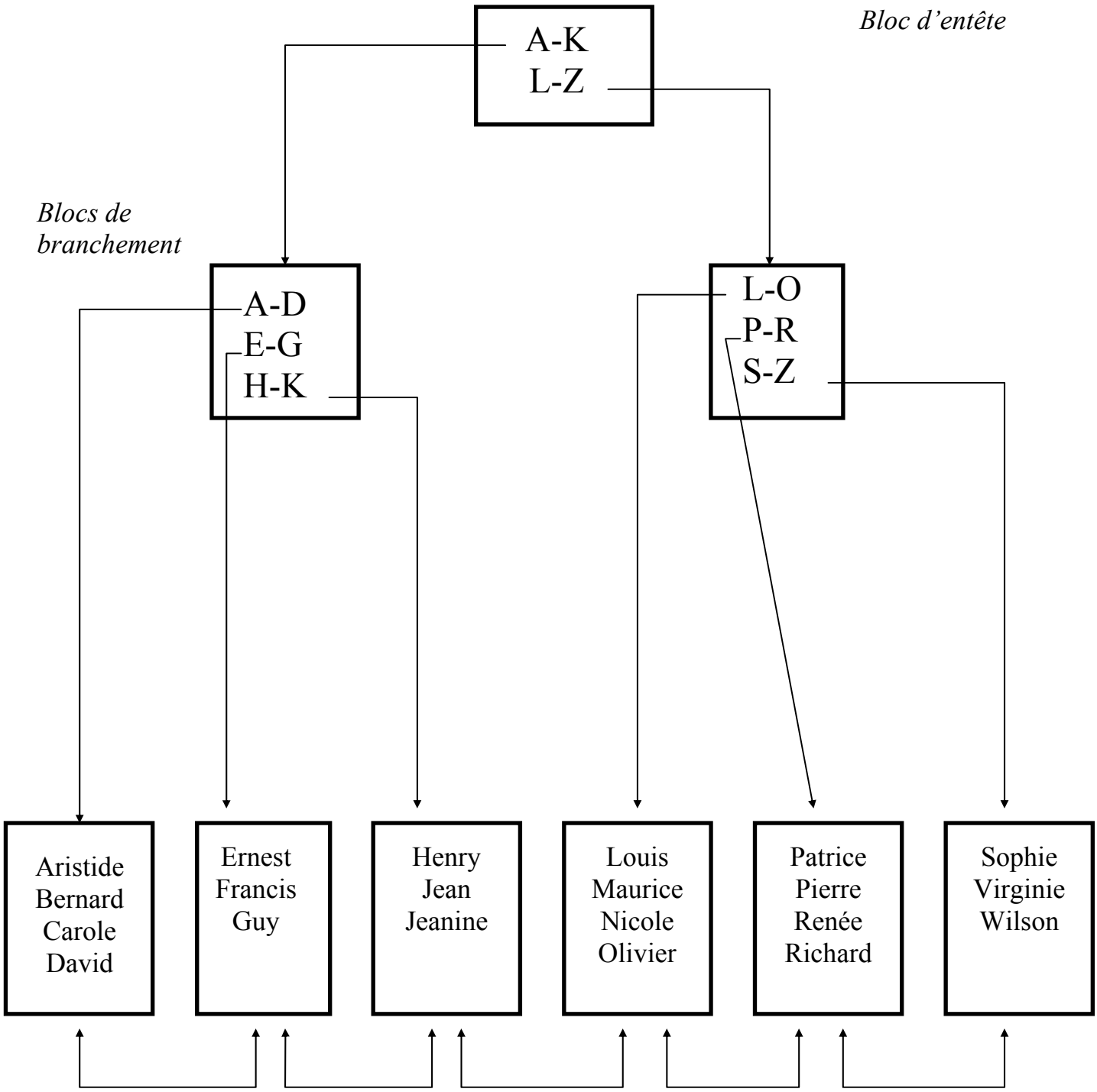
8.1.1.5 Conclusion sur les organisations indexées

Coût d'accès (E/S) Nombre de valeurs par noeud (d)	Taille de la table (N) 10^3 10^4 10^5 10^6 10^7
10	3 4 5 6 7
50	2 3 3 4 4
100	2 2 3 3 4

Exemples d'utilisation :

- Arbre balancé pour réaliser un index dense (associations valeur de clé, adresse relative de l'article dans le fichier séquentiel des données) : méthode indexée IS3 sur les AS 400 d'IBM
- Arbre balancé pour réaliser l'index et le fichier des données (appelé arbre B+) tel que :
  - dans les niveaux internes de l'arbre, on a un index non dense (qui permet un accès direct)
  - dans les feuilles (par ordre croissant, double-chaîné), on a un index dense et le contenu des lignes (qui permet un accès séquentiel trié)

méthodes séquentielles indexées ISAM et VSAM d'IBM (utilisée par DB2 et SQL/DS), UFAS de BULL, index B-Tree d'Oracle (sans le contenu des lignes dans les feuilles).



- *Les blocs feuille sont liés dans les deux directions*
- *Dans le bloc feuille, Oracle stocke la valeur et la liste des rowids correspondant dans la table*

8.1.1.6 Conclusion sur les B-arbres d'Oracle :

- La performance d'accès à l'index est constante grâce à la structure d'arbre équilibré. Tous les blocs feuille se trouvent à la même profondeur dans l'arbre.
- La performance d'accès à une donnée est bonne pour les grandes tables. En moyenne, on accède à une donnée en 4 opérations d'E/S.
- L'index peut être utilisé à la fois pour les recherches exactes (prédicat d'égalité) mais aussi pour les recherches sur des intervalles de valeurs (between, <, >).

8.1.1.7 Index implicites

Oracle crée automatiquement un index pour les contraintes de type *primary key* et *unique* spécifiées lors de la création des tables.

8.1.1.8 Index concaténés

Ce sont les index qui comprennent plus d'une colonne. Ils sont plus sélectifs que les index mono-colonne. Lorsque l'on fait souvent des interrogations sur plusieurs colonnes (clause ***where*** des requêtes), il est très intéressant de créer ce genre d'index. Par exemple :

Dans Employe(Id_emp, Nom_emp, Prenom_emp, age, ...), si on interroge souvent simultanément sur nom et prénom, on crée un index concaténé sur nom et prénom.

```
create index nom_prenom_idx
on Employe(nom_emp, prenom_emp)
```

Une requête peut utiliser un index concaténé si parmi les colonnes interrogées dans la clause **where** Oracle trouve les premières colonnes de l'index. Par exemple,

- si l'on interroge sur nom, prénom, âge : Oracle utilise l'index sur nom, prénom,
- si l'on interroge sur nom, âge : Oracle utilise l'index sur nom,
- si l'on interroge sur prénom, âge : Oracle n'utilise pas l'index.

8.1.1.9 Fusion d'index

Si dans une requête, on interroge sur nom, prénom et si il existe un index sur nom et un index sur prénom, alors Oracle réalise une fusion d'index.

- Il interroge par les deux index et détermine ensuite l'intersection des deux listes de **Rowids** obtenus par les index.
- Lorsque l'on trouve cette opération dans un plan d'exécution d'une requête à optimiser (opération AND EQUALS), une bonne solution consiste à créer un index concaténé (accès à un seul index et pas de calcul d'intersection).

8.1.1.10 Valeurs NULL

Elles ne sont pas stockées dans les index. Une interrogation sur NULL sera donc toujours résolue par un balayage séquentiel de la table.

8.1.1.11 Index sur clé étrangère

A la différence des contraintes de clé primaire, Oracle ne crée pas automatiquement d'index pour vérifier les contraintes d'intégrité référentielle (par exemple entre la table Employé et la table Département sur la colonne IdDépartement). Il est donc préférable de créer systématiquement un index sur chaque colonne clé étrangère pour optimiser les contrôles d'intégrité référentielle (réalisés par select implicite, par exemple) et éventuellement les jointures.

8.1.2 Les clusters index

Dans un cluster index, deux tables sont stockées physiquement dans les blocs de données directement en jointure.

Cette organisation des données est en pratique très rarement utilisée car les autres types d'opérations sur les tables sont pénalisées par l'organisation même (balayage d'une seule table, insertion de données dans le cluster).

8.1.3 Les hash-clusters

8.1.3.1 Principe

- La table est de taille constante fixée à la création
- Elle est divisée une fois pour toute en p blocs de taille fixe L
- Un bloc contient plusieurs lignes
- Un bloc est lu en une opération d'E/S
- A partir de la clé de hashage, la méthode calcule le numéro du bloc N (bucket) dans lequel est placée la ligne
- L'adresse relative AR = N * L

La fonction de calcul est appelée **fonction de hachage**

8.1.3.2 Choix d'une fonction de hachage

- Distribuer de manière uniforme les lignes dans les paquets
- Réduire le nombre de collisions

Technique de pliage : on choisit et combine des bits de la clé

Exemple : soit une table stockée dans 16 paquets

clé 155 → (base 2) 1001 1011

→ (pliage) 1001 XOR 1011

adresse (155) = 0010 → 2

clé 34 → 0010 XOR 0010

adresse (34) = 0000 → 0

Technique de conversion de la clé en nombre entier

Exemple :

clé 155 → 1 + 5 + 5 = 11

clé 34 → 7

clé 3455 → 17

Technique du modulo : fonction la plus utilisée (optimale par rapport aux deux critères énoncés)

Exemple :

155 mod 16 → 11

34 mod 16 → 2

3455 mod 16 → 15

Remarque : pour les clés alphanumériques, on se ramène à une valeur numérique (table ASCII)

8.1.3.3 Gestion des collisions

- Les lignes sont stockées de manière séquentielle dans les blocs.
- A l'intérieur d'un bloc, une ligne est accédée par balayage séquentiel.
- S'il s'agit d'une insertion et d'une clé de hachage correspondant à une clé primaire, il y a balayage de l'intégralité du bloc pour contrôler l'unicité de la valeur de la clé et la ligne est stockée à la première place libre dans le bloc.
- Quand le paquet est plein en insertion : mise en oeuvre d'une technique de débordement

8.1.3.4 Techniques de débordement possibles

- On applique une deuxième fonction de hachage (rehachage) pour placer les lignes dans des blocs de débordement
- On établit un lien de chaînage entre le bloc plein et un bloc de débordement, ce qui constitue un bloc logique

8.1.3.5 Hachage dynamique

La chaîne de bits résultat de l'application de la fonction de hachage à la clé est exploitée progressivement bit par bit au fur et à mesure des extensions de la table.

Exemple : Le hachage extensible

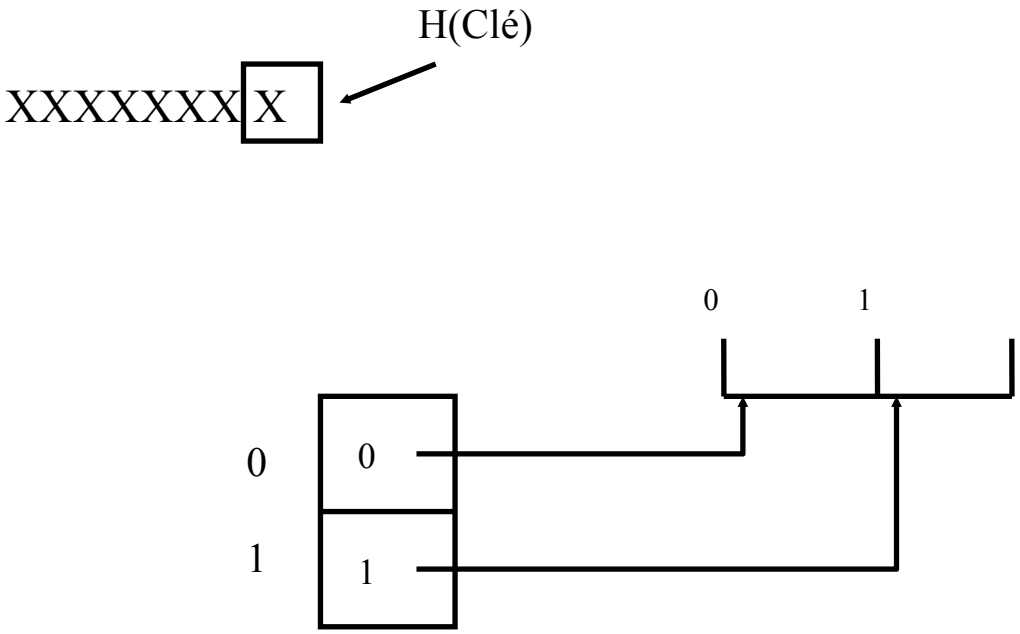
1. La table est étendue à chaque fois qu'un bloc est plein; un nouveau bloc est ajouté à la table.

2. On considère un bit supplémentaire de la valeur de clé hachée. Les valeurs du bloc plein sont répartis dans les deux blocs (le plein et le nouveau) en fonction de la valeur du nouveau bit considéré (0 ou 1).

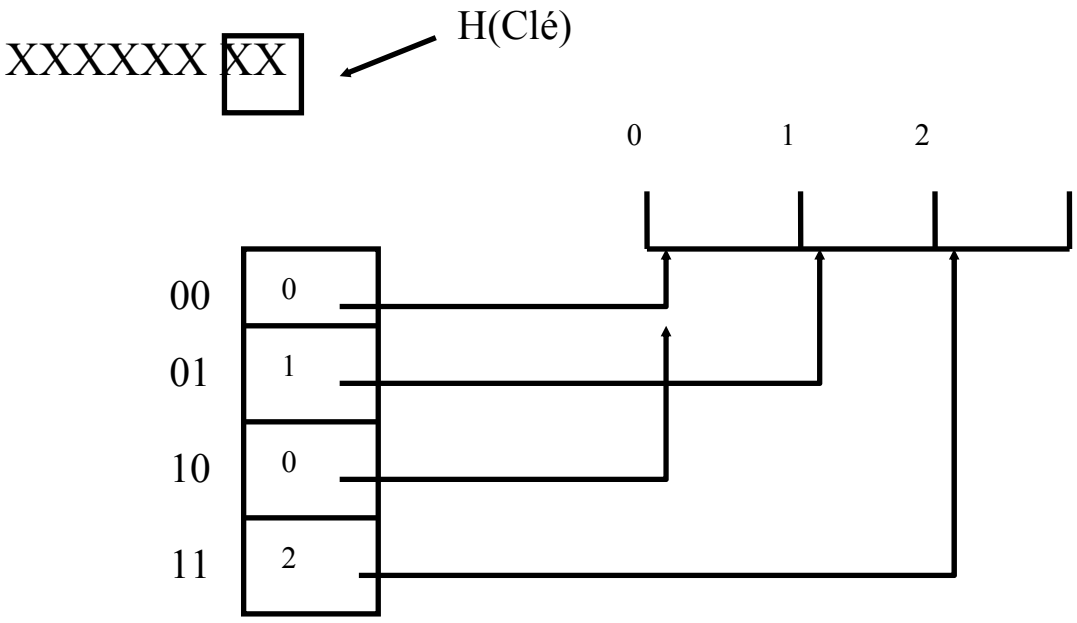
3. Les adresses (sur le disque) des blocs de la table sont stockées dans un répertoire accédé en utilisant les M premiers bits pris en compte de la clé hachée.

4. Il n'y a plus besoin de gérer de bloc de débordement

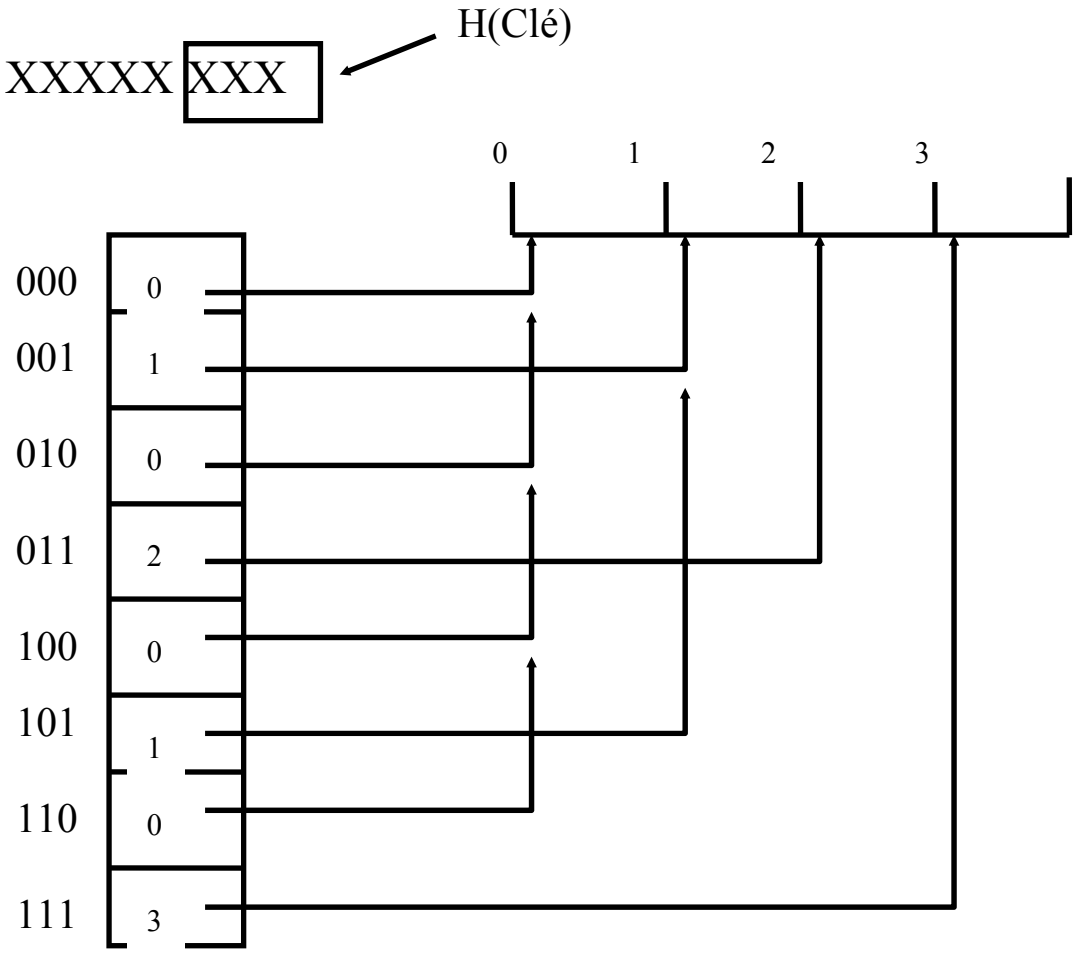
Exemple : Une table est créée avec deux blocs et adressée par le premier bit de la fonction de hachage (celui de droite)



S'il est plein, le paquet 1 éclate et les lignes sont distribuées dans les paquets 01 et 11 :



Le paquet 11 éclate à son tour et les lignes sont distribuées dans les blocs 011 et 111 :


H(Clé)
XXXXX XXX
0
1
2
3
000
001
010
011
100
101
110
111
0
1
0
2
0
1
0
3

8.1.3.6 Conclusion sur le hachage :

Avantages :

- Adaptation aux tables de clé quelconque. Le hachage est souvent opéré sur la clé de la table qui garantit par définition une répartition uniforme.
- Bonne performance tant qu'il n'y a pas de débordement (1 E/S en lecture du paquet, 2 E/S en écriture)

  ⇨ *Bien adapté pour les fichiers peu volatiles*

Inconvénients :

- Taux d'occupation de la table peut être éloigné de 100 %
- Dégradation des performances d'accès en cas de débordement
- Taille de la table doit être fixée à priori (contrainte levée par le hachage dynamique). Le hash cluster d'Oracle n'est pas basé sur un hachage dynamique.
- Si le nombre de lignes de la table devient plus important que prévu, la table doit être réorganisée.
- Gros inconvénient : pas de possibilité efficace d'accès ordonné (sur la valeur de clé) ou d'interrogation sur une plage de valeurs.

8.1.4 Les index bitmap

On associe à chaque valeur de la colonne indexée une chaîne de bits dont la longueur correspond au nombre total de lignes de la table.

- Si le ième bit vaut 0, cela signifie que la ligne i ne porte pas cette valeur,
- Si le ième bit vaut 1, cela signifie que la ligne i porte cette valeur.

Intéressant pour :

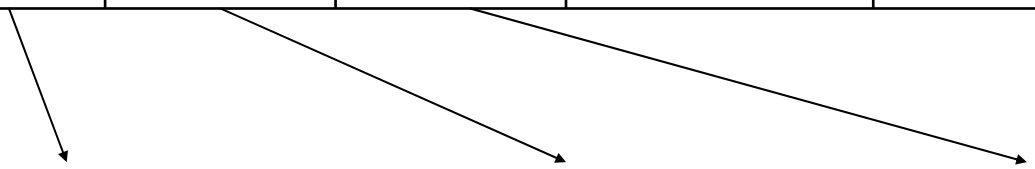
- des colonnes à faible cardinalité (car il y aura peu de lignes dans l'index),
- en interrogation multi-critères (résoudre une requête revient à combiner des bits par des **and** et des **or** dans les index bitmap).

Pas intéressant pour :

- les tables très souvent mises à jour en transactionnel (le verrouillage de l'index étant défini au niveau bloc, quand un bloc est verrouillé, beaucoup de lignes de la table sont verrouillées),
- les tables interrogées sur des intervalles de valeurs,
- les colonnes à cardinalité élevée.

Exemple : la table Personne (IdPersonne, Nom, Prénom, sexe, EtatCivil, EstPropriétaire, ADesEnfants, ...)

Sexe	EtatCivil	ADesEnfants	EstPropriétaire	...
M	Célibataire	oui	oui	
F	Mariée	non	non	
M	Divorcé	non	non	
F	Célibataire	non	oui	



M	F
1	0
0	1
1	0
0	1

Célib.	Marié	Divorcé
1	0	0
0	1	0
0	0	1
1	0	0

Oui	Non
1	0
0	1
0	1
0	1

Select nom from Personne
where **sexe='M'** and **EtatCivil='Divorcé'** and **ADesEnfants='N'** ;

M
1
0
1
0

AND

Divorcé
0
0
1
0

AND

Non
0
1
1
1

→

Résul
0
0
1
0

8.2 Vision globale sur l'exécution d'une requête SQL

Comment Oracle analyse une requête SQL ?

Comment Oracle exécute une requête SQL ?

Comment Oracle optimise l'exécution d'une requête SQL ?

Comment le programmeur peut intervenir sur l'optimisation de l'exécution de la requête ?

8.2.1 Algorithme global d'exécution d'une requête

8.2.1.1 Définition d'un curseur

Un **curseur** est une zone de mémoire dans laquelle Oracle stocke la requête et les informations qui lui permettent de l'exécuter, à savoir :

- la requête analysée,
- le plan d'exécution,
- un pointeur sur la ligne courante.

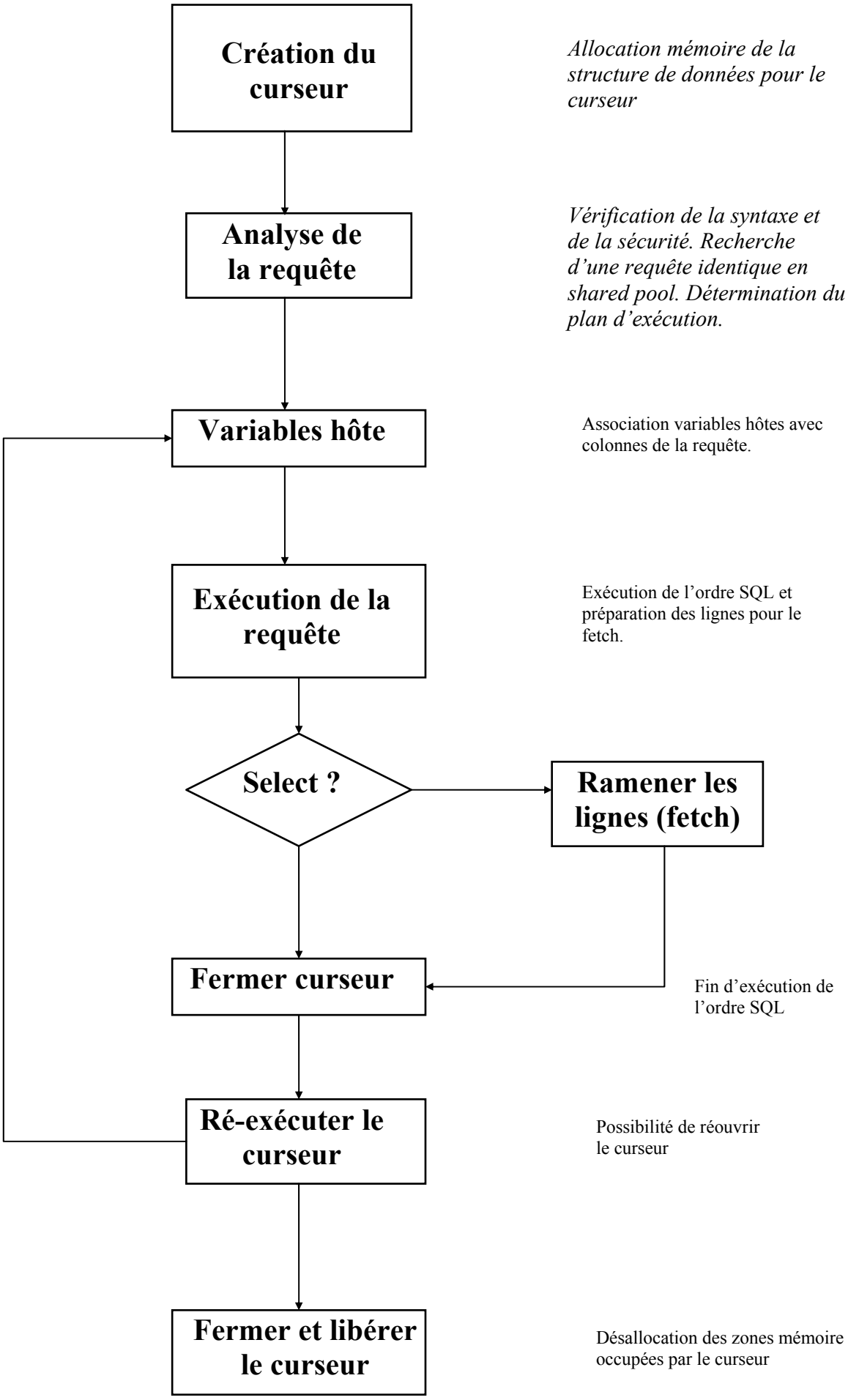
Remarque : Selon l'interface utilisée, la gestion du curseur est plus ou moins visible. En programmation PL/SQL, l'allocation du curseur est masquée. Dans les interfaces avec les langages de programmation, le programmeur doit créer, manipuler et détruire explicitement les curseurs (exemple : PRO*C).

L'analyse de la requête est analogue à la phase de compilation ou d'interprétation d'un langage de programmation.

La requête SQL est traduite en plan d'exécution qui met en oeuvre les algorithmes implémentés dans le moteur du SGBD (algorithmes d'accès aux éléments d'une table, algorithmes de jointure, ...).

L'analyse de la requête se décompose en 4 étapes :

1. **l'analyse syntaxique** vérifie que la requête SQL est valide d'un point de vue syntaxique (analyse de la conformité à la grammaire du langage),
2. **l'analyse sémantique** vérifie que les objets référencés dans la requête sont bien des objets présents dans la base de données (tables, colonnes, fonctions, ...),
3. la vérification que l'ordre SQL a **le droit d'être exécuté** (droit conféré par le DBA ou le propriétaire des objets référencés),
4. la détermination du **plan d'exécution** de la requête, i.e. la liste des algorithmes utilisés pour accéder et/ou mettre à jour les données.


Création du curseur
Allocation mémoire de la structure de données pour le curseur
Analyse de la requête
Vérification de la syntaxe et de la sécurité. Recherche d'une requête identique en shared pool. Détermination du plan d'exécution.
Variables hôte
Association variables hôtes avec colonnes de la requête.
Exécution de la requête
Exécution de l'ordre SQL et préparation des lignes pour le fetch.
Select ?
Ramener les lignes (fetch)
Fermer curseur
Fin d'exécution de l'ordre SQL
Ré-exécuter le curseur
Possibilité de réouvrir le curseur
Fermer et libérer le curseur
Désallocation des zones mémoire occupées par le curseur

8.2.2 La shared SQL et les variables hôtes

Principe : Oracle cherche à éviter de réanalyser des requêtes souvent exécutées. Il a implémenté pour cela un cache dans lequel il mémorise les requêtes récemment exécutées. Oracle stocke dans la **shared pool** de la **SQL Area** le texte analysé de la requête et le plan d'exécution associé.

Implémentation : Oracle utilise un algorithme de hashage qui calcule à partir du texte de la requête un entier. Cette valeur lui permet de retrouver la requête dans la shared pool.

Pour bénéficier de cette fonctionnalité, il faut que les requêtes soient écrites de manière identique (mêmes alias, même nombre d'espaces notamment, mêmes valeurs constantes dans les clauses **where** et **having,** mêmes noms de variables hôtes).

Il faut utiliser systématiquement des variables hôtes plutôt que des valeurs constantes pour éviter de refaire l'analyse de la requête.

Par exemple, en PL/SQL :

```
select nb_roles_titre from acteur where
nom='Depardieu' ;
```

devient

```
nom_acteur acteur.nom%TYPE :='Depardieu' ;
begin
select nb_roles_titre from acteur where
nom= nom_acteur;
end
```

8.2.3 Les algorithmes d'exécution

Oracle dispose de plusieurs algorithmes pour accéder les données stockées dans les tables de la base et pour joindre les tables.

8.2.3.1 Techniques d'accès aux données dans une table

Accès séquentiel à la table (**full table scan**)

- Oracle lit séquentiellement bloc par bloc les données stockées dans la table.
- Il lit du premier bloc jusqu'à la marque de fin de table (high water mark).
- Le high water mark est le plus grand numéro de bloc qui a contenu des données **depuis la création de la table**. Il ne diminue jamais. La seule façon de le faire baisser est de recréer la table (par un export/import par exemple).

Accès par ROWID

- Le ROWID est une pseudo-colonne (i.e elle peut être référencée dans un **select** mais ne fait pas partie de la table).
- Elle contient la localisation physique de la ligne dans la base de données. C'est la technique la plus rapide pour accéder à une ligne dans la base de données.
- Cette technique d'accès est utilisée dans les index et dans les tables organisées selon une clé de hashage.

Accès par un index (**index lookup**)

- Oracle recherche dans l'index les lignes vérifiant un critère sur une ou plusieurs colonnes.
- Pour cela, Oracle mémorise dans l'index l'ensemble des valeurs portées par la (ou les) colonne(s) sur laquelle est construit l'index avec en association la liste des lignes (représentées par leur ROWID) où on les trouve dans la table.
- Oracle propose deux types de structure de données pour construire les index : les B-arbres et les index bitmap.

Accès par clé de hashage (**hash key lookup**)

- Un hash cluster est une table organisée selon une technique de hashage.
- Oracle utilise une fonction mathématique qui transforme la valeur d'une colonne en une valeur numérique.
- Cette valeur calculée permet de déterminer dans quel bloc de données est stockée la ligne .

8.2.3.2 Techniques de jointure entre les tables

Oracle utilise trois techniques de jointure :

- algorithme de tri-fusion,
- algorithme des boucles imbriquées,
- algorithme de hashage.

Algorithme tri-fusion (**sort merge join**)

Il n'utilise pas d'index. Il trie les deux tables sur la ou les colonne(s) de jointure. Puis, il les fusionne.

Algorithme des boucles imbriquées (**nested loops join**)

- Il effectue un balayage séquentiel de l'une des deux tables.
- Pour chaque ligne balayée, il recherche dans l'autre table la ou les lignes qui lui correspondent selon le critère de jointure.
- L'accès à la deuxième table est réalisé en général via un index.

Algorithme de hashage (Hash Join)

- Une table de hashage est construite sur la plus grosse des deux tables.
- La clé de hashage correspond au critère de jointure.
- Il balaye séquentiellement la plus petite des deux tables et recherche via la table de hashage les lignes correspondantes dans la grosse table.

8.2.4 Les processus d'optimisation

Pour la plupart des requêtes, il y a plusieurs manières de ramener les lignes. ***L'optimiseur*** doit choisir l'approche qui lui paraît être la plus rapide. Il est activé pour tous les ordres SQL qui incluent une interrogation ou au moins une clause ***where***.

Oracle offre deux approches pour l'optimisation :

- L'optimiseur basé sur les règles,
- L'optimiseur basé sur les coûts.

L'optimiseur basé sur les règles :

- C'est le plus ancien des deux,
- il base sa décision sur un ensemble de règles et un ordonnancement des techniques d'accès aux données,
- **il ne tient pas compte des volumes de données** stockées dans les tables.

☞ Par exemple, il privilégie toujours un accès par index à un balayage séquentiel de la table : il ne distingue donc pas un balayage séquentiel d'une table de deux lignes d'un balayage d'une table de 2 millions de lignes.

L'optimiseur basé sur les coûts :

- introduit dans la version 7 d'Oracle,
- il reprend l'acquis de l'optimiseur règles, mais il est capable en plus de prendre en compte **des statistiques sur le volume et la distribution des données** dans les tables,
- il est donc capable de distinguer un balayage séquentiel d'une table de deux lignes d'un balayage d'une table de 2 millions de lignes.

8.2.4.1 Notion de plan d'exécution

Le plan d'exécution d'une requête est la traduction de l'ordre SQL par définition **déclaratif** en un algorithme procédural qui permet l'accés aux données. Plus précisément, le plan d'exécution détermine l'ordre dans lequel les tables sont accédées, la technique d'accès aux données dans les tables, l'algorithme de jointure utilisé pour joindre les tables.

8.2.4.2 Notion de trace d'exécution

La trace d'exécution d'un ordre SQL indique pour chaque étape (c'est à dire pour chaque opération de base réalisée) les ressources mises en œuvre (le nombre de lignes traitées, le temps CPU utilisé, …).

8.2.4.3 Quel optimiseur choisir ?

- l'approche coût est à choisir pour les nouveaux projets,
- l'approche règles sera abandonnée progressivement,

néanmoins,

- revoir les applications existantes optimisées par l'approche règles représente un travail important,
- le comportement de l'optimiseur coût est plus difficile à prédire lorsque les données changent. Il faut tout particulièrement faire attention lorsque l'on passe d'une base de test à une base de production.

8.2.4.4 L'approche de l'optimiseur règles

L'optimiseur détermine d'abord le chemin d'accès aux données qu'il va choisir pour chaque table :

les accès considérant peu de lignes sont préférés aux accès par balayage de plusieurs lignes :

1. accès à une seule ligne par rowid,
2. accès à une seule ligne sur clé primaire sur table en hash cluster,
3. accès à une seule ligne sur clé primaire avec un index,
4. accès à une ou plusieurs ligne(s) sur une colonne clé de hash cluster,
5. accès à une ou plusieurs ligne(s) par un index multi-colonnes,
6. accès à une ou plusieurs ligne(s) par un index mono-colonnes
7. full table scan.

Il détermine ensuite l'ordre dans lequel il va réaliser les jointures :

- la première table dans l'ordre des jointures, encore appelée la table maîtresse (outer table), est celle qui optimise le moins l'accès à ses données (celle pour laquelle on a choisi dans l'étape 1 le rang d'accès le plus élevé),
- à priorité égale, de la droite vers la gauche dans l'ordre de la clause from.
- l'optimiseur choisit l'opération nested loop si le chemin d'accès à la table secondaire (inner table) est strictement inférieur à 7 sinon, il choisit l'opération sort-merge

Exemple :

```
select /*+ RULE */ nom_employe, nom_departement

from employe E, client C, departement D

where D.id_departement=E.id_departement

and  nom_client = nom_employe
```

Execution Plan

```
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=HINT: RULE
   1    0   NESTED LOOPS
   2    1     MERGE JOIN
   3    2       SORT (JOIN)
   4    3         TABLE ACCESS (FULL) OF 'CLIENT'
   5    2       SORT (JOIN)
   6    5         TABLE ACCESS (FULL) OF 'EMPLOYE'
   7    1     TABLE ACCESS (BY INDEX ROWID) OF 'DEPARTEMENT'
   8    7       INDEX (UNIQUE SCAN) OF 'PK_DEPT' (UNIQUE)
```

Analyse du plan :

- Ordre selon les chemins d'accès : Employe (7) Client(7) , Departement (3),
- Jointure par Sort-merge entre Employe et Client (même rang= 7)
- Jointure par Nested Loop entre Employe-Client et Departement (rang= 3)

8.2.4.5 L'approche de l'optimiseur coûts

L'optimiseur retient le plan d'exécution pour lequel il a estimé le plus faible coût d'exécution.

- Le calcul du coût est basé principalement sur le nombre de lignes lues dans la base de données et sur le coût des tris nécessaires pour exécuter le plan.
- **Attention !** : Le comportement de cet optimiseur est plus difficile à prédire car il se base sur les statistiques générées sur la table (qui peuvent ne plus être à jour) et aussi parce que les règles de cet optimiseur ne sont pas publiées et sont susceptibles de changer d'une version du produit à une autre.

Calcul des statistiques : on dispose de deux possibilités (calcul exhaustif pour les petites tables et estimation pour les grosses).

```
analyze table acteur compute statistics ;

analyze table acteur estimate statistics sample 10 percent;

analyze index fk_acteur compute statistics ;
```

Les informations générées sont :

- pour les tables, le nombre de lignes, nombre de blocs utilisés et vides, longueur moyenne d'une ligne, volume moyen utilisé dans chaque bloc, statistiques sur les colonnes (nb valeurs distinctes, histogrammes).
- pour les index, le nombre de valeurs de clé distinctes, le nombre de blocs feuille, la hauteur de l'arbre.

```
select /*+ ALL_ROWS */ nom_employe, nom_departement

from employe E, client C, departement D

where D.id_departement=E.id_departement

and  nom_client = nom_employe;
```

```
Execution Plan
----------------------------------------------------------
0      SELECT STATEMENT Optimizer=HINT: ALL_ROWS (Cost=8 Card=10100 Bytes=717100)
1    0   HASH JOIN (Cost=8 Card=10100 Bytes=717100)
2    1     TABLE ACCESS (FULL) OF 'CLIENT' (Cost=2 Card=101 Bytes=3232)
3    1     HASH JOIN (Cost=4 Card=300 Bytes=11700)
4    3       TABLE ACCESS (FULL) OF 'DEPARTEMENT' (Cost=1 Card=3 Bytes=36)
5    3       TABLE ACCESS (FULL) OF 'EMPLOYE' (Cost=2 Card=300 Bytes=8100)
```

Calcul d'histogrammes sur les colonnes d'une table :

- L'optimiseur n'utilise pas d'index si le nombre de valeurs différentes pour la colonne indexée est faible. En effet, dans ce cas, l'index est peu discriminant et il peut être préférable de faire un full table scan.
- Cependant, si l'optimiseur connaît la fréquence de distribution sur chaque valeur, il peut décider d'utiliser l'index pour une valeur peu fréquente qui sera préférable à un full table scan.

Commandes SQL pour obtenir les statistiques sur un nombre limité de colonnes :

```
analyze table acteur compute statistics for all
indexed columns;

analyze table acteur estimate statistics sample 25
percent for column age;
```

Attention : l'optimiseur n'utilise les histogrammes que lorsque les valeurs accédées sont spécifiées dans la requête. Cela exclut l'utilisation de variables hôtes.

- l'optimiseur utilisera les histogrammes pour cette requête :

```
Select nom, age from acteur where age > 100 ;
```

- l'optimiseur n'utilisera pas les histogrammes pour cette requête :

```
age_min:= 100 ;
Select nom, age
into w_nom, w_age
from acteur
where age > age_min ;
```

Conclusion sur l'optimiseur coûts :

- il faut mettre à jour régulièrement les statistiques,
- il faut analyser toutes les tables invoquées dans les requêtes,
- s'il n'a pas de statistiques, l'ordre de jointure par défaut de l'optimiseur est de la gauche vers la droite selon l'ordre des tables dans la clause from (pour l'optimiseur Règles, c'est exactement l'inverse).

8.2.4.6 Choisir un optimiseur et ses objectifs

On peut préciser à Oracle quel optimiseur utiliser pour analyser une requête (au niveau de la base de données, d'une session ou d'une requête). Les options sont :

- RULE : Oracle utilise l'optimiseur Règles,
- CHOOSE : Oracle utilise l'optimiseur Coûts si au moins une des tables invoquées dans la requête a été analysée, sinon il utilise l'optimiseur Règles,
- ALL_ROWS : Oracle utilise l'optimiseur Coûts (sans vérifier s'il y a des statistiques) et choisit le plan qui minimise le coût de traitement de toutes les lignes invoquées par la requête,
- FIRST_ROWS : Oracle utilise l'optimiseur Coûts et choisit le plan qui minimise le coût pour obtenir la première ligne invoquée par la requête.

L'option ALL_ROWS est adaptée aux requêtes invoquées dans un traitement différé (batch) ou la préparation d'un état.

L'option FIRST_ROW est adaptée aux transactions invoquées dans des applications intéractives où l'on cherche à optimiser l'affichage du premier écran de données.

Pour choisir l'optimiseur au niveau de la base de données, on ajoute dans le fichier init.ora :

```
OPTIMIZER_MODE= FIRST_ROWS
```

Pour choisir l'optimiseur au niveau d'une session, on exécute l'ordre SQL :

```
ALTER SESSION SET OPTIMIZER_GOAL= FIRST_ROWS
```

Pour choisir l'optimiseur au niveau d'une requête, on exécute l'ordre SQL en ajoutant une recommandation (a hint) :

```
SELECT /*+ FIRST_ROWS */ NOM, AGE
FROM ACTEUR
WHERE AGE > 100 ;
```

8.3 Optimisation de l'accès à une table

8.3.1 Choix entre balayage table et passage par index

Il vaut mieux utiliser un index si l'on accède ***à moins de 20%*** du contenu de la table, sinon un balayage table est préférable.

Stratégies des optimiseurs :

- L'optimiseur règles va toujours privilégier un passage par index,
- L'optimiseur coûts n'utilisera un index que s'il est sélectif (nombre de valeurs différentes pour la colonne indexée est élevé).

8.3.2 Eviter les balayages table accidentels

Il faut faire attention à la façon dont on écrit une requête SQL. Dans certains cas, l'optimiseur n'utilisera pas un index existant alors qu'il aurait pu être utile. Les exemples type sont :

- utilisation d'une condition !=,
- recherche de valeurs NULL,
- utilisation de fonctions.

8.3.2.1 Utilisation du !=

L'optimiseur réalise un balayage de la table. C'est correct si la valeur exclue est très peu présente dans la table. Dans le cas contraire, ce n'est pas très efficace.

Exemple :

La table Facture(IdFacture, IdClient, EtatFacture, ...) avec comme valeurs possibles pour EtatFacture *Encours, EnRetard et Annulée*. On suppose que l'état Encours est très majoritaire. Dans ce cas :

```
select * from facture where EtatFacture !='Encours' ;
```

est moins bon que

```
select * from facture
   where EtatFacture in ('EnRetard', 'Annulée') ;
```

Remarque : Seul l'optimiseur coûts est capable de détecter automatiquement l'intérêt de passer par l'index s'il dispose de l'histogramme de répartition des valeurs de la colonne EtatFacture.

Dans tous les cas, on peut utiliser un hint pour être sur qu'Oracle utilise l'index :

```
select /*+ use_concat index(facture EtatFactureIdx) */ *

from facture

where EtatFacture in ('EnRetard', 'Annulée') ;
```

Remarque : il faut mettre deux hints :

- **index** pour rechercher les deux valeurs dans l'index,
- **use_concat** pour fusionner les deux listes de rowid.

8.3.2.2 Interrogation sur la valeur NULL

Comme la valeur NULL n'est pas stockée dans les index, une interrogation sur NULL dans une requête est exécutée sous la forme d'un balayage de la table. Par exemple : la table Facture(IdFacture, IdClient, EtatFacture, DateRéglement,...)

```
select * from facture where DateReglement is NULL ;
```

Si le nombre de lignes concernées est faible, il est préférable de prévoir une valeur par défaut pour pouvoir bénéficier d'un accès par l'index.

Solution possible : Plutôt que de laisser la colonne DateRéglement à NULL lorsque l'on crée une ligne dans la table, il est préférable d'y mettre une valeur par défaut, par exemple 'inconnue'.

```
alter table facture

modify DateReglement default 'Inconnue' ;

select * from facture where DateReglement='Inconnue';
```

8.3.2.3 Utilisation de fonctions

Si l'on applique une fonction sur une colonne indexée, l'index n'est pas utilisé par l'optimiseur. Par exemple, la table Employe(IdEmploye, Nom, Prenom, ...) :

```
select IdEmploye, Nom

from employe

where upper(Nom)= upper(:var1)

 and upper(Prenom) = upper(:var2) ;
```

Solutions possibles : être sur que les valeurs stockées dans les colonnes sont toujours en majuscule ou gérer deux colonnes supplémentaires NomMaj et PrenomMaj gérées par trigger dans lesquelles on mémorise nom et prénom en majuscule.

```
select IdEmploye, Nom

from employe

where Nom= upper(:var1)

 and Prenom = upper(:var2) ;
```

8.3.3 Optimisation des recherches par index

8.3.3.1 Index concaténés

Rappel : Penser à créer des index concaténés sur l'ensemble des colonnes interrogées dans la clause **where** de la requête.

Autre optimisation possible : Si l'on ramène une colonne supplémentaire en projection, l'ajouter dans l'index évite de faire un accès à la table.

Par exemple : dans Employe(Id_emp, Nom_emp, Prenom_emp, Age, NoTel...),

```
select Nom, Prenom, NoTel

from employe

where Nom= upper(:var1)

 and Prenom = upper(:var2) ;
```

on crée l'index concaténé :

```
create index nom_prenom_tel

on employe(nom, prenom, tel) ;
```

8.3.3.2 Opérateur like

L'optimiseur utilise l'index si le caractère % n'est pas utilisé en début de chaîne.

```
Select count(*) from Employe

where Nom like 'Dup%' ;
```

On peut tout de même forcer l'utilisation de l'index par un hint. Balayer un index est plus rapide que balayer une table car l'index est plus petit.

```
select /*+ index(Employe Nom_Idx) */ count(*)

from Employe

where Nom like '%ssian' ;
```

8.3.3.3 Opérateur and

En cas de critère de sélection multi-colonnes avec des index séparés pour chaque colonne, Oracle réalise une intersection des listes de rowid sélectionnés par chaque index (opération AND-EQUAL dans le plan d'exécution).

Pour optimiser cette requête, on peut créer un index multi-colonnes qui est plus sélectif que les index séparés.

8.3.4 Table en hash cluster

Il faut déterminer deux paramètres pour organiser une table en hash cluster :

- HASHKEYS : le nombre de blocs dans lesquels sont réparties les lignes,
- SIZE : l'espace alloué pour chaque bloc de données

Pour HASHKEYS :

- si le cluster est réalisé sur la clé primaire, *hashkeys* est égal au nombre de lignes de la table,
- si le cluster est organisé sur une autre colonne, *hashkeys* est égal au nombre de valeurs distinctes pour cette colonne.

Pour SIZE :

$$SIZE = \frac{NbTotalLignes}{HASHKEYS} \times average\_row\_length \times 1{,}1$$

On peut obtenir ces informations sur la table (après avoir calculé ses statistiques) par l'intermédiaire des vues suivantes :

```
select avg_row_len, num_rows
   from user_tables
   where table_name='FACTURE' ;
```

```
select num_distinct
   from user_tab_columns
   where table_name='FACTURE'
   and column_name='ID_CLIENT' ;
```

Création du cluster :

```
create cluster hash_facture (id_client varchar2(3))
   haskeys 100
   size 50K ;
```

Création de la table :

```
create table facture (
   id_facture varchar2(3) primary key,
   id_client varchar2(3),
   etat_facture varchar2(20),
   date_reglement varchar2(20) default 'Inconnue')
   cluster hash_facture(id_client);
```

8.3.5 Optimisation des balayages table

Lorsque l'accès à une table se fait par un balayage séquentiel, on peut essayer d'optimiser la lecture de plusieurs manières.

8.3.5.1 Réinitialiser le marqueur de fin de table (high water mark)

Dans un balayage séquentiel, Oracle lit du premier bloc de données jusqu'au dernier bloc qui a contenu des données depuis la création de la table. Il n'y a jamais de recompactage de la table. Si la table est souvent mise à jour, on peut être amené à lire beaucoup de blocs par rapport au nombre de lignes se trouvant réellement dans la table.

2 solutions possibles :

- Utiliser les utilitaires d'exportation et de réimportation de données d'Oracle,
- Recréer la table en utilisant un stockage temporaire.

2ème solution :

```
create table facture2 (
   id_facture primary key,
   id_client ,
   etat_facture ,
   date_reglement default 'Inconnue')
   as
      select id_facture, id_client, etat_facture,
      date_reglement
      from facture;
```

```
drop table facture ;

create table facture (
   id_facture primary key,
   id_client ,
   etat_facture ,
   date_reglement default 'Inconnue')
   as
   select id_facture, id_client, etat_facture,
   date_reglement
   from facture2;
```

8.3.5.2 Maintenir une table dans le cache-disque

Les grandes tables parcourues en balayage séquentiel sont les moins prioritaires pour l'algorithme LRU qui gère le cache-disque.

Si l'on a besoin de faire deux balayages consécutifs d'une grande table, il y a donc de fortes chances qu'Oracle ait besoin de lire à nouveau les blocs de données sur le disque.

On peut modifier ce comportement en utilisant le hint ***cache***. Par exemple :

```
select /*+ cache(Employe) */ * from employe ;
```

8.4 TD Etudes de cas d'optimisation d'accès à une table

Si vous utilisez la version Unix de l'interpréteur SqlPlus, recopiez les fichiers se trouvant sous :

```
/home/test/td_oracle/td10
```

Si vous utilisez la version Windows de l'interpréteur SqlPlus, recopiez les fichiers se trouvant sous :

```
Sur le lecteur td et dans le répertoire /Td_Bdspe/Td10
```

8.4.1 Prise en main de l'utilitaire de génération de trace

L'objectif de cet exercice est de générer la trace d'exécution d'une requête. Pour tester cela, vous allez demander à Oracle de générer la trace d'exécution correspondant à l'ordre SQL suivant :

```
select * from acteur where nom = 'Coluche' ;
```

Pour générer une trace d'exécution d'une requête SQL, la procédure est la suivante :

1) Connectez vous à Oracle avec l'utilitaire SQLPLUS. Vous passez en mode trace d'exécution par l'ordre SQL :

```
alter session set sql_trace true ;
```

2) Afin de retrouver facilement votre fichier trace par la suite, commencez la session en exécutant l'ordre SQL :

```
select 'speinfo1_1023' from dual ;
```

Vous remplacerez dans la commande ci-dessus *speinfo1* par votre nom de compte Oracle et *1023* par l'heure a laquelle vous exécutez la requête.

Tous les fichiers trace sont générés dans un répertoire commun à tous les utilisateurs de la base de données. L'ordre SQL ci-dessus permet d'insérer dans le fichier qui contient la trace d'exécution de votre session un identifiant unique pour la session dans laquelle vous exécutez la requête.

Exécutez ensuite votre requête SQL et sortez de la session (exit).

3) Le fichier trace est généré dans le système de fichiers du serveur Oracle (ie Castor). Vous devez donc vous connecter obligatoirement sur la machine serveur pour exécutre les opérations qui suivent. Le répertoire contenant les fichiers trace est :

```
/oracle/admin/speinfo816/udump
```

Recherchez le nom de votre fichier trace par la commande shell (grep recherche la chaîne de caractères passée en premier argument dans l'ensemble des fichiers filtrés par le deuxième argument) :

```
grep speinfo1_1023 *.trc
```

Si le nom du fichier trace est speinfo_ora_61651.trc, recopiez ce fichier dans votre répertoire de travail et depuis le shell, exécutez la commande (en ayant préalablement remplacé speinfo1 par votre nom de compte Oracle) :

```
tkprof speinfo_ora_61651.trc resul.prf
       explain=speinfo1/speinfo1
       sort='(prsela,exeela,fchela)'
```

La trace d'exécution est générée par l'utilitaire tkprof dans le fichier resul.prf. Générez la trace d'exécution correspondant à la requête SQL. Analysez la trace en la comparant à celle donnée ci-dessous :

- La colonne rows (ligne fetch) indique que la requête a ramené un seul tuple.
- Le plan d'exécution généré pour le select est un accès à la table par rowid (table access by index rowid), les rowid sont obtenus par une recherche (index range scan) dans l'index pk_acteur

```
********************************************************************************
select *
from
 acteur where nom='Coluche'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.05       0.01          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          3          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.05       0.01          0          3          0           1

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 35  (SPEINFO11)

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID ACTEUR
      2   INDEX RANGE SCAN (object id 4551)


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      1   TABLE ACCESS (BY INDEX ROWID) OF 'ACTEUR'
      2    INDEX (RANGE SCAN) OF 'PK_ACTEUR' (UNIQUE)
```

8.4.2 Utilisation des index par les optimiseurs

Créez la table test_index et ses index associés en exécutant le script base.sql. Puis, exécutez les scripts ins200.sql, ins400.sql, ins600.sql et ins800.sql qui insèrent au total 2000 lignes dans la table. Générez les statistiques sur la table test_index pour l'optimiseur statistique.

Dans une autre session SQLPLUS, générez la trace d'exécution des scripts requetes1.sql et requete2.sql d'abord en utilisant l'optimiseur règles, puis l'optimiseur statistique.

Le choix de l'optimiseur est obtenu en exécutant la commande :

```
alter session set optimizer_goal = rule ;
```

ou

```
alter session set optimizer_goal = all_rows ;
```

Quelles conclusions tirez-vous de l'analyse des traces d'exécution ? L'optimiseur statistique fait, par défaut, une hypothèse de distribution uniforme sur les valeurs distinctes prises par une colonne donnée. Pour calculer l'histogramme de répartition sur n plages de valeurs distinctes, utilisez la fonction suivante du package dbms_stats (pour le propriétaire *speinfo11*, la table *test_index* et la colonne *nom*, n est ici fixé à 4) :

```
execute dbms_stats.gather_table_stats('speinfo11', 'test_index',
method_opt => 'for columns size 4 nom');
```

Générez de nouveau la trace d'exécution correspondant à une optimisation statistique des requêtes contenues dans le fichier requetes1.sql. Que constatez-vous ?

8.4.3 Création d'une table en hash cluster

Mettre la table Film en hash cluster sur la colonne Réalisateur. La démarche à suivre est la suivante :

- générer les statistiques de la table film,
- calculer les deux paramètres pour créer le cluster,
- créer le cluster,
- créer la table film_h,
- insérer les données de la table film dans film_h,
- comparer la trace d'exécution des requêtes :

```
select * from film where realisateur='OURY' ;
select * from film_h where realisateur='OURY' ;
```

8.4.4 Optimiser un balayage séquentiel

Un bloc de données sous Oracle fait 2K octets (valeur par défaut qui peut être modifiée). A partir du fichier *debut_bal_seq.sql*, créez une table à deux colonnes (un entier et une chaîne fixe de 100 caractères).

- Avec une séquence, initialisez la table avec 2000 lignes.
- Comptez le nombre de blocs physiques occupés (cf commande sql ci-dessous).
- Supprimez les 1950 dernières lignes.
- Générez la trace d'exécution d'un balayage séquentiel de toute la table.

- Comptez le nombre de blocs physiques lus (colonnes Query et Current sur la ligne Fetch).

Réinitialisez le marqueur de fin de table (high water mark) avec la procédure présentée dans le cours et comptez de nouveau le nombre de buffers lus pour l'exécution d'un balayage séquentiel. Comparez.

Vous pouvez également connaître le nombre de blocs utiles (correspondant aux données logiquement présentes dans la table) se trouvant dans la table :

```
select count (distinct substr(rowid, 10, 6)) blocks
from test_sequ ;
```

9. Architecture du SGBD ORACLE

Objectif : Description de l'architecture interne d'un SGBD relationnel, i.e. :

- Les structures de données mises en oeuvre (fichiers, structures en mémoire centrale, unités de stockage logiques)
- Les actions sur les structures de données (les processus UNIX)

Intérêt : Etre capable de configurer une base de données et d'en assurer la maintenance (sauvegarde, optimisation). C'est le travail du **DBA** (administrateur de la base de données).

9.1 Les structures de données

9.1.1 Deux points de vue

Point de vue sur les **structures physiques** :

- Dépendantes du système d'exploitation : fichiers UNIX qui contiennent les objets de la base et les zones de mémoires partagées.
- Seul l'administrateur de la base est concerné par leur gestion.
- Un fichier se décompose en n blocs physiques.

Point de vue sur les **structures logiques**

- Unités de stockage logiques gérées par l'administrateur et utilisées par les développeurs (tablespaces, segments, tables, index).
- Tablespace : Un tablespace est stocké physiquement dans un ou plusieurs fichier(s) UNIX. La taille du fichier UNIX est fixée une fois pour toute (dans une fourchette min/max) lorsqu'on l'associe au tablespace. D'un point de vue logique, un tablespace est un ensemble de segments.
- Segment : C'est l'unité logique de stockage d'une table, d'un index, de journaux-avant (rollback segment) et de zones temporaires de tri (pour réaliser les ordres SQL distinct, order by et group by par exemple).
- Extent : Un segment se décompose en n extents. Un extent est composé de m blocs physiques contigus.

Assurent l'indépendance de la base de donnée vis à vis du système d'exploitation : la portabilité d'une base de données d'un système d'exploitation à un autre (UNIX, DOS, NT, VMS, VM, ...) est assurée.

9.1.2 Composition minimale d'une base de données ouverte

- une instance,
- un tablespace SYSTEM,
- deux fichiers de journalisation (journal-après),
- un fichier de contrôle.

9.1.3 Une instance

Un ensemble de mécanismes qui permet **l'accès et le contrôle** d'une base de données. Elle est composée d'une structure en mémoire centrale (la SGA) et d'un ensemble de processus Unix qui agissent sur cette structure et sur les fichiers constituant la base.

9.1.4 La System Global Area

Elle joue le rôle de **cache-disque** de la base de données.

Intérêt : retarder au maximum les lectures et écritures sur le disque en laissant en mémoire centrale les données les plus fréquemment accédées.

C'est un ensemble de zones de **mémoire partagée** (mécanisme Unix SYSTEM V), i.e. des zones de mémoire accessibles par plusieurs processus simultanément. La SGA est donc une zone de stockage temporaire des données et d'échange d'informations entre les processus Unix qui gérent l'accès à la base de données.

Elle contient plusieurs types d'informations :

- des zones de données (images-avant et après des tuples),
- des parties du dictionnaire de données (tab$, col$, user$, ...),

- une file d'attente de requêtes,
- des files d'attente de réponse,
- des zones d'exécution de requêtes SQL (partagées et privées),
- des zones de Redo Log.

Sa taille est fixée à l'ouverture de la base.

9.1.5 Le tablespace SYSTEM

Il est créé automatiquement à la génération de la base.

Il contient les tables du **dictionnaire de données**. Par exemple, **tab$** est la table des noms de tables stockées dans la base de données, **user$** la table des noms d'utilisateurs enregistrés, **ts$** la table des noms de tablespaces, ...

Il contient un **rollback segment** dans lequel est enregistré **le journal-avant** des transactions (réservé aux transactions effectuées dans le tablespace SYSTEM). C'est dans cette unité de stockage qu'ORACLE va chercher **l'image-avant** des tuples modifiés lorsque l'on demande le rollback d'une transaction (cas de la reprise à chaud).

9.1.6 Deux fichiers de journalisation (journal-après)

Ce sont des fichiers UNIX externes à la base de données (i.e. à l'ensemble des tablespaces) appelés les fichiers **Redo Log**.

- Oracle enregistre sur ces fichiers les mises à jour sur la base au fur et à mesure du déroulement des transactions.
- Plus précisément, c'est le seul fichier dans lequel il y a écriture effective sur disque des transactions qui ont été commitées (remarque : le commit d'une transaction est terminé lorsque l'écriture dans le fichier Redo Log s'est correctement terminée).

Pourquoi deux au minimum ?.

ORACLE ne peut pas écrire à l'infini dans le fichier Redo Log actif. Lorsque le fichier Redo Log actif est plein, ORACLE bascule l'écriture des transactions sur le deuxième qui devient actif. Pendant ce temps, ORACLE consolide la base de données (**checkpoint**) en recopiant toutes les transactions commitées du fichier Redo Log plein dans les fichiers de données. Lorsque le checkpoint est terminé, le fichier Redo Log plein est prêt à être réutilisé. ORACLE le réutilisera lorsque le fichier Redo Log actif sera plein à son tour.

Utilité : Ce sont les fichiers qui sont utilisés dans les **reprises à froid** :

- **Reprise à froid, cas 1** : Une panne de courant entraine la perte de la SGA. Des transactions commitées n'ont pas été descendues sur les fichiers de données. La cohérence de la base est perdue. A la prochaine ouverture de la base, ORACLE effectue automatiquement une consolidation de la base. Pour cela, il applique toutes les transactions commitées se trouvant dans le fichier Redo Log courant au moment de la panne. La base est à nouveau cohérente.

- **Reprise à froid, cas 2** : Lorsqu'une partie de la base de données a été perdue ou endommagée, on arrète la base. On repart d'une sauvegarde totale de la base (version cohérente antérieure) et on déroule le journal-après des transactions *commitées*, i.e. un ensemble de fichiers Redo Log. Ce type de reprise n'est possible que si ORACLE travaille en mode d'archivage des fichiers Redo Log.

9.1.7 Un fichier de contrôle :

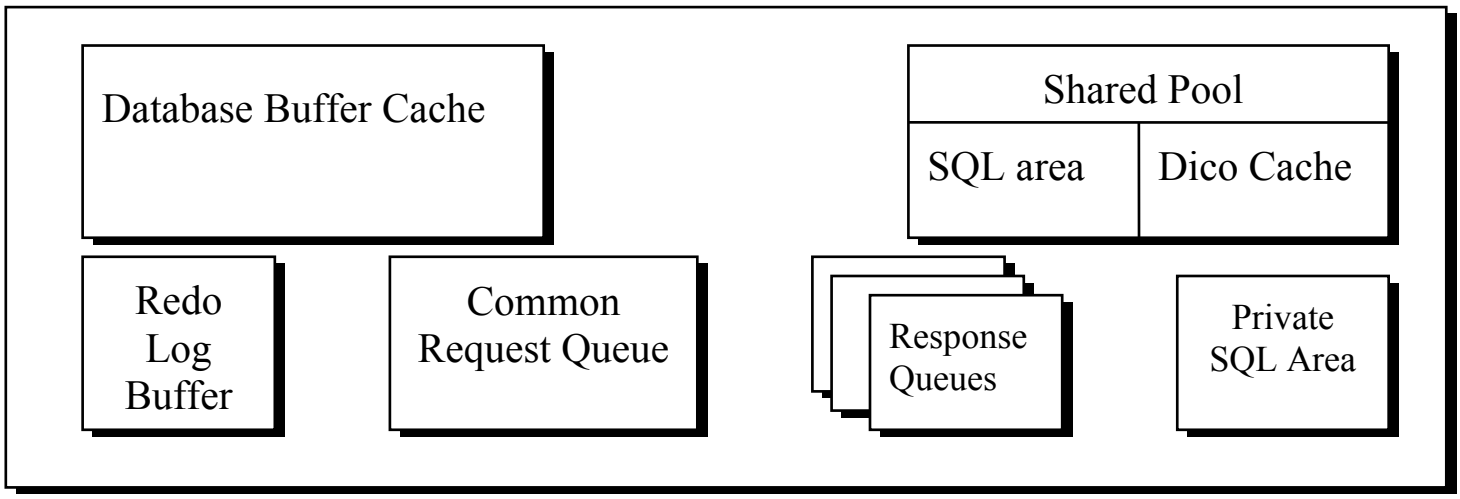
C'est le **fichier maitre** de la base de données.

Il permet d'initialiser le lien entre une instance et les fichiers d'une base de données au moment du démarrage de la base. Il contient pour cela le nom de tous les fichiers Unix associés aux unités de stockage logiques de la base ORACLE.

Il est accédé en permanence par les processus Unix, car il est le lieu de mémorisation des états de la base. Par exemple, il contient le numéro du fichier Redo Log actif.

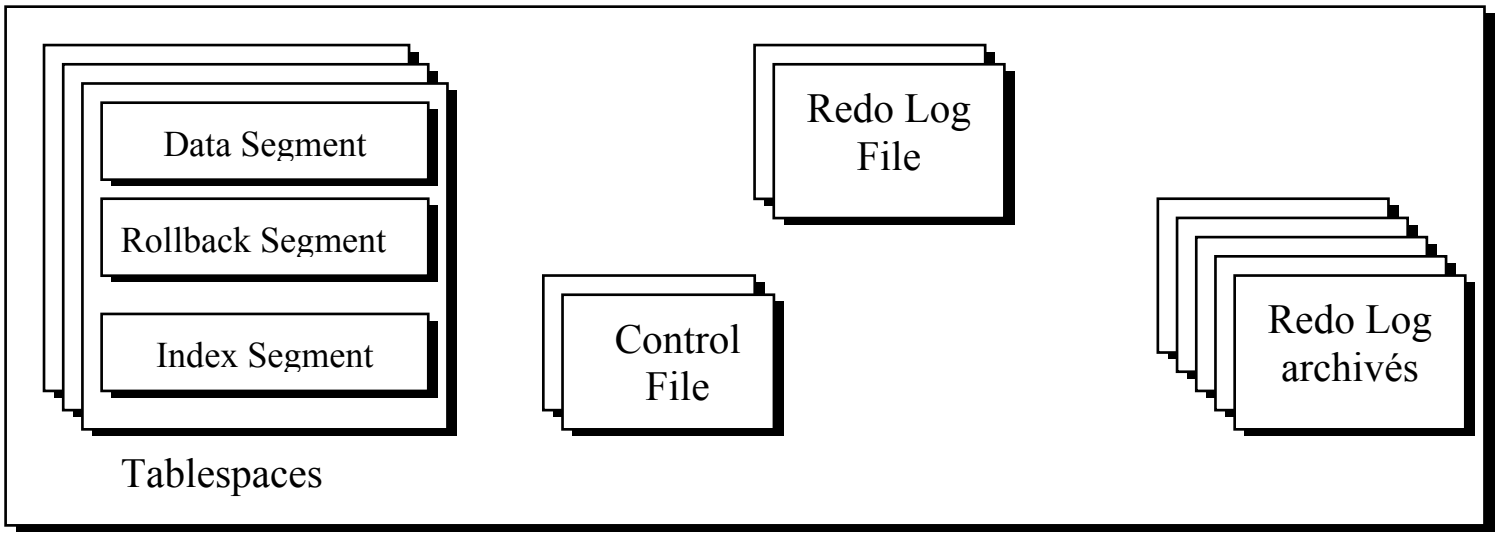
Comme c'est un fichier important, il est conseillé d'en créer plusieurs exemplaires, stockés sur des disques différents. A chaque écriture, ORACLE met à jour tous les exemplaires du fichier de contrôle (**mirroring**).

9.1.8 Représentation d'une base de données ORACLE ouverte



La SGA en mémoire centrale

Les objets ORACLE sur le disque



9.2 Le fonctionnement du SGBD

Sous Unix, deux types de processus assurent le fonctionnement du SGBD : les processus serveurs et les processus d'arrière-plan

9.2.1 Les processus serveurs

Sxxx (Server) : lecture dans la base

9.2.2 Les processus d'arrière-plan

DBWR (DataBase WRiter), écriture dans la base.

LGWR (LoG WRiter), écriture dans le fichier Redo Log.

CKPT (ChecKPoinT), écriture dans le fichier Redo Log.

SMON (System MONitor), moniteur du système.

PMON (Process MONitor), moniteur des processus.

ARCH (ARCHivor), archivage des transactions.

RECO (RECOver), gestion des transactions distribuées.

Dxxx (Dispatcher), dispatchers des transactions.

9.2.3 Sxxx, le processus Serveur

Processus sollicité lorsqu'un client (sqlplus, C/SQL) transmet une requête à la base.

Il récupère dans la Common Request Queue les requêtes à exécuter.

Il analyse la requête dans la shared pool (SQL area) et dans la private SQL area.

Il recherche les données demandées par l'utilisateur :

- Dans la SGA.
- Si non trouvées dans la SGA, lit les blocs de données dans le tablespace et les charge dans la SGA.

Remarque : S'il n'y a plus de place en SGA pour les accueillir, les données les moins récemment utilisées en SGA sont redescendues sur le disque pour leur laisser la place. La SGA est donc gérée avec un mécanisme de pagination virtuelle selon un algorithme LRU (Least Recently Used).

Il restitue les informations demandées dans la file d'attente des réponses (Response Queue) du dispatcher qui l'a sollicité.

ORACLE fournit deux modes de fonctionnement du processus utilisateur : **single task** et **multi-thread.**

Single task (pour MS-DOS) : le processus utilisateur a les accès en lecture/écriture sur les fichiers qui contiennent la base.

L'architecture **multi-thread :** deux processus communiquent entre eux

- Le processus **UPI** (User Process Interface) : c'est le processus client qui émet la demande de données et s'adresse à l'OPI,
- Le processus **OPI** (ORACLE Process Interface) : c'est le processus serveur qui va réaliser l'accès à la base. Le client ne s'adresse pas en général directement au processus serveur mais transite par un Dispatcher.
- En architecture client-serveur Unix/DOS : l'UPI tourne sur le PC où s'exécute l'application, l'OPI tourne sur le serveur Unix. Dans ce cas, la communication entre les deux processus implique l'existence d'une couche de transport réseau TCP/IP et le produit SQL/NET d'ORACLE.
- En Multi-thread, un OPI peut servir plusieurs UPI (il fait du multiplexage). En effet, un UPI n'émet pas en permanence des requêtes et donc l'OPI peut être partagé. L'intérêt est d'économiser l'espace-mémoire utilisé par les processus serveur OPI.

9.2.4 Dxxx, les processus Dispatcher

Lorsqu'un client veut adresser une requête à la base de données, il va s'adresser au Dispatcher, directement ou indirectement via le processus listener (lorsqu'il s'agit de clients qui tournent sur des machines différentes).

Le Dispatcher va déposer la requête du client dans la file d'attente de type FIFO(Request Queue) qui est consultée par les processus serveur.

Le Dispatcher est responsable du suivi de l'exécution de la requête. A ce titre, il restitue au client la réponse à la requête. Il accède pour cela à sa file d'attente des réponses (Response Queue) qui est alimentée par les processus serveur.

9.2.5 DBWR, le processus écrivain dans les fichiers de la base

Ecrit les buffers modifiés (dirty buffers) par les processus utilisateurs se trouvant en SGA dans les fichiers de la base.

C'est un processus d'arrière-plan (background) non attaché à un terminal.

C'est un processus asynchrone : il n'y a pas d'écriture systématique dans la base à chaque commit. L'intégrité est garantie par l'écriture des données dans le fichier Redo Log.

C'est lui qui exécute les consolidations (checkpoints). Le processus CKPT le réveille lorsque le fichier Redo Log actif est plein.

Le processus serveur Sxxx le réveille lorsqu'il ne trouve pas assez de buffers libres dans la SGA pour monter les blocs de données dont il a besoin en mémoire centrale. DBWR se charge alors d'écrire sur disque les données qui se trouvent dans la liste LRU. Le processus serveur le réveille également lorsque le nombre de buffers modifiés dépasse un certain seuil (données commitées ou non).

9.2.6 LGWR, le processus écrivain dans les fichiers Redo Log

C'est le processus qui garantit l'intégrité de la base en sauvegardant chaque transaction commitée sur le fichier Redo Log actif.

Les données modifiées par les transactions sont écrites dans un buffer Redo Log dans la SGA.

LGWR écrit le contenu du buffer Redo Log quand :

- une transaction demande un commit.
- le buffer Redo Log est plein (au tiers), car ce buffer est circulaire. Il faut que les processus serveur puissent écrire dans le Redo Log buffer sans attendre sinon il y a risque de contention du système.

L'écriture dans le fichier Redo Log est séquentielle. C'est plus rapide à exécuter qu'une mise à jour au bon endroit dans la base.

Oracle permet de gérer simultanément plusieurs versions identiques du fichier Redo Log actif (fonctionnement en miroir). L'intérêt est de couvrir la situation où arrivent simultanément une panne de courant et une panne du disque sur lequel est stocké le fichier Redo Log actif. Dans ce cas, le Log Writer écrit de manière synchrone sur le groupe des fichiers Redo Log actifs.

9.2.7 CKPT, le processus écrivain, auxiliaire de LGWR

Il supervise la consolidation du fichier Redo Log plein dans les fichiers de la base de données dans la phase de CheckPoint (réalisée par DBWR). Il écrit dans l'entête des fichiers de la base le numéro du dernier fichier Redo consolidé.

C'est donc lui qui garantit que le fichier Redo Log plein sera réutilisable lorsque le fichier Redo Log actif sera plein à son tour.

9.2.8 SMON, le moniteur du système

C'est lui qui lit le fichier de contrôle au démarrage de la base pour établir le lien entre l'instance et les fichiers de la base.

Il a un rôle de surveillance générale de l'activité du système :

- Il contrôle que les processus de l'instance sont toujours en mémoire centrale (exemple: DBWR ou LGWR).
- Il génère les erreurs détectées dans le fichier alert.log.

Au démarrage de la base, c'est lui qui consolide éventuellement la base avec les transactions enregistrées dans le fichier Redo Log actif (noté dans le fichier de contrôle) après une panne de courant.

9.2.9 PMON, le moniteur des processus utilisateur

C'est un processus asynchrone qui scrute de manière périodique le bon fonctionnement des processus dispatcher, serveur et utilisateurs.

Si un processus utilisateur tombe en panne (exemple: coupure physique d'une connexion), PMON est responsable du ménage.

Il libère toutes les ressources utilisées par le processus serveur pour exécuter la requête du processus utilisateur tombé en panne. Par exemple, il libère les zones occupées par le processus serveur en SGA, les verrous sur les tuples en cours de modification, les zones mémoire utilisées par le processus serveur pour exécuter la requète (la PGA, Program Global Area, la private SQL area, éventuellement la shared SQL area).

C'est lui qui détecte les verrous mortels (deadlocks) entre les transactions. Pour cela, il gère dans la SGA un buffer des verrous posés ou demandés par les transactions.

Il gère également les rollbacks demandés par les transactions. Il reconstitue l'image-avant des données grâce aux rollback segment et libère les verrous posés sur les données en cours de modification.

9.2.10 ARCH, le processeur d'archivage

Ce processus n'est pas forcément présent. En effet, l'archivage des fichiers Redo Log est optionnel. Il est indispensable si l'on veut faire une reprise à froid à partir d'une sauvegarde de la base.

C'est un processus asynchrone qui est réveillé par le processus LGWR lorsque le fichier Redo Log actif est plein.

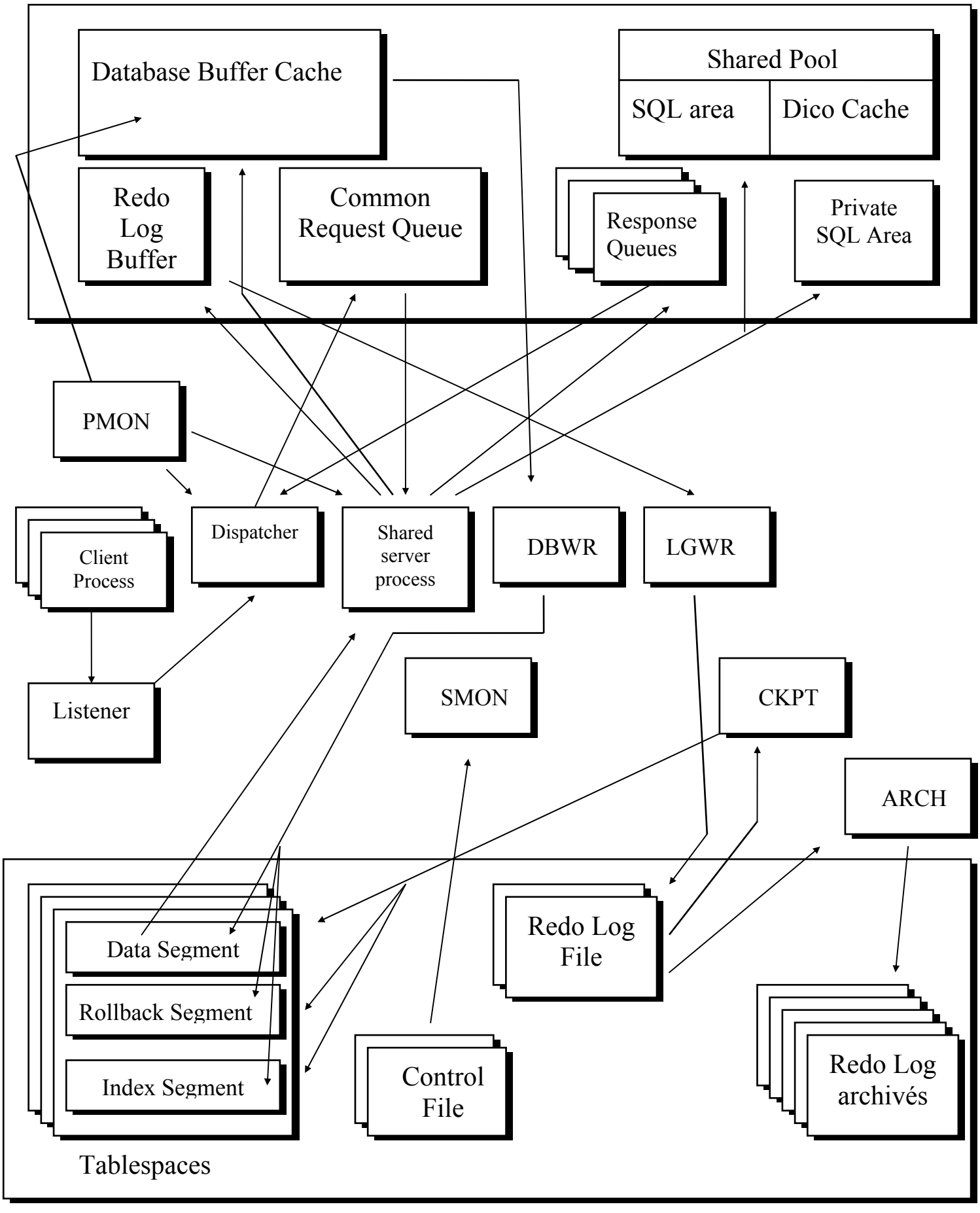
Pendant que DBWR consolide la base de données, ARCH recopie le fichier Redo Log plein dans un fichier d'archivage.

9.2.11 RECO, le gestionnaire des transactions distribuées

C'est un processus asynchrone qui gère la liste des transactions distribuées en cours de validation (commit à deux phases).

Il gère notamment les problèmes de connection avec les serveurs de bases de données impliqués dans la transaction.

9.2.12 Un schéma d'ensemble du fonctionnement de la base



9.2.13 Quelques questions

Que se passe-t-il si une coupure de courant intervient pendant un checkpoint ?

Que se passe-t-il si l'on perd le fichier Redo Log actif ?

Dans une situation idéale (sans problème financier), combien de disques faut-il prévoir pour concevoir un serveur de données avec ORACLE et comment répartir les fichiers sur les différents disques ?

Réponses :

Une coupure de courant intervient pendant un checkpoint : durant l'exécution du checkpoint, les deux fichiers Redo Log sont marqués comme actifs dans le fichier de contrôle. SMON reprendra donc les transactions commitées mémorisés sur les deux fichiers jusqu'au moment de la panne (toutes les transactions enregistrées sont datées).

On perd le fichier Redo Log actif : c'était une faiblesse d'ORACLE V6. L'intégrité de la base n'est plus assurée. Si une panne de courant intervient dans la foulée, on risque d'avoir perdu des transactions commitées. ORACLE V7 permet de faire du mirroring sur le fichier Redo Log actif.

Une configuration pour un serveur de données :

- Un disque pour le tablespace SYSTEM et le fichier de contrôle n°1
- Un disque pour les fichiers de la base (tablespaces de données),
- Un disque pour les fichiers de la base (tablespaces des index),
- Un disque pour les fichiers Redo Log n°1 et le fichier de contrôle n°2,
- Un disque pour les fichiers Redo Log n°2,
- Un périphérique lent pour les fichiers Redo Log archivés (un hexabyte par exemple),
- Une sauvegarde (export) de la base sur disque.

9.3 Agir sur le fonctionnement du SGBD : le fichier init.ora

C'est le fichier qui permet de configurer l'instance. Il est consulté au démarrage de la base de données.

Un exemple de fichier de configuration init.ora:

```
#
# $Header: init.ora,v 6.12 89/11/14 16:43:00 cyang Exp $ init.ora Copyr
(c) 1987 Oracle
#
db_name = C1
db_block_buffers = 32
# Nombre maximum de verrous posés sur des tuples
dml_locks = 100
log_checkpoint_interval = 10000
# Nombre maximum de processus utilisateurs
processes = 10
control_files=(/home/mathinfo/buche/cours_bd/tp3/cntrl2_C1.dbf,
/home/mathinfo/buche/cours_bd/tp3/cntrl1_C1.dbf)
# Nombre maximum de transactions concurrentes
transactions = 11
transactions_per_rollback_segment = 21
rollback_segments = (RS1)
#log_archive_start = true
#log_archive_dest=$HOME/cours_bd/tp3/arch/C1.arc
```

log_checkpoint_interval = 10000 :

ORACLE donne la possibilité d'effectuer un checkpoint à chaque fois que le nombre de blocs spécifié est atteint. le nombre de blocs écrits entre deux checkpoints. Ca permet de réduire le temps d'exécution d'une reprise à froid à partir du fichier Redo Log actif.

db_block_buffers = 32 :

- Il détermine le nombre de buffers de la SGA. Un buffer ORACLE vaut 2K octets sur Unix.
- Plus on augment le nombre de buffers en SGA, moins on fera d'accés disque.
- Le cache hit ratio : 1 accès disque pour 10 demandes d'accès est correct. Pour le déterminer, il faut lancer des statistiques et calculer le rapport logicals reads/physical reads.

9.4 Configuration d'une base de données

On présente les ordres SQL qui permettent de configurer une base de données, à savoir :

- création d'une base de données,
- création d'un tablespace,
- création d'une table dans un tablespace,
- création d'un utilisateur,
- création d'un rollback segment.

9.4.1 Création d'une base de données

```
/* connexion a sqlplus par : sqlplus /nolog */

connect username/password as sysdba

startup nomount pfile=$HOME/initTdSpe1_0.ora

create database "TdSpe1"
   maxinstances 1
   maxlogfiles  16
   character set "WE8DEC"
   datafile '$HOME/systTdSpe1.dbf'
   size 20M autoextend on next 5M maxsize 75M
   logfile
   '$HOME/log1TdSpe1.dbf'   size 500K,
   '$HOME/log2TdSpe1.dbf'   size 500K,
   '$HOME/log3TdSpe1.dbf'   size 500K;
```

La base a été créée dans le répertoire d'accueil ($HOME) avec les fichiers suivant :

1. trois fichiers Redo Log nommés **log1TdSpe1.dbf, log2TdSpe1.dbf** et **log3TdSpe1.dbf**, de taille 500K octets chacun,
2. un tablespace **SYSTEM** stocké dans le fichier **systTdSpe1.dbf** de taille initiale 20M octets pouvant s'étendre par incrément de 5M jusqu'à une taille maximale de 75M,
3. un fichier de contrôle nommé **ctrl1TdSpe.ctl** (spécifié dans le fichier de configuration de l'instance **initTdSpe1_0.ora**).

9.4.2 Tablespace

Un tablespace est l'objet de la base de données qui fait la liaison entre le système de fichiers et la base de données.

Un tablespace est un espace de stockage dans lequel on crée des tables, des index, des procédures, des triggers, ...

Créer un tablespace :

```
create tablespace speinfo datafile
'/usr/local/oracle/dbs/speinfo/speinfo.dbf' size 70M ;
```

Augmenter la taille d'un tablespace en lui ajoutant un fichier :

```
alter tablespace speinfo add datafile
'/usr/local/oracle/dbs/speinfo/speinfo2.dbf' size 30M ;
```

Deux vues pour gérer les tablespaces :

- dba_tablespaces
- dba_data_files

9.4.3 Création et dimensionnement d'une table dans un tablespace

Créer et dimensionner une table dans un tablespace :

```
create table acteur(
   id_acteur number
      constraint pk_acteur primary key,
   nom_acteur varchar2(30) )

   storage (initial 100K next 50k
      minextents 1 maxextents 50 pctincrease 5)
   tablespace speinfo ;
```

initial : taille en nombre d'octets du premier extent alloué pour la table. Par défaut, Oracle alloue 5 blocs de données (ie 10k).

next : taille en nombre d'octets des extents suivant alloués pour la table. Par défaut, Oracle alloue 5 blocs de données (ie 10k).

pctincrease : pourcentage d'augmention de la taille des nouveaux extents (après le second) par rapport à l'extent précédent. La valeur par défaut est 50. Oracle arrondit au bloc de données de taille supérieure. Dans l'exemple, le troisième extent fera 27 blocs (ie 54k).

minextents : spécifie le nombre total d'extents alloués à la table lorsqu'elle est créée. Cela permet d'allouer un espace **important non obligatoirement contigue**. La valeur par défaut est 1.

maxextents : permet de contrôler le nombre maximum d'extents pour une table. La valeur par défaut dépend du système.

9.4.4 Création d'un utilisateur

Créer l'utilisateur :

```
create user speinfo1 identified by speinfo1
   default tablespace speinfo
   temporary tablespace temp
   quota 100k on speinfo ;
```

- la clause **default tablespace** indique dans quel tablespace est créée la table sans indication particulière,
- la clause **temporary tablespace** indique dans quel tablespace sont réalisés les tris lors de l'évaluation des requêtes demandées par l'utilisateur,
- la clause **quota** permet de limiter l'espace alloué à un utilisateur.

Attribuer un rôle à l'utilisateur (ie un ensemble de privilèges système) :

```
grant connect to speinfo1
```

- attribue à l'utilisateur speinfo1 l'ensemble des privilèges système regroupés dans le rôle **connect** (à savoir alter session, create cluster, create database link, create sequence, create session, create synonym, create table, create view),
- les autres rôles prédéfinis sont **resource** et **dba**.

9.4.5 Gestion de rollback segment

Création du rollback

```
create rollback segment RSEG
storage
(initial 50K next 50K minextents 2 maxextents 50)
tablespace speinfo ;
```

Une vue pour lister les rollback segments :

dba_rollback_segs ;

Rendre *online* un rollback segment :

```
rollback_segments=(RSEG)
```

dans le fichier de configuration de l'instance (initSpeinfo.ora par exemple).

```
alter rollback segment RSEG online ;
```

si la base est ouverte (sous le compte system ou sous svrmgrl).

Table des matières