République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Hassiba BENBOUALI de CHLEF
Faculté des Sciences

Filière : Informatique
Niveau : Master 1
Spécialités : IL & ISIA

Base de données avancées

Polycopié du cours

2015-2016

Dr Nassim DENNOUNI
DEPARTEMENT D'INFORMATIQUE

Table des matières

Préambule 5
Introduction générale 6
Chapitre I. Base de données relationnelles 7
Introduction 7
I. Base de données 7
I.1. Définition 7
I.2. Objectifs des bases de données 7
II. Système de Gestion de Base de Données 8
II.1. Définition 8
1I.2. Evolution des SGBD 8
a) Navigationnels 8
b) Relationnels 8
c) Post-relationnels 8
III. Les formes normales 10
III.1.Première Forme Normale (1FN) 10
III.2.Deuxième Forme Normale (2FN) 10
III.3.Troisème Forme Normale (3FN) 10
IV. SQL : Structured Query Language 10
IV.1. Langage de définition de données : LDD 11
a).Créer une table 11
b).Modification de la structure d'une table 11
c).Suppression de tables 11
IV.2. Langage de Manipulation de Données : LMD 12
Conclusion du chapitre I 12
Chapitre II. Base de données objet-relationnelles 13
Introduction 13
I. Les SGBDR 13
I.1.Les avantages du modèle relationnel 13
I.2. Les faiblesses du modèle relationnel : 14
II. Les SGBDOO 14
II.1.Les avantages des SGBDOO 14
II.2.Les inconvénients des SGBDOO 15
II.3.Le passage entre SGBDR/SGBDRO/SGBDOO 15
II.3.1.Pourquoi ne pas passer directement du SGBDR vers SGBD OO ? 15

II.3.2. SGBDOO ou SGBDRO ? 15
II.3.3.Discussions : 15
III. Les SGBDRO 16
III.1.Modèles NF2 16
III.2.Types utilisateurs 18
III.3.Les tables imbriquées dans le modèle relationnel-objet 19
a) Alias obligatoire 22
b) Extension THE 22
III.4. Les tables objets 23
IV. Le passage conceptuel vers relationnel-objet 25
IV.1.Classe 25
IV.2. Attributs et méthodes 26
a) Attributs composites 26
b) Attributs multi-valués 26
c) Attributs dérivés et méthodes 26
IV.3.Association 1 : N et 1 :1 27
IV.4.Association N : M 27
IV.5. Composition 27
IV.6.Héritage 28
V. Le modèle Objet relationnel et le SQL3 28
V.1.Caractéristiques d'une table objet-relationnel 28
V.2.Création de type objet 29
V.2.1.Utilisation du type dans un autre type 29
V.2.2.Utilisation d'un type utilisateur dans une table relationnelle 29
V.2.3.Utilisation d'un type objet dans une table relationnelle 30
V.3.Interrogation dans une table objet relationnelle 31
V.4. Définition de types utilisant des références (ou pointeurs) 33
V.5.Fonction DEREF et VALUE 33
V.6.Simulation de l'héritage 34
a) Sous-type et sur-type 34
b) Tables et sous-tables 34
V.7.Tables imbriquées 35
V.8.Les Tableaux (ou VARRAY) 38
Conclusion du chapitre II 42
Chapitre III. Les données semi-structurées 43

Introduction 43
I. Le langage XML 43
I.1.Définition d'une DTD 44
I.2. Validation d'un document XML 45
I.2.1.Validation syntaxique d'un document XML 45
I.2.Validation d'un document XML par une DTD 45
II. Bases de données et XML 49
II.1.Utilisation d'une base de donnée relationnelle 49
II.2.Utilisation des bases de données intégrant des types XML 50
II.3.Mapping Relationnel/XML 50
III. Le passage entre le Relationnel et XML 51
III.1.Le passage du Relationnel vers le XML 51
III.2.Le passage du XML vers le Relationnel 52
IV. Bases de données XML natives 54
Conclusion du chapitre III 54
Chapitre IV. Les bases de données réparties 55
Introduction 55
I. SGBD réparti 55
I.1.12 règles de Date 55
I.2.Fragmentation verticale : 56
I.3.Fragmentation horizontale 56
II. Migration vers une BDR 56
II.1.Décomposition physique en BD locales 56
II.2.Intégration logique des BD locales existantes 57
III. Modèles d'intégration de bases de données 57
III.1. Intégration en relationnel 57
III.2.Intégration en objet 58
III.3.Entrepôt de données 58
III.4.Médiateurs d'information 58
III.5.Réseaux Pair-à-pair 59
Conclusion du chapitre IV 59
Chapitre V. La gestion des accès concurrents 60
Introduction 60
I. Le concept de transaction 60
I.1.Propriétés d'une transaction 61

I.2. Les opérations élémentaires d'une transaction 61
I.3.Etats d'une transaction 62
II. Les accès concurrents 62
II.1. Les systèmes multitâches 62
II.2. Problèmes posés par les transactions concurrentes 63
a) Perte de mise à jour 64
b) Lecture impropre (Dirty read) 64
c) Lecture non reproductible 65
III. Control des accès concurrents 65
III.1. Notion d'équivalence d'exécution 66
III.2.Des opérations conflictuelles 66
III.3.La sérialisabilité 67
III.4.Utilisation des verrous 69
a) Collision de type perte de mise à jour : le problème 70
b) Collision de type perte de mise à jour : souhait 70
III.5.Estampillage 73
Conclusion du chapitre IV 74
Conclusion générale 75
TD 1-1 : Diagrammes Conceptuels 76
TD 1-2 : Normalisation de schéma conceptuel 77
TD 2 : L'objet-relationnel 78
TD 3 : Les données semi structurées 80
TD4 : Base de données réparties 81
TD5 : Les transactions et les accès concurrents 82
TP 1 : BD relationnel sous Oracle 83
TP 2 : BD objet-relationnelle sous oracle 85
TP 3 : Données semi-structurées 98
TP 4 : Gestion de la répartition des données 101
TP 5 : Gestion des accès concurrents sous oracle 102
Accès e-learning au module 109
Bibliographie 111

Préambule

Ce polycopié aborde les limites du modèle relationnel et donne une ouverture sur les modèles post-relationnels. Il traite les concepts suivants : modèle relationnel, langage SQL, intégrité et gestion des transactions, bases de données post-relationnelles, bases de données semi-structurées, bases de données réparties, bases de données objets, bases de données géographiques, base de données multimédia, etc.

Ce cours permets aussi d'acquérir des savoir-faire comme la modélisation conceptuelle des données en utilisant le modèle E/A, le passage vers le relationnel objet, la mise en œuvre d'une base de données Oracle et l'utilisation des standards SQL et PL/SQL pour la création des schémas de base de données objets, l'alimentation et la manipulation des données, etc.

Dans ce contexte, ce module vise à préparer les apprenants à la conception et l'implémentation des bases de données relationnelles ou post-relationnelles. A l'issue de ce cours, les apprenants doivent être capables : (1) d'utiliser une méthodologie de conception de base de données, (2) de maîtriser des éléments d'architecture logique et physique d'une base de données relationnelles ou post-relationnelles et (3) de gérer les accès concurrents.

Ce polycopié est destiné aux étudiants inscrits en première année de master en informatique dans l'une des deux spécialités : Ingénierie des Logiciels (IL) ou Ingénierie des Systèmes d'Information Avancés (ISIA). Ce support concerne un cours du 2ème semestre qui comporte une séance de cours et une séance de TD pour chaque groupe.

D'autre part, le nombre d'heures d'enseignement associé à ce module est de 42 H réparti sur 14 semaines. Chaque semaine comporte 1h30 de TP et 1h30 de travail personnel (sous forme de TP) pour l'étudiant. Enfin, le nombre de crédits associé à cette matière est égal à 5 crédits et le coefficient qui lui a été affecté est égal à 3 (Mode d'évaluation : Examen écrit, travaux personnels notés).

Introduction générale

Le chapitre I commence par définir les Base de Données de façon générale puis il explique l'évolution historique des SGBD depuis les fichiers vers les schémas NoSQL. Ce chapitre donne aussi un aperçu sur les différentes formes à utiliser pour la normalisation des schémas relationnels. Enfin, ce chapitre conclut par un survol des possibilités offertes par les langages de définition et de manipulation des données à l'aide du SQL.

Le chapitre II introduit les différentes faiblesses des SGBD relationnels du point du vue sémantique afin de mieux justifier les nouvelles perspectives des bases de données. Ces perspectives ont permis une évolution vers un nouveau modèle de bases de données post relationnels et la naissance des SGBD-OO (Orientés Objet). Cependant, peu d'utilisateurs se sont familiarisés à l'utilisation de ce type de système qui se base essentiellement sur des langages procéduraux difficiles à apprendre par les non informaticiens. Pour cette raison, les SGBD objets-relationnels sont considérés comme une solution qui profite de la simplicité d'utilisation du modèle relationnel et de la richesse sémantique du modèle objet.

Le Chapitre III présente les données semi structurées comme une solution à l'interopérabilité des SGBD, le partage des formats d'échange et l'intégration des données issues du processus d'importation à partir de plusieurs BD hétérogènes. Ces données permettent de traiter de prendre en compte des structures peuvent être irrégulière ou incomplète. Cela est particulièrement bien adapté aux nouvelles applications qui ont besoin de gérer des éléments qui proviennent souvent de plusieurs bases distinctes provenant du Web au format HTML et XML. Ce chapitre aborde le mappage entre les bases de données relationnelles et les documents XML qui peut être orienté par une DTD ou simplement basé sur des règles syntaxiques.

Le Chapitre IV aborde les bases de données réparties car elles ont une architecture plus adaptée à l'organisation des entreprises et sont considérée comme un bon exemple d'approche décentralisées. Cette approche permet d'avoir plus de fiabilité car la panne d'un site n'a pas d'impact sur l'accès aux données (le système peut s'adresser à autre site pour assurer la disponibilité de la base de données).

Le Chapitre V traite le control des accès concurrents dans un contexte où plusieurs utilisateurs peuvent accéder à plusieurs bases de données. En effet, la plupart des systèmes de gestion de base de données (SGBD) actuels sont des systèmes multi-utilisateurs : Cette utilisation multi-utilisateurs de systèmes informatiques pose des problèmes quant à la cohérence et l'intégrité de la base de données. Cette notion de partage de ressources est très intéressante pour l'utilisateur et constitue un enjeu majeur dans un SGBD qui devra gérer les utilisations concurrentes sur les données avec le plus d'efficacité possible

Chapitre I. Base de données relationnelles

Introduction

Une base de données (database) permet de stocker et de retrouver des données en rapport avec un thème ou une activité. Ces informations sont très structurées, et la base est localisée dans un même lieu et sur un même support.

La manipulation de données est une des utilisations les plus courantes des ordinateurs. Les bases de données sont par exemple utilisées dans les secteurs de la finance, des assurances, des écoles, de l'épidémiologie, de l'administration publique (statistiques notamment) et des médias.

I. Base de données

I.1. Définition

Une Base de Données (BD) est un ensemble de données structurées modélisant un univers précis et accessible à plusieurs utilisateurs en même temps

Les bases de données servent à stocker des informations sur un support informatique pendant une longue période de taille importante en autorisant des accès multi-utilisateurs. Il faut donc : gérer de manière efficace les accès aux disques et proposer une définition structurée des données afin d'éviter les redondances (Sans, 2000)

I.2. Objectifs des bases de données

1. **Élimination** de la redondance des données.
2. **Indépendance entre les programmes et les données** : on peut isoler le niveau utilisation (application) de l'organisation physique de données.
3. **Intégration de données** : intégration de toutes les données de l'entreprise dans un réservoir unique de données.

II. Système de Gestion de Base de Données

II.1. Définition

Un Système de Gestion de Base de Données (SGBD) est un ensemble de programmes qui permettent à l'utilisateur d'une Base de Données :

- de la créer (LDD : Langage de Définition de Données),
- de la manipuler (LMD : Langage de Manipulation de données)
- et de la contrôler (LCD : Langage de Contrôle de Données).

Ex. : Oracle[1], PostgreSQL[2], Access[3], MySQL[4], SQL server[5], DB2[6], Informix[7] etc.

1I.2. Evolution des SGBD

Les progrès de la technologie des processeurs mais surtout des mémoires, des disques et des réseaux ont permis de développer de manière considérable les performances des systèmes de gestion de base de données. L'évolution du domaine a suivi l'évolution des modèles de données utilisés. On peut considérer trois grandes classes de modèles : navigationnels, déclaratifs/relationnels, et post-relationnels (Wikipédia, Base de données relationnelle, 2016).

a) Navigationnels

Les deux modèles principaux de cette période sont le modèle hiérarchique d'IBM et le modèle réseau, CODASYL. Dans ces modèles, on "navigue" dans les données en manipulant des pointeurs logiques. La distance entre la représentation logique des données et leur représentation physique est faible.

b) Relationnels

Entre 1965 et 1975, avec le développement de l'informatique dans les grands comptes, le besoin d'organiser les données selon un modèle qui permettrait d'établir une séparation plus claire entre la représentation logique des données et leur organisation physique s'est fait de plus en plus sentir. Le modèle de données relationnel a été défini en 1970 par l'informaticien britannique d'IBM Edgar F. Codd, et publié dans son article A Relational Model of Data for Large Shared Data Banks. Depuis les années 1990, le modèle de données relationnel est utilisé dans la grande majorité des bases de données. Il reste toujours dominant en 2015.

c) Post-relationnels

Plusieurs directions ont été suivies pour dépasser le modèle relationnel. Partant du constat que la première forme normale est une contrainte très importante du modèle relationnel. Différentes suggestions fondées sur la notion de N1NF (Non First Normal Form) ont été proposées pour permettre de gérer des informations structurées et s'affranchir du fait que le modèle relationnel

[1] https://fr.wikipedia.org/wiki/Oracle_Database
[2] https://fr.wikipedia.org/wiki/PostgreSQL
[3] https://fr.wikipedia.org/wiki/Microsoft_Access
[4] https://fr.wikipedia.org/wiki/MySQL
[5] https://fr.wikipedia.org/wiki/Microsoft_SQL_Server
[6] https://fr.wikipedia.org/wiki/IBM_DB2
[7] https://fr.wikipedia.org/wiki/Informix

met l'ensemble des informations « à plat ». Cela a conduit à introduire la notion d'objets persistants. Deux approches se sont affrontées : La première approche consiste à utiliser l'existant et à l'améliorer (approche supportée par exemple par Stonebraker) ; et la seconde à introduire la notion de persistance dans les langages de programmation orientés objet (approche supportée par des universitaires comme Atkinson ou Bancilhon). Ces deux approches ont permis le développement de bases de données dites relationnelles à objets pour la première approche et de bases de données orientées objet pour la seconde.

Dans une direction complètement différente, une approche a consisté à introduire dans les bases de données des capacités de déduction empruntées aux systèmes experts, capables de grandes capacités déductives mais pas très performants sur la gestion de gros volumes de données. Cette approche, avec des systèmes fondés sur le langage Datalog, n'a jusqu'à présent eu que peu de succès dans l'industrie même si on a pu assister à un certain retour de cette approche depuis 2010.

Depuis 2010 également, le mouvement NoSQL vise à développer des applications de gestion de données massives en utilisant des modèles N1NF. Les modèles utilisés sont des modèles clé-valeur (associant une valeur à chaque clé) ou des bases de documents. Le tableau ci-dessous résume les principales phases d'évolution des SGBD.

Tableau 1 : historique des SGBD

Historique	**SGBD**	**Caractéristiques fonctionnelles**
1955-1960	SGF	- BD = fichiers reliés par des pointeurs
1960-1980 SGBD Hiérarchique et Réseaux	IMS (hiérarchique), CODASYL (réseau)	- Séparation de la description des données et des programmes d'application - Traitement de fichiers reliés par des structures de graphe
1970 SGBD Relationnel	Oracle, DB2, Informix, Sybase, SQL Server	- Emergence du modèle relationnel dans les laboratoires de recherche - Formulation du langage SQL, basé sur le langage algébrique de CODD
1980 SGBD objet	O2 Objectivity	- Intégration des données et des traitements (procédure, fonction) qui les manipulent grâce à la notion d'objet
1980 SGBD Objet Relationnel	Oracle 8, DB2, Informix	- Une structuration conjointe des programmes et des données tout en conservant les acquis du relationnel
1990-2000	Oracle SQL server	- Supporter les informations non-structurées, provenant du Web, des multimédias etc. - SGBDS orientés aide à la prise de décision, et extraction des connaissances (Data Mining)
2010	*Apache Hadoop*	Les données deviennent très volumineuses, variées et véloces. On parle de BIG DATA

III. Les formes normales

III.1.Première Forme Normale (1FN)

Une relation est en 1FN si tous les attributs qui la composent sont non décomposables.

Exemple :

Personne (numP, patronyme) → Personne (numP, nom, prénom)

III.2.Deuxième Forme Normale (2FN)

Une relation est en 2FN si elle est en 1FN et si toutes les DFs entre la clé et les autres attributs sont élémentaires.

Exemple :

Employé (numE, numP, nomE, temps) → Employé (numE, nomE)

→ TempsProjet (numE, numP, temps)

III.3.Troisème Forme Normale (3FN)

Une relation est en 3FN si elle est en 2FN et si toutes les DFs entre la clé et les autres attributs sont élémentaires et directes.

Exemple :

Fournisseur (numF, nomF, NumP, prixP)→ fournisseur (numF, nomF, NumP)

→ Produit (NumP, prixP)

IV. SQL : Structured Query Language

SQL est un langage pour les BDR. Créé en 1970 par IBM. Ses principales caractéristiques sont :

1. **Normalisation** : SQL implémente le modèle relationnel.
2. **Standard** : la plupart des éditeurs de SGBDR intègrent SQL à leurs produits (Oracle, MS SQL Server, DB2, etc.) pour que les données, requêtes et applications soient facilement portables d'une BD à une autre.
3. **Non procédural** : SQL est un langage déclaratif non procédural permettant d'exprimer des requêtes dans un langage relativement simple. En contrepartie il n'intègre aucune structure de contrôle permettant par exemple d'exécuter une boucle itérative.

D'autre part, le SQL peut être utilisé à tous les niveaux dans la gestion d'une BDR :

Le langage de définition de données (LDD) : permet la description de la structure de la base de données (tables, vues, attributs, index).

Le langage de manipulation de données (LMD) : permet la manipulation des tables et des vues avec les quatre commandes : SELECT, INSET, DELETE, UPDATE.

Le langage de contrôle de données (LCD) : comprend les primitives de gestion des transactions : COMMIT, ROLLback et des privilèges d'accès aux données : GRANT et REVOKE.

IV.1. Langage de définition de données : LDD

La définition de données dans SQL permet la définition des objets manipulés par le SGBD. Ces objets sont des tables, vues ou index. Les commandes du LDD sont :

- **CREATE** : création des objets.
- **ALTER** : modification de la structure des objets.
- **DROP** : suppression des objets.

a).Créer une table

```
CREATE TABLE Produit
(Numprod number (6) not null,
 Desprod varchar (15) unique,
 Couleur char,
 Poids number (8,3),
 Qte_stk number (7,3),
 Qte_seuil number (7,3),
 Prix number (10,3),
CodMag number(5,3),
Constraint Ck1_Produit CHECK (Poids >=0),
Constraint PK_Produit primary key (NumProd),
Constraint FK_Produit Foreing Key (CodMag) references Magasin (NumMag));
```

b).Modification de la structure d'une table

- Ajout de nouvelles colonnes à une table

```
ALTER TABLE CLIENT ADD type_clt char(3) ;
```

- Modification de la structure d'une colonne existante

```
ALTER TABLE CLIENT MODIFY type_clt char(5) ;
```

- Suppression de colonnes existantes

```
ALTER TABLE Magasin DROP ville ;
```

- Ajout d'une contrainte

```
ALTER TABLE Magasin ADD Constraint ck1_magasin check (surface between 10 and 100) ;
```

- Suppression de contraintes existantes

```
ALTER TABLE magasin DROP PRIMARY KEY CASCADE ;
ALTER TABLE produit DROP CONSTRAINT Ck4_Produit ;
```

c).Suppression de tables

DROP TABLE nom_table ;

IV.2. Langage de Manipulation de Données : LMD

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT /UNIQUE après le mot SELECT.

SELECT UNIQUE /UNIQUE ma_colonne FROM nom_du_tableau ;

Dans le langage SQL il est possible d'utiliser des alias pour renommer temporairement une colonne ou une table dans une requête. Cette astuce est particulièrement utile pour faciliter la lecture des requêtes.

Alias sur une colonne SELECT colonne1 **AS c1**, colonne2 FROM `table`;

Alias sur une table SELECT *FROM `nom_table` AS **t1**; SELECT *FROM `table`t1 ;

Renommer une table SELECT p_id, p_nom_fr, pc_id, pc_nom_fr_fr FROM `produit` AS **p**

SELECT nom_colonnes FROM nom_table WHERE condition1 AND (condition2 OR condition3)

SELECT nom_colonne FROM table WHERE nom_colonne IN (valeur1, valeur2, valeur3)

SELECT prenom FROM utilisateur WHERE prenom = 'Maurice' OR prenom = 'Marie' OR prenom = 'Thimoté'

SELECT prenom FROM utilisateur WHERE prenom IN ('Maurice', 'Marie', 'Thimoté')

SELECT * FROM table WHERE nom_colonne BETWEEN 'valeur1' AND 'valeur2'

SELECT * FROM table WHERE colonne LIKE 'N%'

L'opérateur IS permet de filtrer les résultats qui contiennent la valeur NULL. Cet opérateur est indispensable car la valeur NULL est une valeur inconnue et ne peut par conséquent pas être filtrée par les opérateurs de comparaison (cf. égal, inférieur, supérieur ou différent).

SELECT *FROM `table`WHERE nom_colonne IS NULL

Conclusion du chapitre I

Après avoir fait un tour d'horizon sur le modèle relationnel, ses avantages et ses atouts. Le prochain chapitre va mettre l'accent sur les faiblesses de ce modèle afin d'introduire une nouvelle structure des bases de données post-relationnelles.

Chapitre II. Base de données objet-relationnelles

Introduction

Si le modèle logique relationnel a prouvé sa puissance et sa fiabilité au cours des 20 dernières années, les nouveaux besoins de l'informatique industrielle ont vu l'émergence de structures de données complexes mal adaptées à une gestion relationnelle (CROZAT, 2014).

La naissance du courant **orienté objet** et des langages associées (Java et C++ par exemple) ont donc également investi le champ des SGBD afin de proposer des solutions pour étendre les concepts du relationnel et ainsi mieux répondre aux nouveaux besoins de modélisation (CROZAT, 2014).

I. Les SGBDR

Le modèle relationnel a été définit par E. Codd en1970, il est basé sur la théorie des ensembles et assure l'indépendance entre les programmes d'applications et la représentation interne des données et il fournit une base solide pour traiter les problèmes d'incohérence et de redondance (Kaabi, 2011). Le schéma relationnel définit la structure de la relation où les n-uplets représentent les différents éléments. Les relations respectent des propriétés définies à partir des dépendances fonctionnelles (en général, troisième forme normale (3NF)) (Sans, 2000).

I.1.Les avantages du modèle relationnel

Selon (CROZAT, 2014) (Sans, 2000), les atouts du modèle relationnel peuvent se résumer comme suit :

1. C'est un modèle fondé sur une théorie rigoureuse et des principes simples. Il est aussi fiable et performant car il assure l'indépendance entre les programmes et données
2. Les SGBDR sont les plus utilisés, connus et maîtrisés.
3. L'utilisation du SQL pour une implémentation standard du modèle relationnel, avec des API pour la plupart des langages de programmation.
4. Les SGBDR incluent des outils performants de gestion de requêtes, de générateurs d'applications, d'administration, d'optimisation, etc.
5. C'est modèle bien implanté dans le monde professionnel car il permet une organisation structurée des données et un accès par des utilisateurs concurrents.

I.2. Les faiblesses du modèle relationnel :

Parmi les faiblesses de ce modèle, on peut citer (CROZAT, 2014) (Kaabi, 2011) (Sans, 2000) :

1. l'absence de pointeurs visibles (jointure par valeurs à éviter : opération lourde et coûteuse, chaînage direct des données : à considérer)
2. le non support de domaines composés (on ne peut pas avoir pas exemple un attribut qui correspond à une adresse avec le numéro de la rue, le nom de la rue, la ville, … cela est impossible à cause de la première forme normale, qui impose l'atomicité des attributs).
3. la non intégration des opérations (la partie dynamique) car la notion de méthode ne peut être intégrée au modèle logique, elle doit être gérée au niveau de l'implémentation physique.
4. le mapping MCD vers MLD entraîne une perte de sémantique car la structure de donnée en tables est pauvre d'un point de vue de la modélisation logique
5. la manipulation de structures relationnelles par des langages objets entraîne une un décalage entre les structures de données pour le stockage et les structures de données pour le traitement.
6. la 1FN est inappropriée à la modélisation d'objets complexes car la normalisation entraîne la genèse de structures de données complexes et très fragmentées, qui peuvent notamment poser des problèmes de performance ou d'évolutivité
7. Ne peut pas traiter des éléments de structure complexe, des opérations sur les éléments des pointeurs reliant les éléments (pour de l'héritage par exemple) car les types de données disponibles sont limités et non extensibles

II. Les SGBDOO

Un SGBDOO (Orienté Objet) doit d'abord supporter les fonctionnalités d'un SGBD comme la persistance des objets, la concurrence d'accès, la fiabilité des objets et la facilité d'interrogation (Kaabi, 2011). En effet, les SGBDOO ont été créés pour gérer des structures de données complexes, en profitant de la puissance de modélisation des modèles objets et de la puissance de stockage des BDs classiques. Ce type de SGBD a pour objectifs (CROZAT, 2014):

- d'offrir aux langages de programmation orientés objets des modalités de stockage permanent et de partage entre plusieurs utilisateurs
- d'offrir aux BD des types de données complexes et extensibles
- de permettre la représentation de structures complexes et/ou à taille variable.
- Fournir un ensemble de fonctionnalités OO (Support d'objets atomiques et complexes, Identité d'objets, Héritage simple et polymorphisme)

II.1.Les avantages des SGBDOO

Les avantages des SGBDOO peuvent être résumés comme suit (CROZAT, 2014) :

1) Le schéma d'une BD objet est plus facile à appréhender que celui d'une BD relationnelle (il contient plus de sémantique, il est plus proche des entités réelles)
2) L'héritage permet de mieux structurer le schéma et de factoriser certains éléments de modélisation
3) La création de ses propres types et l'intégration de méthodes permettent une représentation plus directe du domaine

4) L'identification des objets permet de supprimer les clés artificielles souvent introduites pour atteindre la 3FN et donc de simplifier le schéma
5) Les principes d'encapsulation et d'abstraction du modèle objet permettent de mieux séparer les BD de leurs applications (notion d'interface).

II.2.Les inconvénients des SGBDOO

Les SGBDOO présentent certains inconvénients comme (CROZAT, 2014):

1. La gestion de la persistance et de la coexistence des objets en mémoire (pour leur manipulation applicative) et sur disque (pour leur persistance) complexe
2. La gestion de la concurrence (transactions) plus difficile à mettre en œuvre
3. L'interdépendance forte des objets entre eux
4. La gestion des pannes
5. La complexité des systèmes (problème de fiabilité)
6. Le problème de compatibilité avec les SGBDR classiques

II.3.Le passage entre SGBDR/SGBDRO/SGBDOO

II.3.1.Pourquoi ne pas passer directement du SGBDR vers SGBD OO ?

Le relationnel a ses avantages, en particulier sa grande facilité et efficacité pour effectuer des recherches complexes dans des grandes bases de données. En outre, ce modèle facilite la spécification des contraintes d'intégrité sans programmation car il est basé sur une théorie solide et des normes reconnues. Pour cette raison, il existe de de très nombreuses bases relationnelles en fonctionnement. D'autre part, les SGBDO manquent de normalisation pour et s'appuient sur beaucoup de solutions propriétaires. En outre, les SGBDOO sont moins souple que le relationnel et peu d'informaticiens sont formés dans ce domaine. Pour toutes ces raisons, le modèle OR peut permettre un passage en douceur.

II.3.2. SGBDOO ou SGBDRO ?

On peut constater que les SGBDOO sont plus "propres" du point de vue objet et les mieux adaptés pour traiter les objets mais ils sont complètement absents du monde professionnel. D'autre part, les SGBDRO sont basés sur des SGBD robustes et éprouvés répandus dans le monde professionnel mais qui ne sont pas prévus pour gérer l'objet. Pour cette raison, les concepteurs et les utilisateurs de bases de données ne sont pas prêts à remettre en cause leurs savoirs et à redévelopper toutes leurs applications sur de nouveaux systèmes et donc ils vont privilégier les SGBDRO (Sans, 2000).

II.3.3.Discussions :

Il existe deux manières d'utiliser l'objet dans les SGBD : (1) soit on part des langages objet dans lesquels on intègre les notions des SGBD (persistance des données, aspect multi-utilisateurs, partage de données,…), ce sont les SGBD orientés objet comme o2 (basé sur c++) ou (2) on part des SGBD relationnels dans lesquels on insère des notions objet, ce sont les SGBD relationnels objet : oracle 8 (SQL 3) (Sans, 2000). Dans ce qui suit, on va s'intéresser de près à la deuxième solution qui s'appuie sur l'intégration des notions objets dans des SGBDR existants pour donner naissance à des SGBDRO.

III. Les SGBDRO

Les SGBDRO sont nés du double constat de la puissance nouvelle promise par les SGBDOO et de l'insuffisance de leur réalité pour répondre aux exigences de l'industrie des BD classiques. Leur approche est plutôt d'introduire dans les SGBDR, les concepts apportés par les SGBDOO plutôt que de concevoir de nouveaux systèmes (CROZAT, 2014). Cette approche intègre des éléments de structure complexe dans une relation de type NF2 (Non First Normal Form) et pourra contenir un attribut composé d'une liste de valeurs ou de plusieurs attributs (Sans, 2000).

Dans ce contexte, les SGBDRO peuvent (1) gérer des données complexes (temps, géo-référencement, multimédia, types utilisateurs, etc.), (2) rapprocher le modèle logique du modèle conceptuel et (3) réduire les pertes de performance liées à la normalisation et aux jointures (Sans, 2000).

Par rapport au modèle relationnel standard : on a moins besoin d'aplatir les données, ici on a une seule table avec des tables imbriquées au lieu de 3 tables on a moins besoin de faire des jointures pour récupérer les informations sur les accidents du contrat 1111, on accède simplement à l'attribut accidents.

N° contrat	Nom	Adresse	Conducteurs	Accidents
1111	J. CHIRAC	Elysée 75000 PARIS	Nom: Bernadette; Age: 70	N° contrat: 375; Personne: Nicolas; Date: 2005
...	...	...	...	...

Figure 1: exemple d'une structure issue du modèle objet relationnel

III.1.Modèles NF2

L'introduction de modèles NF2 (ou non first NF) permet de se libérer de la 1ère Forme Normale afin d'avoir la possibilité de définir des attributs comme des listes de valeurs. Pour cette raison, de nouveaux opérateurs doivent être introduits : NEST et UNNEST.

D'autre part, l'extension du LDD pour définir des types structurés engendre la modification des opérateurs de base comme la restriction, la projection et la jointure.

Dans ce contexte, de nouveaux opérateurs (voir la figure 2) : NEST et UNNEST sont définit comme suit : (1) NEST permet de regrouper ensemble les composants (1FN →NF2) et (2) UNNEST permet décompose et ramène en 1ère Forme Normale (NF2 →1FN).

Pour atteindre cet objectif, l'extension des systèmes relationnels nécessite de modifier le noyau de base du SGBD pour supporter l'intégration des objets complexes et leurs manipulations. Par conséquent, le langage de définition de données (LDD) doit évoluer pour créer
des relations définies à partir de ces nouveaux types

Pièce	Nap	Npiece	Nom	Superficie
	1	1	Salon	30
	1	2	Salle à diner	20
	1	3	Cuisine	10
	1	4	Chambre	20
	1	5	Salle de bain	6
	2	2	Chambre	15
	2	1	Salle	40

Nest

Unnest

Appartement	Nap	Pieces
	1	<1,salon,30>, <2,Salle à diner, 20>, <3, cuisine, 10>, …
	2	<1, Salle, 40>, <2,chambre,15>,…

Figure 2: les opérateurs NEST et UNNEST

D'autre part, le modèle objet-relationnel (OR) ou le modèle relationnel-objet (RO) est un modèle relationnel étendu avec des principes objet pour en augmenter les potentialités (Informix, Oracle, Sybase, IBM/DB2, CA-OpenIngres, PostGres, ...). Il offre de nouvelles possibilités comme :

1. La définition de nouveaux types complexes avec des fonctions pour les manipuler (une colonne peut contenir une collection (ensemble, liste) et une ligne considérée comme un objet avec un objet)
2. Utilisation de références aux objets ou des Identificateur (*Object Identifier* OID)
3. Extensions du langage SQL (SQL3) pour la recherche et la modification des données

Cependant, l'OR ne s'appuie pas sur une théorie solide comme le modèle relationnel et il manque d'implémentation standard de SGBD. Le concept central du RO est celui de type utilisateur, correspondant à la classe en POO, qui permet d'injecter la notion d'objet dans le modèle relationnel. Les apports principaux des types utilisateurs sont :

- La gestion de l'imbrication (données structurées et collections)
- Les méthodes et l'encapsulation
- L'héritage et la réutilisation de définition de types
- L'identité d'objet et les références physiques

D'autre part, le modèle RO apporte deux nouveautés distinctes et indépendantes à ne pas confondre :

1. Les tables imbriquées (Nested model) qui permettent de dépasser les limites de la première forme normale et de limiter la fragmentation de l'information ;
2. Les tables objets qui permettent d'ajouter les identifiants d'objets (OID), les méthodes et l'héritage aux tables classiques.

Dans ce qui suit, on va détailler les différents concepts liés au modèle RO :

III.2.Types utilisateurs

Parmi les types standards existant dans les BD classiques, on peut citer le type VARCHAR, le type NUMBER… ces types sont prédéfinis par le SGBDR. Les SGBDRO donne la possibilité de définir de nouveaux types utilisateurs définis par le concepteur de la base afin d'être utilisés comme des types standards. On appelle aussi ces types utilisateurs des types objet car ils ont une structure complexe et peuvent contenir des opérations (méthodes). (Sans, 2000)

Le Type de données utilisateur ou le Type de données abstrait est un Type de données créé par le concepteur d'un schéma relationnel-objet, qui encapsule des données et des opérations sur ces données. Ce concept se rapproche de celui de la classe d'objets.

```
Type nom_type : <
 attribut1:typeAttribut1,
 attribut2:typeAttribut2,
 ...
 attributN:typeAttributN,
 =methode1:(paramètres) typeRetourné1,
 =methode2:(paramètres) typeRetourné2,
 =...
 =methodeN:(paramètres) typeRetournéN
>
```

Figure 3: Syntaxe au niveau logique d'un type défini par l'utilisateur

```
CREATE TYPE nom_type AS OBJECT (
 nom_attribut1 type_attribut1
 ...
 MEMBER FUNCTION nom_fonction1 (parametre1 IN|OUT
type_parametre1, ...) RETURN type_fonction1
 ...
) [NOT FINAL];
/
CREATE TYPE BODY nom_type
 IS
 MEMBER FUNCTION nom_fonction1 (...) RETURN type_fonction1
  IS
  BEGIN
  ...
  END ;
 MEMBER FUNCTION nom_fonction2 ...
  ...
 END ;
END ;
/
```

Figure 4 : Syntaxe de déclaration de type

```
1  CREATE TYPE sous_type UNDER sur_type (
2   Déclarations spécifiques ou surcharges
3  ) ;
```

Figure 5 : Syntaxe de l'héritage de type

Pour être héritable, un type doit être déclaré avec la clause optionnelle NOT FINAL.

❖ Un type avec référence

– CREATE TYPE WITH OID phone (country VARCHAR, area VARCHAR, number int, description CHAR (20))

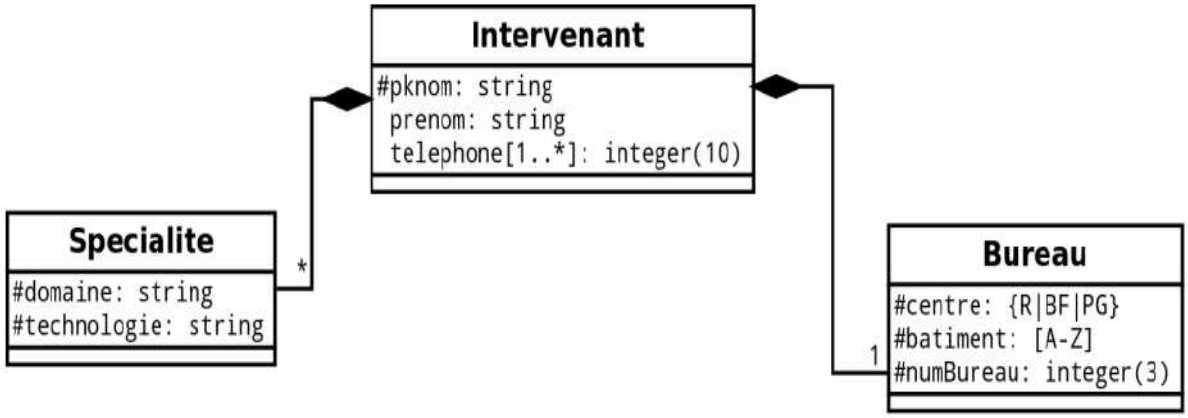
❖ • Un type sans référence

– CREATE TYPE person (ncin INT, nom VARCHAR, telphone)

III.3.Les tables imbriquées dans le modèle relationnel-objet

pknom	prenom	bureau			liste-telephones	listes-specialites	
Crozat	Stéphane	centre	batiment	numero		Domaine	Technologie
		PG	K	256	0687990000	BD	SGBDR
					0912345678	Doc	XML
					0344231234	BD	SGBDRO
Vincent	Antoine	centre	batiment	numero		Domaine	Technologie
		R	C	123	0344231235	IC	Ontologie
					0687990001	Base de données	SGBDRO

```
1  Type typBureau : <centre:char, batiment:char, numero:int>
2  Type typListeTelephones : collection de <entier>
3  Type typSpecialite : <domaine:char, specialite:char>
4  Type typListeSpecialites : collection de <typSpecialite>
5  tIntervenant (#nom:char, prenom:char, bureau:typBureau,
   ltelephones:typListeTelephones, lspecialites:typListeSpecialites)
```



```
1  CREATE OR REPLACE TYPE typBureau AS OBJECT (
2  centre char(2),
3  batiment char(1),
4  numero number(3)
5  );
6  /
7
8  CREATE OR REPLACE TYPE typListeTelephones AS TABLE OF number(10);
9  /
10
11 CREATE OR REPLACE TYPE typSpecialite AS OBJECT (
12 domaine varchar2(15),
13 technologie varchar2(15)
14 );
15 /
16
17 CREATE OR REPLACE TYPE typListeSpecialites AS TABLE OF typSpecialite;
18 /
19
20 CREATE TABLE tIntervenant (
21 pknom varchar2(20) PRIMARY KEY,
22 prenom varchar2(20) NOT NULL,
23 bureau typBureau,
24 ltelephones typListeTelephones,
25 lspecialites typListeSpecialites
26 )
27 NESTED TABLE ltelephones STORE AS tIntervenant_ntl,
28 NESTED TABLE lspecialites STORE AS tIntervenant_nt2;
```

Figure 6: Modèle logique Cours et intervenants

```
1  INSERT INTO tIntervenant (pknom, prenom, bureau, ltelephones,
   lspecialites)
2  VALUES (
3  'Crozat',
4  'Stéphane',
5  typBureau('PG','K',256),
6  typListeTelephones (0687990000,0912345678,0344231234),
7  typListeSpecialites (typSpecialite ('BD','SGBDR'),
   typSpecialite('Doc','XML'), typSpecialite('BD','SGBDRO'))
8  );
```

Figure 7 : Exemple : Insert 1

```
9
10  INSERT INTO tIntervenant (pknom, prenom, bureau, ltelephones,
    lspecialites)
11  VALUES (
12  'Vincent',
13  'Antoine',
14  typBureau('R','C',123),
15  typListeTelephones (0344231235,0687990001),
16  typListeSpecialites (typSpecialite ('IC','Ontologies'),
    typSpecialite('BD','SGBDRO'))
17  );
```

Figure 8 : Exemple : Insert 2

```
1  SELECT pknom, prenom, i.bureau.centre FROM tIntervenant i;
```

```
1  PKNOM                PRENOM               BUREAU.CENTRE
2  -------------------- -------------------- -------------
3  Crozat               Stéphane             PG
4  Vincent              Antoine              R
```

Figure 9 : Exemple : Select (enregistrement imbriqué)

```
1  SELECT i.pknom, t.*
2  FROM tIntervenant i, TABLE(i.ltelephones) t
3
```

```
1  PKNOM                COLUMN_VALUE
2  -------------------- ------------
3  Crozat                  687990000
4  Crozat                  912345678
5  Crozat                  344231234
6  Vincent                 344231235
7  Vincent                 687990001
```

Figure 10 : Exemple : Select (collection de scalaires imbriquée)

```
1  SELECT i.pknom, s.*
2  FROM tIntervenant i, TABLE(i.lspecialites) s
```

```
1  PKNOM                DOMAINE         TECHNOLOGIE
2  -------------------- --------------- ---------------
3  Crozat               BD              SGBDR
4  Crozat               Doc             XML
5  Crozat               BD              SGBDRO
6  Vincent              IC              Ontologies
7  Vincent              BD              SGBDRO
```

Figure 11 : Exemple : Select (collection d'enregistrements imbriquée)

a) Alias obligatoire

```
1  SELECT alias_table.nom_objet.nom_attribut
2  FROM nom_table AS alias_table
```

Figure 12 : Syntaxe d'accès aux attributs des enregistrements imbriqués

```
1  SELECT pknom, prenom, i.bureau.centre FROM tIntervenant i;
```

```
1  PKNOM                PRENOM               BUREAU.CENTRE
2  -------------------- -------------------- -------------
3  Crozat               Stéphane             PG
4  Vincent              Antoine              R
```

Figure 13 : Exemple d'accès aux attributs des enregistrements imbriqués

b) Extension THE

La clause THE est une extension du LMD✯ permettant de manipuler les objets (scalaires ou enregistrements) dans les collections implémentées sous forme de tables imbriquées.

INSERT INTO THE (SELECT ...) VALUES (...)

DELETE THE (SELECT ...) WHERE ...

UPDATE THE (SELECT ...) SET ... WHERE ...

```
1  INSERT INTO tIntervenant (pknom, prenom, bureau, ltelephones,
   lspecialites)
2  VALUES (
3  'Dumas',
4  'Leonard',
5  typBureau('R','C',123),
6  typListeTelephones(0344234423),
7  typListeSpecialites()
8  );
9
10 INSERT INTO THE (SELECT i.ltelephones FROM tIntervenant i WHERE
   i.pknom='Dumas')
11 VALUES (0666666666);
12
13 INSERT INTO THE (SELECT i.lspecialites FROM tIntervenant i WHERE
   i.pknom='Dumas')
14 VALUES (typSpecialite('BD','SGBDR'));
```

Figure 14: Exemple : INSERT d'un scalaire ou d'un enregistrement dans une collection

```
1  DELETE THE (SELECT i.ltelephones FROM tIntervenant i WHERE
   i.pknom='Dumas') nt
2  WHERE nt.COLUMN_VALUE=0344234423;
```

Figure 15: Exemple : DELETE d'un objet dans une collection

```
1  UPDATE THE (SELECT i.lspecialites FROM tIntervenant i WHERE
   i.pknom='Dumas') nt
2  SET nt.technologie='SGBDRO'
3  WHERE nt.domaine='BD';
```

Figure 16: Exemple : UPDATE d'un objet dans une collection

III.4. Les tables objets

Une table objet peut être définie en référençant un type de données. Il est possible de spécifier les mêmes contraintes au moment de la création de la table, et non au moment de la création du type. Les enregistrements d'une table-objet peuvent être identifiés par un OID et des méthodes peuvent être associées à une table-objet. Ce type de table permet de profiter de l'héritage de type pour permettre l'héritage de schéma de table.

```
1  CREATE OR REPLACE TYPE typIntervenant AS OBJECT(
2  pknom varchar2(20),
3  prenom varchar2(20)
4  );
5  /
6
7  CREATE TABLE tIntervenant OF typIntervenant (
8  PRIMARY KEY(pknom),
9  prenom NOT NULL
10 );
```

Figure 17 : exemple de création de table objet relationnel

- Le modèle relationnel-objet permet de disposer d'identificateurs d'objet (OID).
- Une table objet dispose d'un OID pour chacun de ses tuples.
- L'OID est une référence unique pour toute la base de données qui permet de référencer un enregistrement dans une table objets.
- L'OID est une référence physique (adresse disque) construit à partir du stockage physique de l'enregistrement dans la base de données.

a)Avantages

- Permet de créer des associations entre des objets sans effectuer de jointure (gain de performance).
- Fournit une identité à l'objet indépendamment de ses valeurs (clé artificielle).
- Fournit un index de performance maximale (un seul accès disque).

b) Inconvénient

- Plus de séparation entre le niveau logique et physique.
- L'adresse physique peut changer si le schéma de la table change (changement de la taille d'un champ par exemple)
- Manipulation des données plus complexes, il faut un langage applicatif au-dessus de SQL pour obtenir, stocker et gérer des OID dans des variables.

c) Références entre enregistrements avec les OID

Les OID peuvent être une alternative aux clés étrangères pour référencer des enregistrements. SCOPE FOR Renforce l'intégrité référentielle en limitant la portée de la référence à une table particulière (alors que REF seul permet de pointer n'importe quel objet de ce type)

```
1  CREATE OR REPLACE TYPE typCours AS OBJECT(
2  pkannee number(4),
3  pknum number(2),
4  titre varchar2(50),
5  type char(2),
6  refintervenant REF typIntervenant,
7  debut date,
8  MEMBER FUNCTION fin RETURN date
9  );
10 /
11
12 CREATE TABLE tCours OF typCours (
13 CHECK (pkannee>2000 and pkannee<2100),
14 type NOT NULL,
15 CHECK (type='C' or type='TD' or type='TP'),
16 refintervenant NOT NULL,
17 SCOPE FOR (refintervenant) IS tIntervenant,
18 PRIMARY KEY(pkannee, pknum)
19 );
```

Figure 18 : exemple de l'utilisation de SCOPE et REF

d) Insertion de références par OID

Pour faire une référence avec un OID, il est nécessaire de récupérer l'OID correspondant à l'enregistrement que l'on veut référencer. L'OID est accessible grâce à la syntaxe REF en SQL3.

```
DECLARE
 refI REF typIntervenant;
BEGIN
 INSERT INTO tIntervenant (pknom, prenom)
 VALUES ('CROZAT', 'Stéphane');

 SELECT REF(i) INTO refI
 FROM tIntervenant i
 WHERE pknom='CROZAT';

 INSERT INTO tCours (pkannee, pknum, titre, type, debut,
refintervenant)
 VALUES ('2003', 1, 'Introduction','C', '01-JAN-2001', refI);

 INSERT INTO tCours (pkannee, pknum, titre, type, debut,
refintervenant)
 VALUES ('2003', 2, 'Modélisation','TD', '02-JAN-2001', refI);
END;
/
```

Figure 19 : exemple d'insertion de données en PL/SQL

e) Manipulation d'OID

La référence par OID permet la navigation des objets sans effectuer de jointure, grâce à la syntaxe suivante :

```
SELECT t.reference_oid.attribut_table2
FROM table1 t;
```

```
SELECT c.pkannee, c.pknum, c.refintervenant.pknom
FROM tCours c
```

Figure 20 : exemple de navigation d'objets

f) Méthodes de table objet

Si le type sur lequel s'appuie la création de la table définit des méthodes, alors les méthodes seront associées à la table (méthodes de table).Il sera possible d'accéder à ces méthodes de la même façon que l'on accède aux attributs (projection, sélection...)

```
CREATE OR REPLACE TYPE BODY typCours IS
MEMBER FUNCTION fin RETURN DATE
  IS
  BEGIN
    RETURN SELF.debut + 5;
  END;
END;
/

SELECT c.pkannee, c.pknum, c.fin()
FROM tCours c;
```

Figure 21 : exemple d'accès aux méthodes d'une table objet

Lorsque l'on écrit une méthode on a généralement besoin d'utiliser les attributs propres (voire d'ailleurs les autres méthodes), de l'objet particulier que l'on est en train de manipuler. On utilise pour cela la syntaxe SELF qui permet de faire référence à l'objet en cours.

g) Héritage et réutilisation de types

Un type de données utilisateur peut hériter d'un autre type de données utilisateur. Pour qu'une table hérite du schéma d'une autre table, il faut définir les tables depuis des types. L'héritage entre les types permet ainsi l'héritage entre les schémas de table.

IV. Le passage conceptuel vers relationnel-objet

Cette partie permet de traiter la traduction d'un modèle conceptuel UML en modèle relationnel-objet.

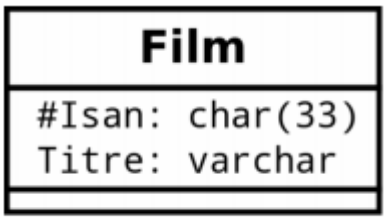
IV.1.Classe

Pour chaque classe (ou entité), créer un type d'objet avec les attributs de la classe (ou entité).Créer une table d'objets de ce type pour instancier la classe (ou entité), en ajoutant les contraintes d'intégrité.

Le fait d'instancier la classe (l'entité) via une table d'objets plutôt qu'une relation classique, permet l'accès à l'héritage de type, à l'implémentation des méthodes et éventuellement à l'usage

d'OID à la place de clés étrangères classiques. Si aucun de ces aspects n'est utilisé, il est possible d'instancier directement la classe (l'entité) en table

Exemple : proposer un modèle RO correspondant au modèle UML.

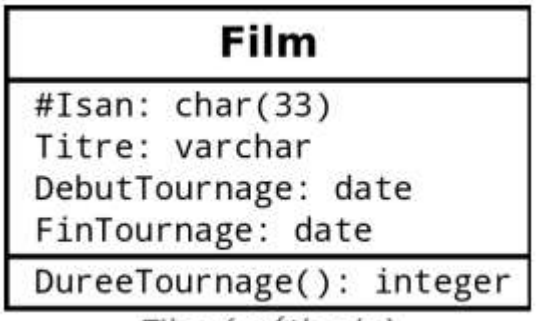


Film (classe)

IV.2. Attributs et méthodes

a) Attributs composites : pour chaque type d'attribut composé, créer un type d'objet.
*b) Attributs multi-valués : p*our chaque attribut multi-valué créer une collection du type de cet attribut.
c) Attributs dérivés et méthodes : pour chaque attribut dérivé et chaque méthode créer une méthode associée au type d'objet de la classe (l'entité).
Exemple : un film avec des attributs.

Film (attribut multi-valué)



Film (méthode)

```
1  CREATE TYPE FilmT AS OBJECT (
2  isan:char(33),
3  titre:varchar2,
4  debut:date,
5  fin:date,
6  MEMBER FUNCTION dureeTournage RETURN integer
7  );
8  /
9
10 CREATE TYPE BODY FilmT
11 IS
12 MEMBER FUNCTION duree RETURN integer)
13 IS
14 BEGIN
15 RETURN duree(SELF.debut,SELF.fin);
16 END;
17 END;
18 /
19
20 CREATE TABLE Film AS FilmT (
21 PRIMARY KEY (isan));
```

IV.3.Association 1 : N et 1 :1

Les associations 1 : N et 1:1 sont gérées comme en relationnel. On peut néanmoins favoriser l'usage d'objets pour les clés étrangères composées de plusieurs attributs, ainsi que pour les attributs des classes d'association. Il est aussi possible de gérer la clé étrangère avec un OID à la place d'une clé étrangère classique.

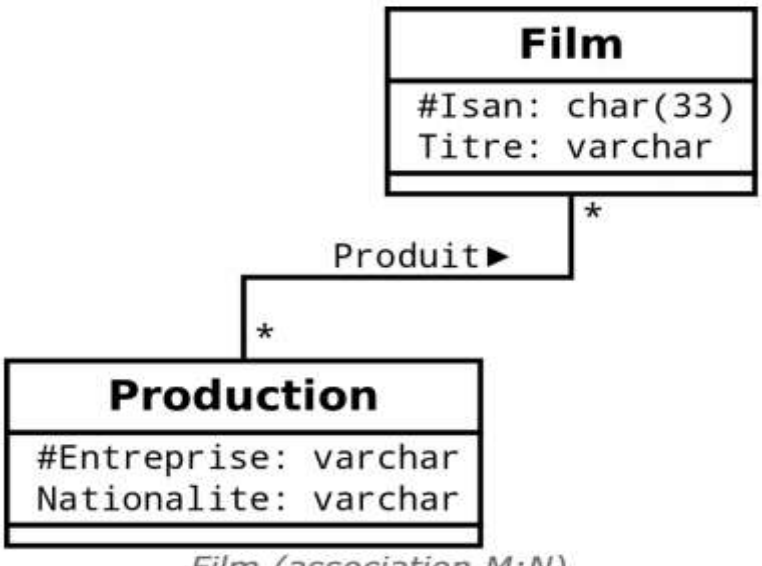
Exemple : proposer un modèle RO correspondant au modèle UML, en utilisant les OID.

Film (association 1:N)

IV.4.Association N : M

Les associations N : M peuvent être gérées comme en relationnel. Il est aussi possible de gérer ces relations, en utilisant une collection de références (une collection de clés étrangères) (NB : on favorise ainsi une des deux relations). Il est aussi possible de gérer ces relations en utilisant un OID comme clé étrangère. Il est aussi possible de gérer ces relations en utilisant une collection de référence OID.

Exemple : proposer un modèle RO correspondant au modèle UML, en utilisant les OID, mais pas de table imbriquée.

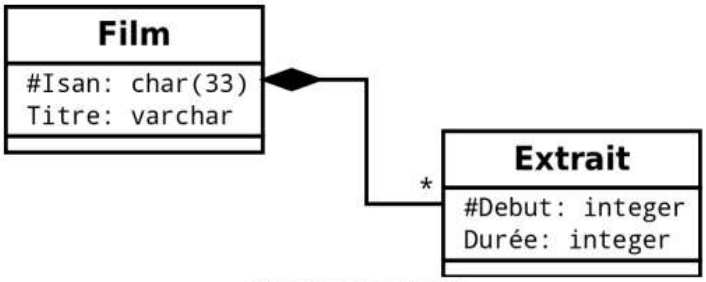


Film (association M:N)

IV.5. Composition

En RO, les compositions sont gérées avec le modèle imbriqué, à l'instar des attributs composés et multi-valués : en effet par définition l'élément composant ne peut être partagé par plusieurs composites, le modèle imbriqué n'introduit donc pas de redondance.

Exemple : proposer un modèle RO correspondant au modèle UML, en utilisant les OID.



Film (composition)

IV.6.Héritage

L'héritage est géré comme en relationnel classique. En cas d'héritage par les classes filles, on peut profiter de l'héritage de type pour éliminer la redondance dans la définition des schémas.

V. Le modèle Objet relationnel et le SQL3

V.1.Caractéristiques d'une table objet-relationnel

Une table objet-relationnelle est construite à partir d'un type. Les lignes de cette table sont considérées comme des objets avec un identifiant (OID).

Exemple

```
SQL> connect system/inchalah
Connected.
SQL> create type employe_type as object
  2  (MATR number,
  3  nom varchar(30),
  4  SAL number(8,2));
  5  /

Type created.
```

Figure 22 : création du type EMPLOYE TYPE

```
SQL> create table temploye of employe_type(primary key (MATR));

Table created.
```

Figure 23 : la création d'une table à partir de type et spécification des contraintes d'intégrité

```
SQL> insert into temploye values (employe_type(33,'aridj',1000.50));

1 row created.
```

Figure 24 : mise à jour d'une table objet

```
SQL> select * from temploye;

      MATR NOM                                SAL
---------- ------------------------------ ----------
        33 aridj                            1000,5

SQL> select e.nom
  2  from temploye e;

NOM
------------------------------
aridj
```

Figure 25 : Consultation d'une table objet

V.2.Création de type objet

```
SQL> create type t_adresse as object
  2  (num number,
  3  rue varchar(30),
  4  ville varchar(20),
  5  codepostal  varchar(5));
  6  /

Type created.
```

On peut utiliser ce type objet (ou type utilisateur) soit pour définir une table relationnelle standard soit pour définir une table relationnelle objet (ou table objet relationnelle) soit pour définir d'autres types objet qui contiennent cette structure.

V.2.1.Utilisation du type dans un autre type

```
SQL> create type t_personne as object
  2  (nom varchar(30),
  3  prenom varchar(30),
  4  adresse t_adresse);
  5  /

Type created.
```

V.2.2.Utilisation d'un type utilisateur dans une table relationnelle

Dans une table relationnelle, on l'utilise comme un type prédéfini standard.

```
SQL> create table employes
  2  (num number primary key,
  3  dept number,
  4  salaire number,
  5  adresse t_adresse,
  6  nom varchar(30));

Table created.
```

-Insertion dans une table relationnelle :

```
SQL> insert into employes values (1000,15,2000,t_adresse(804,'oudjlida','tlemcen
','13000'),'nassim');
```

-Interrogation dans une table relationnelle :

```
SQL> select * from employes;

       NUM       DEPT    SALAIRE
---------- ---------- ----------
ADRESSE(NUM, RUE, VILLE, CODEPOSTAL)
--------------------------------------------------

NOM
------------------------------
      1000         15       2000
T_ADRESSE(804, 'oudjlida', 'tlemcen', '13000')
nassim
```

```
SQL> delete from employes where (num=1000);

1 row deleted.
```

```
SQL> select * from employes;

no rows selected
```

```
SQL> insert into employes values (1000,15,2000,t_adresse(2,'Bd Lavoisier','ANGER
S','49000'),'toto');

1 row created.

SQL> select * from employes;

       NUM       DEPT    SALAIRE
---------- ---------- ----------
ADRESSE(NUM, RUE, VILLE, CODEPOSTAL)
--------------------------------------------------------------------------------

NOM
------------------------------
      1000         15       2000
T_ADRESSE(2, 'Bd Lavoisier', 'ANGERS', '49000')
toto
```

num	dept	salaire	adresse (num, rue, ville, cp)	nom
1000	15	2000	t_adresse (2, 'Bd Lavoisier', 'ANGERS', '49000')	toto

-Utilisation d'alias

```
SQL> select e.adresse from employes e;

ADRESSE(NUM, RUE, VILLE, CODEPOSTAL)
--------------------------------------------------------------------------------

T_ADRESSE(2, 'Bd Lavoisier', 'ANGERS', '49000')
```

adresse (num, rue, ville, cp)
t_adresse (2, 'Bd Lavoisier', 'ANGERS', '49000')

```
SQL> select e.adresse.num from employes e;

ADRESSE.NUM
-----------
          2
```

adresse.num
2

V.2.3.Utilisation d'un type objet dans une table relationnelle

Une table objet relationnelle est une table qui contient des éléments de type objet.Chaque élément est identifié par un numéro appelé OID (Object Identifier)

```
SQL> create table adresses of t_adresse;

Table created.
```

Figure 26 : Exemple avec le type t_adresse

On a une relation dont chaque élément est un objet de type t_adresse. On peut voir la relation de deux manières différentes. On a deux manières d'insérer des n-uplets :

-soit avec le constructeur de type (vision objet)

```
SQL> insert into adresses values (t_adresse(30,'Bd Foch','ANGERS','49000'));

1 row created.
```

-soit en précisant chacun des champs (vision relationnelle)

```
SQL> insert into adresses values (30,'Bd Foch','ANGERS','49000');

1 row created.
```

Les deux requêtes sont équivalentes : on insère un n-uplet un OID lui est attribué lors de l'insertion

V.3.Interrogation dans une table objet relationnelle

- On peut accéder aux valeurs comme dans le cas du relationnel standard.

```
SQL> select * from adresses;

       NUM RUE                            VILLE                CODEP
---------- ------------------------------ -------------------- -----
        30 Bd Foch                        ANGERS               49000
        30 Bd Foch                        ANGERS               49000
```

- On peut accéder aux objets (a est un alias de table et VALUE est un mot clé pour récupérer les objets)

```
SQL> select value(a) from adresses a ;

VALUE(A)(NUM, RUE, VILLE, CODEPOSTAL)
-----------------------------------------

T_ADRESSE(30, 'Bd Foch', 'ANGERS', '49000')
T_ADRESSE(30, 'Bd Foch', 'ANGERS', '49000')
```

- On peut accéder aux OID (a est un alias de table et REF est un mot clé pour récupérer les OID)

```
SQL> select REF(a) from adresses a ;

REF(A)
--------------------------------------------------------------------------------

0000280209643803887A54D41A7D4BCEC7F52598BFCE72A812FAC4950BD4498E34C76287A0040B4
110000

0000280209394D4C3F564A3FB5287F8B3AB7DB6DFCE72A812FAC4950BD4498E34C76287A0040B4
110001
```

Remarque :

En accédant aux OID, on obtient une relation de référence où chaque référence correspond à un code assez long fait de chiffres et de lettres. Dans le cas d'une table relationnelle standard utilisant des types objets, on n'aura pas d'OID pour les n-uplets donc pas de SELECT REF(…) ou de SELECT VALUE(…).

- Utilisation des références pour représenter les informations dans les relations

Exemple : **Table employés**

num	dept	salaire	adresse	nom
1	15	2000		toto
6	13	1000		titi
9	12	3000		tata
…	…	…	…	…

Plutôt que cette représentation qui pose des problèmes de redondance, de maintien de cohérence lors des mises à jour, on préfèrera la représentation suivante : TABLE employés (avec des OID d'objets de la table adresses)

Table employés

num	dept	salaire	adresse	nom
1	15	2000	ABC1234	toto
6	13	1000	XYZ9999	titi
9	12	3000	XYZ9999	tata

Table adresses — table objet relationnelle

	num	rue	ville	cp
ABC1234	2	Bd Lavoisier	ANGERS	49000
XYZ9999	10	Bd Foch	ANGERS	49000

Pas de changement pour l'interrogation des relations. Par contre, en terme de représentation, un objet est conservé en un seul exemplaire dans une autre table et non pas à l'intérieur de chaque n-uplet qui l'utilise.

V.4. Définition de types utilisant des références (ou pointeurs)

```
SQL> create type t_ville as object
  2  (nom varchar(10),
  3  population number);
  4  /

Type created.

SQL> create table pays
  2  (nom varchar(30),
  3  capitale REF t_ville,
  4  population number);

Table created.
```

```
SQL> create type t_pays as object
  2  (nom varchar(30),
  3  capitale REF t_ville,
  4  population number);
  5  /

Type created.

SQL> create table pays2 of t_pays;

Table created.

SQL> create table ville2 of t_ville;

Table created.
```

Remarque :

Un objet tout seul n'a pas d'OID. Un objet d'une table objet relationnelle possède un OID.

V.5.Fonction DEREF et VALUE

```
SQL> select DEREF(REF(v)) from ville2 v;

DEREF(REF(V))(NOM, POPULATION)
------------------------------------------

T_VILLE('paris', 2000000)
T_VILLE('rome', 2700000)

SQL> select VALUE(v) from ville2 v;

VALUE(V)(NOM, POPULATION)
------------------------------------------

T_VILLE('paris', 2000000)
T_VILLE('rome', 2700000)
```

Figure 27: DEREF qui renvoie un objet à partir de sa référence

V.6.Simulation de l'héritage

```
SQL> create type type_personne as object
  2  (nom varchar(30),
  3  prenom varchar(30),
  4  datenaiss date);
  5  /

Type created.

SQL> create type type_etudiant as object
  2  (num_ine varchar(10),
  3  pers REF type_personne);
  4  /

Type created.
```

Pour simuler l'héritage, on utilise des requêtes utilisant, par exemple, **e.pers.nom** : e est un alias de table, pers un pointeur qui simule l'héritage et nom le champ hérité de personne.

a) Sous-type et sur-type

La définition des sous-types à partir d'sur-type permet de réutiliser la définition de types et de spécialiser des types.

→ **CREATE TYPE sous-type UNDER sur-type AS (liste des attributs spécifiques)**

Exemple : CREATE TYPE TEnfant UNDER Tpersonne AS (JouetPréféré char(20))

b) Tables et sous-tables

La définition des sous-tables à partir d'une table donne la possibilité de définir l'héritage lors de la création de tables.

Exemple :
CREATE TYPE **personne** AS OBJECT
(
nom varchar(20),
prenom varchar(20)
)
NOT FINAL (: mot clé obligatoire si le type a des sous-types)
CREATE TYPE **etudiant** UNDER **personne**
(
faculte varchar(20),
cycle varchar(20)
)
;
CREATE TYPE **employe** UNDER **personne**
(
affectation varchar(30),
joursderepos varchar(50)
)
CREATE TABLE **personnes** OF **personne** **(primary key (nom));**
CREATE TABLE **Tetudiants** OF **etudiant** **(primary key (nom));**
CREATE TABLE **Temployes** OF **employe** **(primary key (nom));**

INSERT INTO personnes VALUES (personne ('Fellahi', 'Khaled'));

INSERT INTO Tetudiants VALUES (etudiant ('Fellahi', 'Khaled', 'FS', 'Master'));

INSERT INTO Temployes VALUES (employe ('Fellahi', 'Khaled', 'FS', 'Master'));

Remarque :
Il n'y a pas d'inclusion de population entre Tetudiants, Temployes et TPersonnes, les tables sont indépendantes les unes des autres.

V.7.Tables imbriquées

Une table relationnelle (pas nécessairement objet-relationnelle) peut contenir une ou plusieurs tables imbriquées : (1) NESTED TABLE : collection non-ordonnées et non limitée en nombre d'éléments et (2) tableau pré dimensionné (VARRAY) : collection d'éléments de même type, ordonnées et limitée en taille.

Une table imbriquée est une collection illimitée, non ordonnée d'éléments de même type. Sous Oracle, il ne peut y avoir qu'un seul niveau d'imbrication (c'est-à-dire qu'il ne peut pas y avoir de table imbriquée dans un élément d'une table imbriquée) (Sans, 2000).

Le principe du modèle imbriqué est qu'un attribut d'une table ne sera plus seulement valué par une unique valeur scalaire (principe de la 1NF), mais pourra l'être par un vecteur (enregistrement) ou une collection de scalaires ou de vecteurs, c'est à dire une autre table. La 1NF est relâchée pour permettre d'affecter des valeaurs non atomiques à un attribut afin de modéliser des objets complexes. Ce non-respect de la 1NF ne pose pas de problème en relationnel-objet car la déclaration de type permet de contrôler la structure interne des enregistrements imbriqués (voir le tableau 1). Dans le modèle relationnel classique, le problème du non-respect de la 1NF était associé à la structure interne des attributs qui interdisait l'accès à des sous informations extraites de la valeur de l'attribut (CROZAT, 2014).

Tableau 2: Exemple du modèle imbriqué

Nom	Prénom	typBureau			typTelephones	typSpecialite	
		Centre	Bâtiment	Numéro		Domaine	Technologie
DENNOUNI	Nassim	UHBC	FAC-S	21	0550505050	BD	Oracle
					0660606060	Réseaux	TCP/IP
						E-commerce	Prestashop
						E-learning	Moodle
SLIMANE	Mohamed	UHBC	FAC-S	22	021212121	Algorithmique	JAVA
					0770707070	UML	Power AMC
					0990909090	Sécurité	EBIOS
						WEB	JAVASCRIP

- Le Modèle logique associé à cet exemple :

*Type **typBureau** : <centre:char, batiment:char, numero:int>*
*Type **typListeTelephones** : collection de <entier>*
*Type **typSpecialite** : <domaine:char, specialite:char>*
*Type **typListeSpecialites** : collection de <typSpecialite>*
***tIntervenant** (#nom:char, prenom:char, bureau:typBureau, ltelephones:typListeTelephones, lspecialites:typListeSpecialites)*

- Implémentation en SQL3 de ce modèle Logique :

```
CREATE OR REPLACE TYPE typBureau AS OBJECT
 (
 centre char(2),
 batiment char(1),
 numero number(3)
 );
/
CREATE OR REPLACE TYPE typListeTelephones AS TABLE OF number (10);
CREATE OR REPLACE TYPE typSpecialite AS OBJECT
(
 domaine varchar2(15),
 technologie varchar2(15)
 );
/

CREATE OR REPLACE TYPE typListeSpecialites AS TABLE OF typSpecialite;
/
CREATE TABLE tIntervenant
(
   pknom varchar2(20) PRIMARY KEY,
   prenom varchar2(20) NOT NULL,
   bureau typBureau,
   ltelephones typListeTelephones,
   lspecialites typListeSpecialites
)
NESTED TABLE ltelephones STORE AS tIntervenant_nt1,
NESTED TABLE lspecialites STORE AS tIntervenant_nt2;
/

INSERT INTO tIntervenant (pknom, prenom, bureau, ltelephones, lspecialites)
VALUES ('Crozat', 'Stéphane',
typBureau ('PG','K',256),
typListeTelephones (0687990000,0912345678,0344231234),
typListeSpecialites (typSpecialite ('BD','SGBDR'), typSpecialite ('Doc','XML'), typSpecialite
('BD','SGBDRO')));

INSERT INTO tIntervenant (pknom, prenom, bureau, ltelephones,lspecialites)
VALUES ('Vincent','Antoine',
typBureau ('R','C',123),
typListeTelephones (0344231235,0687990001),
typListeSpecialites (typSpecialite ('IC','Ontologies'),
typSpecialite ('BD','SGBDRO')));

SELECT i.pknom, t.*
FROM tIntervenant i, TABLE (i.ltelephones) t;
```

PKNOM	COLUMN_VALUE
Crozat	687990000
Crozat	912345678
Crozat	344231234
Vincent	344231235
Vincent	687990001

SELECT i.pknom, s.*
FROM tIntervenant i, TABLE (i.lspecialites) s;

PKNOM	DOMAINE	TECHNOLOGIE
Crozat	BD	SGBDR
Crozat	Doc	XML
Crozat	BD	SGBDRO
Vincent	IC	Ontologies
Vincent	BD	SGBDRO

SELECT i.pknom, t.COLUMN_VALUE, s.domaine
FROM tIntervenant i, TABLE (i.ltelephones) t, TABLE (i.lspecialites) s

PKNOM	COLUMN_VALUE	DOMAINE
Crozat	687990000	BD
Crozat	687990000	Doc
Crozat	687990000	BD
Crozat	912345678	BD
Crozat	912345678	Doc
Crozat	912345678	BD
Crozat	344231234	BD
Crozat	344231234	Doc
Crozat	344231234	BD
Vincent	344231235	IC
More than 10 rows available. Increase rows selector to view more rows.		

Extension THE

La clause THE est une extension du LMD permettant de manipuler les objets (scalaires ou enregistrements) dans les collections implémentées sous forme de tables imbriquées.
INSERT d'un scalaire ou d'un enregistrement dans une collection

INSERT INTO tIntervenant (pknom, prenom, bureau, ltelephones,lspecialites) VALUES ('Dumas','Leonard', typBureau ('R','C',123), typListeTelephones(0344234423), typListeSpecialites ()) ;

INSERT INTO THE (SELECT i.ltelephones FROM tIntervenant i WHERE i.pknom='Dumas') VALUES (0666666666);

INSERT INTO THE (SELECT i.lspecialites FROM tIntervenant i WHERE i.pknom='Dumas') VALUES (typSpecialite ('BD','SGBDR')) ;

Remarque

L'emploi de **typListeSpecialites ()** permet de déclarer une collection **imbriquéevide** dans tIntervenant. La collection ne contient aucun élément initialement, ils peuvent être ajoutés ultérieurement grâce à l'instruction INSERT INTO THE.

DELETE d'un objet dans une collection

DELETE THE (SELECT i.ltelephones FROM tIntervenant i WHERE i.pknom='Dumas') nt WHERE nt.COLUMN_VALUE=0344234423;

UPDATE d'un objet dans une collection

UPDATE THE (SELECT i.lspecialites FROM tIntervenant i WHERE i.pknom='Dumas') nt
SET nt.technologie='SGBDRO'
WHERE nt.domaine='BD';

V.8.Les Tableaux (ou VARRAY)

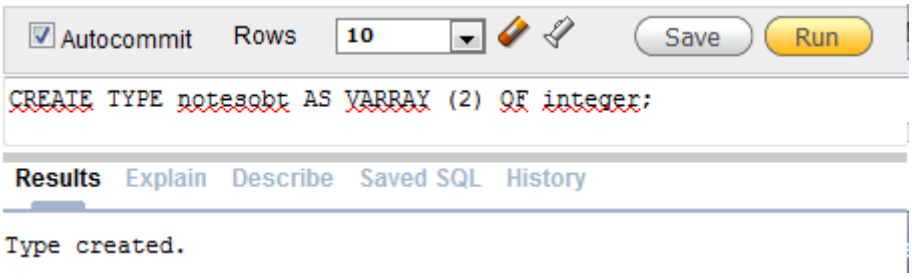
Un tableau est une collection limitée, ordonnée d'éléments de même type donc, il faut déclarer sa taille dans le modèle logique des données.

Un tableau fixe permet d'avoir plusieurs niveaux d'imbrication (à cause de l'utilisation des références, les NESTED TABLE peuvent définir qu'un seul niveau d'imbrication). Cependant, on ne peut pas accéder à un élément particulier du VARRAY dans une requête SQL standard. En effet, contrairement aux NESTED TABLE qui se comportent comme des tables relationnelles standard, l'accès à des éléments particulier d'un tableau nécessite l'utilisation d'un bloc PL/SQL (langage procédural qui intègre les requêtes SQL)

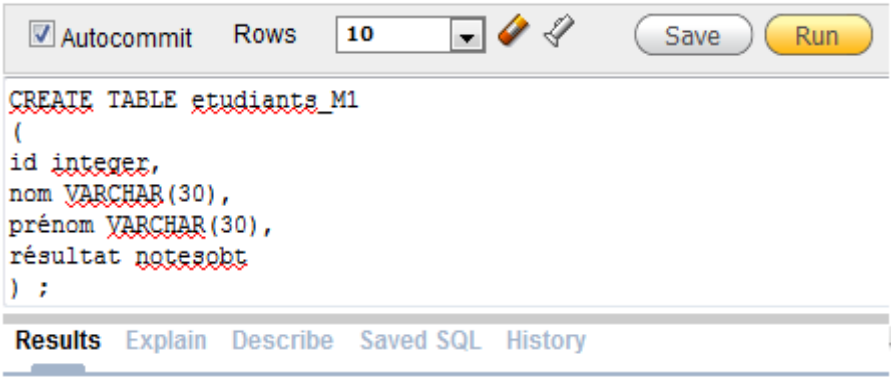
Pour cette raison, on peut manipuler un VARRAY de deux manières : (1) en utilisant une requête SQL pour manipuler le VARRAY en entier ou (2) en utilisant un bloc PL/SQL pour manipuler des éléments particuliers du VARRAY.

- Exemples d'implémentation en SQL

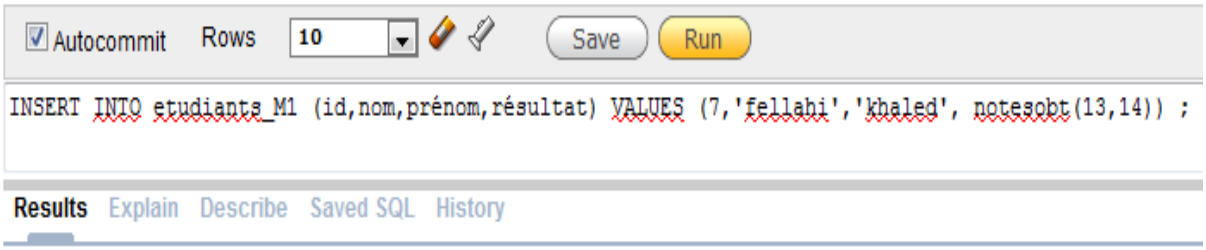
CREATE TYPE notesobt AS VARRAY (2) OF integer;



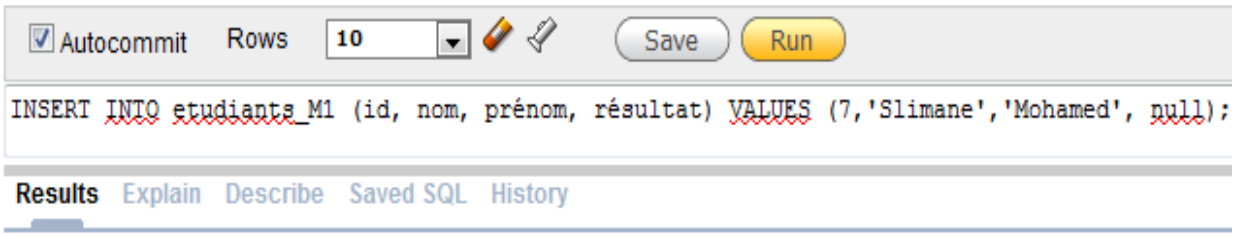
CREATE TABLE etudiants_M1
(
id integer,
nom VARCHAR(30),
prénom VARCHAR(30),
résultat notesobt
) ;



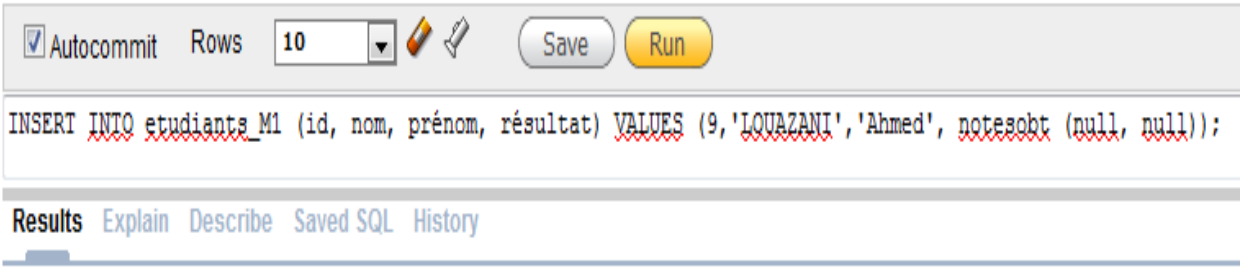
INSERT INTO etudiants_M1 (id, nom, prénom, résultat)
VALUES (7,'fellahi','khaled', notesobt (13, 14));



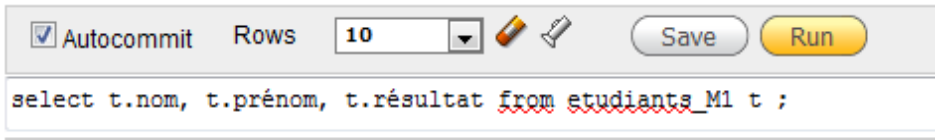
INSERT INTO etudiants_M1 (id, nom, prénom, résultat)
VALUES (7,'Slimane','Mohamed', null);



INSERT INTO etudiants_M1 (id, nom, prénom, résultat)
VALUES (7,'louazani','ahmed', notesobt (null, null));



Select t.nom, t.prénom, t.résultat from etudiants_M1 t ;




```
select t.nom, t.prénom, t.résultat from etudiants_M1 t ;
```

Results Explain Describe Saved SQL History

NOM	PRÉNOM	RÉSULTAT
LOUAZANI	Ahmed	[unsupported data type]
fellahi	khaled	[unsupported data type]
Slimane	Mohamed	[unsupported data type]
LOUAZANIfellahi	Ahmed	[unsupported data type]
LOUAZANI	Ahmed	[unsupported data type]


```
DELETE
from etudiants_M1
WHERE nom = 'LOUAZANIfellahi' ;
```

Results Explain Describe Saved SQL History

```
Statement processed.
```


```
select *
from etudiants_M1 ;
```

Results Explain Describe Saved SQL History

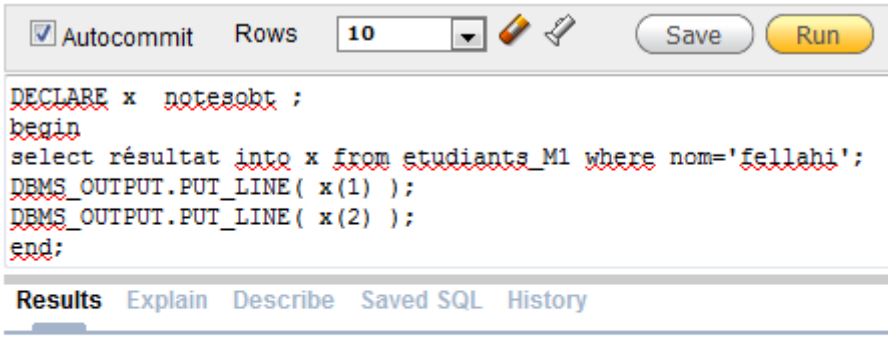
ID	NOM	PRÉNOM	RÉSULTAT
9	LOUAZANI	Ahmed	[unsupported data type]
7	fellahi	khaled	[unsupported data type]
7	Slimane	Mohamed	[unsupported data type]
9	LOUAZANI	Ahmed	[unsupported data type]


```
UPDATE etudiants_M1
SET prénom = 'DENNOUNI'
WHERE id=9 ;
```

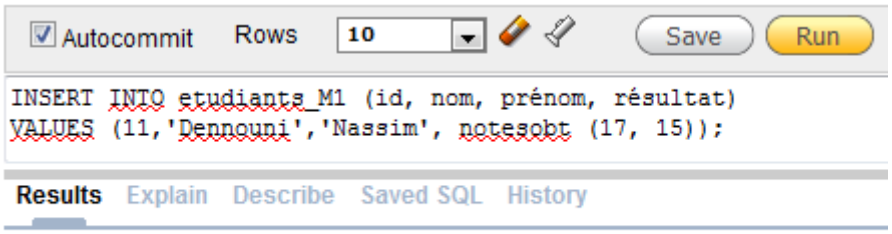
Results Explain Describe Saved SQL History

```
2 row(s) updated.
```

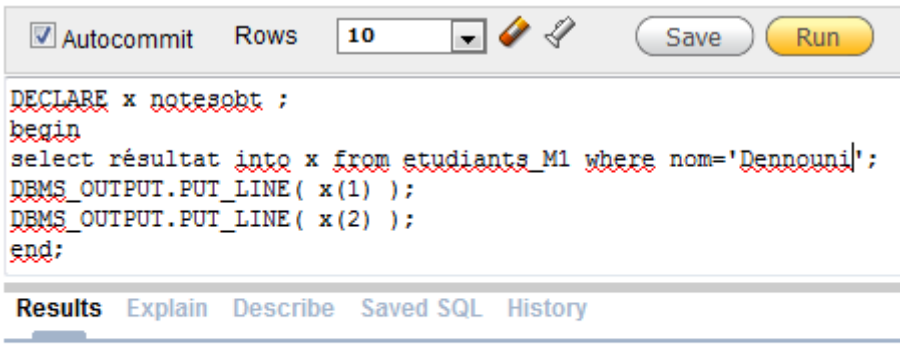
```
DECLARE x notesobt ;
begin
select résultat into x from etudiants_M1 where nom='fellahi';
DBMS_OUTPUT.PUT_LINE( x(1) );
DBMS_OUTPUT.PUT_LINE( x(2) );
end;
```




```
INSERT INTO etudiants_M1 (id, nom, prénom, résultat)
VALUES (11,'Dennouni','Nassim', notesobt (17, 15));
```




```
DECLARE x notesobt ;
begin
select résultat into x from etudiants_M1 where nom='Dennouni';
DBMS_OUTPUT.PUT_LINE( x(1) );
DBMS_OUTPUT.PUT_LINE( x(2) );
end;
```



V.9.l.Les méthodes

Une méthode (ou opération) est la modélisation d'une action applicable sur un objet, caractérisée par un en-tête appelé signature définissant son nom, ses paramètres d'appel et de retour, et qui permet de modifier l'état de l'objet ou de renvoyer un résultat. En relationnel objet, les types peuvent admettre soit des fonctions soit des procédures.

On déclare les méthodes :

- soit au début lors de la déclaration de l'objet
- soit plus tard avec commande ALTER TYPE

Le corps de la méthode correspond aux opérations effectuées sur l'objet, il peut faire référence à l'objet concerné grâce à SELF.

Exemple : type objet contenant une fonction

```
CREATE TYPE tpersonne AS OBJECT
(
 nom VARCHAR(10),
datenaiss date,
MEMBER FUNCTION age RETURN number
)
CREATE TYPE BODY tpersonne AS MEMBER FUNCTION age
RETURN number IS n number;
BEGIN
n := TRUNC((SYSDATE - SELF.datenaiss)) ;
RETURN n;
END age;
END;
```

Ce type contient une fonction nommée **age** sans paramètre qui retourne un nombre. On peut utiliser une fonction dans un bloc PL/SQL ou une requête SQL.

Exemple :

```
create table tpersonnes of tpersonne;
insert into tpersonnes (nom,datenaiss) values ('youcef','01/01/1988') ;
SELECT p.age () FROM tpersonnes p ;
```

Conclusion du chapitre II

Le modèle objet-relationnel offre de nouvelles possibilités pour la modélisation des besoins croissants des utilisateurs grâce à sa richesse sémantique. Cependant, ce type de modèle trouve vite ces limites face à des données hétérogènes ou non structurées. Pour cette raison, nous allons aborder dans le prochain chapitre, une nouvelle façon de représentation à l'aide du langage XML.

Chapitre III. Les données semi-structurées

Introduction

Le modèle relationnel n'est pas adapté aux données hétérogènes ou de structure mal définie. Par exemple, si on veut stocker des informations clients de deux banques indépendantes qui représentent leurs clients de manière différente, l'approche relationnelle semble peu adaptée à ce type de sauvegarde. Par contre, une représentation à l'aide du langage XML (*eXtended Markup Language*) permet de manipuler plus facilement ces informations irrégulières, grâce à la possibilité de structurer et d'annoter des informations sans la définition d'un schéma fortement structuré et contraignant. En effet, le langage XML représente les informations sous forme de documents textuels annotés et structurés par des balises dont la structure correspond à une arborescence d'éléments (AMANN & SCHOLL, 2016).

D'autre part, les données semi-structurées peuvent se voir comme une relaxation du modèle relationnel classique dans lequel on autorise une structure moins rigide et homogène des champs de données. Par conséquent, ce modèle de données est devenu très utile dans la représentation des documents de type multimédia, hypertexte, données scientifiques ... (DAL ZILIO & LUGIEZ, 2002)

I. Le langage XML

Le langage XML est un format textuel qui permet de créer des documents contenant des données semi-structurées. Conçu à l'origine comme un dérivé simple et flexible de la norme SGML, il joue aujourd'hui un rôle de plus en plus important dans l'échange d'informations sur le Web. L'utilisation la plus simple de XML consiste à créer des documents sans avoir défini de contrainte sur leur structure. Mais on peut également, comme dans le modèle relationnel, contraindre un (ensemble de) document(s) à respecter une structure définie sous forme de schémas ou types de documents (AMANN & SCHOLL, 2016) (DAL ZILIO & LUGIEZ, 2002).

En 1969, Charles Goldfarb (chef de projet chez IBM) lance un langage descriptif nommé GML (Generalized Markup Language) destiné à encapsuler l'ancien langage Script trop lié physiquement aux possibilités techniques des imprimantes. Goldfarb développe un successeur de GML appelé SGML (Standard Generalized Markup Language) qui sera publié en 1986 comme norme ISO (ISO 8879:1986). L'un des principes fondamentaux sur lequel repose le SGML est la séparation complète entre la structure logique d'un document (titres,

chapitres, paragraphes, illustrations…) et sa mise en page (feuilles de style), qui dépend du support de présentation (livre, journal, écran …). Ensuite, Tim Berners-Lee a créé le langage HTML qui est une application de SGML destinée World Wide Web (Wikipédia, Standard Generalized Markup Language, 2016). XML est un métalangage informatique de balisage générique qui dérive du SGML. Cette syntaxe est dite « extensible » car elle permet de définir différents espaces de noms, c'est-à-dire des langages avec chacun leur vocabulaire et leur grammaire, comme XHTML, MathML, XSLT, RSS, SVG… (Wikipédia, Extensible Markup Language, 2016). Par exemple, le SVG (Scalable Vector Graphics) est un format de données conçu pour décrire des ensembles de graphiques vectoriels qui est basé sur XML. D'autre part, le MathML est aussi un langage basé sur XML permettant l'affichage de symboles mathématiques, notamment sur Internet. On peut citer aussi le XHTML (Extensible HyperText Markup Language) conçu comme le successeur de HTML car ce langage utilise la syntaxe définie par XML (voir la figure ci-dessous).

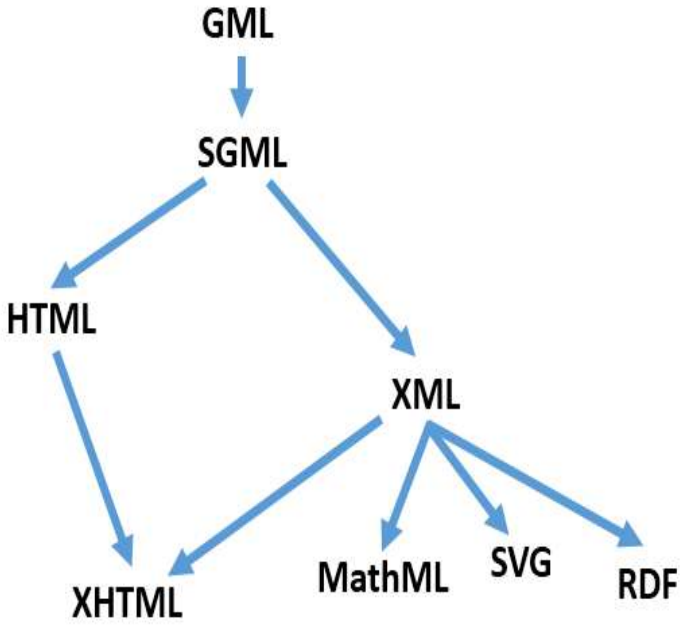


Figure 28: la relation entre GML, SGML, HTML, XML, XHTML, MATHML, SVG et RDF

L'objectif de XML est de faciliter l'échange automatisé de contenus complexes (arbres, texte riche…) entre systèmes d'informations hétérogènes (interopérabilité).

I.1.Définition d'une DTD

XML emprunte en grande partie sa simplicité et sa syntaxe à HTML qui est un fragment du SGML. En particulier, un document XML se présente comme une succession de balises imbriquées (ou plus abstraitement comme un arbre étiqueté). Cependant, afin d'imposer un peu plus de structure à un document XML, on lui associe une DTD (Document Type Definition).

Une DTD est un ensemble de règles qui spécifient pour chaque type d'élément les types de ses éléments fils, leur ordonnancement et leur fréquence.

Les DTDs définissent ainsi l'ensemble des balises qu'il est possible de trouver dans un document valide et établissent des contraintes sur l'ordre d'apparition de ces balises.

I.2. Validation d'un document XML

Un document est appelé "document XML" s'il est **bien formé**, ce qui signifie qu'il respecte les règles syntaxiques de XML. Il est **valide** si, en plus, il respecte la grammaire du langage, contenue dans un fichier appelé DTD (Définition de Type de Document).

I.2.1.Validation syntaxique d'un document XML

Un document XML peut être découpé en 2 parties : le **prologue** et le **corps**.

Le prologue correspond à la première ligne de votre document XML. Il donne des informations de traitement.

Exemple : < ? xml version = "1.0" encoding="UTF-8" standalone="yes" ?>

- Le prologue est une balise unique qui commence par < ? xml et qui se termine par ?>.
- Dans le prologue, on commence généralement par indiquer la version de XML que l'on utilise pour décrire nos données. Pour rappel, il existe actuellement 2 versions : 1.0 et 1.1.
- Par défaut, l'encodage de XML est l'UTF-8, mais si votre éditeur de texte enregistre vos documents en ISO8859-1, il suffit de la changer dans le prologue
- La dernière information présente dans le prologue est standalone="yes".Cette information permet de savoir si votre document XML est autonome ou si un autre document lui est rattaché.

Le **corps** d'un document XML est constitué de l'ensemble des balises qui décrivent les données. Il y a cependant une règle très importante à respecter dans la constitution du corps : *une balise en paires unique doit contenir toutes les autres*. Cette balise est appelée **élément racine** du corps.

Un document XML est dit *bien formé* lorsque le document est correct sans toutefois posséder une DTD. Le prologue du document ne contient pas de Définition de Type de Document (DTD) et la structure arborescente du document respecte les standards XML (nom des balises et attributs, imbrication des marqueurs XML,...)

I.2.Validation d'un document XML par une DTD

Une DTD structure un document XML en définissant : (1) le nom des éléments, leur contenu, le nombre de fois et l'ordre d'apparition, (2) les attributs éventuels et leurs valeurs par défaut et (3) les noms des entités qui peuvent être utilisées.

Les documents XML valides doivent respecter les règles données d'une DTD. La déclaration d'une DTD doit apparaître après la déclaration XML, mais avant l'élément racine.

<!xml version="1.0" ....>
<!DOCTYPE élément_racine ....>

La déclaration de la DTD peut contenir : (1) la DTD elle-même à l'intérieur du fichier XML (DTD interne) ou (2) une adresse URL qui indique le fichier contenant la DTD (DTD externe).En cas de conflit, les déclarations de la DTD interne prime.Une déclaration d'une DTD commence par :

<!DOCTYPE ElementRacine et se termine par >

Le nom d'un élément **utilisé** dans le document XML doit être **identique** à celui déclaré dans la DTD.

A). Description des attributs d'une DTD

Un élément peut : (1) contenir du texte, (2) contenir d'autres éléments, (3) contenir un mélange de texte et d'éléments (contenu mixte), (4) être vide.
Chaque type d'élément doit être déclaré, cette déclaration respecte un des formats suivants :
(1) ***< ! ELEMENT NOM (CONTENU)>***
(2) ***< ! ELEMENT NOM (CONTENU_MIXTE)*>***
(3) ***< ! ELEMENT NOM ANY> n'importe quelles données***
(4) ***< ! ELEMENT NOM EMPTY > élément vide (<NOM/>)***

Exemple

• DTD

< ! ELEMENT personne (nom, prenom +, tel?, email, adresse >
< ! ELEMENT nom (#PCDATA) >
< ! ELEMENT prenom (#PCDATA) >
< ! ELEMENT tel (#PCDATA) >
< ! ELEMENT email (#PCDATA) >
< ! ELEMENT adresse (ANY) >

• Document XML associé

<personne>
<nom> Bennani </nom>
<prenom> Mohammed </prenom>
<prenom> Ali </prenom>
<tel> 0683000000 </tel>
<email> bennani@fsr.ac.ma </email>
<adresse> <rue> <rue/> <ville>Rabat</ville></adresse>
</personne>

La spécification du contenu d'un élément précise : le genre d'informations que l'élément peut contenir (texte, sous éléments, mixte) et les contraintes sur son contenu. Les mots clés de description du contenu sont :

(#PCDATA) : Parsed Character Data, du contenu littéral.
(ELEMENT) : le sous-élément ELEMENT.
(ELEMENT1, ELEMENT2,...) : une liste d'éléments appelée séquence. L'ordre d'apparition des éléments doit être respecté dans le document XML.
(ELEMENT1|ELEMENT2|...) choix d'un sous-élément.
ELEMENT ? : zéro ou une fois.
ELEMENT+ : une ou plusieurs fois.
ELEMENT* : zéro ou plusieurs fois.

Par exemple si on dit : qu'une liste de films **lfilms** contient des films, ***au moins un*** qu'un **film** contient **un titre** et **zéro ou plusieurs** acteurs (dans cet ordre), qu'un titre et un acteur sont des chaines de caractères PCDATA (Parsed Character Data) On écrira la DTD suivante :

< ! ELEMENT **lfilms** (film+)>
< ! ELEMENT **film** (titre, acteur*)>
< ! ELEMENT **titre** (#PCDATA)>
< ! ELEMENT **acteur** (#PCDATA)>

Exemple 1 :

```
<? xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
< ! DOCTYPE parents [
< ! ELEMENT parents (fille, garcon)>
< ! ELEMENT fille (#PCDATA)>
< ! ELEMENT garcon (#PCDATA)>
]>
< parents >
   < fille >Jalila</fille>
   < garcon >Sami</garcon>
</ parents >
```

Exemple 2 :

```
<? xml version='1.0' encoding='ISO-8859-1' standalone="yes"?>
< ! DOCTYPE BIBLIOTHEQUE [
< ! ELEMENT BIBLIOTHEQUE (LIVRE)* >
< ! ELEMENT LIVRE (AUTEUR, TITRE, EDITEUR)>
< ! ELEMENT AUTEUR (PRENOM, NOM) >
< ! ELEMENT TITRE (#PCDATA) >
< ! ELEMENT EDITEUR (NOM, ANNEE) >
< ! ELEMENT PRENOM (#PCDATA) >
< ! ELEMENT NOM (#PCDATA) >
< ! ELEMENT ANNEE (#PCDATA) >
]>
<BIBLIOTHEQUE>
<LIVRE>
<AUTEUR>
<PRENOM>Rolf</PRENOM>
<NOM> MAURERS</NOM>
</AUTEUR>
<TITRE>JAVA</TITRE>
<EDITEUR>
<NOM>Micro Application</NOM>
<ANNEE> 1996</ANNEE>
</EDITEUR>
</LIVRE>
</BIBLIOTHEQUE>
```

B) Description des attributs d'une DTD

La description des attributs se fait par une déclaration d'une liste d'attributs (**ATTLIST)** La syntaxe est la suivante :

- < ! ATTLIST Elément Attribut Type #FIXED Valeur> FIXED signifie que l'attribut a une valeur fixe.
- < ! ATTLIST Elément Attribut Type #REQUIRED> REQUIRED signifie que l'attribut est obligatoire et n'a pas de valeur par défaut.

- < ! ATTLIST Elément Attribut Type #IMPLIED> IMPLIED signifie que l'attribut n'est pas obligatoire et n'a pas de valeur par défaut.

Exemple1

1. L'élément MESSAGE contient des données textuelles et peut contenir un attribut nommé LANGUE, sa valeur par défaut est "Français".

< ! ELEMENT MESSAGE (#PCDATA)>
< ! ATTLIST MESSAGE LANGUE CDATA "Français">

2. On peut, dans une même déclaration ATTLIST, définir plusieurs attributs associés au même élément :

< ! ATTLIST IMG WIDTH CDATA "100" HEIGHT CDATA "100">

3. Nous pouvons limiter la liste de valeurs possibles pour un attribut. On le définit comme un type énuméré. Nous déclarons un attribut format d'un élément **img**. Cet attribut peut prendre une valeur parmi **GIF**, **JPEG** et **PNG**. La valeur par défaut est **GIF.**

< ! ELEMENT img EMPTY>
<! ATTLIST img format (GIF|JPEG|PNG) "GIF">

Exemple2

<! ELEMENT personne (nom, prenom+, tel?, email,adresse >
<!ELEMENT nom (#PCDATA) >
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT tel (#PCDATA) >
<!ELEMENT email (#PCDATA) >
<!ELEMENT adresse ANY>
<! ATTLIST personne
age CDATA #IMPLIED
genre (Masculin | Feminin) #REQUIRED >
<!ELEMENT auteur (#PCDATA) >
<!ATTLIST auteur
genre (Masculin | Feminin) #REQUIRED
ville CDATA #IMPLIED>
<!ELEMENT editeur (#PCDATA) >
<!ATTLIST editeur
ville CDATA #FIXED "Rabat">
C) Attributs ID et IDREF

Ce type sert à indiquer que l'attribut concerné peut servir d'identifiant dans un fichier XML.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Document [
<!ELEMENT Document (Personne*)>
<!ELEMENT Personne (#PCDATA)>
<!ATTLIST Personne PNum ID #REQUIRED>
<!ATTLIST Personne Mere IDREF #IMPLIED>
<!ATTLIST Personne Pere IDREF #IMPLIED>
]>
```

< Document >
< Personne PNum = "P1">Latifa<//Personne>
< Personne PNum = "P2">Rachid</Personne>
< Personne PNum = "P3" Mere ="P1" Pere ="P2">Ali</Personne>
< Personne PNum = "P4" Mere ="P1" Pere ="P2">Samia</Personne>
</Document >

II. Bases de données et XML

Deux grandes catégories de bases de données peuvent intégrer du XML : (1) l'extension des bases de données relationnelles "classiques" et (2) des bases de données XML natives. Soit les données sont transformées et mémorisées avec un SGBD relationnel, soit elles sont intégrées directement en XML. Le schéma ci-dessous illustre ces deux familles (Andreas & Lars, 2000).

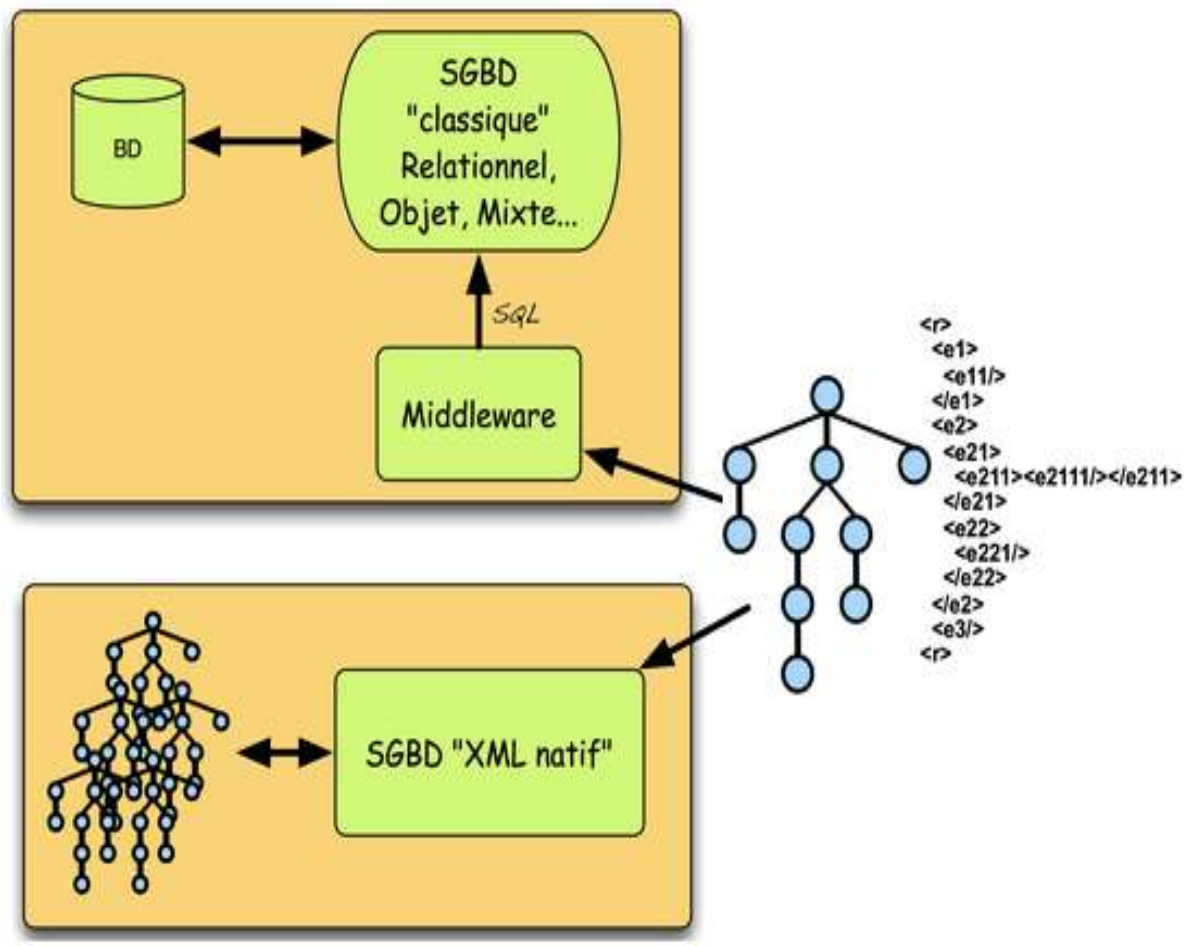
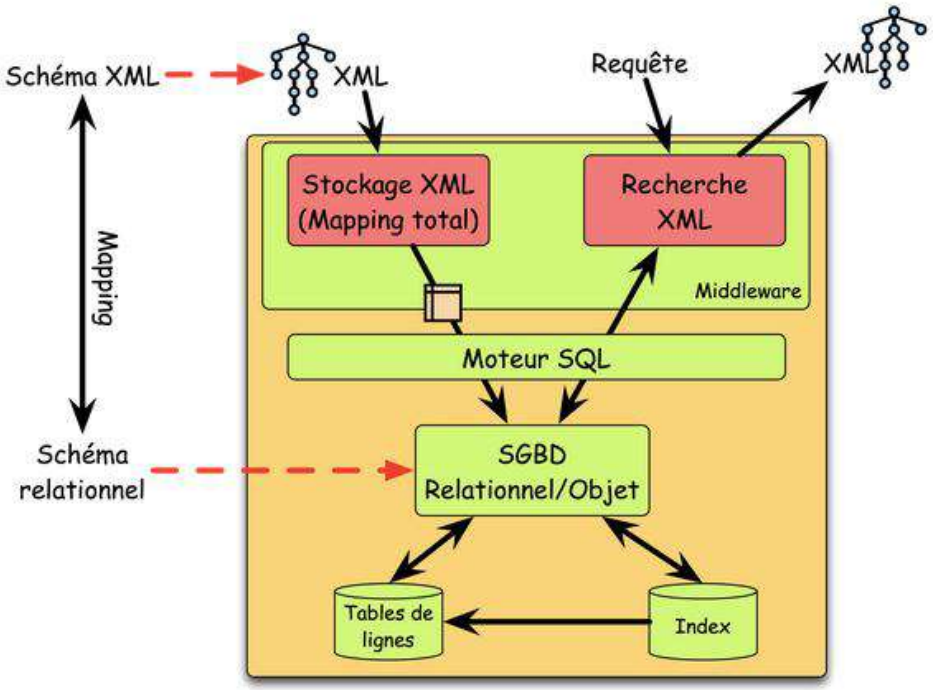


Figure 29: bases de données intégrant du XML

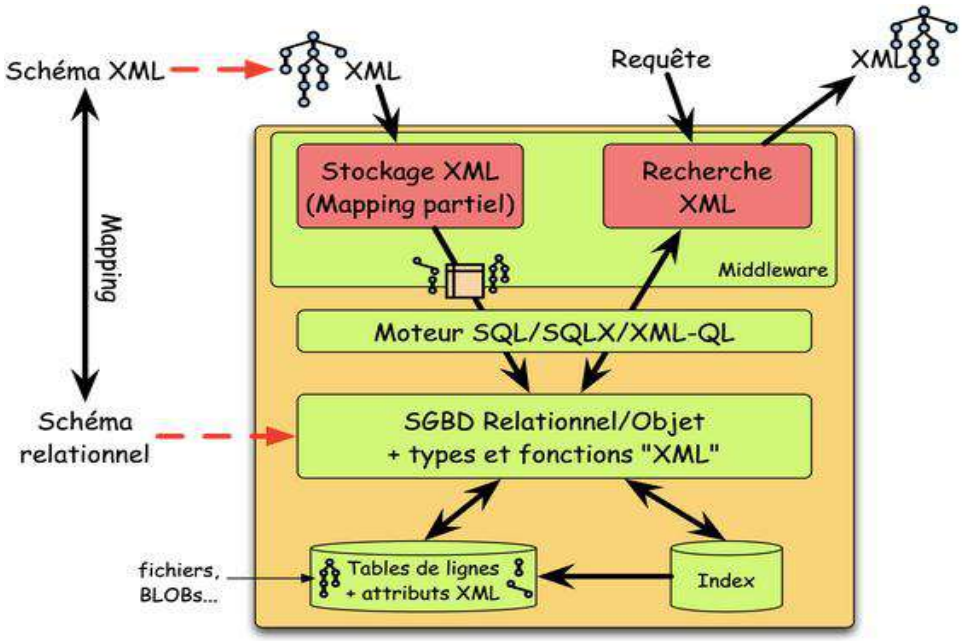
II.1.Utilisation d'une base de donnée relationnelle

L'hypothèse de départ est la suivante : comment mémoriser des flux XML dans une base de données relationnelle ? Une interface (middleware) joue l'intermédiaire entre données ou requêtes XML et la base de données relationnelle comme le montre la figure ci-dessous :



II.2.Utilisation des bases de données intégrant des types XML

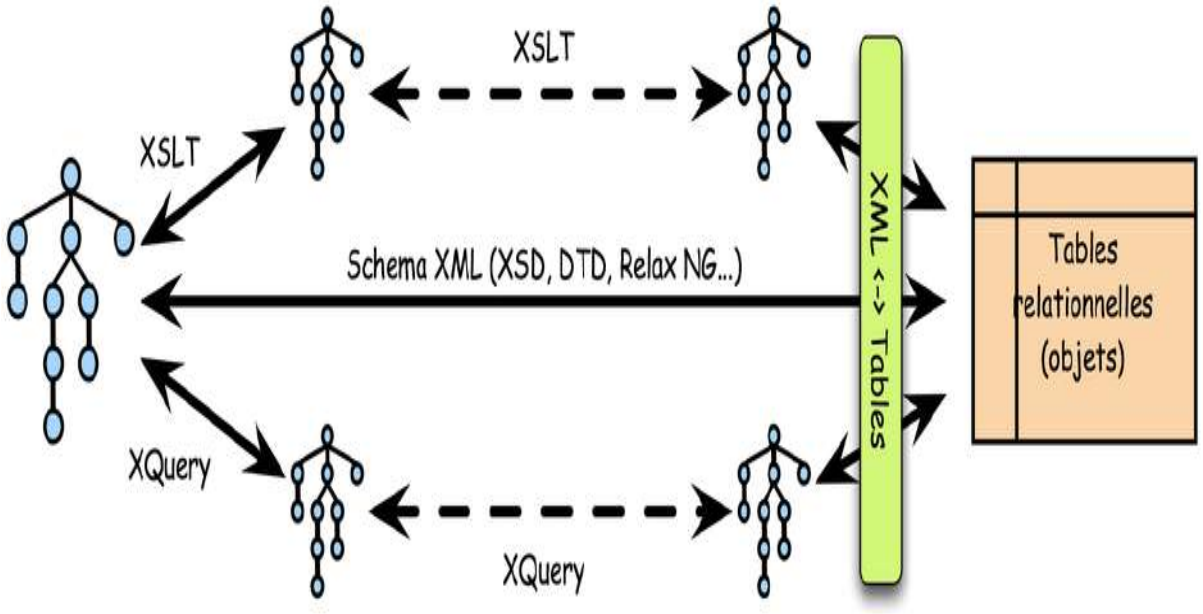
Concrètement, le mapping complet de données XML n'est pas nécessaire, car la plupart des SGBD ont intégré des types XML, voire même des fonctions pour les manipuler (XPath ou autres). L'idée principale est donc de ne transformer qu'une partie des données, en laissant le reste en XML. On a ainsi un mapping partiel comme le montre la figure ci-dessous.



II.3.Mapping Relationnel/XML

Le mapping relationnel/XML se base sur deux phases : (1) la détermination des schémas XML et du schéma relationnel et (2) la mise en oeuvre des transformations dans les deux sens. L'opération de détermination des schémas doit prendre en compte non seulement les schémas généraux, mais aussi les schémas liés aux différentes requêtes de mise à jour et de recherche.

L'opération de transformation s'effectue en deux étapes : une série de transformations du modèle XML (en utilisant XSLT, voire XQuery) puis une transformation entre XML et le modèle relationnel comme le montre la figure ci-dessous (Andreas & Lars, 2000).



III. Le passage entre le Relationnel et XML

La situation favorable est d'avoir à sa disposition un modèle de données de type MCD. Ainsi, la génération conjointe des deux schémas, sans optimisation particulière, facilitera le passage de l'un à l'autre. Lorsqu'un seul des deux modèles est imposé, il faut alors construire l'autre. Il est plus facile d'opérer du relationnel vers XML, car XML est plus structuré (Andreas & Lars, 2000).

III.1.Le passage du Relationnel vers le XML

Parmi les caractéristiques des documents XML, on peut citer les points suivants (JUGANARU-MATHIEU, 2012):

(1) Les objets XML nécessitent moins de modifications que les enregistrements des tables
(2) Consulter XML signifie avoir soit le contenu complet, soit une partie du document
(3) Présenter un résultat : forme textuelle ou autre (XSLT)
(4) valider un document XML : par rapport à sa description (DTD, XML XSchema) mais moins par rapport à d'autres documents XML (clé étrangère ?)
(5) Les documents XML peuvent être partitionnés de diverses manières en collections

Dans cette section, nous posons comme hypothèse la connaissance du modèle relationnel pour construire le modèle XML (sous forme de DTD). La méthodologie de construction du schéma XML, assez triviale, est la suivante :

1. pour chaque table "t_i" est généré un élément XML "e_i" ;
2. pour chaque attribut de la table, créer un attribut ("CDATA", en le mettant "#IMPLIED" s'il peut être null et "#REQUIRED" sinon) ou un élément #PCDATA (avec "?" s'il peut être null) inclus uniquement dans l'élément associé à la table ;
3. créer un élément "k_e_i ID #REQUIRED", obtenu en appliquant une fonction (bijective et compatible avec les noms XML) sur les attributs clés de la table ;

4. pour chaque clé étrangère (clé d'une autre table "t_j"), procéder de la même manière en créant "fk_e_j IDREF #REQUIRED" ;
5. créer un élément "liste_e_i" défini par "e_i*" ;
6. créer l'élément racine contenant chacun des éléments "liste_e_i".

III.2.Le passage du XML vers le Relationnel

Le passage du modèle XML au modèle relationnel est un peu plus complexe à mettre en œuvre du fait de l'approche très structurée d'XML. Plusieurs approches sont possibles. Dans tous les cas il faut (1) s'assurer de ne pas perdre d'informations et (2) faire attention aux modifications directes de la base de données. En effet, le modèle relationnel étant peu structurer, le risque majeur est de perdre la structure, et donc les informations qu'elle porte. Par ailleurs, toute modification directe de la base risque de produire un résultat "ne respectant plus le schéma XML".

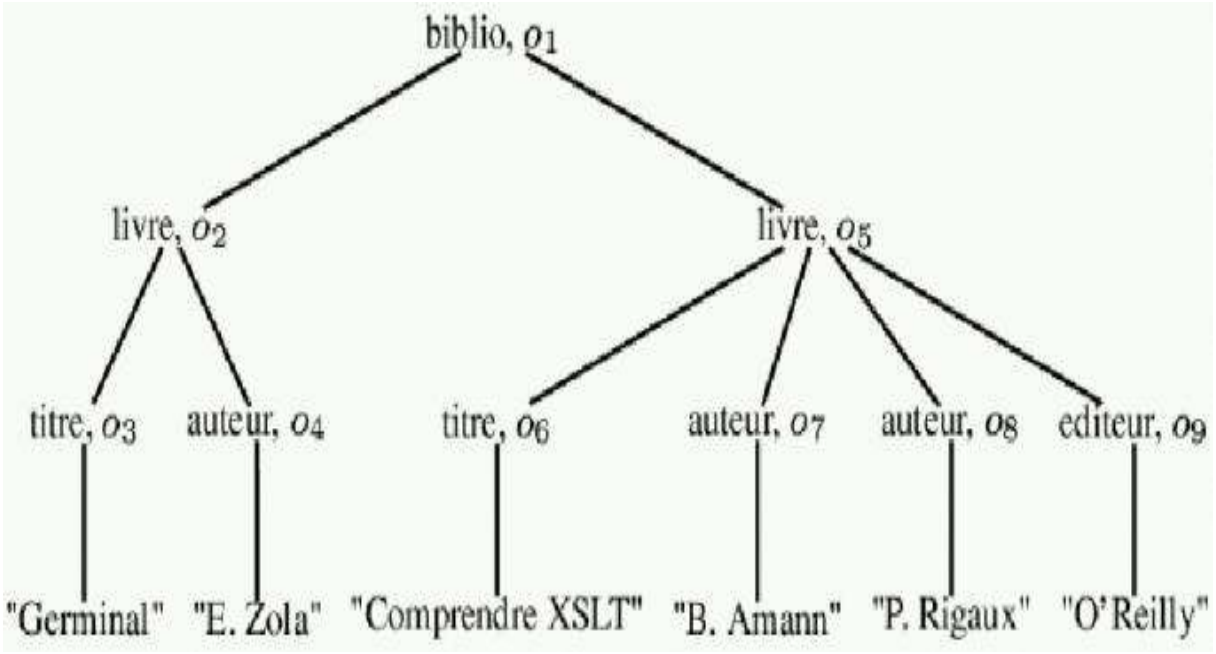
Un document XML est par nature une structure hiérarchique (un arbre), aussi les méthodes de construction du modèle relationnel cherchent à mémoriser cette structure. Cette structuration permet de mettre en place des liens souvent représentés par des clés (étrangères) dans le modèle relationnel. Il est donc souvent nécessaire, lors du passage XML vers le Relationnel de créer des clés. Cependant, il ne faut pas non plus perdre de vue le fait que l'arbre XML est ordonné, c'est-à-dire que les fils d'un élément ont un ordre qui peut avoir son importance et soit porteur d'informations. Cette dernière remarque est à prendre en compte, car le modèle relationnel ne possède, lui, pas de notion d'ordre (Andreas & Lars, 2000).

Exemple

Soit le document suivant :

```
<? xml version="1.0» ?>
<biblio>
        <livre>
                <titre>Germinal</titre>
                 <auteur>E. Zola</auteur>
         </livre>
         <livre>
                  <titre>Comprendre XSLT</titre>
                  <auteur>B. Amann</auteur>
                  <auteur>P. Rigaux</auteur>
                  <editeur>O’Reilly</editeur>
          </livre>
</biblio>
```

Ce document XML peut être représenté par l’arbre suivant :



Pour représenter n'importe quel document XML deux tables suffisent : (1) une qui garde la structure arborescente avec le numéro du nœud, sa balise, son type, son père et le numéro d'ordre parmi ses frères et (2) une table qui garde les valeurs des feuilles.

R:

Part	Pos	Lab	Type	Id
o_0	1	biblio	ref	o_1
o_1	1	livre	ref	o_2
o_1	2	livre	ref	o_5
o_2	1	titre	cdata	o_3
o_2	2	auteur	cdata	o_4
o_5	1	titre	cdate	o_6
o_5	2	auteur	cdata	o_7
o_5	3	auteur	cdata	o_8
o_5	4	editeur	cdata	o_9

S:

Noeud	Val
o_3	Germinal
o_4	E. Zola
o_6	Comprendre XSLT
o_7	B. Amann
o_8	P. Rigaux
o_9	O'Reilly

→Exemple : trouver les titres des livres d'E. Zola :

En XQuery

```
FOR $i IN doc ("biblio.xml")/biblio/livre
WHERE $i/auteur = "E. Zola"
RETURN $i/titre
```

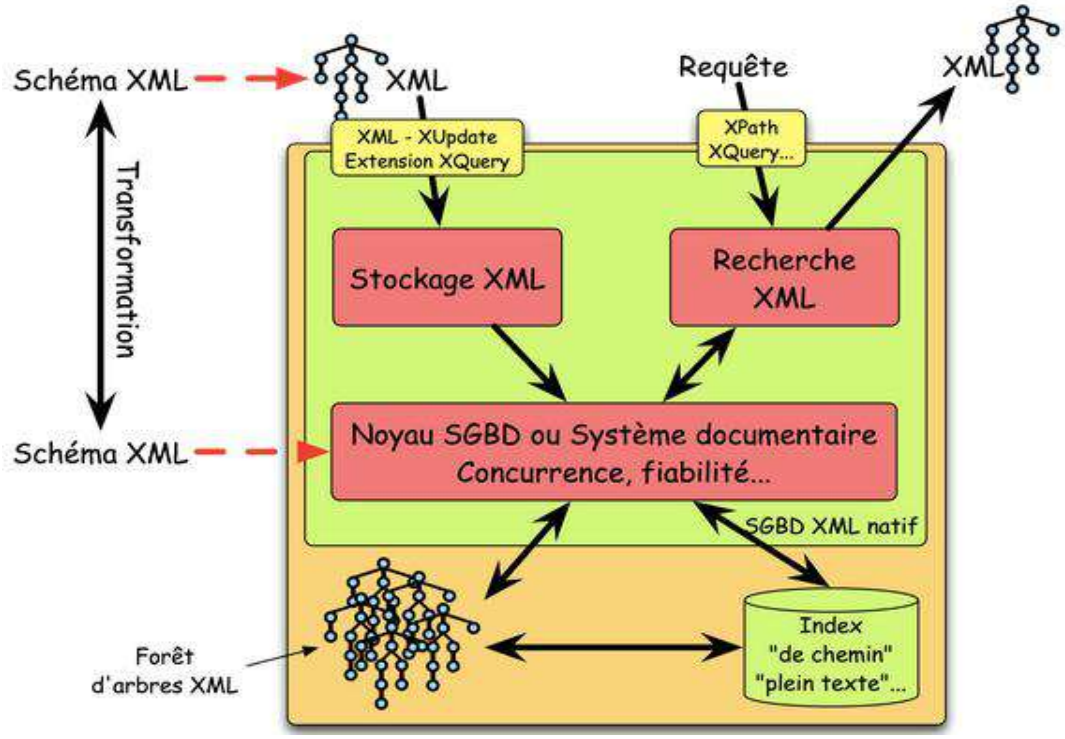
En SQL avec une seule jointure sur les deux tables :

```
SELECT L.*
FROM LIVRE L, AUTEUR A
WHERE L.CODE_LIVRE = A.CODE_LIVRE AND A.AUTEUR = 'E. Zola' ;
```

Cette méthode (en SQL ou XQuery) n'est pas générique, car elle impose la connaissance de la sémantique des balises et l'augmentation du nombre de tables (même ordre que le nombre de balises) (JUGANARU-MATHIEU, 2012).

IV. Bases de données XML natives

L'utilisation de bases de données relationnelles a pour avantages principaux l'utilisation de SGBD communs et des performances connues. Cependant, le passage de l'XML vers le Relationnel a un coût à ne pas négliger. De plus, nous l'avons vu avec la mise en place du mapping, il y a une perte d'informations notable nécessitant de nombreux contrôles par le middleware. Dans ce contexte, les SGBD XML natifs appelés aussi "SGBD-XML" sont apparus comme des SGBD traitant directement des données XML sans faire une transformation de modèle comme le montre la figure ci-dessous (Andreas & Lars, 2000).



Dans ce type de système, des "forêts" d'arbres XML sont mémorisés. Les SGBD-XML permettent d'optimiser l'accès aux informations, en particulier par la mise en place, de manière automatique ou manuelle plusieurs stratégies d'indexations des chemins, de sous-arbres, de noeuds DOM, etc.

Comme dans les SGBD-R, les SGBD-XML possèdent un langage de recherche et un langage de modification nommé XQuery. Ce langage est assez utilisé dans les bases de données natives et Il utilise XPath pour décrire les localisations des modifications. Les opérations autorisées sont des opérations : (1) d'insertion : "insert-before" et "insert-after", (2) d'ajout : "append" et (3) de mise à jour : "update", "rename" et "remove" (Andreas & Lars, 2000).

Conclusion du chapitre III

Ce chapitre a montré que XML est un format de bases de données qui ne permet pas de faire un stockage efficace car il n'est pas très structuré et il ne dispose pas de langage de requête. Une des solutions pour l'indexation des transactions, les mises à jour, etc consiste à associer une BD à XML car XML est considéré comme un format d'échange et d'intégration.

Chapitre IV. Les bases de données réparties

Introduction

Les bases de données réparties ont une architecture plus adaptée à l'organisation des entreprises décentralisées car elles permettent : (1) plus de fiabilité par exemple, la panne d'un site n'est pas très importante pour l'utilisateur qui peut s'adresser à autre site (2) meilleures performances grâce à la réduction du trafic sur le réseau et (3) faciliter l'accroissement par l'ajout de machines sur le réseau (Moussa, 2006).

I. SGBD réparti

Une base de données centralisée est gérée par un seul SGBD et elle est stockée dans sa totalité à un emplacement physique unique avec une seule et même unité de traitement. Par opposition, une base de données distribuée[1] est gérée par plusieurs processeurs, sites ou SGBD.

Du point de vue organisationnel, nous distinguons deux architectures :

(1) Architecture Client-Serveur : les serveurs, ont pour rôle de servir les clients. Par servir, on désigne la réalisation d'une tâche demandée par le client.

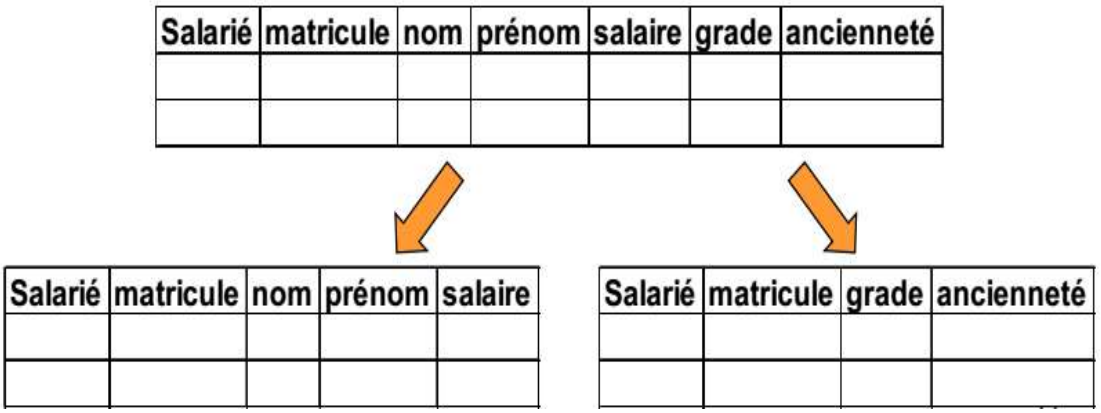
(2) Architecture Pair-à-Pair (Peer-to-Peer, P2P) : par ce terme on désigne un type de communication pour lequel toutes les machines ont une importance équivalente.

I.1.12 règles de Date

1. Autonomie locale
2. Pas de dépendance d'un site central
3. Opère en continu
4. Indépendance vis-à-vis de la localisation
5. Indépendance vis-à-vis de la fragmentation
6. Indépendance vis-à-vis de la réplication
7. Traitement de requêtes distribuées
8. Traitement de transactions distribuées
9. Indépendance vis-à-vis du matériel
10. Indépendance vis-à-vis du système d'exploitation
11. Indépendance vis-à-vis du réseau
12. Indépendance vis-à-vis du SGBD

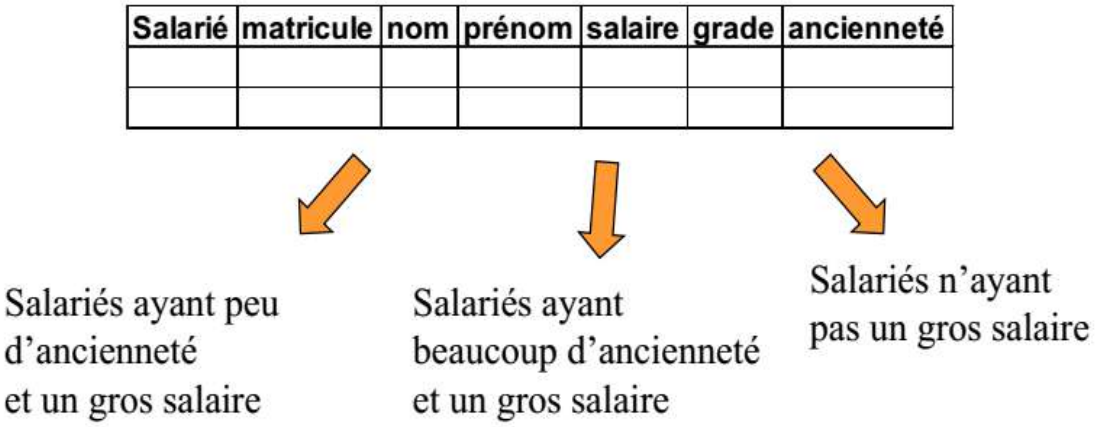
I.2.Fragmentation verticale :

Cette opération peut se faire en faisons une subdivision des attributs dans des groupes en dupliquant une clé commune comme le montre l'exemple suivant :



I.3.Fragmentation horizontale

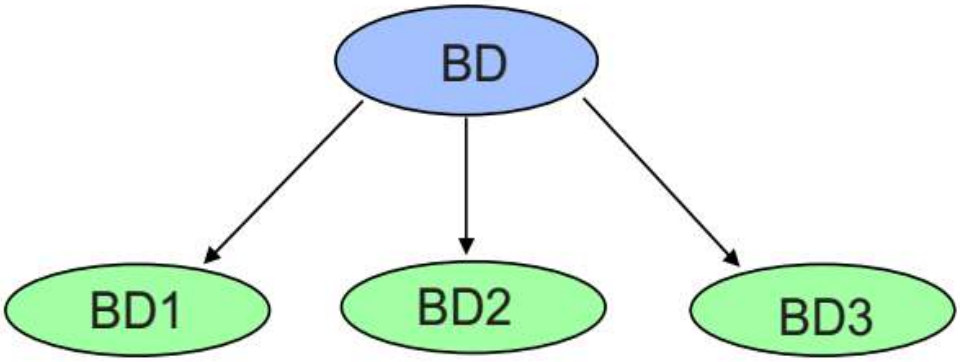
Cette opération peut être faite en utilisant une Sélection selon un prédicat de qualification et doit être réversible par union comme le montre l'exemple suivant :



II. Migration vers une BDR

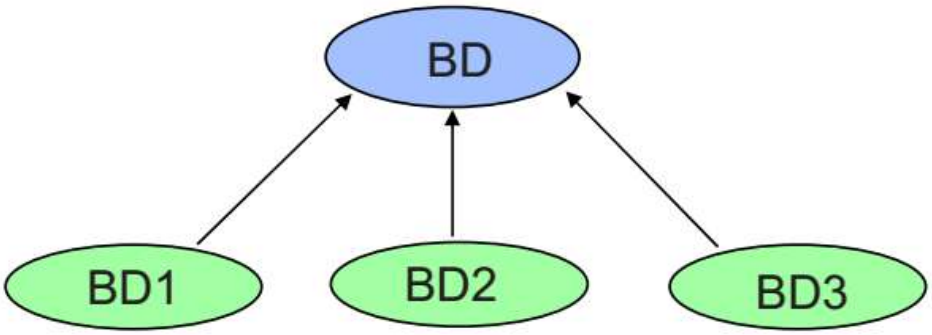
II.1.Décomposition physique en BD locales

Ce type de décomposition peut se faire à l'aide d'un schéma de placement règles de correspondance avec les données locales (Offres = Offres@site1 ∪ Offres@site2).



II.2.Intégration logique des BD locales existantes

L'opération d'intégration peut se faire en utilisant un Schéma conceptuel global donne la description globale et unifiée de toutes les sources de données. ex. Offres (emploi, ville, date)



Les phases d'intégration de plusieurs sources de données comprennent :

1) La pré-intégration : cette opération consiste à faire l'identification des éléments reliés et établissement des règles de conversion (e.g. 1 pouce = 2,54 cm)
2) La comparaison permet d'identifier des conflits de noms (synonymes et homonymes) et des conflits structurels (types, clés)
3) La mise en conformité consiste à faire une résolution des conflits de noms et des conflits structurels (changements de types, de clés)
4) La fusion et la restructuration : cette phase permet de faire la fusion des schémas intermédiaires pour créer le schéma intégré

III. Modèles d'intégration de bases de données

III.1. Intégration en relationnel

Ce type d'intégration donne naissance à des structures de données simples et régulières. Cependant, cette solution présente des problèmes de renommage et d'introduction de valeurs nulles.

Emp = Emp@Site1 U Emp@Site2

prenom	nom	ville	tel.
null	P. Dupont	Paris	0140...
Anne	Martin	Nantes	null
null	A. Martin	Nantes	0235…
Jean	Smith	Lille	null

Emp@Site1

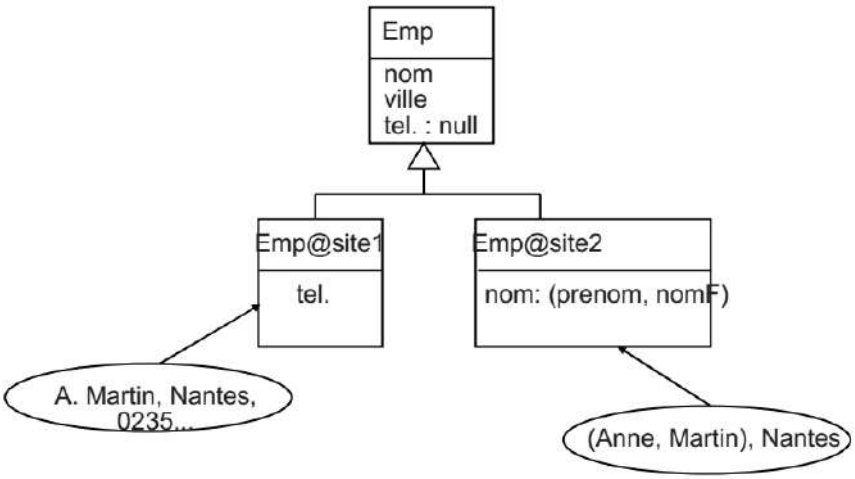
nom	ville	tel.
P. Dupont	Paris	0140...
A. Martin	Nantes	0235...

Emp@Site2

prenom	nomF	ville
Anne	Martin	Nantes
Jean	Smith	Lille

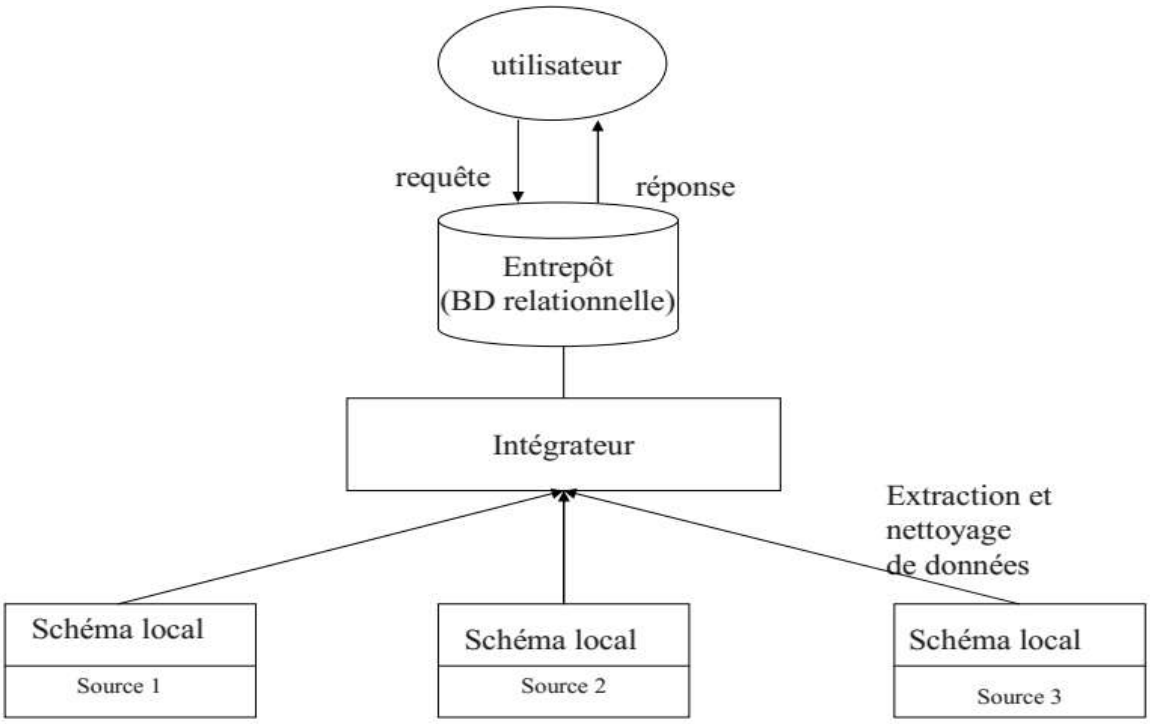
III.2.Intégration en objet

Cette intégration permet de créer des structures de données complexes et régulières. Cependant, elle présente un problème de redéfinition d'attribut.



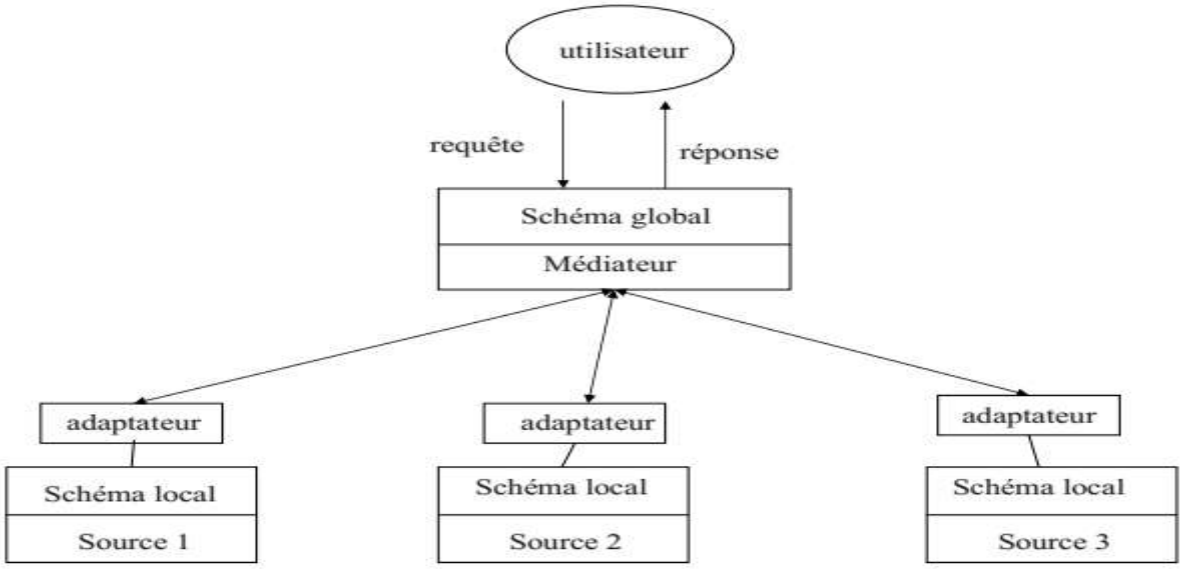
III.3.Entrepôt de données

Parmi les fonctions des entrepôts, il y a le regroupement et la récupération de données existantes ainsi que le référencement des données de manière uniforme. D'un autre côté, les entrepôts des données permettent de sauvegarder l'historique des données afin de mettre à disposition les données pour l'analyse. Bien que les entrepôts de données présentent de bonnes performances, les données ne sont pas toujours fraîches et nécessitent le nettoyage et le filtrage des données.



III.4.Médiateurs d'information

La médiation des données permet l'intégration du modèle semi-structuré (XML) en utilisant des structures de données complexes et irrégulières sans se contraindre par un schéma obligatoire.



III.5.Réseaux Pair-à-pair

Les réseaux P2P sont des réseaux d'ordinateurs, où chaque nœud est à la fois client et serveur (met à disposition des ressources, et accède à des données se trouvant sur d'autres nœuds). Cependant, ce genre de système n'a pas de vision centralisée du système et ne permet pas de coordination centralisée car il y a une absence de schéma global. D'un autre côté, les réseaux P2P permet l'accès à des fichiers non structuré et une grande autonomie des nœuds.

Conclusion du chapitre IV

L'intégration des données peut se faire grâce à des SGBD répartis, des entrepôts, des médiateurs, des P2P comme c'est indiqué dans le tableau suivant :

	Modèle	Requêtes	Admin.	Hétérogénité	Autonomie	Nb de sources
SGBD répartis	Relationnel	Requêtes complexes, transactions	Globale	Faible	Faible	dizaines
Entrepôts	Relationnel	Complexes, lecture	Globale	Faible	Forte	dizaines
Médiateurs	Relationnel, objet, semi-structuré	Simples, textuelles	Locale	Forte	Forte	centaines
P2P	Fichiers	Simples	Locale	Forte	Forte	milliers

Chapitre V. La gestion des accès concurrents

Introduction

La plupart des systèmes de gestion de base de données (SGBD) actuels sont des systèmes multi-utilisateurs : Plusieurs utilisateurs peuvent accéder à la base de données de façon concurrente, "en même temps". Par exemple, un même système de réservations de billets d'avions est utilisé en concurrence par des centaines d'employés d'agences de voyage. Cette utilisation multi-utilisateurs de systèmes informatiques est rendue possible grâce au concept de multitâche, qui autorise le système qui ne dispose que d'une unité centrale de calculs à gérer, en même temps, plusieurs programmes. Seulement ce concept pose des problèmes quant à la cohérence et l'intégrité de la base de données. Cette notion de partage de ressources est très intéressante pour l'utilisateur et constitue un enjeu majeur dans un SGBD : Celui-ci devra donc gérer les utilisations concurrentes sur les données avec le plus d'efficacité possible (telecom-paris, 2011).

I. Le concept de transaction

Une transaction est une unité de programme qui accède aux données de la base en lecture et/ou écriture. Une transaction accède à un état cohérent de la base. Cependant, durant l'exécution d'une transaction, l'état de la base peut ne pas être cohérent. Quand une transaction est validée (commit), l'´état de la base doit être cohérent. Deux types de problèmes peuvent surgir : (1) les problèmes systèmes (récupérabilité) et (2) les exécutions concurrentes de plusieurs transactions (sérialisabilité). Par ailleurs, une transaction est une unité logique de traitement ou une suite d'opérations interrogeant et/ou modifiant la BD et pour laquelle l'ensemble des opérations doit être soit validé, soit annulé.

L'accès concurrent se produit lorsque plusieurs utilisateurs accèdent en même temps à la même donnée dans la BD. Cette BD est dite cohérente si les contraintes d'intégrité sont respectées. L'intégrité des données est sauvegardée car pendant le déroulement d'une transaction, une BD passe d'un ´état cohérent à un autre ´état cohérent. Enfin, la consistance des données est garantie par le SGBD car les données utilisées par une transaction ne sont pas modifiées par des requêtes d'autres transactions.

Les transactions représentent des évènements du monde réel tels que la vente d'un produit ou un dépôt d'argent sur un compte bancaire. Si on lit à partir d'une base de données ou si on écrit pour une mise à jour, on réalise une transaction. Une transaction est ainsi un ensemble de requêtes élémentaires sur les données d'une base. Elle peut consister en un simple SELECT pour générer une liste de tuples d'une table ou en une série de commandes UPDATE pour mettre à jour les données. Par exemple, si on effectue une vente de produit à un client, la transaction consiste au moins en deux parties : il faut mettre à jour l'inventaire en soustrayant la quantité vendue du produit d'une part, et mettre à jour la table des encaissements à réaliser pour facturer le client d'autre part (telecom-paris, 2011).

I.1.Propriétés d'une transaction

Pour préserver l'intégrité des données, le système doit garantir certaines propriétés :

Atomicité : Soit toutes les opérations de la transaction sont validées, ou bien aucune opération ne l'est. Une transaction doit être soit entièrement réalisée, soit annulée. Aucun état intermédiaire n'est accepté. Une transaction est considérée comme une unité logique de travail unique et indivisible.

C**ohérence :** L'exécution d'une transaction préserve la cohérence de la base. Une transaction doit transformer une base de données cohérente en une autre base aussi cohérente.

Isolation : Même si plusieurs transactions peuvent être exécutées en concurrence, aucune n'est censée prendre en compte les autres transactions ; i.e pour chaque paire Ti, Tj, pour Ti tout se passe comme si Tj s'est terminée avant le début de Ti ou bien qu'elle commence après la fin de Ti (les résultats intermédiaires de Tj lui sont invisibles). Durant l'exécution de la transaction, les données ne peuvent pas être utilisées par une seconde transaction jusqu'à ce que la première soit terminée.

Durabilité : Si une transaction est validée, alors tous les changements qu'elle a faits sont persistants (même s'il y a un crash). Ceci indique la permanence de l'état cohérent de la base de données. Lorsqu'une transaction est validée, la base reste dans un état cohérent et cet état ne peut pas être perdu même en cas de défaillance du système.

I.2. Les opérations élémentaires d'une transaction

Les utilisateurs ont la possibilité de manipuler les données de la base suivant des opérations élémentaires. Une transaction a une marque de début "begin of transaction" et une marque de fin "end of transaction". Entre ces deux marques il y a les opérations élémentaires. Dans les bases de données, et bien sûr dans le standard SQL, les requêtes qu'un utilisateur fait se traduisent par les opérations élémentaires suivantes sur les données :
READ : Lecture. On lit la valeur des données d'un objet.
WRITE : Ecriture. On écrit la nouvelle valeur des données d'un objet.
COMMIT : Validation. Après avoir réalisé certaines opérations (READ, WRITE), le processus qui a commencé indique qu'il désire contrôler si tout s'est bien déroulé. Si tous les processus sont d'accord, les changements des opérations précédentes sont enregistrés de manière permanente. Cette opération termine automatiquement une transaction. Sinon, les processus abandonnent la transaction (ABORT).
ABORT : Abandon. Tous les changements des opérations précédentes sont annulés et la base de données revient dans l'état cohérent qui était le sien avant le début de la transaction.
Lorsqu'une transaction est lancée par un utilisateur ou un programme d'application, ses opérations s'exécutent avec succès jusqu'à ce qu'une opération COMMIT ou ABORT soit rencontrée, ceci mettant fin à la transaction.

Exemple : une transaction qui transfert 1000 Eur du compte A vers le compte B

1. Lire(A)
2. A := A − 1000
3. Ecrire(A)
4. Lire(B)
5. B := B + 1000
6. Ecrire(B)

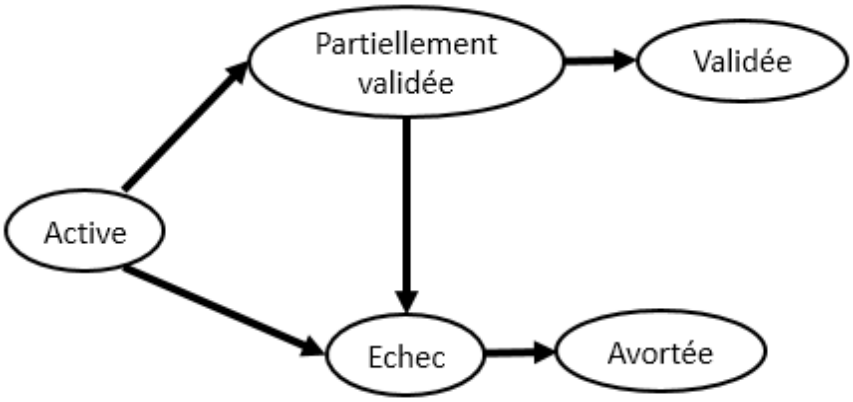
La base est cohérente si la somme A+B ne change pas suite à l'exécution de la transaction (cohérence).

Si la transaction "échoue" après l'étape 3, alors le système doit s'assurer que les modifications de A ne soient pas persistantes (atomicité)

Une fois l'utilisateur est informé que la transaction est validée, il n'a plus `à s'inquiéter du sort de son transfert (durabilité)

Si entre les ´étapes 3 et 6, une autre transaction est autorisée `à accéder à la base, alors elle "verra" un ´état incohérent (A+B est inférieur `à ce qu'elle doit être). L'isolation n'est pas assurée. La solution triviale consiste `à exécuter les transactions en s´séquence.

I.3.Etats d'une transaction



II. Les accès concurrents

Quand plusieurs utilisateurs manipulent les mêmes données en même temps, il faut arbitrer entre :

(1) la disponibilité de l'information (ne pas bloquer tout le monde parce que une personne travaille par exemple ne pas arrêter toutes les connexions à la SNCF quand une personne fait une réservation)
(2) la cohérence de l'information (ne pas rendre les données incohérentes en tenant compte de plusieurs demandes en concurrence en même temps par exemple donner la même place de train `a 2 personnes...)

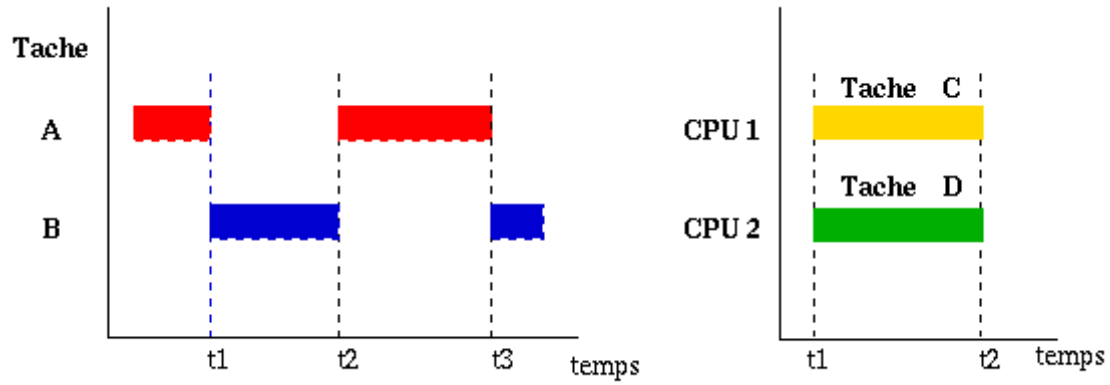
II.1. Les systèmes multitâches

Un critère pour classifier les systèmes de base de données est le nombre d'utilisateurs qui peuvent utiliser le système en concurrence, c'est à dire en même temps. Un système est multi-utilisateur si plusieurs utilisateurs peuvent y accéder en concurrence. Les utilisateurs peuvent utiliser des systèmes en concurrence parce que les systèmes informatiques actuels ont la possibilité de travailler en multitâches.

Si une seule CPU est présente, elle peut exécuter au plus une tâche (programme ou transaction) à la fois. Le travail en multitâche en vue d'optimisation du temps CPU est géré par l'ordonnanceur. Suivant un ordre de priorité (s'il y en a un) ou suivant un ordre arbitrairement défini, l'ordonnanceur va attribuer le processeur à chaque tâche à tour de rôle : il exécute quelques commandes d'une tâche, suspend ensuite cette tâche pour exécuter quelques

commandes de la tâche suivante et ainsi de suite. Chaque tâche est reprise là où elle avait été suspendue. Chaque utilisateur a ainsi l'impression d'être seul avec sa machine.

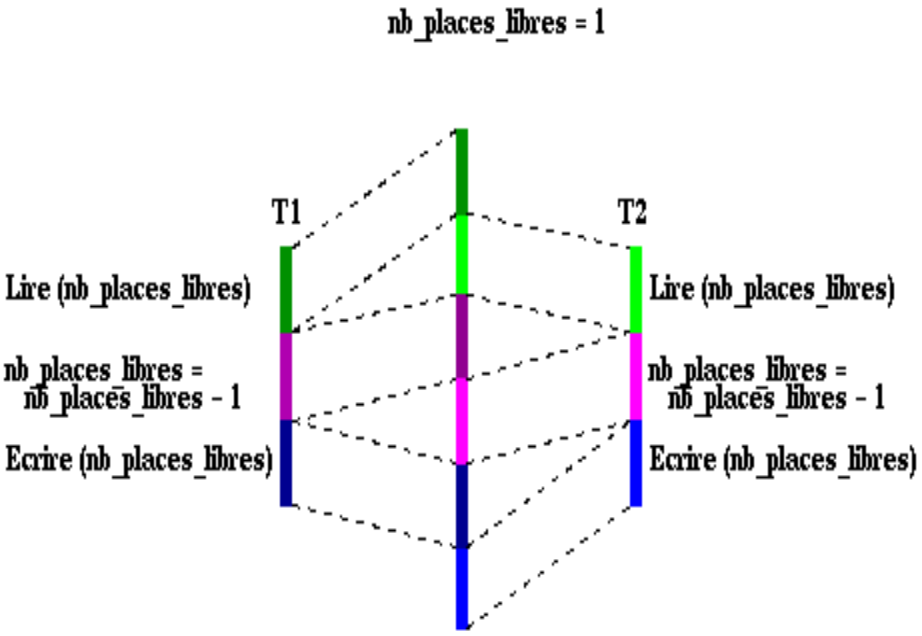
Comme l'illustre la figure suivante, l'exécution concurrente des programmes A et B se fait de manière imbriquée. S'il y a plusieurs CPUs, l'exécution simultanée de plusieurs tâches est possible (comme illustré par les tâches C et D), conduisant à une concurrence simultanée plutôt qu'imbriquée.



Nous considèrerons le problème du contrôle de la concurrence sur les bases de données en termes de concurrence imbriquée. Il peut être adapté à la concurrence simultanée.

II.2. Problèmes posés par les transactions concurrentes

Considérons deux transactions faisant une réservation d'une place d'avion, alors qu'il n'y a plus qu'une place de disponible. Les deux transactions pourront néanmoins effectuer les deux réservations, conduisant à une incohérence de la base de données.

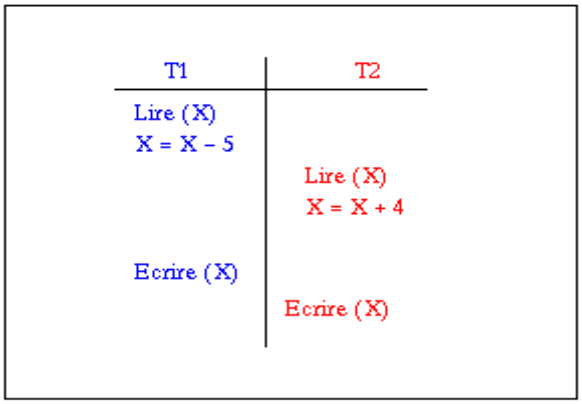


Plusieurs problèmes peuvent ainsi se poser lorsque des transactions concurrentes s'exécutent de manière non contrôlée.

a) Perte de mise à jour

La perte de la mise à jour d'une donnée peut avoir lieu lorsque deux transactions qui accèdent aux mêmes données ont leurs opérations imbriquées, si bien qu'une valeur d'une donnée devient incorrecte.

Ainsi sur la figure suivante, la valeur finale de X est incorrecte, car T2 lit la valeur de X avant que T1 la change dans la base de données. Si X = 80 au début, T1 retranche 5 et T2 ajoute 4, la valeur finale devrait être 79. Mais elle sera de 84, car la mise à jour X-5 de T1 est perdue.

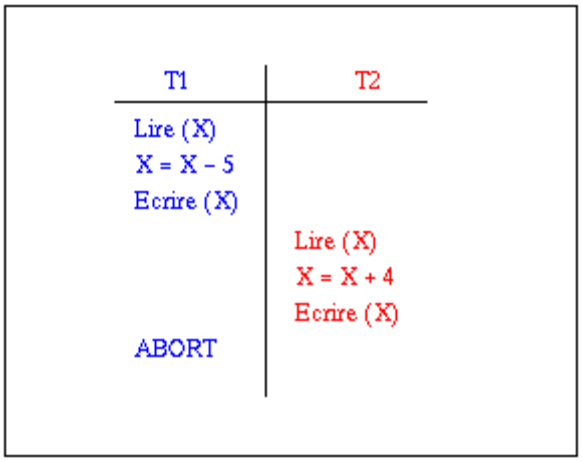


L'anomalie de perte de mise à jour résulte de la dépendance Ecriture-Ecriture entre actions de plusieurs transactions interférentes.

b) Lecture impropre (Dirty read)

Une lecture d'une valeur temporaire de donnée peut conduire à une valeur fausse. Ceci a lieu lorsqu'une transaction met à jour une donnée de la base et que cette transaction échoue ou est annulée. La valeur mise à jour est lue par une seconde transaction avant que la donnée ait retrouvé sa valeur initiale conduisant à un "dirty read".

Dans la figure suivante, T1 met à jour la donnée X et ensuite échoue et est annulée : X retrouve sa valeur initiale (80). T2 a lu entre-temps la valeur temporaire de X (75) qui ne sera pas enregistrée à cause de l'échec de T1. La valeur de X lue par T2 est donc fausse.

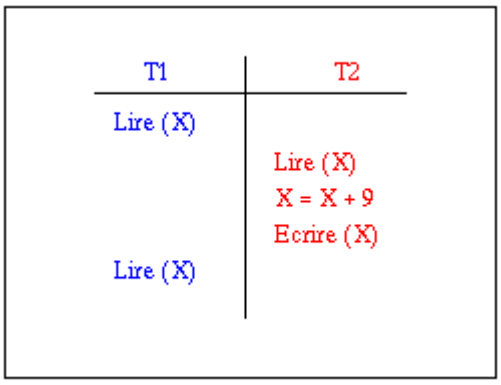


L'anomalie de lecture impropre résulte de la dépendance Ecriture-Lecture entre actions de différentes transactions.

c) Lecture non reproductible

Dans le corps de la même transaction, la relecture d'une donnée peut produire un résultat différent du fait de la modification de la donnée par une autre transaction entre les deux lectures successives.

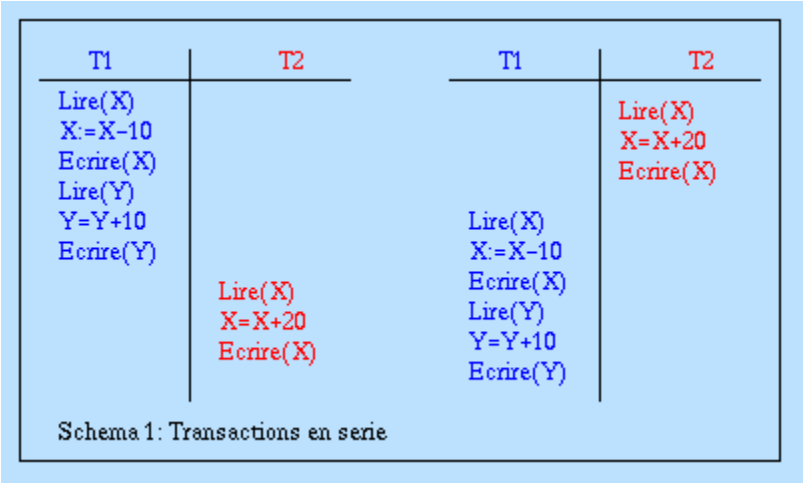
Sur la figure suivante, T1 relit la valeur de X après que T2 la change dans la base de données. Si X = 80 au début, T1 lue 80 et T2 aussi. T2 ajoute 9 et quand T1 relue la valeur de X, elle est 71. Mais elle devrait être 80 car T1 n'a pas changé la valeur de X.



L'anomalie de lecture non reproductible est un cas particulier de lecture incohérente due à l'observation d'états momentanément incohérents. Cette anomalie qui résulte de la dépendance Lecture-Ecriture entre actions correspond à une destruction de l'intégrité interne des données.

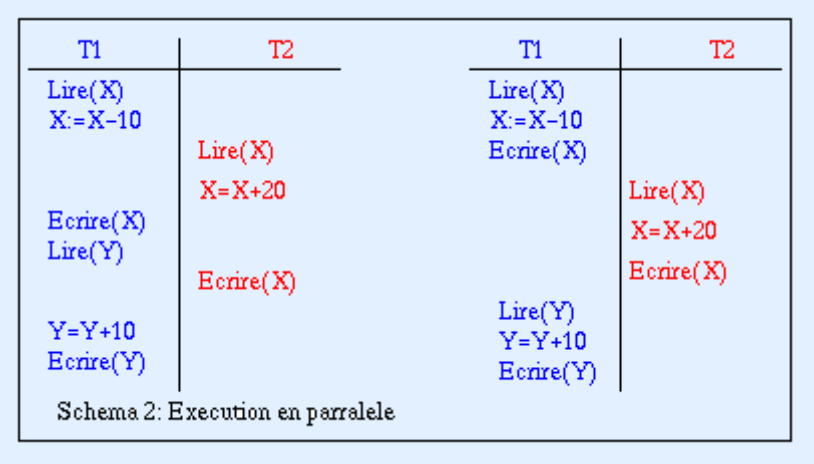
III. Control des accès concurrents

Supposons maintenant que deux agents dans deux agences de voyage différentes veulent effectuer chacun une transaction sur une même base de données. Supposons de plus que ces transactions sont exécutées séquentiellement. Pour l'exécution, il existe deux façons d'ordonner les opérations constituant les deux transactions :(1) on exécute toutes les opérations de T1 consécutivement, puis on fait de même avec les opérations de T2 et (2) On exécute toutes les opérations de T2 consécutivement, puis on fait de même avec T1.Le schéma suivant montre la différence entre les 2 exécutions :



Ces deux exécutions en série, qui ne diffèrent que par l'ordre des transactions, laissent la base par définition cohérente, dans un état stable.

Supposons maintenant que les transactions ne sont plus exécutées les unes après les autres, mais sont exécutées en multitâche. On se rend alors compte que plus de deux alternatives sont possibles. Un exemple en est montré ci-dessous :

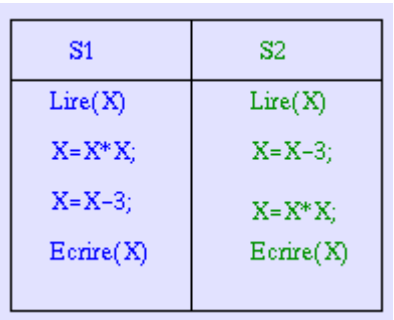


Schema 2: Execution en parralele

Un aspect important du contrôle de concurrence est la théorie de la sérialisabilité : Cette théorie vise à déterminer quelles exécutions sont «correctes » ou pas et permettent de développer des techniques qui forcent la sérialisabilite d'exécutions. Avant de définir clairement la notion de sérialisabilité, il est nécessaire d'introduire la notion d'exécutions équivalentes.

III.1. Notion d'équivalence d'exécution

On dit que deux exécutions sont équivalentes si elles produisent le même résultat : même état final de la base de données. Cependant, cette définition est ambigüe puisqu'il peut arriver que deux exécutions donnent accidentellement le même résultat pour une donnée particulière sans pour autant être équivalentes. L'exemple qui suit illustre bien le problème. Soit les deux exécutions suivantes :



III.2.Des opérations conflictuelles

Deux opérations sont conflictuelles lorsqu'elles proviennent de deux transactions différentes, qu'elles veulent accéder à un même objet de la base de données et que l'une de ces opérations est une opération d'écriture. C'est en effet dans ce cas, qu'apparaissent des conflits. En voici un exemple :
L'opérateur1 veut lire une donnée O1 et L'opérateur 2 veut changer la donnée O1 et donc avoir un accès en écriture de O1. Si les transactions sont exécutées en parallèle, il va falloir déterminer qui de l'opérateur 1 ou 2 aura la main en premier sur l'objet O1 : L'ordre de ces 2 transactions détermine le résultat. On peut en déduire une deuxième définition de l'équivalence entre exécutions : « On dira que deux exécutions sont équivalentes si l'ordre d'exécution de toutes opérations conflictuelles est le même dans les deux exécutions ».

III.3.La sérialisabilité

Il existe un algorithme qui permet de déterminer la sérialisabilité d'une exécution. Cet algorithme met en œuvre des relations de précédence entre les transactions : Ces relations déterminent par rapport à une exécution en parallèle choisie l'ordre dans lequel doivent s'exécuter les transactions. Une fois toutes les relations de précédence établies, on peut générer un graphe dit de sérialisablité. La théorie dit alors que l'on a une exécution non sérialisable si et seulement si le graphe de sérialisabilité de cette exécution possède un cycle. Autrement dit, une exécution est sérialisable si et seulement si son graphe de sérialisabilité ne présente aucun cycle.
L'algorithme de sérialisabilité est décrit ci-dessous :

1. *Chaque transaction est un nœud du graphe ;*

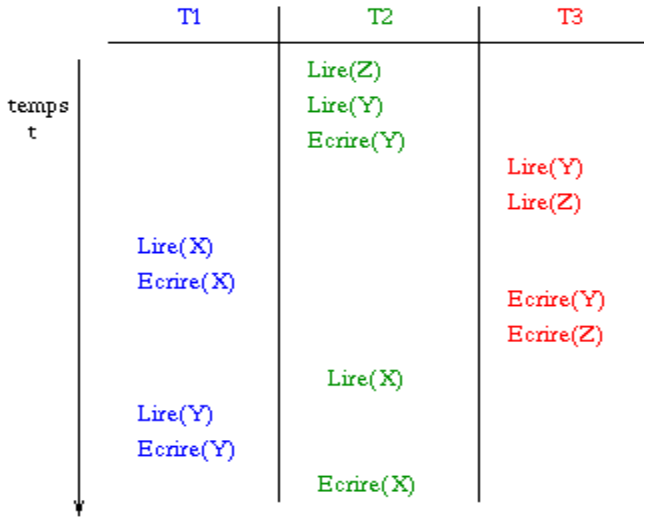
2. *Pour un objet O : Si une transaction T1 fait une écriture de l'objet O avant qu'une transaction T2 ne fasse une opération sur cet objet alors, dans le graphe, T1 précède T2 : Cette relation de précédence est concrétisée par une branche orientée de T1 vers T2.*

3. *Pour un objet O : Si une transaction T1 fait une lecture sur l'objet O avant qu'une transaction T2 ne fasse une opération d'éciture sur cet objet, alors, dans le graphe, T1 précède T2 :*

4. *L'exécution est sérialisable si et seulement si le graphe ne présente aucun cycle.*

Dans ce qui suit, nous allons voir sur trois exemples, comment se traduit une exécution en termes de relations de précédence.

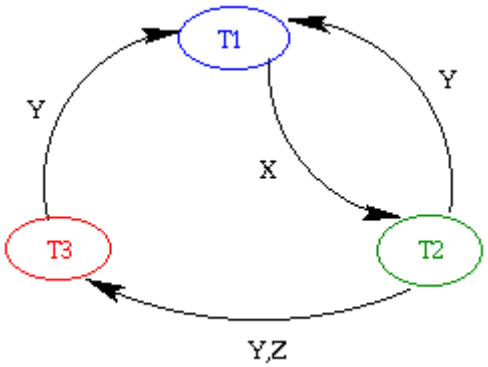
Exemple 1.



Pour L'objet X:
Lecture de T1, Ecriture de T1, Lecture de T2, Ecriture de T2.
Relation de précédence : T1 précède T2 (T1→T2).

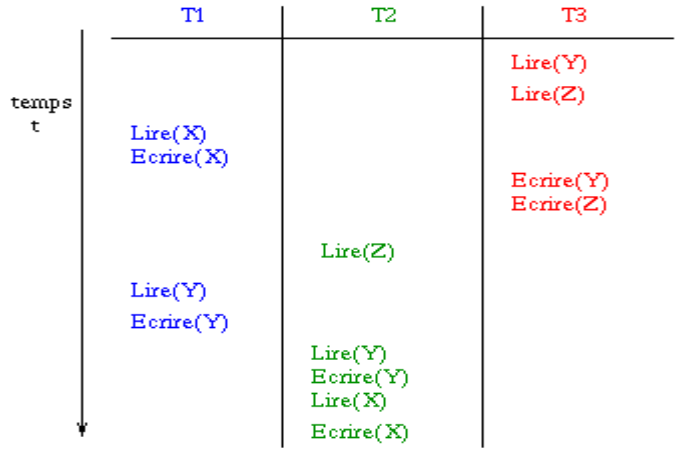
Pour l'objet Y:
Lecture de T2, Ecriture de T2, Lecture de T3, Ecriture de T3, Lecture de T1, Ecriture de T1.
Relation de précédence : T2 écrit avant T3 donc T2 précède T3 (T2→T3)
T2 écrit avant T1 donc T2 précède T1 (T2→T1)
T3 écrit avant T1 donc T3 précède T1 (T3→T1)

Pour l'objet Z :
Lecture de T2, Lecture de T3, Ecriture de T3.
Relation de précédence : Le fait que T2 lise avant que T3 ne lise ne provoque pas de conflit : les deux premières opérations de la liste précédentes ne signifient pas que T2 précède T3. Cependant, la précédence doit être exprimée ici, parce que T3 fait par la suite une opération d'écriture sur Z (T2→T3).



X (T1→T2) Y (T2→T3) (T2→T1) (T3→T1) Z (T2→T3).

Exemple 2.



Pour L'objet X:

Lecture de T1, Ecriture de T1, Lecture de T2, Ecriture de T2.

Relations de précédence : On détermine sans difficulté que T1 précède T2 (T1→T2).

Pour l'objet Y :

Lecture de T3, Ecriture de T3, Lecture de T1, Ecriture de T1, Lecture de T2, Ecriture de T2.

Relations de précédence :

T3 écrit avant T1 donc T3 précède T1 (T3→T1)

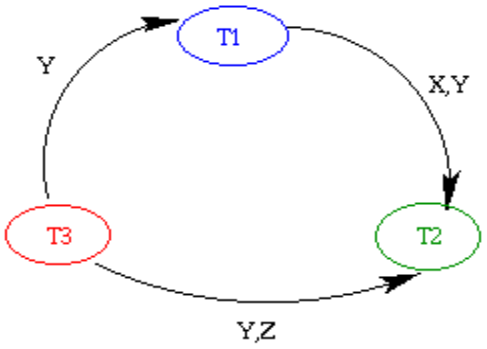
T3 écrit avant T2 donc T3 précède T2 (T3→T2)

T1 écrit avant T2 donc T1 précède T2 (T1→T2)

Pour l'objet Z :

Lecture de T3, Ecriture de T3, Lecture de T2.

Relations de précédences : T3 écrit avant T2 donc T3 précède T2 (T3→T2).



(T1→T2) (T3→T1) (T3→T2) (T1→T2) (T3→T2)

Il est clair que l'exécution de transactions en série n'est absolument pas efficace en termes d'optimisation de temps CPU, temps de traitement de l'information pour le processeur. Une exécution sérialisable de transactions nous offre les bénéfices d'une exécution en multitâche sans les inconvénients d'incohérence des données engendrés par ce type de traitement.

En pratique, il est quasiment impossible de tester une exécution en termes de sérialisabilité : en effet, l'ordonnancement des opérations des différentes transactions impliquées est typiquement traité par le processeur : Il est très difficile de le connaître en général, à priori. Il dépend de nombreux facteurs : les priorités des transactions, le temps de soumission d'une transaction... Déterminer l'ordre d'exécution des opérations des différentes transactions est en pratique impossible à faire. Il devient alors impossible de créer le graphe de sérialisabilité et donc de définir si oui ou non, cette exécution est sérialisable.

On résout alors le problème du en utilisant des méthodes qui assure qu'une exécution sera sérialisable. Ces méthodes sont en fait des ensembles de protocoles et de règles qui sont imposées/appliquées par le système de gestion de base de données.

III.4.Utilisation des verrous

Le contrôle de concurrence consiste à coordonner l'exécution simultanée de transactions dans un système multitâches de gestion de base de données. L'objectif est d'assurer la sérialisabilité des transactions.

Le verrouillage est un bon moyen d'assurer la sérialisabilité et d'imposer que les accès aux données de la base se fassent de façon mutuellement exclusive.
Un verrou garantit un usage exclusif d'une donnée à une transaction. Un verrou sert à interdire l'accès à une donnée qui est en cours d'utilisation par une autre transaction. Si une transaction T1 pose un verrou sur l'accès d'une donnée, une transaction T2 n'aura pas accès à cette donnée. Le verrou est levé lorsque la transaction T1 est terminée, la transaction T2 pourra alors poser elle-même un verrou pour son usage.

Notons qu'une transaction doit maintenir son verrou sur une donnée aussi longtemps qu'elle y a accès. De plus, il n'est pas toujours souhaitable qu'une transaction déverrouille la donnée qu'elle traite immédiatement après y avoir eu accès, puisqu'on n'est pas sûr de sa sérialisabilité. Pour assurer la consistance : verrouiller ces données pendant la durée de la transaction.3 types de verrous peuvent être utilisés : (1) **verrous partagés** (shared lock), pour lire des données avec l'intention d'y faire des mises à jour (2) **verrous exclusifs** (exclusive lock), pour modifier des données et (3) **verrous globaux** (global lock) pour bloquer un ensemble de données, généralement une table toute entière.

Lorsque deux transactions accèdent en même temps à la même donnée, il y a un grand risque pour engendrer une perte de cohérence de la BD. Pour cette raison, une des solutions consiste à placer un verrou à la fin de la transaction en cours. Après COMMIT ou ROLLBACK, les verrous placés par une transaction peuvent être relâchées.

Dans ce contexte, un verrou garantit un usage exclusif d'une donnée à une transaction car il sert à interdire l'accès à une donnée qui est en cours d'utilisation par une autre transaction. Si une transaction T1 pose un verrou sur l'accès d'une donnée, une transaction T2 n'aura pas accès à cette donnée. Le verrou est levé lorsque la transaction T1 est terminée, la transaction T2 pourra alors poser elle-même un verrou pour son usage.

a) Collision de type perte de mise à jour : le problème

T1 et T2 modifient simultanément la quantité qte. A la fin, ` qte : 1500 ? ou 4500 ?

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2			Lire qte
t_3		qte←qte+3000	
t_4	qte=4000	Écrire qte COMMIT	
t_5			qte←qte+500
t_6	**qte=1500**		Écrire qte COMMIT

b) Collision de type perte de mise à jour : souhait

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Écrire qte COMMIT	
t_4			Lire qte
t_5			qte←qte+500
t_6			Écrire qte
	qte=4500		COMMIT

❖ Collision de type perte de mise à jour : solution

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		**Verrou exclusif** sur qte + Lire qte	
t_2			**Attente** pour poser **verrou exclusif** sur qte
t_3		qte←qte+3000	
t_4	qte=4000	Écrire qte COMMIT	
t_5			**Verrou exclusif** sur qte
t_6			Lire qte
t_7			qte←qte+500
t_8	**qte=4500**		Écrire qte COMMIT

❖ Collision de type lecture impropre (ou inconsistante) : le problème

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Ecrire qte	
t_4			Lire qte
t_5			qte←qte+500
t_6	qte=4500		Écrire qte COMMIT
t_7	**qte=4500**	Annuler T_1 (ROLLBACK)	

❖ Collision de type lecture impropre (ou inconsistante) : le souhait

Temps	État de la base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Écrire qte	
t_4	qte=1000	Annuler T_1 (ROLLBACK)	
t_5			Lire qte
t_6			qte←qte+500
t_7	**qte=1500**		Écrire qte COMMIT

❖ Collision de type lecture impropre : solution

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		**Verrou exclusif** sur qte + Lire qte	
t_2		qte←qte+3000	
t_3	qte=4000	Écrire qte	
t_4			**Attente** d'un **verrou exclusif** sur qte
t_5	qte=1000	Annuler T_1 (ROLLBACK)	**Verrou exclusif** sur qte
t_6			Lire qte
t_7			qte←qte+500
t_8	**qte=1500**		Écrire qte COMMIT

❖ Collision de type lecture non reproductible : le problème

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		Traitement impliquant qte	
t_3	qte=1000		**Verrou exclusif** sur qte Lire qte qte←qte+1000
t_4	qte=2000		Écrire qte COMMIT
t_5	**qte=2000**	Lire qte	

❖ Collision de type lecture non reproductible : souhait

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		Lire qte	
t_2		Traitement impliquant qte	
t_3	**qte=1000**	Lire qte	
t_4	qte=1000		**Verrou exclusif** sur qte Lire qte qte←qte+1000
t_5	qte=2000		Écrire qte COMMIT

❖ Collision de type lecture non reproductible : solution

Temps	État base	Transaction T_1	Transaction T_2
t_0	qte=1000		
t_1		**Verrou partagé** sur qte + Lire qte	
t_2			**Attente verrou exclusif** sur qte
t_3		Traitement impliquant qte	
t_4	**qte=1000**	Lire qte + Traitement impliquant qte COMMIT	
t_5	qte=1000		**Verrou exclusif** sur qte Lire qte qte←qte+1000
t_6	**qte=1500**		Ecrire qte + COMMIT

Le mécanisme de verrouillage permet à une transaction de se réserver l'usage exclusif d'une donnée aussi longtemps que c'est nécessaire. Un inconvénient du verrouillage est son coût élevé lorsque les transactions font référence à de nombreuses données de la base.

En contrepartie, le degré de parallélisme est maximum puisque seules sont verrouillées les données réellement manipulées par les transactions mais on doit aussi tenir en compte le problème de verrous mortels.

III.5.Estampillage

La méthode de verrouillage garantit la sérialisabilité d'une exécution de transactions en parallèle. Si une transaction demande un objet qui est bloqué, il doit attendre que l'objet se libère. L'ordre est fixé à l'exécution suivant l'ordre dans lequel les transactions successivement demandent les objets. La méthode de l'estampillage est une autre approche du problème car l'ordre est fixé à priori suivant l'ordre d'apparition des transactions afin de garantir la sérialisabilité d'une exécution.

Une estampille doit identifier de façon unique une transaction et doit être créée avant l'exécution de la transaction. Les estampilles sont gérées par le SGBD et sont assignées typiquement dans l'ordre chronologique de soumission des transactions au système. Ainsi, on peut considérer l'estampille d'une transaction comme la date de début de celle-ci. Le système doit gérer les estampilles de façon que chaque estampille soit unique. Pour ce faire, il y a deux méthodes :

1. Utilisation de l'horloge système pour dater les transactions. Il faut alors vérifier l'unicité des estampilles.
2. Utilisation d'un compteur qui s'incrémente à chaque fois que sa valeur est attribuée à une transaction : cette valeur est alors l'estampille de la transaction. Comme le compteur a une valeur maximale finie, il faut veiller à le remettre à zéro pendant une période où il n'y a pas de transactions à exécuter.

L'estampille d'un objet correspond à l'estampille de la dernière transaction qui a accédé à cet objet. On distingue alors deux types d'estampille sur un objet :

(a) **L'estampille en écriture :** correspond à l'estampille la plus élevée parmi celles des transactions ayant effectué avec succès une opération de lecture sur l'objet : la plus jeune de ces transactions.
(b) **L'estampille en lecture :** correspond à l'estampille la plus élevée parmi celles des transactions ayant effectué avec succès une opération d'écriture sur l'objet : la plus jeune de ces transactions.

L'idée de cette méthode est de mettre en œuvre un protocole d'ordonnancement par estampilles afin qu'il s'assure que les opérations conflictuelles de lecture et d'écriture sont exécutées par ordre d'estampille.

Supposons par exemple une exécution avec deux transactions en concurrence T1 et T2.

E(T1) est l'estampille de T1 et E(T2) l'estampille de T2.

Supposons que nous avons E(T1)=1 et E(T2)=2.

Comme E(T1)>E(T2), on sait que l'ordre de soumission des transactions au système est T1 puis T2. Le protocole d'estampillage doit faire respecter cet ordre T1 puis T2. Si T1 lit O avant que

T2 ne fasse une opération d'écriture sur l'objet. T1 précède bien T2 : On respecte l'ordre des estampilles. Cependant, T2 fait une écriture sur O avant que T1 ne fasse d'opérations sur l'objet. L'ordre des estampilles n'est pas respecté. Le protocole règle ce problème, en abandonnant T1 : T1 est alors soumis à nouveau au système avec une nouvelle estampille. L'ordre des estampilles sera alors : T2 puis T1.

Il est clair que le protocole vérifie le jeu des estampilles si une opération conflictuelle (lecture et écriture par 2 transactions différentes ou écriture et écriture) s'opère dans l'ordre dicté par les estampilles. Si c'est le cas, on abandonne la transaction la plus vieille (celle qui a l'estampille la plus faible) : Cet algorithme privilégie les transactions les plus récentes. En effet, l'estampillage définit un protocole qui permet de décider laquelle de deux transactions qui veut accéder à un objet doit être éliminée. Ce protocole garantit la sérialisabilité de l'exécution. De plus, contrairement au verrouillage, on est exempt de problème de blocage puisqu'aucune transaction n'attend une autre transaction.

Conclusion du chapitre IV

La notion d'accès concurrent décrit la situation où plusieurs applications veulent accéder à la même donnée en même temps. Par exemple, il est possible que deux applications bancaires accèdent simultanément au compte du même client (à la même donnée). Si les accès sont conflictuels, c'est-à-dire si au moins une des deux applications veut modifier le compte, le S.G.B.D. doit réagir afin d'éviter que la base de données se trouve dans un état incohérent à la fin des opérations. Le groupement d'opérations en transactions permet une vision cohérente des données.

Une transaction rassemble un ensemble d'opérations qui doivent toutes avoir la même vision d'une base de données. Ainsi, deux opérations successives de lecture du compte d'un client à l'intérieur d'une transaction doivent retourner la même valeur sans être dérangées par des opérations d'écriture par une autre transaction.

Les applications sont ainsi perçues par le S.G.B.D. comme un ensemble de transactions indépendantes auxquelles il doit fournir un accès cohérent aux données. Cette cohérence est formalisée par la notion de sérialisabilité.

L'exécution des opérations d'un ensemble de transactions est sérialisable s'il existe un ordre (partiel) < entre les transactions tel que toutes les opérations d'une transaction T qui sont en conflit avec les opérations d'une autre transaction T' < T, sont exécutées après ces opérations (deux opérations, comme on l'a dit, sont en conflit si elles accèdent à la même donnée et si au moins une opération est une écriture).

On peut montrer que chaque transaction dans un ensemble sérialisable peut posséder une vision différente des données, mais que chacune de ces visions est cohérente. Pour augmenter le débit transactionnel et réduire l'attente des utilisateurs, les opérations de transactions concurrentes sont interclassées, mais, pour une transaction, tout se passe comme si l'exécution des transactions concurrentes était séquentielle (AMANN & SCHOLL, 2016)

Conclusion générale

Ce polycopié a permis de préparer les apprenants à la conception et l'administration de base de données relationnelle ou post-relationnelles. Le but étant de les initier à utiliser une méthodologie de conception de base de données afin de maîtriser des éléments d'architecture logique et physique d'une base de données relationnelles ou post-relationnelles.

Pour cette raison, ce document a abordé les limites du modèle relationnel afin de donner une ouverture sur les modèles post-relationnels. Il a traité les concepts suivants : le modèle relationnel, le langage SQL, l'intégrité et gestion des transactions, les bases de données post-relationnelles, les entrepôts de données (Datawarehouse), les bases de données semi-structurées, les bases de données réparties et parallèles, les bases de données à objets, les bases de données géographiques, les bases de données multimédia...

A travers ce polycopié, les apprenants peuvent acquérir des savoir-faire comme la modélisation conceptuelle des données, le passage vers le relationnel objet, la mise en œuvre d'une base de données Oracle, l'utilisation des standard SQL et PL/SQL, la création, la gestion des droits d'accès, l'alimentation et la manipulation des entrepôts de données...

Comme future perspective à nos lecteurs, nous recommandons les bases de données de type NoSQL (Not only SQL) désignant une famille de systèmes de gestion de base de données (SGBD) qui s'écarte du paradigme classique des bases relationnelles. Ce genre d'architecture logicielle distribuée permet de remettre en cause de nombreux fondements de l'architecture SGBD relationnelle traditionnelle, notamment les propriétés ACID.

TD 1-1 : Diagrammes Conceptuels

On propose de développer un système permettant de gérer une bibliothèque et mettant en œuvre une base de données. On suppose que la bibliothèque ne contient que des livres (pas de revues. de mémoires) c'est-à-dire un seul type d'ouvrage. Les principales fonctions à prendre en compte sont les suivantes :

- Une demande peut concerner ; Un livre répertorié en bibliothèque. Identifié par l'emprunteur l'aide de sa référencé (référence notée sur l'étiquette située sur la tranche du livre).un livre, dont L'emprunteur ne sait s'il est ou non répertorié en bibliothèque. Dans ce cas l'emprunteur indique Je désirerais un livre qui a été écrit par "x" ? Les recherches et l'identification de l'ouvrage se feront à partir du nom de l'auteur ? : Je désirerais un livre qui concerne "l'informatique" ? L'emprunteur ne désire pas un livre précis, mais plutôt un ouvrage concernant un thème. Il souhaite alors qu'on lui fasse des propositions de titres et de références parmi lesquelles il pourra choisir. On se limitera à ces deux types de recherche. La référence de l'ouvrage ayant été déterminée. Il faut alors déterminer si un exemplaire de l'ouvrage est disponible, et si oui, déterminer sa localisation dans la bibliothèque (N° de rayon, N° étagère).
- Une personne qui a trouvé en bibliothèque un ouvrage qui l'intéresse doit faire enregistrer le prêt. Pour cela on enregistre : l'identification de l'emprunteur, l'identification de l'ouvrage et la date de sortie (AAMMJJ) Il y aura mise à jour du nombre d'exemplaires disponibles. On doit garder une trace des prêts pour assurer le suivi et pour faire des statistiques (détermination des ouvrages les plus consultés). On met à jour les caractéristiques du prêt en rajoutant la date restitution. On doit donc saisir le code emprunteur, le code livre, la date de restitution.
- A chaque entrée de livres en bibliothèque on lui affecte une référence et on précise : son titre, le nom des auteurs, les thèmes concernés. Sa localisation en bibliothèque, le nombre d'exemplaires entrés. Pour préciser la liste des thèmes le bibliothécaire dispose d'une nomenclature permettant d'associer un livre à la liste des thèmes. Lors de l'enregistrement de l'ouvrage, il y a mise à jour des index par auteur et par thème. Les livres entrés sont étiquetés et rangés dans les rayons.
- Lors du premier emprunt, avant l'enregistrement du prêt, on enregistre pour l'emprunteur : son no Insee, nom, le Prénom, son adresse, son téléphone ; et on lui affecte un numéro de code (interne à la bibliothèque).

QUESTIONS

QI) Etablir une liste des attributs susceptibles d'être prises en compte dans la base de données.

Q2) Représenter par un diagramme Entités - Associations le schéma conceptuel de la Base de données.

Q3) Dresser la liste des dépendances fonctionnelles.

Q4) Tracer le graphe des dépendances fonctionnelles mettant en évidence les relations.

Q5) Construire un schéma conceptuel relationnel en troisième forme normale (3 FN).

Q6) Ecrire les instructions SQL qui permette de créer les tables associées à ce modèle.

Q7) Exprimer au moyen du langage SQL les requêtes suivantes :

a) Trouver les références et les titres la bibliothèque.

b) Quels sont les titres de livres écrits par "Victor Hugo" ?

c) Donner toutes les informations concernant les livres d' « informatique"

d) Trouver le nom, le prénom et l'adresse de tous les emprunteurs du livre intitulé "SYSTEMES DE GESTION DE BASES DE DONNEES"

TD 1-2 : Normalisation de schéma conceptuel

Exercice 1 :

Soit un schéma de bases de données contenant les relations suivantes :

- Bureau (NumBureau, NumTelephone, Taille) avec {NumTelephone → NumBureau}
- Occupant (NumBureau, PersonneID) avec {NumBureau → PersonneID}
- Matériel (NumBureau, NumPC) avec {NumPC → NumBureau}

1. Les contraintes ci-dessous sont-elles vérifiées par ce schéma de bases de données ?

(a) "Un bureau peut contenir plusieurs postes téléphoniques."

(b) "Il y a une et une seule personne par bureau."

(c) "Un bureau contient un seul ordinateur."

2. En prenant ces trois contraintes quelle sera la clé minimale possible de ces trois relations ?

3. Proposer un schéma conceptuel normalisé pour cette Base de Données.

Exercice 2 :

Soit R une relation dont le schéma est le suivant :

R (UtilisateurID, Nom, Prénom, AdresseEmail, Login, Passwd, ServeurMail).

1. Exprimez, à l'aide de dépendances fonctionnelles, les contraintes suivantes :

(a) "On peut déduire le nom et le prénom d'un utilisateur à partir de son identicateur."

(b) "Un utilisateur (identifié par son identificateur) possède un seul login et un seul password par serveur de mails."

(c) "Une adresse email est associée à un et un seul identificateur d'utilisateur."

Attention : un utilisateur peut avoir plusieurs adresses de mails.

(d) "Une adresse email est associée à un et un seul serveur de mails."

2. Indiquez, à partir des dépendances fonctionnelles les clés minimales de R.

3. Indiquez, à partir des dépendances fonctionnelles la forme normale de la relation R

Exercice 3 :

Soit le schéma de la relation R (A, B, C, D, E, G) et un ensemble donné de dépendances fonctionnelles pour cette relation :

A → B, C

A, C → E

A, D, E → B, G

C, G → D

B, G → C

C → B

a) Donner le graphe minimal des dépendances fonctionnelles de R

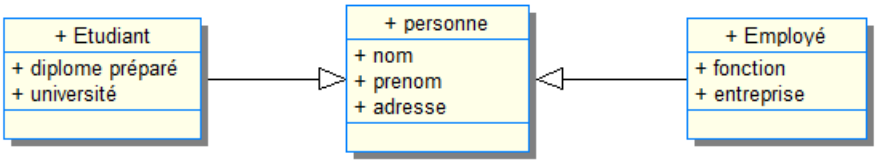
b) Donner une décomposition de R en relations 3FN sans perte d'informations et sans perte de dépendances. Précisez l'identifiant de chaque relation obtenue.

TD 2 : L'objet-relationnel

Exercice 1 :

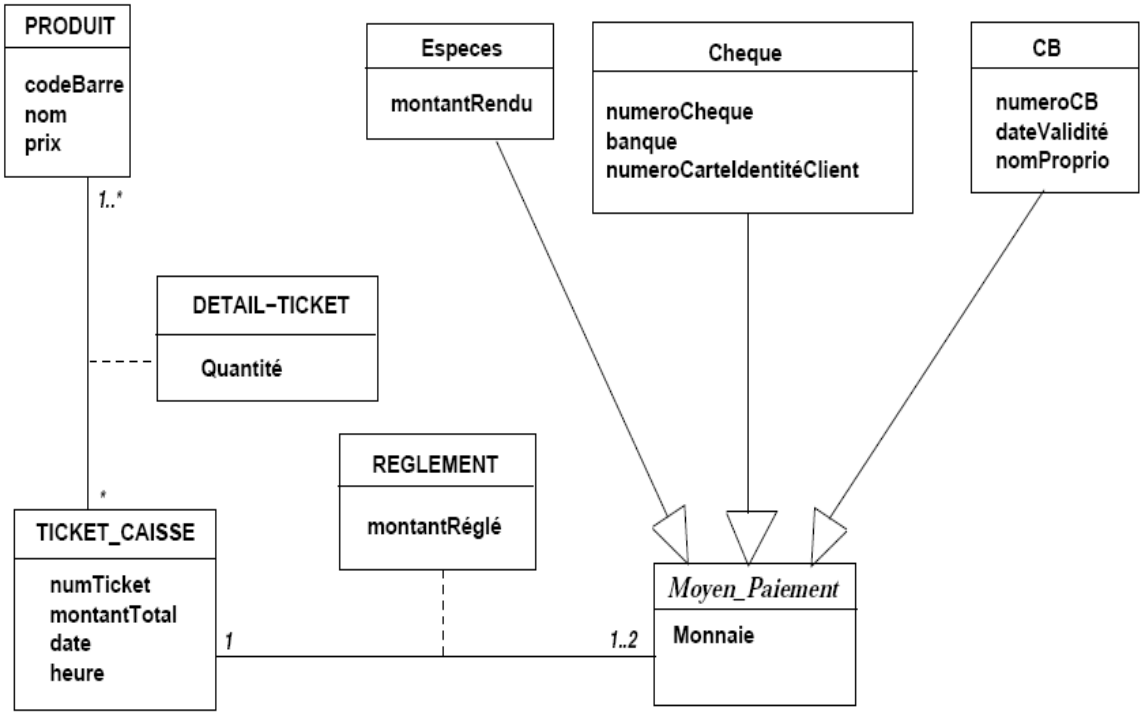
Définir les types objets pour des données suivantes :

1. Un point du plan euclidien.
2. Un segment du plan.
3. Un polygone défini par un ensemble ordonné de points.
4. Définir une table Carrelage pour stocker des carreaux de forme polygonale avec leur couleur et un numéro d'identification.
5. Traduire ce diagramme de classe UML en modèle objet relationnel à l'aide du SQL3



Exercice 2 :

La figure ci-dessous représente une partie du diagramme de classes d'une application de comptabilité de supermarché. Chaque ticket de caisse émis par le supermarché contient une liste de produits vendus avec la quantité de chaque produit. Le supermarché offre la possibilité aux clients de payer le total d'un ticket de caisse avec deux moyens de paiement différents. Un ticket de caisse est par conséquent associé à un ou deux moyens de paiements. Le magasin accepte les paiements en deux monnaies différentes, euros ou dollars.



❖ Etablir le modèle objet relationnel correspondant à ce diagramme.

Exercice 3 :

Soit une base de données concernant des bateaux de guerre. On souhaite exprimer cette base de données en objet- relationnel. Chaque bateau de guerre a les informations suivantes qui lui sont associées : son nom, son poids et son type mais il y a certains bateaux qui ont des spécificités :

1. Les canonniers sont des bateaux qui transportent de grands canons. Pour cela, on souhaite enregistrer le nombre et le calibre des principaux canons.
2. Les porteurs d'avions, pour lesquels nous enregistrons la longueur de la piste d'envol et les ensembles de modèles d'avion qui lui sont assignés.
3. Les sous-marins, pour lesquels la profondeur de la plongée maximum doit être connue.

Aucun sous-marin n'est canonnier, ni même porteur d'avion. Par contre, les cuirassés sont des porteurs d'avion et des canonniers en même temps.

a). Donner le diagramme de classe UML relatif à ce domaine.

b). Donner le script SQL relatif à la création des tables de ce diagramme.

Exercice 4 :

On souhaite concevoir une base de données décrivant des pièces et leurs composants. Une pièce P est composée de plusieurs composants C, chacun des composants pouvant apparaître plusieurs fois dans une pièce. Une pièce peut elle-même être composante d'une autre pièce. Par exemple, un moteur est un composant d'une voiture, et les bougies sont des composants du moteur.

- ❖ Modéliser ces informations sous forme d'un schéma UML.

Exercice n°5 :

L'objectif est d'étudier et de modéliser une application en vue d'en établir une base de données objet-relationnelle. On utilisera le système objet-relationnel Oracle8i comme support d'implantation. Dans une entreprise, on souhaite développer une application CAO pour connaître l'ensemble des éléments entrant dans la composition d'une pièce.

On considère deux types de pièces : (1) les pièces de base et (2) les pièces composites obtenues par assemblage d'autres pièces (composites ou non).

On veut constituer une base de données comportant tous ces types de pièces :

- pour les pièces de base, on stockera son nom et sa matière. On suppose en effet qu'il existe différents types de pièces de base (cubique, sphérique, cylindrique, parallélépipède...) réalisées dans différente matière (bois, acier, plastique...) et dont on connaît les dimensions. Pour chaque matière on connaît son nom, son prix au kilo et sa masse volumique.
- pour les pièces composites, on stockera son nom, le coût de son assemblage, l'ensemble des différentes pièces entrant dans sa fabrication en précisant la quantité et l'ensemble des pièces dans la fabrication desquelles elles entrent en précisant la quantité.
- On veut pouvoir calculer le prix de revient et les différentes caractéristiques physiques (masse, volume) d'une pièce dont on connaît le nom.
- On veut aussi pouvoir connaître le nombre exact de pièces de base entrant dans la fabrication d'une pièce composite.

- ❖ Représenter le diagramme de classe UML associé à cette application.

TD 3 : Les données semi structurées

Exercice 1 :

Une liste de films lfilms contient des films (au *moins un).* Un film contient un titre et zéro ou plusieurs acteurs (dans cet ordre). Un titre et un acteur sont des chaines de caractères PCDATA (Parsed Character Data).

Ecrire la DTD correspondante à la description de cette liste de films ensuite écrivez un exemple du fichier XML correspondant à cette DTD.

Exercice 2 :

Ecrire une DTD pour une bibliographie d'articles. Les informations associées à un article sont :

1. son titre ;
2. les noms des auteurs ;
3. ses références de publication : nom du journal, numéro des pages, année de publication et numéro du journal.
4. On réserve aussi un champ optionnel pour un avis personnel.

Ecrivez un exemple de fichier XML correspondant à cette DTD.

Exercice 3 :

1. Modifier la DTD de l'exercice 2 :
   a. En faisant de l'élément nom_journal un attribut de l'élément journal et en lui donnant comme valeur par défaut ACM ;
   b. En faisant de l'élément annee un attribut de type énuméré, prenant comme valeurs possibles 2012, 2013, 2014, "avant_2012" et proposant comme valeur par défaut "avant_2012".
2. Utiliser cette DTD pour créer un fichier XML valide.

Exercice 4 :

Donner une DTD qui valide des documents sous forme de carnets d'adresses. Un carnet d'adresse a la forme suivante :

• la personne possède un identifiant unique (obligatoire), un nom, un prénom.
• on veut connaître le sexe de la personne (attribut optionnel)
• on veut connaître son email (optionnel)

Exercice 5 :

Ecrire une DTD qui permet d'identifier une personne par un code tout en lui associant son père et sa mère.

Ecrire le fichier XML correspondant à cette DTD.

TD4 : Base de données réparties

Trois universités parisiennes (Jussieu, Sorbonne, Dauphine) ont décidé de mutualiser leurs bibliothèques et leur service de prêts, afin de permettre à l'ensemble des étudiants d'emprunter des ouvrages dans toutes les bibliothèques des universités participantes. Par exemple, un étudiant de Jussieu pourra emprunter des ouvrages à la bibliothèque de la Sorbonne. La gestion commune des bibliothèques et des emprunts est effectuée par une base de données répartie, dont le schéma global est le suivant :

EMPLOYE (Id_pers, nom, adresse, statut, affectation)

L'attribut affectation désigne ici la bibliothèque où travaille l'employé.

ETUDIANT (Id_etu, nom, adresse, université, cursus, nb_emprunts)

L'attribut université indique l'université où est inscrit l'étudiant.

OUVRAGES (Id_ouv, titre, éditeur, année, domaine, stock, site)
L'attribut site indique la bibliothèque qui gère cet ouvrage. L'attribut domaine permet de classer les ouvrages en catégories (physique, maths, informatique, médecine, etc.). L'attribut stock désigne le nombre d'ouvrages restant disponibles au prêt.

AUTEURS (Id_ouv, nom_auteur)

PRETS (Id_ouv, Id_etu, date_emprunt, date_retour)

La gestion de cette application s'appuie sur les hypothèses suivantes :

-Un employé est affecté à un seul site

- un étudiant est inscrit dans une seule université, mais peut emprunter dans toutes les bibliothèques.

- un ouvrage emprunté dans une bibliothèque est rendu dans la même bibliothèque.

- Le champ nb_emprunts de la relation ETUDIANT est utilisé pour limiter le nombre d'ouvrages empruntés simultanément par un étudiant sur l'ensemble des bibliothèques. Il est mis à jour lors de chaque emprunt et chaque retour, quelle que soit la bibliothèque d'emprunt.

- Chaque université gère ses propres étudiants

- Chaque bibliothèque gère son personnel et les ouvrages qu'elle détient.

Les relations globales sont fragmentées et réparties sur les différents sites.

1) Donner la définition des différents fragments en utilisant les opérateurs de l'algèbre relationnelle ainsi que le schéma d'allocation des fragments.
2) Donnez la définition d'une fragmentation correcte. Montrez que la fragmentation que vous proposez pour la relation ETUDIANT est correcte.
3) Donner les opérations de reconstruction des relations globales

TD5 : Les transactions et les accès concurrents

Exercice 1 : « Contrôle de concurrence »

On suppose que x, y, z et t sont des granules d'une base de données. On suppose, de plus, que l'on a cinq transactions qui souhaitent accéder à ces granules. On donne une exécution ci-dessous des cinq transactions :

r1(x) r2(y) w1(y) w3(x) w1(t) w5(x) r4(z) r2(z) w4(z) w5(z) r3(t) r5(t)

Où r désigne une opération de lecture, w une opération d'écriture. Ces opérations sont indicées par la transaction demanderesse et suivies entre parenthèses par le granule concerné.

1) Tracer le graphe de dépendances.
2) Donnez une exécution sérialisable en série.

Exercice 2 : « Comportement d'exécution »

Construisez les graphes de sérialisation pour les trois exécutions suivantes. Indiquez les exécutions sérialisables et vérifiez si parmi ces exécutions il y a des exécutions équivalentes.

E1 : w2[x] w3[z] w2[y] c2 r1[x] w1[z] c1 r3[y] c3

E2 : r1[x] w2 [y] r3 [y] w3 [z] c3 w1 [z] c1 w2[x] c2

E3 : w3[z] w1[z] w2[y] w2[x] c2 r3[y] c3 r1[x] c1

Exercice 3 : « Pannes »

On considère un système de reprise sur panne fonctionnant de la façon suivante : Dans un premier temps, une analyse du journal permet de déterminer les pages « sales », c'est-à-dire les pages dont les modifications n'ont pas encore été écrites sur le disque, et à déterminer les transactions actives. La deuxième phase consiste à refaire les opérations qui doivent être refaites (algorithme REDO). La troisième phase consiste à défaire les opérations qui doivent être défaites (algorithme UNDO). Soit l'extrait de journal suivant :

1. T1 : begin
2. T1 : write (P1)
3. T1 : write (P2)
4. T2 : begin
5. T2 : write (P3)
6. T1 : write (P5)
7. T1: commit
8. T3 : begin
9. T3 : write (P2)
10. T2 : write (P1)
11. T4 : begin
12. T4 : write (P4)
13. T2 : abort
14. T3 : write (P3)

Panne !!!!!!

1) Quelles sont les informations produites par la phase d'analyse ? (Pages sales, Transactions actives)
2) Détaillez les actions de l'algorithme REDO, puis celles de l'algorithme UNDO, en précisant l'ordre de parcours du journal.
3) On suppose qu'une panne se produit pendant l'algorithme UNDO. Que risque-t-on ?

TP 1 : BD relationnel sous Oracle

1. Connexion à la base de données Oracle grâce au compte administrateur :

```
SQL> connect system/inchalah
Connected.
```

2. Création d'un nouvel utilisateur :

```
SQL> create user omar1 identified by 123;

User created.
```

3. Assignation du privilège Ressource à l'utilisateur Omar1 :

```
SQL> grant resource to omar1;

Grant succeeded.
```

4. Connexion de l'utilisateur Omar1 à son schéma de la base de données Oracle

```
SQL> connect omar1/123;
Connected.
```

5. Création de la table relationnelle employé :

```
SQL> create table employe (code number primary key, nom varchar2(20), DN date);
```

6. Création d'une vue employé

```
SQL> connect system/inchalah
Connected.
SQL> grant create view to omar1;

Grant succeeded.

SQL> connect omar1/123;
Connected.
SQL> create view v_employe (code,nom)
  2  as select code,nom
  3  from employe
  4  where code=1;

View created.
```

7. Insertion des données dans la table employé :

```
SQL> insert into employe values (1,'a','01/01/2012');

1 row created.

SQL> insert into employe values (2,'b','02/01/2012');

1 row created.

SQL> insert into employe values (3,'c','03/01/2012');

1 row created.
```

8. Description de la table employé et de la vue employé

```
SQL> desc employe;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------

 CODE                                      NOT NULL NUMBER
 NOM                                                VARCHAR2(20)
 DN                                                 DATE
```

```
SQL> desc v_employe;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------

 CODE                                      NOT NULL NUMBER
 NOM                                                VARCHAR2(20)
```

9. Manipulation des vues relationnelles

```
SQL> insert into v_employe values (4,'c');

1 row created.

SQL> insert into v_employe values (5,'c');

1 row created.

SQL> select * from v_employe;

      CODE NOM
---------- --------------------
         1 a

SQL> create view vemploye (code,nom)
  2  as select code,nom
  3  from employe
  4  ;

View created.

SQL> select * from vemploye;

      CODE NOM
---------- --------------------
         1 a
         2 b
         3 c
         4 c
         5 c

SQL> create view vemp (code,nom)
  2  as select code,nom
  3  from employe;

View created.
```

```
SQL> create view vemp2 (code,nom)
  2  as select code,nom
  3  from employe
  4  with read only;

View created.

SQL> insert into vemp2 values (5,'c');
insert into vemp2 values (5,'c')
*
ERROR at line 1:
ORA-42399: cannot perform a DML operation on a read-only view
```

TP 2 : BD objet-relationnelle sous oracle

L'objectif de ce TP est d'utiliser les nouvelles possibilités offertes par l'objet-relationnel introduit dans le langage SQL 3.

I. Création de nouveaux types

1. Créez un type **adresse_type** avec un numéro de rue, un nom de rue et un nom de ville.
2. Créez un type **departement_type** sur le même modèle que la table **dept**.
3. Créez un type **employe_type** avec un matricule, un nom, une adresse (de type adresse_type), un salaire, une référence à un supérieur, une référence à un département.
4. Créez les 2 tables **departement** et **employe** associées à ces 2 types. N'oubliez pas les contraintes d'intégrité (au moins les clés primaires).
5. Faites afficher les noms de types dont vous disposez

Correction

```
1. CREATE TYPE adresse_type AS OBJECT
   (
    numero INTEGER,
    rue VARCHAR(30),
    ville VARCHAR(20)
   )
   /
2. CREATE TYPE departement_type AS OBJECT
   (
    numDept SMALLINT,
    nomDept VARCHAR(20),
    lieu VARCHAR(50)
   )
   /
3. CREATE TYPE employe_type AS OBJECT
   (
   matricule SMALLINT,
   nom VARCHAR(30),
   adresse adresse_type,
   salaire DECIMAL(8,2),
   superieur REF employe_type,
   departement REF departement_type
   )
   /
 4. CREATE table departement OF departement_type
  (
    constraint pk_departement primary key (numDept)
   )
   CREATE table employe OF employe_type
  (
   constraint pk_employe primary key (matricule)
   )
```

5. Affichage des types et tables crées :

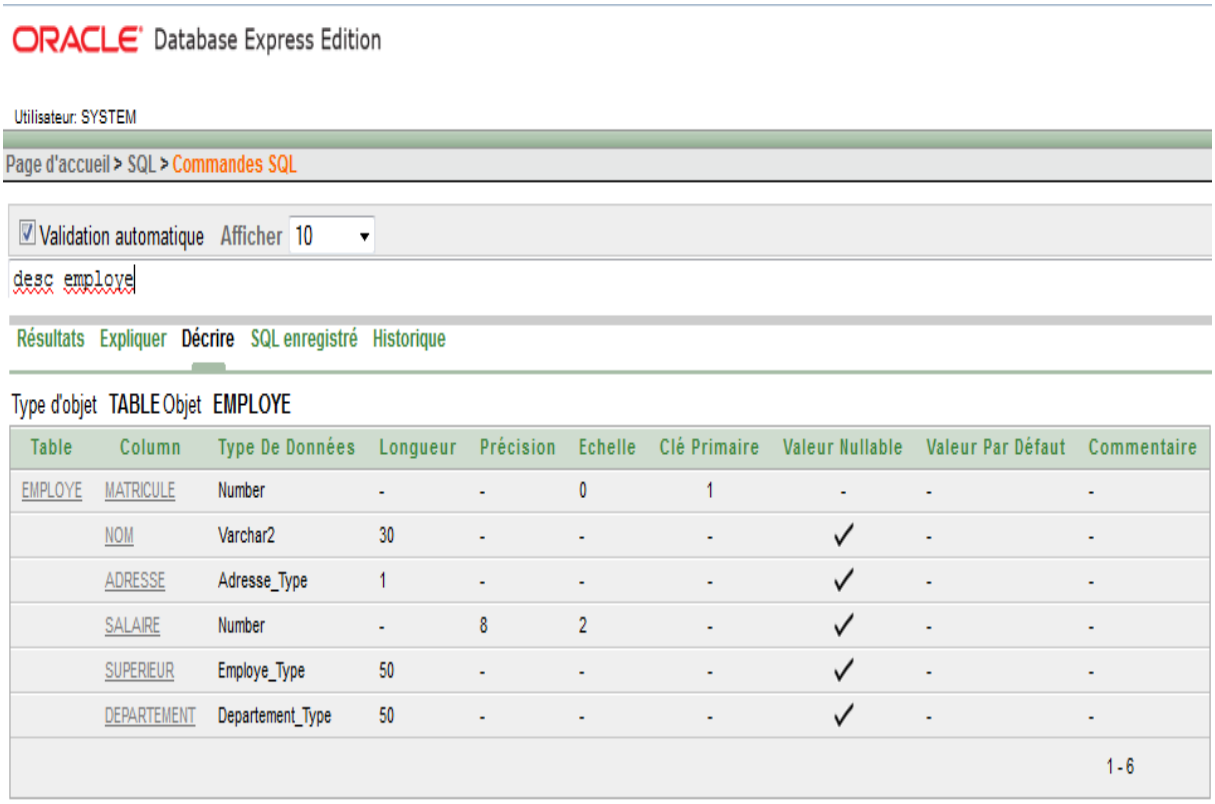
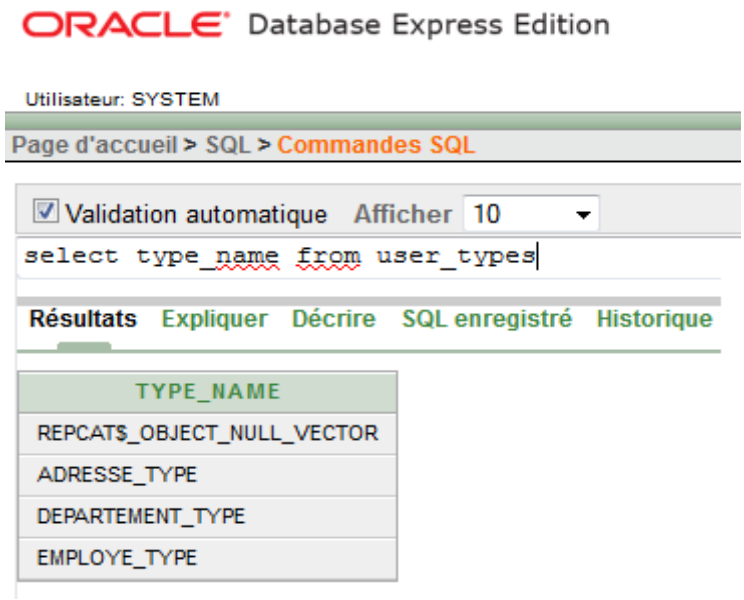


Table	Column	Type De Données	Longueur	Précision	Echelle	Clé Primaire	Valeur Nullable	Valeur Par Défaut	Commentaire
EMPLOYE	MATRICULE	Number	-	-	0	1	-	-	-
	NOM	Varchar2	30	-	-	-	✓	-	-
	ADRESSE	Adresse_Type	1	-	-	-	✓	-	-
	SALAIRE	Number	-	8	2	-	✓	-	-
	SUPERIEUR	Employe_Type	50	-	-	-	✓	-	-
	DEPARTEMENT	Departement_Type	50	-	-	-	✓	-	-

1 - 6



TYPE_NAME
REPCAT$_OBJECT_NULL_VECTOR
ADRESSE_TYPE
DEPARTEMENT_TYPE
EMPLOYE_TYPE

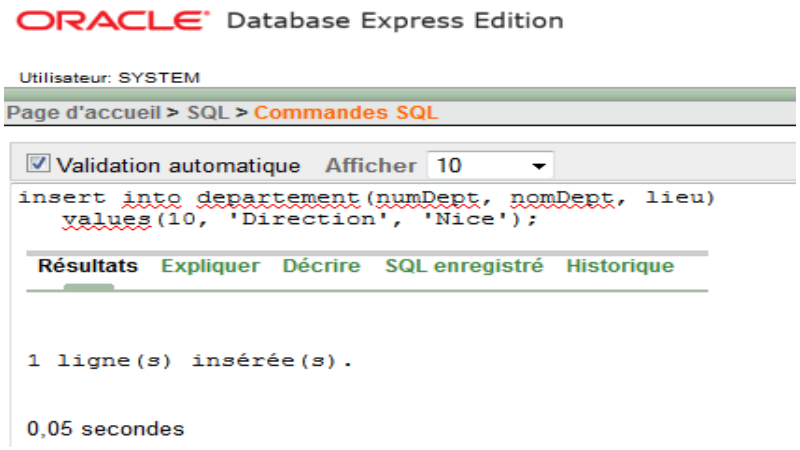
II. Ajouter des données dans les nouvelles tables

1. Ajoutez des données dans les nouvelles tables (au moins 2 départements et 3 employés). Attention, la version actuelle (fin 2010) de SQL Developer n'affiche pas toujours correctement les références. Pour vérifier, vous pouvez lancer un select qui renvoie les valeurs du type référencé, par exemple departement.numDept.
2. Faites afficher pour chaque employé : le matricule, la ville où il habite, le nom de son supérieur, la ville où il travaille (celle de son département).
3. Faites afficher le nom des employés du département 10 (ou d'un autre numéro).

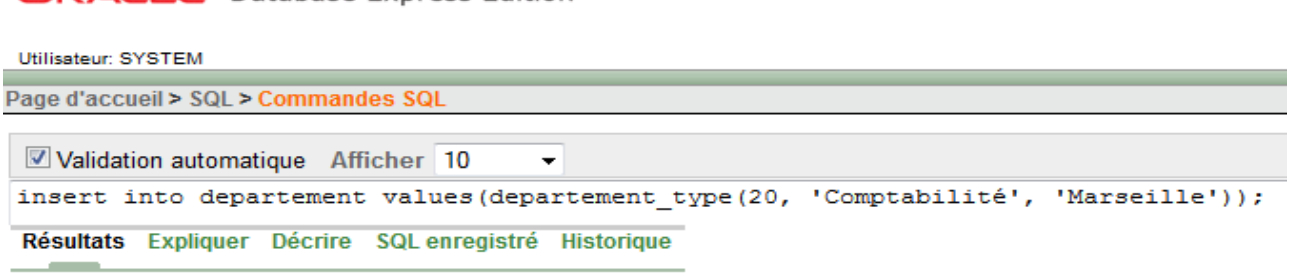
Correction

1. INSERTION DES DONNEES

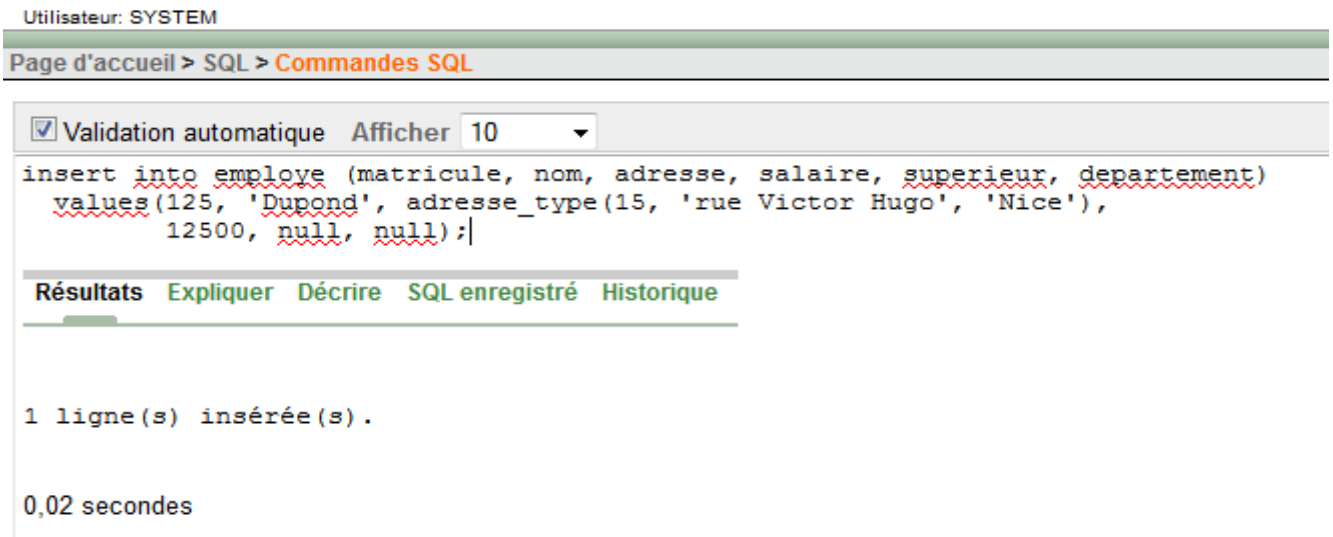
→*insert into departement (numDept, nomDept, lieu) Values (10, 'Direction', 'Nice')*



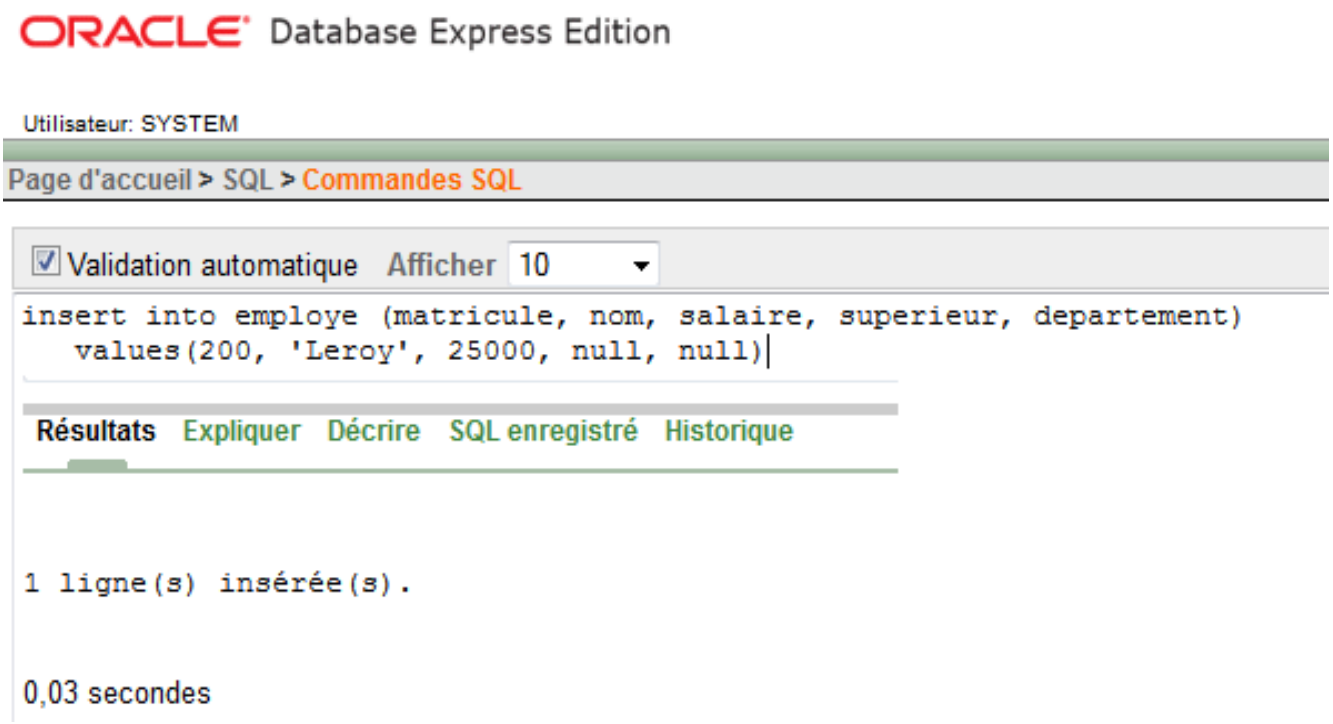
→ *insert into departement values (departement_type (20, 'Comptabilité', 'Marseille'))*



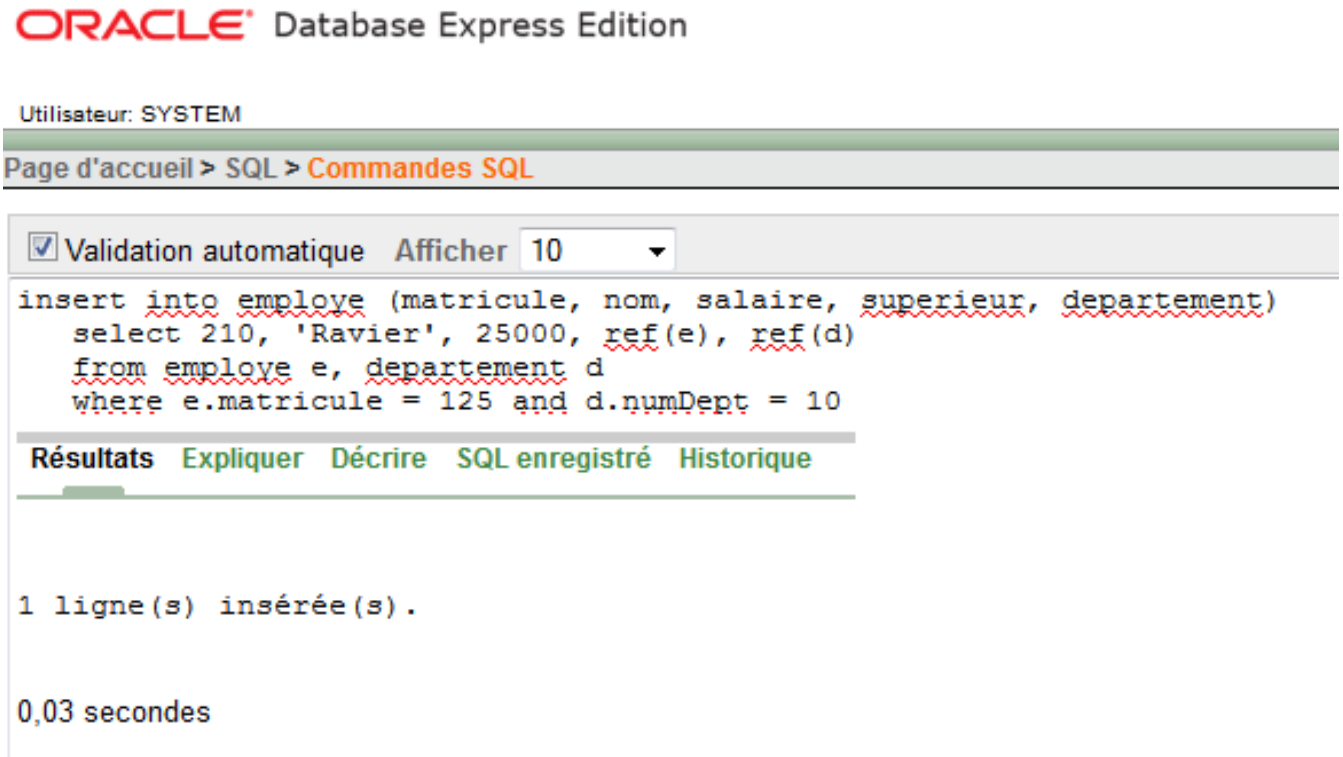
→insert into employe (matricule, nom, adresse, salaire, superieur, departement)
values(125, 'Dupond', adresse_type(15, 'rue Victor Hugo', 'Nice'), 12500, null, null)



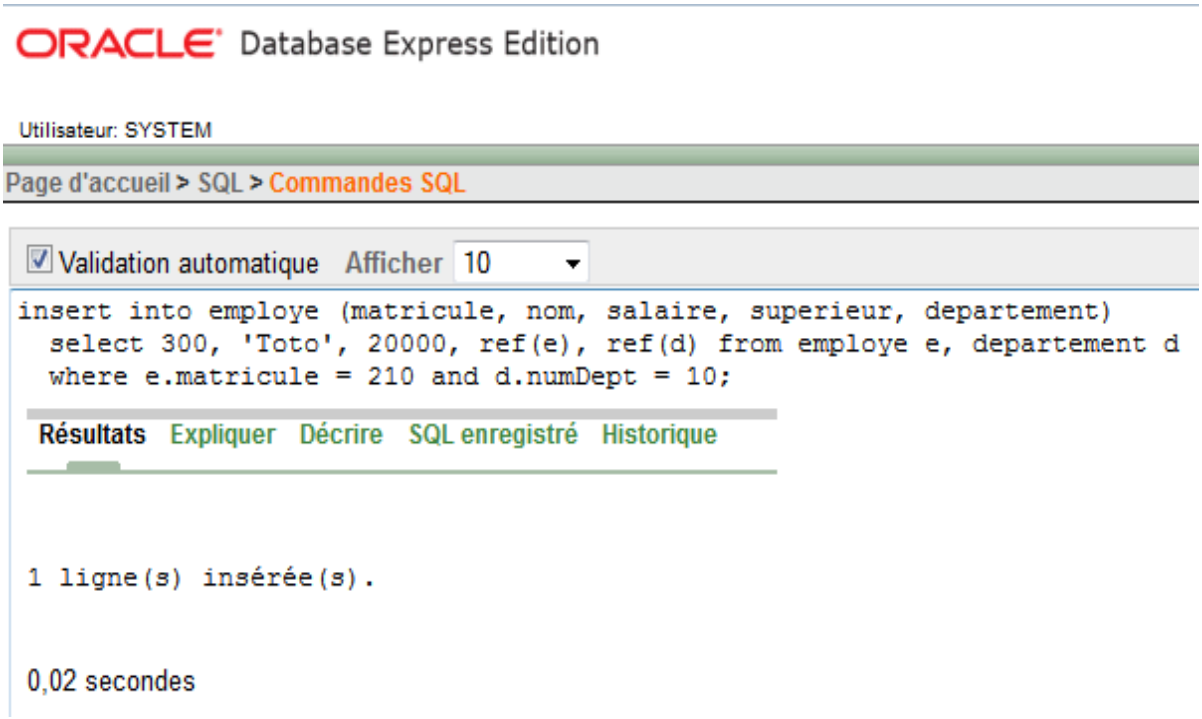
→*Insert into employe (matricule, nom, salaire, superieur, departement) values (200, 'Leroy', 25000, null, null)*



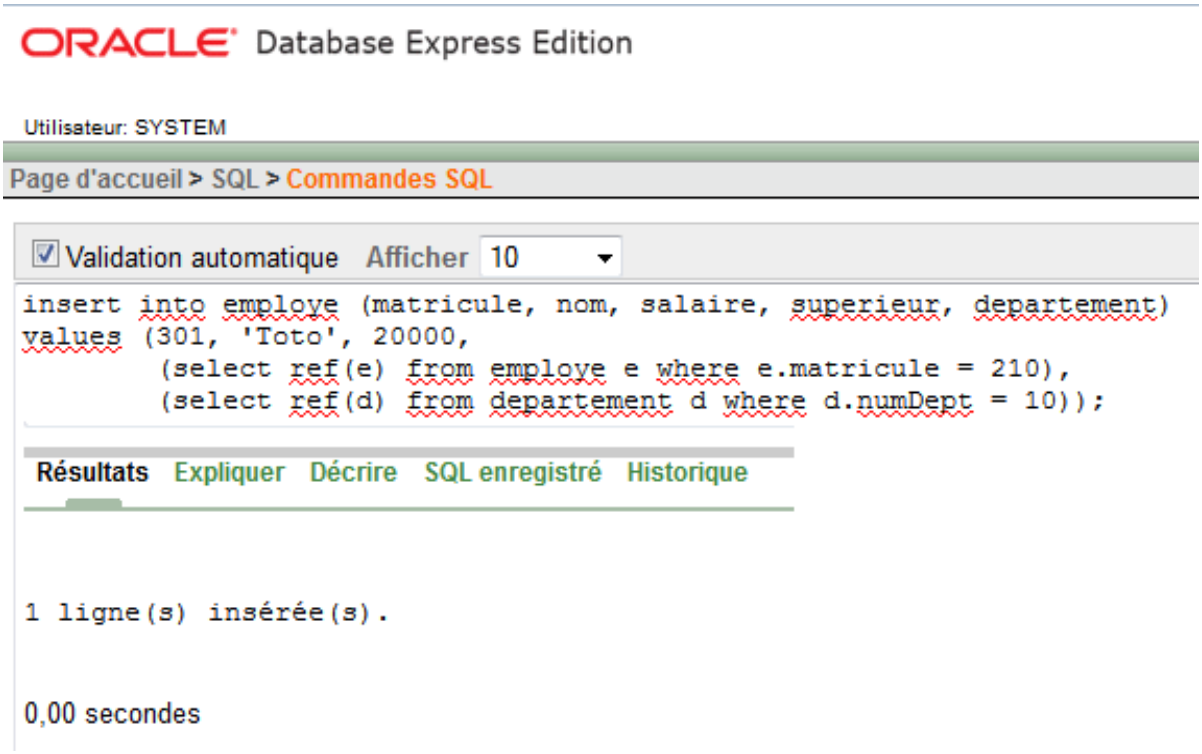
→*Insert into employe (matricule, nom, salaire, superieur, departement)*
Select 210, 'Ravier', 25000, ref(e), ref(d)
from employe e, departement d
where e.matricule = 125 and d.numDept = 10



→*Insert into employe (matricule, nom, salaire, superieur, departement)*
Select 300, 'Toto', 20000, ref (e), ref(d) from employe e, departement d
Where e.matricule = 210 and d.numDept = 10;

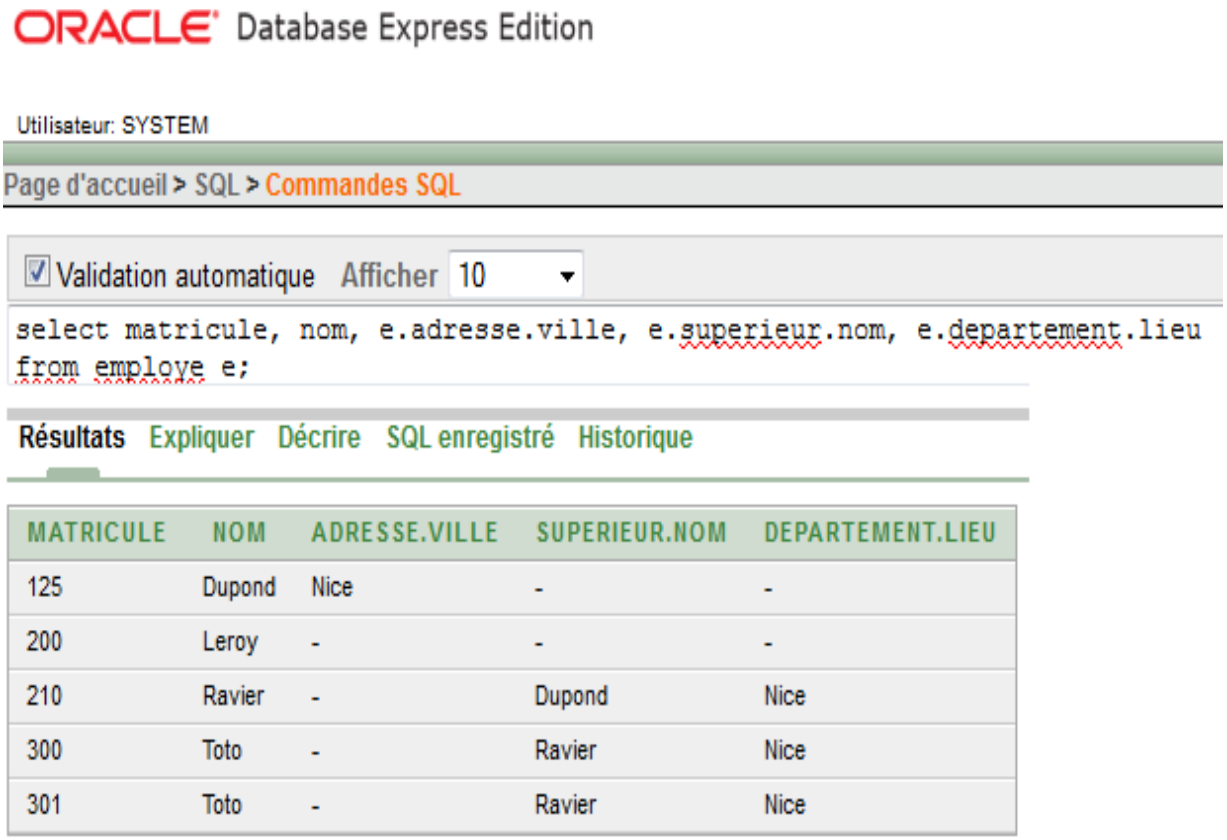


→Une autre façon d'ajouter une ligne contenant un REF :
Insert into employe (matricule, nom, salaire, superieur, departement)
values (301, 'Toto', 20000,
(select ref(e) from employe e where e.matricule = 210),
(select ref(d) from departement d where d.numDept = 10))



2. Afficher des informations sur les employés

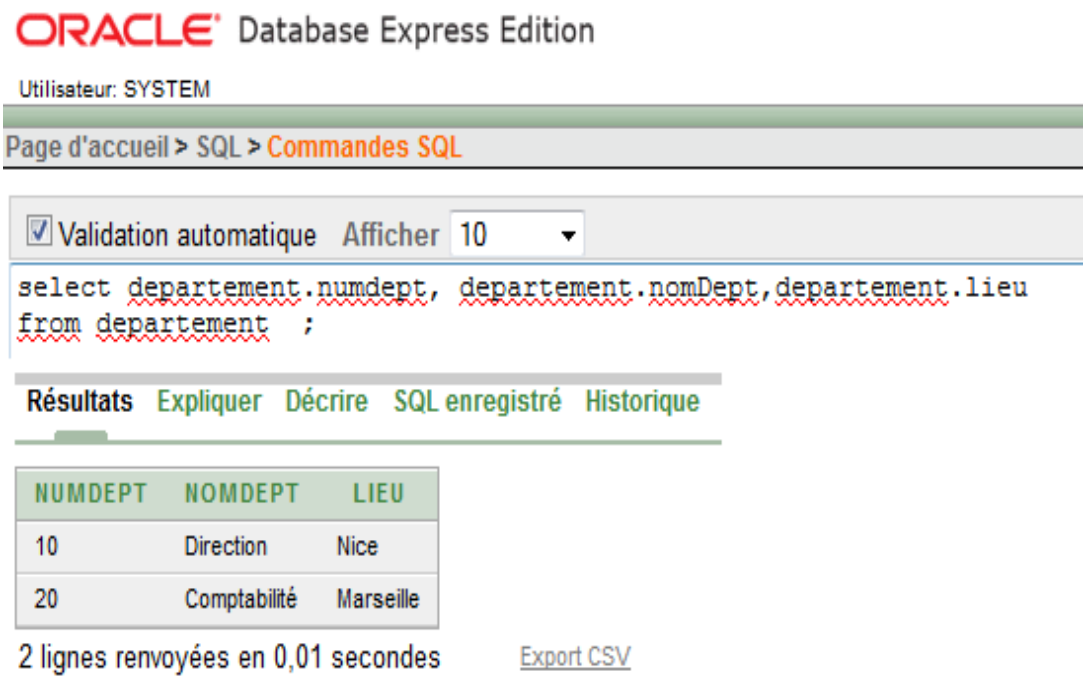
→*select matricule, nom, e.adresse.ville, e.superieur.nom, e.departement.lieu*
from employe e;



MATRICULE	NOM	ADRESSE.VILLE	SUPERIEUR.NOM	DEPARTEMENT.LIEU
125	Dupond	Nice	-	-
200	Leroy	-	-	-
210	Ravier	-	Dupond	Nice
300	Toto	-	Ravier	Nice
301	Toto	-	Ravier	Nice

3. Afficher des informations sur les départements

select departement.numdept, departement.nomDept,departement.lieu
from departement ;

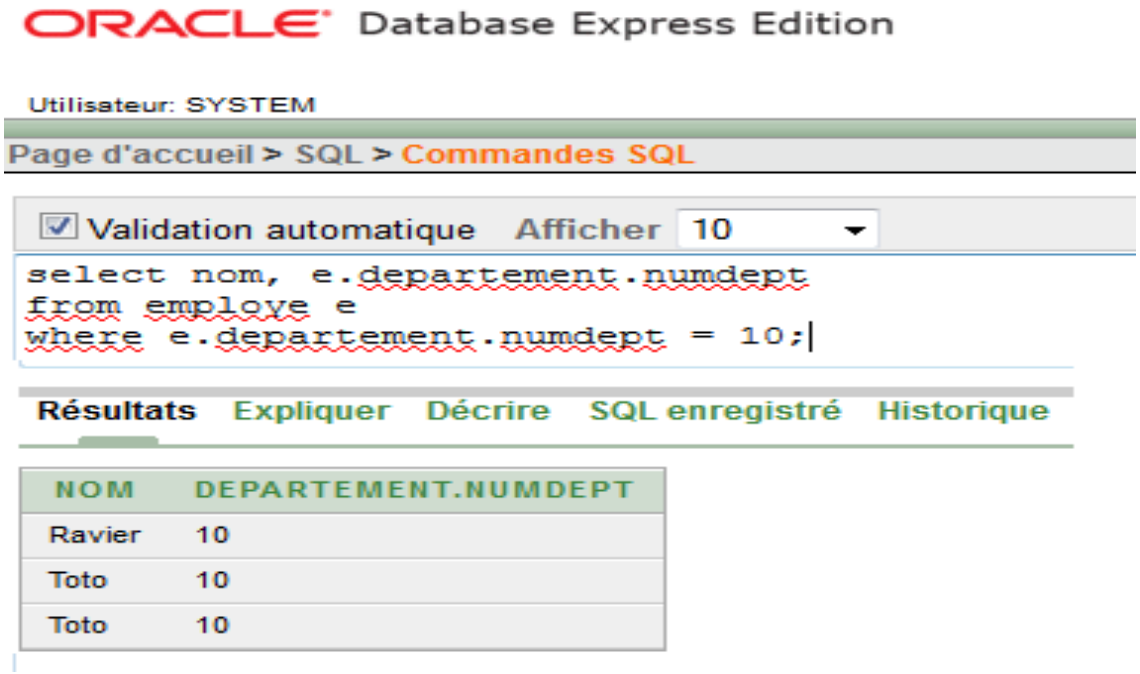


NUMDEPT	NOMDEPT	LIEU
10	Direction	Nice
20	Comptabilité	Marseille

2 lignes renvoyées en 0,01 secondes Export CSV

4. **Employés du département 10**

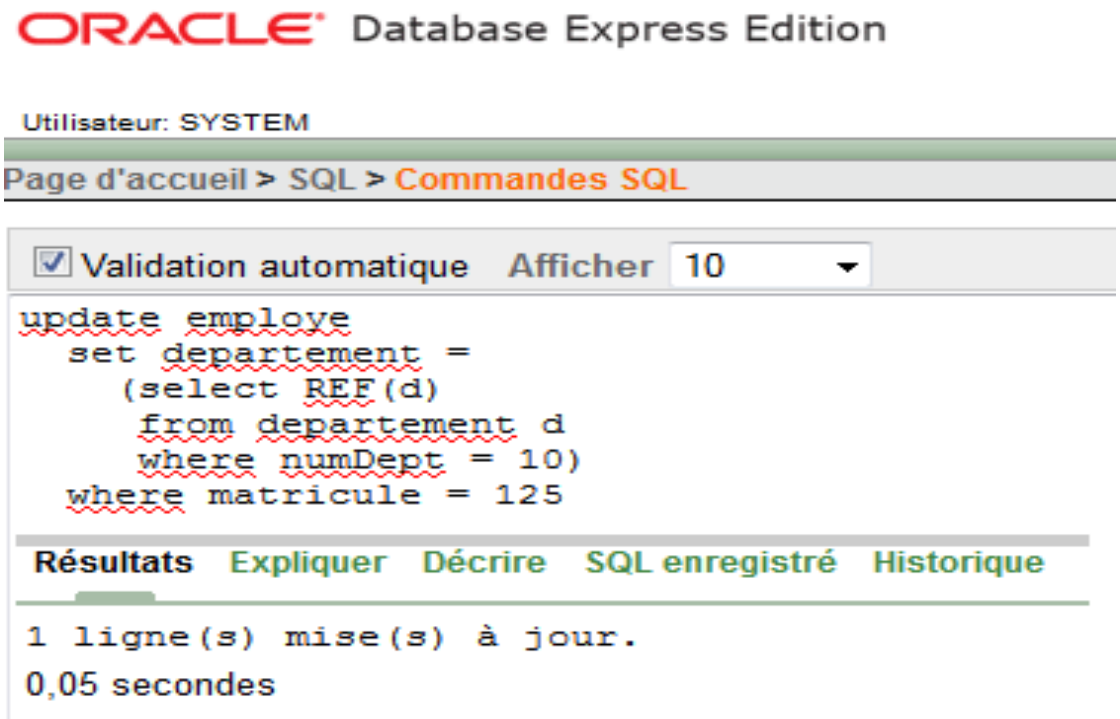
```
Select nom, e.departement.numdept
from employe e
where e.departement.numdept = 10;
```



III. Modifier les données

1. Changez le département d'un des employés.

```
update employe
set departement =  (select REF(d)
                   from departement d
                   where numDept = 10)
where matricule = 125
```



```
select matricule, nom, e.adresse.ville, e.superieur.nom, e.departement.lieu
from employe e;
```

Résultats Expliquer Décrire SQL enregistré Historique

MATRICULE	NOM	ADRESSE.VILLE	SUPERIEUR.NOM	DEPARTEMENT.LIEU
125	Dupond	Nice	-	Nice
200	Leroy	-	-	-
210	Ravier	-	Dupond	Nice
300	Toto	-	Ravier	Nice
301	Toto	-	Ravier	Nice

Changez le supérieur d'un des employés

```
update employe    set superieur =     (select REF(e)
                                       from employe e
                                       where matricule = 200)
  where matricule = 125
```

ORACLE Database Express Edition

Utilisateur: SYSTEM

Page d'accueil > SQL > Commandes SQL

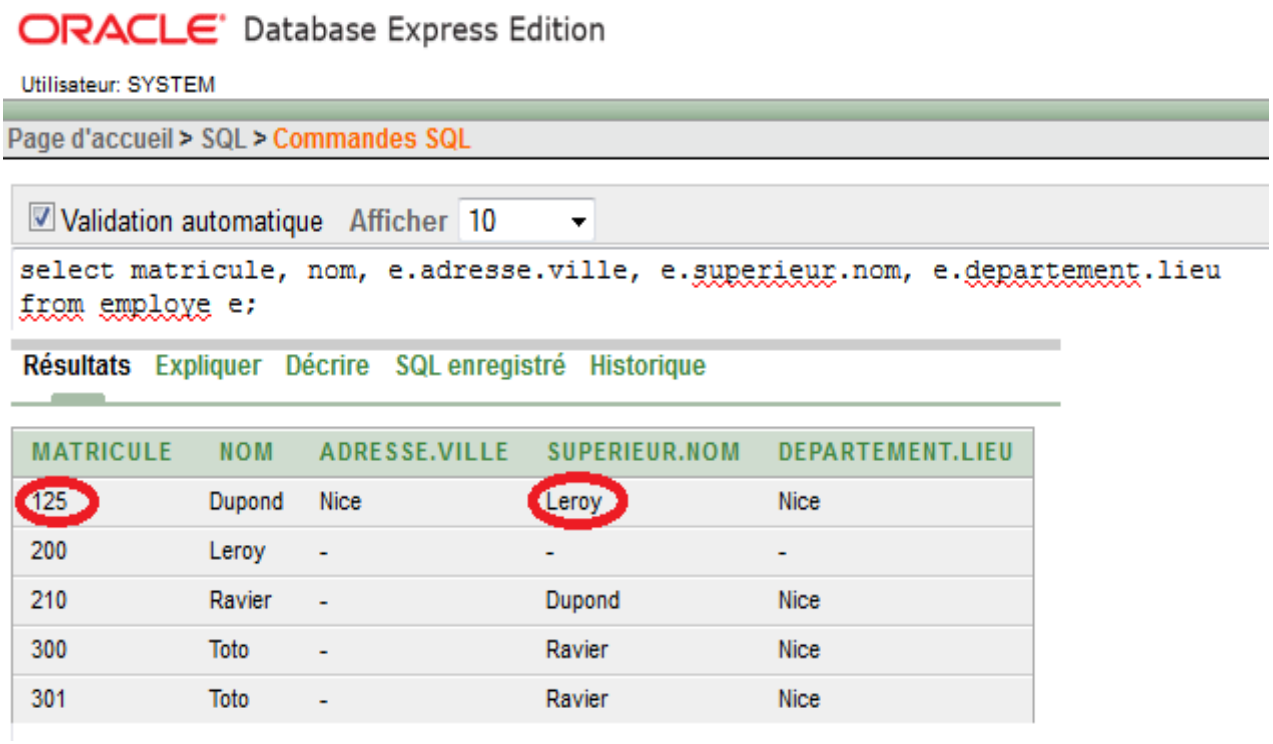
Validation automatique Afficher 10

```
update employe
   set superieur =
     (select REF(e)
      from employe e
      where matricule = 200)
   where matricule = 125
```

Résultats Expliquer Décrire SQL enregistré Historique

```
1 ligne(s) mise(s) à jour.
```

0,03 secondes



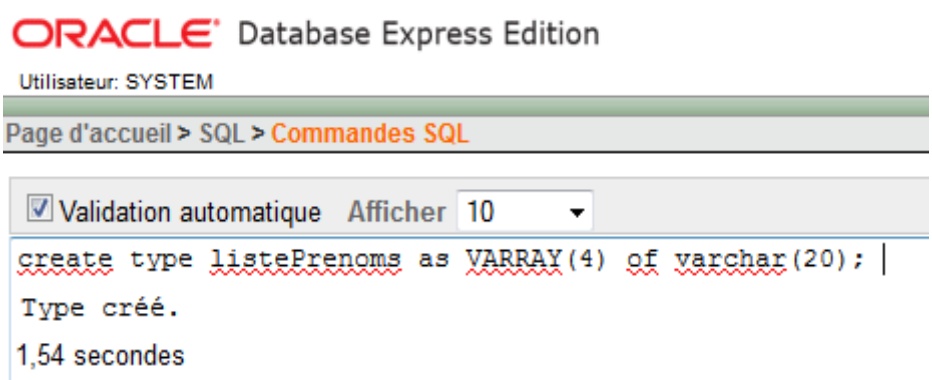
En une fois :

update employe
set superieur =
(select REF(e)
from employe e
where matricule = 200),
departement =
(select REF(d)
from departement d
where numDept = 10)
where matricule = 125

IV. Exemple sur le type Array

1. Créez un nouveau type personne_type avec un nom et un tableau de prénoms (au plus 4 prénoms).

CREATE type listePrenoms as VARRAY (4) of varchar(20);



1. Créez la table personne correspondante (n'oubliez pas les contraintes d'intégrité).

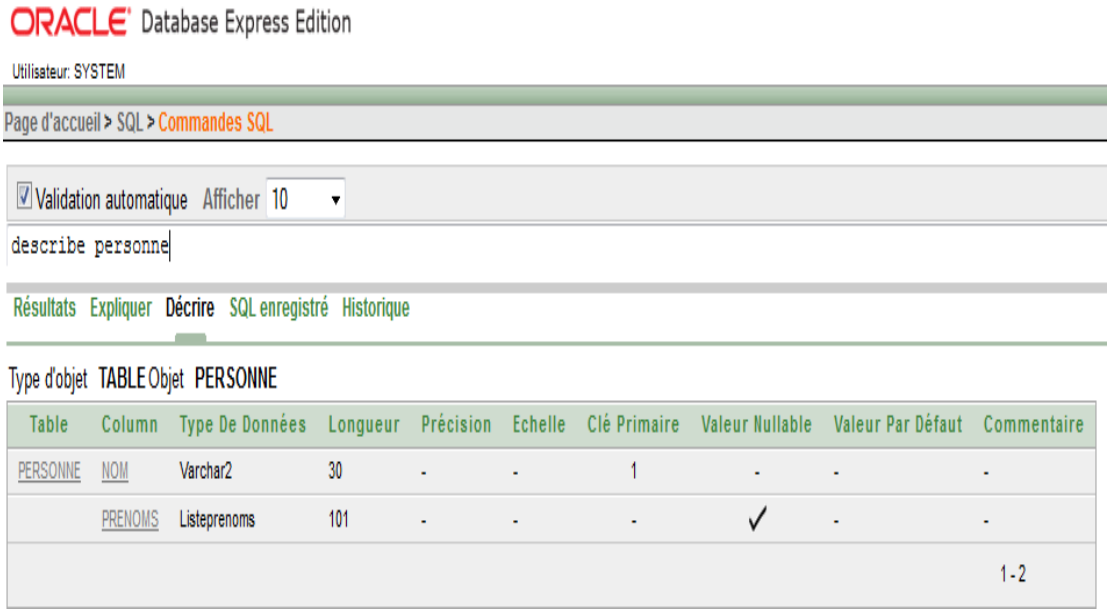
→*CREATE TYPE personne_type AS OBJECT (nom varchar(30), prenoms listePrenoms) ;*

```
CREATE TYPE personne_type AS OBJECT
    (nom varchar(30),
     prenoms listePrenoms);

Type créé.

0,70 secondes
```

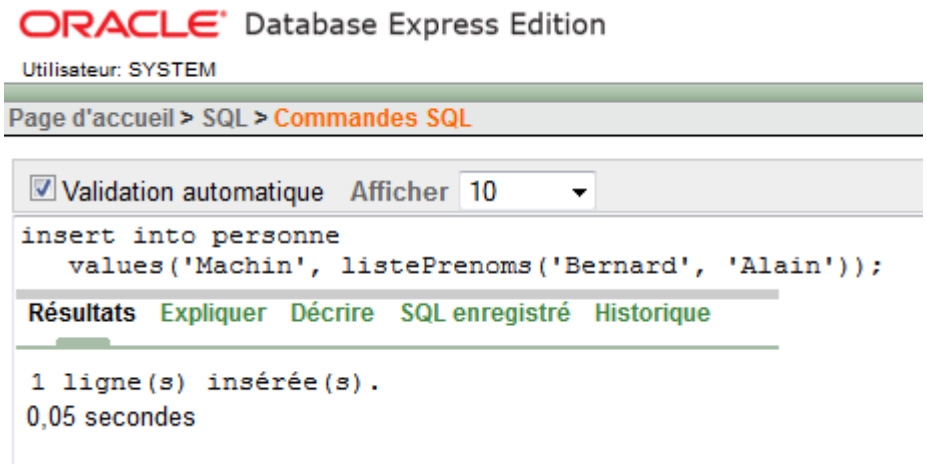
→ *CREATE TABLE personne OF personne_type (primary key (nom));*

```
create table personne OF personne_type (primary key(nom));
Table créée.

1,39 secondes
```

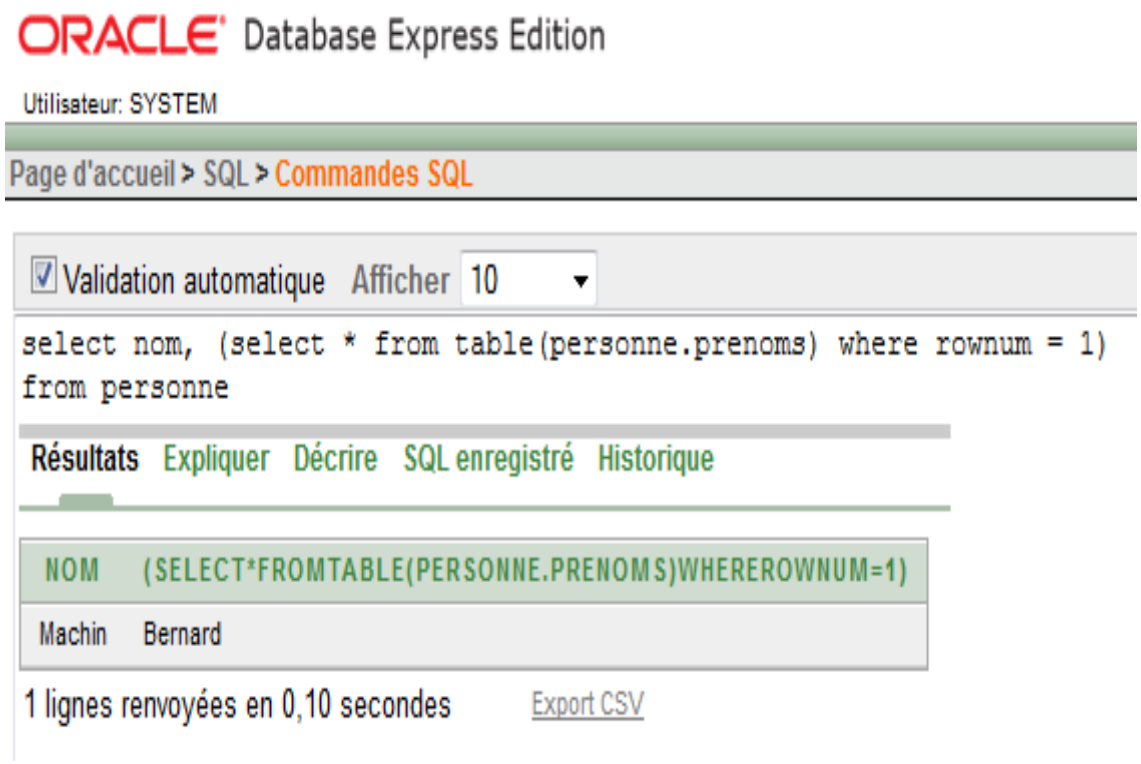
2. Faites afficher une description de personne avec la commande describe.



3. Ajoutez des lignes dans la table `personne`.
   insert into personne values ('Machin', listePrenoms('Bernard', 'Alain'));

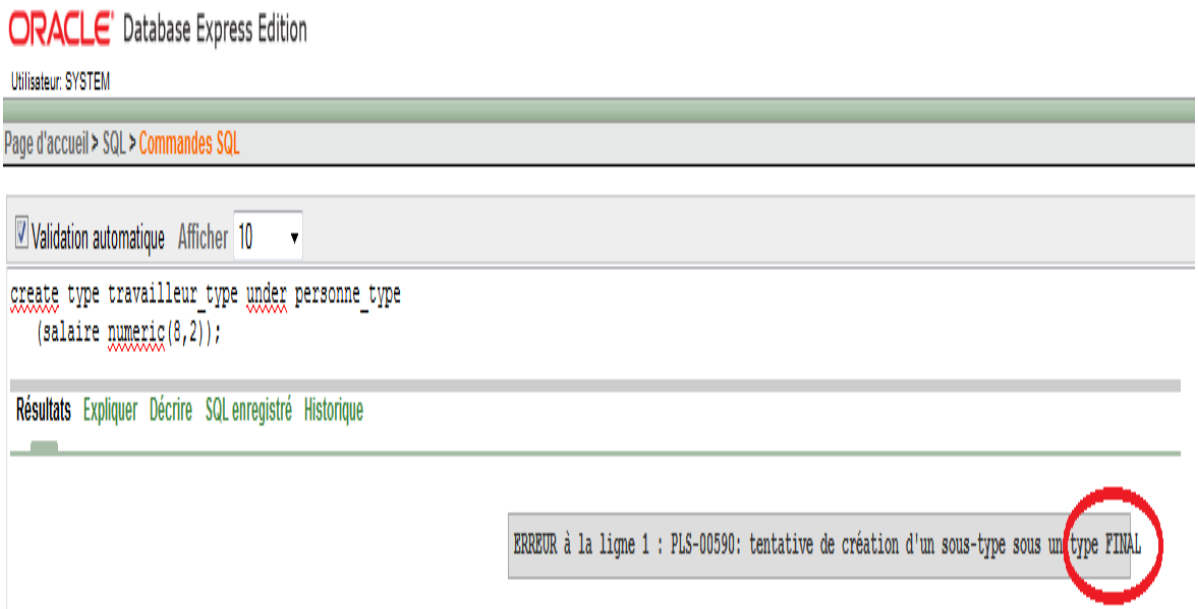


4. Affichez **les noms et tous les prénoms** que vous avez entrés ou mieux encore afficher le 1er prénom de chacune des personnes ?

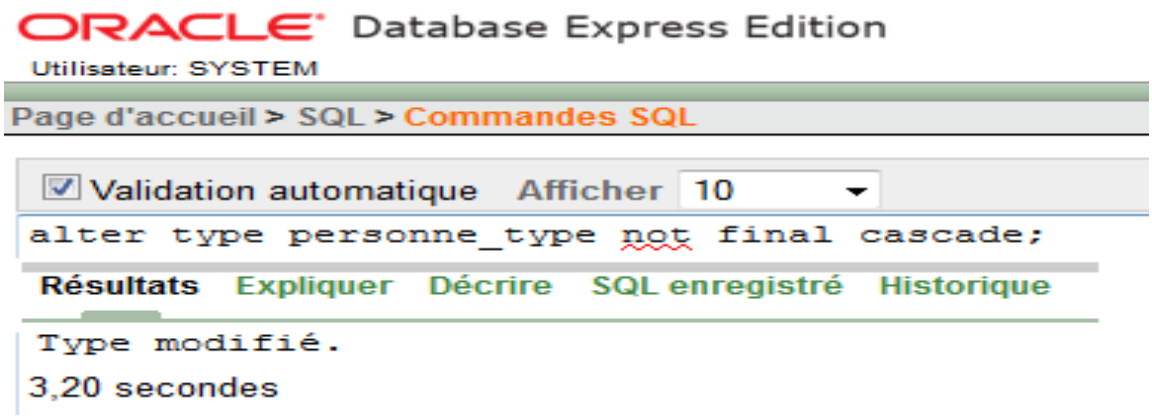


V. Héritage

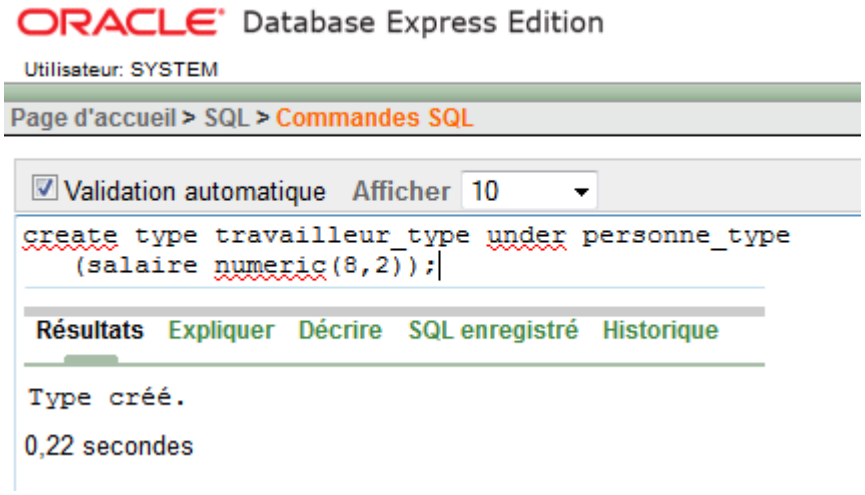
1. Créez un type `travailleur_type` qui hérite de `personne_type`. Ce nouveau type a seulement un attribut salaire en plus.



Ça ne marche pas car un type est final par défaut. Il faut d'abord rendre personne_type non final :

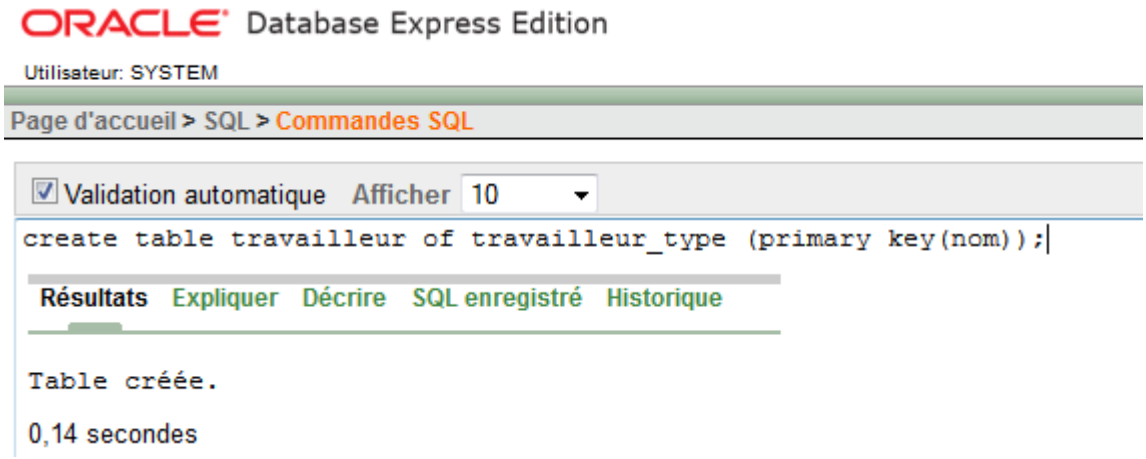


→ *create type travailleur_type under personne_type (salaire numeric(8,2)) ;*



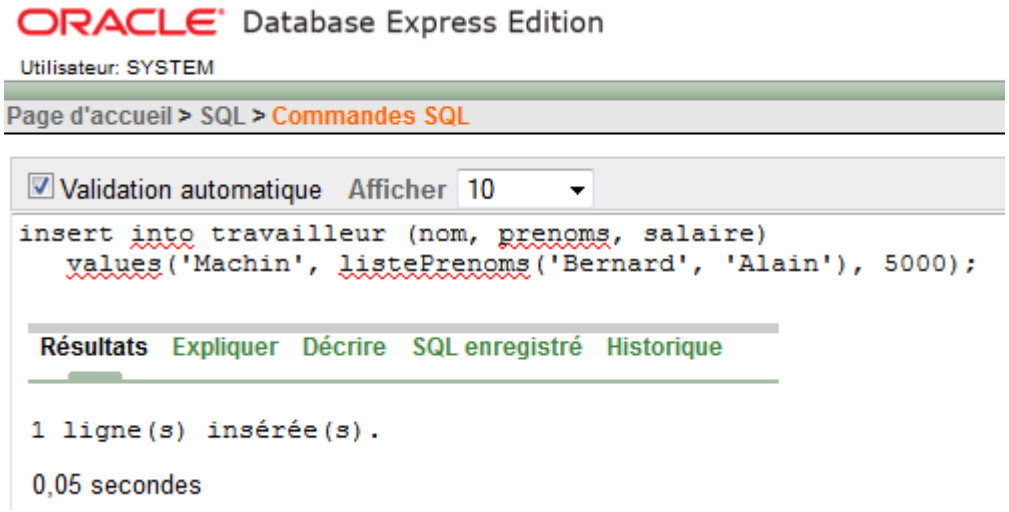
2. Créez une table `travailleur` (n'oubliez pas les contraintes d'intégrité).

→CREATE TABLE travailleur of travailleur_type (PRIMARY KEY (nom)) ;

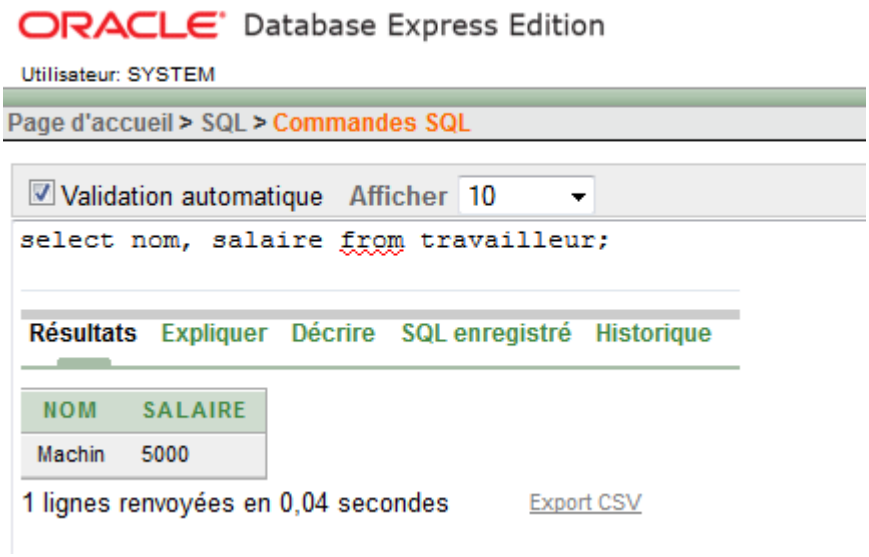


3. Ajoutez quelques travailleurs dans cette table.

```
Insert into travailleur (nom, prenoms, salaire)
   values('Machin', listePrenoms('Bernard', 'Alain'), 5000);
```



4. Faites afficher des informations sur les travailleurs.



TP 3 : Données semi-structurées

L'objectif de ce TP est d'introduire le langage XML, créer un document XML bien formé, créer un document XML valide, définir une description DTD d'une structure d'un document XML.

1. Créer un document XML bien formé

Dans un fichier de texte nommé **test.xml** saisir le document XML ci-dessous. Ce document décrit plusieurs types de livres.

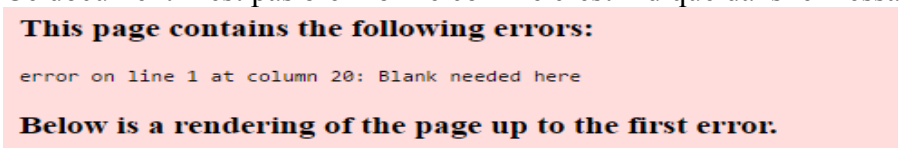
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<biblio>
 <livre>
 <!-- Élément enfant titre -->
 <titre>Les Misérables</titre>
 <auteur>Victor Hugo</auteur>
 <nb_tomes>3</nb_tomes>
 </livre>
 <livre>
 <titre>L'Assomoir</titre>
 <auteur>Émile Zola</auteur>
 </livre>
 <livre lang="en">
 <titre>David Copperfield</titre>
 <auteur>Charles Dickens</auteur>
 <nb_tomes>3</nb_tomes>
 </livre>
```

2. Visualisez votre document XML dans le browser

3. Déterminez s'il est bien formé, et corrigez les erreurs

```
<?xml version="1.0"? encoding="UTF-8">
<Contacts>
        <Person>
                <Firstname>John</Firstname>
                <Lastname>Smith</Lastname>
                <Birthday>1965/03/02</Birthday>
                <Company>IBM</Company>
                <Position>CEO</Position>
                <Email>jsmith@ibm.com<Email/>
                <Email>jsmith@yahoo.com</Email>
        </Person>
        <Person>
                <Firstname>Tom</Firstname>
                <Lastname>Dunne</Lastname>
                <Company>Today FM</Company>
                <Position/>
                <Email>tom.dunne@todayfm.com</Email>
        </Person>
</Contacts>
```

Ce document n'est pas bien formé comme c'est indiqué dans le message d'erreur suivant :



On remarque que les balises **xml** et **Email** mal fermées, d'où la correction suivante :

```
<Contacts>
<Person>
<Firstname>John</Firstname>
<Lastname>Smith</Lastname>
<Birthday>1965/03/02</Birthday>
<Company>IBM</Company>
<Position>CEO</Position>
<Email>jsmith@ibm.com</Email>
<Email>jsmith@yahoo.com</Email>
</Person>
<Person>
<Firstname>Tom</Firstname>
<Lastname>Dunne</Lastname>
<Company>Today FM</Company>
<Position/>
<Email>tom.dunne@todayfm.com</Email>
</Person>
</Contacts>
```

On peut Visualisez ce document XML dans le browser comme suit :

4. Créer un document XML valide

Dans un nouveau fichier de texte nommé biblio.dtd, définissez une DTD pour le document biblio.xml. (1) un élément livre doit être composé des trois éléments dans l'ordre : titre, auteur et nb_tomes de manière optionnelle. (2) l'attribut « lang » de l'élément livre ne prend que les valeurs « en » ou « fr » par défaut.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT biblio (livre)+>

<!ELEMENT livre (titre, auteur, nb tomes ?)>

<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT nb_tomes (#PCDATA)>
<!ATTLIST livre lang  (en|fr) "fr">
```

5. Intégrez l'appel à la DTD dans le prologue de biblio.xml. Tester.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE biblio SYSTEM "biblio.dtd">
```

6. Vérifiez la validité de votre document XML. Comment le faire ?

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<biblio>
  <livre>
    <!-- Élément enfant titre -->
    <titre>Les Misérables</titre>
    <auteur>Victor Hugo</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
  <livre>
    <titre>L'Assomoir</titre>
    <auteur>Émile Zola</auteur>
  </livre>
  <livre lang="en">
    <titre>David Copperfield</titre>
    <auteur>Charles Dickens</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
</biblio>
```

Pour vérifier la validité d'un document XML, il faut mettre le fichier XML et le fichier de DTD dans le même répertoire puis essayer de visualiser le fichier XML dans le navigateur :

Fichier biblio.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT biblio (livre)+>
<!ELEMENT livre (titre, auteur, nb_tomes ?)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT nb_tomes (#PCDATA)>
<!ATTLIST livre lang (en|fr) "fr">
```

Fichier exemplebiblio.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE biblio SYSTEM "biblio.dtd">
<biblio>
 <livre>
 <!-- Élément enfant titre -->
 <titre>Les Misérables</titre>
 <auteur>Victor Hugo</auteur>
 <nb_tomes>3</nb_tomes>
 </livre>
 <livre>
 <titre>L'Assomoir</titre>
 <auteur>Émile Zola</auteur>
 </livre>
 <livre lang="en">
 <titre>David Copperfield</titre>
 <auteur>Charles Dickens</auteur>
 <nb_tomes>3</nb_tomes>
 </livre>
</biblio>
```

7. Pour s'assurer que le parseur effectue la vérification, testez les cas d'erreurs suivants : (1) valeur d'attribut lang erroné, différent de fr ou en (2) un élément titre ou auteur manquant, ou pas dans le bon ordre, et nb_tomes plus d'une fois.

TP 4 : Gestion de la répartition des données

On dispose d'une base de données viticoles sur une base Oracle située sur l'instance d'adresse oranet TANNA que l'on veut répartir entre deux instances, l'une d'adresse oranet POAS et l'autre d'adresse oranet CALCITE. La base de données est décrite par le schéma suivant :

- VIN(num, cru, année, degré)
- PRODUCTEUR(num, nom, prénom, région)
- RECOLTE(nprod, nvin, quantité)

Cette base existe de manière centralisée dans la base *defude/bruno*du SGBD Oracle sur l'instance d'adresse oranet *TANNA* .
La répartition proposée pour les données est la suivante :

- relation PRODUCTEUR placée par fragmentation horizontale selon un prédicat portant sur la région :

PRODUCTEURPOAS = sélection(PRODUCTEUR, région IN ('Jura', 'Alsace', 'Bourgogne'))
PRODUCTEURCALCITE = sélection(PRODUCTEUR, région NOT IN ('Jura','Alsace', 'Bourgogne'))

- relation RECOLTE placée par voisinage de la relation PRODUCTEUR :

RECOLTEPOAS = semi-jointure(RECOLTE, PRODUCTEURPOAS)
RECOLTECALCITE = semi-jointure(RECOLTE, PRODUCTEURCALCITE)

- relation VIN dupliquée sur les deux sites : VINPOAS = VINCALCITE = VIN

Travail à faire

1/ Création des relations PRODUCTEURPOAS, PRODUCTEURCALCITE, RECOLTEPOAS, RECOLTECALCITE dans les bases correspondantes (création d'un "database link" puis un *create table ... as select* ...puisque les relations existent déjà sur la base *defude/bruno* de *TANNA*),
2/ Créer des liens entre les différentes instances pour pouvoir accéder aux relations distantes,
3/ Création des vues permettant de donner une vision globale de la base de données,
4/ Test de la base ainsi obtenue par des requêtes d'interrogation,
5/ Test de mises à jour distantes (par exemple à partir de l'instance POAS on insère un tuple dans une relation stockée dans CALCITE),
6/ Vérifier le comportement d'une transaction répartie (par exemple, faire une insertion dans une relation locale à l'instance dans laquelle on se trouve suivie d'une insertion dans une instance distante). Il est plus parlant de faire une transaction qui échoue (le travail est annulé) qu'une qui réussit. Par exemple faire une transaction qui fait une première insertion qui marche suivie d'une deuxième qui échoue,
7/ Comparer le plan d'exécution (utiliser le bouton plan d'exécution) de trois requêtes réparties avec la solution optimale (obtenue à la main).

Requêtes de test (à titre d'exemple)

1- Donner la liste des producteurs de la région du Beaujolais,
2- Donner le nom des producteurs de vins de cru Morgon,
3- Donner le cru et la quantité des vins produits par le producteur de numéro 10,
4- Calculer la quantité totale de vin numéro 12 produite,
5- Donner pour chaque producteur (numéro) le nombre total de vins produits,
6- Donner pour chaque vin (numéro) la quantité totale récoltée.

TP 5 : Gestion des accès concurrents sous oracle

Déroulement du TP : (campioni, 2009)

- Créer l'utilisateur « **test »** identifié par le mot de passe **« 123 »**.
- Attribuer les droits **« connect », « ressource »** à l'utilisateur **« test »**.
- Créer la table **chambre (code, type, surface, prix)** avec l'utilisateur **« test ».**

I. Les accès concurrents

1. Ouvrez **deux sessions** sous le **même** nom.
2. Faites des **modifications** dans **une des sessions** et voyez si les modifications sont connues de **l'autre session**.
3. Faites un **COMMIT** des modifications dans **une des sessions** et voyez si les modifications sont connues de **l'autre session**.
4. Modifiez le prix d'une **même chambre** dans les **deux sessions** (avec des valeurs différentes). Que se passe-t-il ?
5. Faites un **COMMIT** sur **la session** qui a fait les modifications **en premier**. Que se passe-t-il ? Faites un select dans les 2 sessions pour voir la modification.
6. Faites un **COMMIT** dans **la deuxième session**. Faites un select dans les 2 sessions pour voir la modification.
7. Utilisez un SELECT FOR UPDATE sur une des sessions et essayez de modifier les lignes bloquées avec l'autre session.

II. Mode de fonctionnement par défaut d'Oracle

En travaillant sur la table des chambres vérifiez que dans ce mode

1. Les lectures ne bloquent ni les autres lectures ni les écritures.
2. Les lectures ne sont bloquées par rien, même pas par un blocage d'une table en mode exclusif.
3. Il n'y pas de lecture impropre.
4. Il n'y a pas de pertes de mises à jour.
5. Il peut y avoir des lectures non reproductibles.
6. Il peut y avoir des lignes fantômes.

Empêcher les lectures non reproductibles

Que pouvez-vous faire pour empêcher les lectures non reproductibles,

1. dans le cas où la transaction ne modifie aucune donnée.
2. dans le cas où elle modifie des données.

Empêcher les lignes fantômes

Mêmes questions que l'exercice précédent, mais pour les lignes fantômes.

Interblocages

1. Ouvrez 2 sessions et provoquez un interblocage en commençant par faire des modifications, puis en bloquant des tables. Voyez comment Oracle réagit.
2. Provoquez un blocage en commençant par lancer un update que vous ne validez pas tout de suite et un blocage en mode share dans l'autre session. Voyez comment Oracle réagit.

Read only

1. Ouvrez une nouvelle transaction en "READ ONLY".
2. Ouvrez en parallèle une deuxième transaction dans laquelle vous modifiez des prix.
3. Validez cette deuxième transaction. Voyez-vous les modifications dans la première transaction ?

4. Essayez de modifier des données dans la première transaction.
5. Que se serait-il passé si la première transaction n'avait pas été en "READ ONLY" ? Vérifiez-le après avoir terminé la première transaction.

Mode serialisable

Essayez de faire cet exercice en devinant ce qui va se passer avant de lancer chaque commande.

- Ouvrez 2 sessions de travail avec des transactions T1 et T2.
- Passez T1 en mode sérialisé.
- T1 affiche tous les noms et prix des chambres.
- T2 modifie le prix d'une chambre.
- T1 affiche à nouveau tous les prix. Quel prix voit-il pour cette chambre ? Pourquoi ?
- T2 valide sa transaction.
- T1 affiche à nouveau tous les prix. Quel prix voit-il pour cette chambre ? Pourquoi ?
- T1 modifie le prix de cette chambre. Que se passe-t-il ? Pourquoi ?
- T1 modifie le prix d'une autre chambre. Que se passe-t-il ?

Voyez-vous une différence avec le mode par défaut d'Oracle ? Explications ?

Correction

→Se connecter à la BD oracle avec un compte administrateur (qui a le privilège DBA)

```
SQL> connect system/inchalah;
Connected.
```

→Créer l'utilisateur **« amine »** identifié par le mot de passe **« 123 »** puis attribuer les droits **« connect », « ressource »** à l'utilisateur **« amine ».**

```
SQL> create user amine identified by 123;

User created.

SQL> grant connect, resource to amine;

Grant succeeded.
```

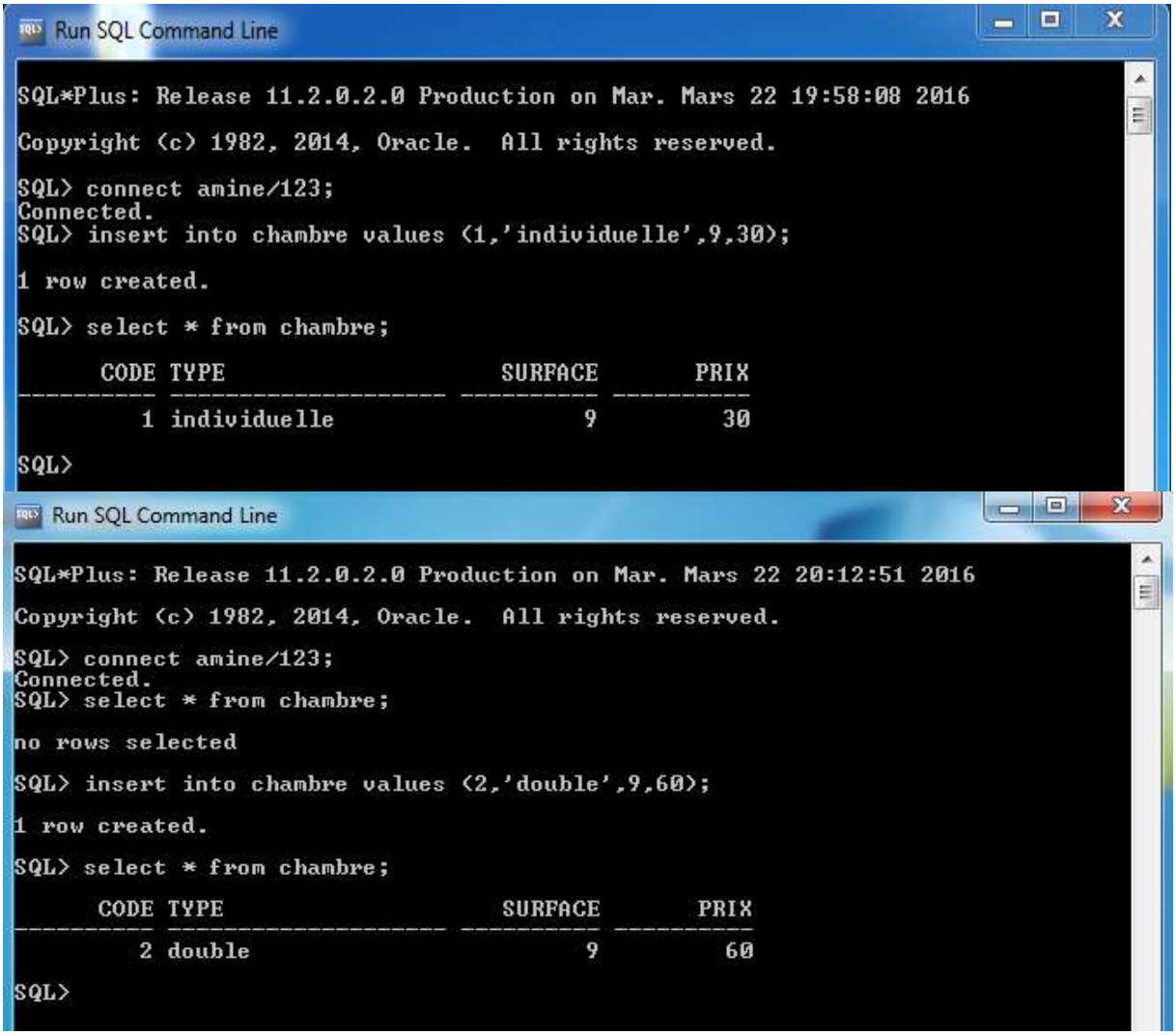
→Se connecter avec le compte **« amine ».**

```
SQL> connect amine/123;
Connected.
```

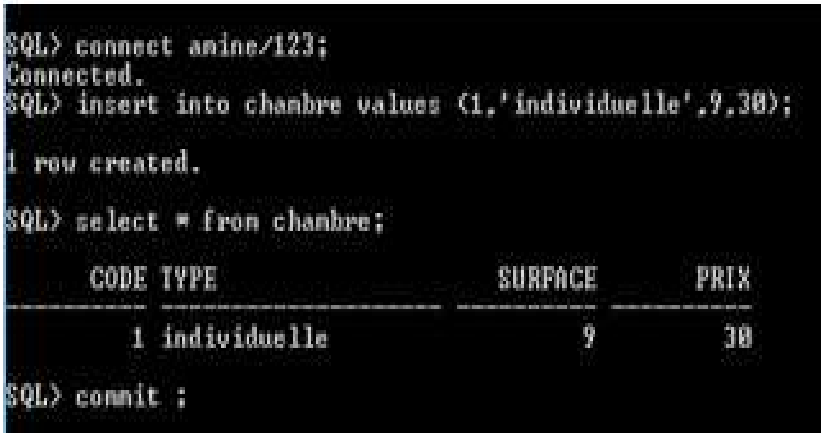
→Créer la table **chambre (code, type, surface, prix)**.

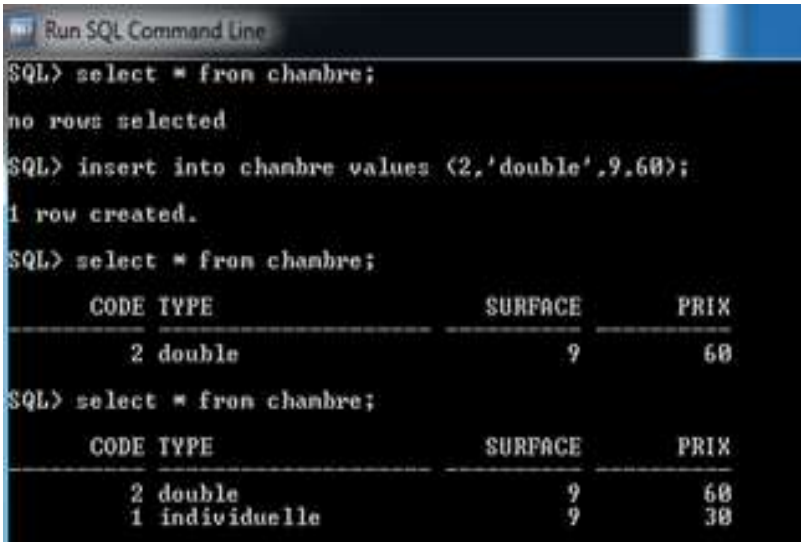
```
SQL> create table chambre
  2  (code number primary key,
  3  type varchar(20),
  4  surface number(3,2),
  5  prix number (8,2));

Table created.
```

→Ouvrez **deux sessions** sous le **même** nom et faites des **modifications** dans **une des sessions** et voyez si les modifications sont connues de **l'autre session**.

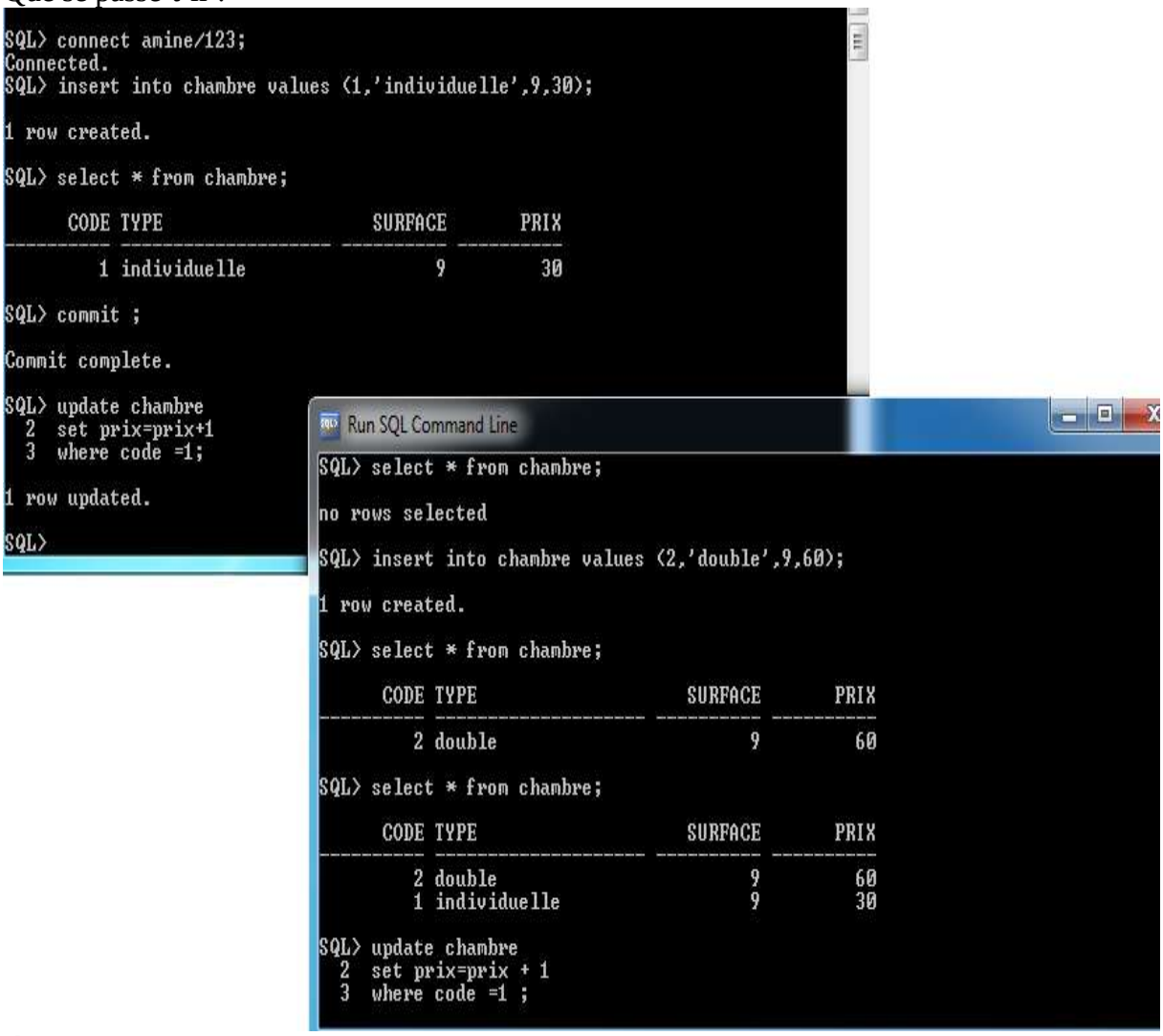


→Faites un **COMMIT** des modifications dans **une des sessions** et voyez si les modifications sont connues de **l'autre session**.

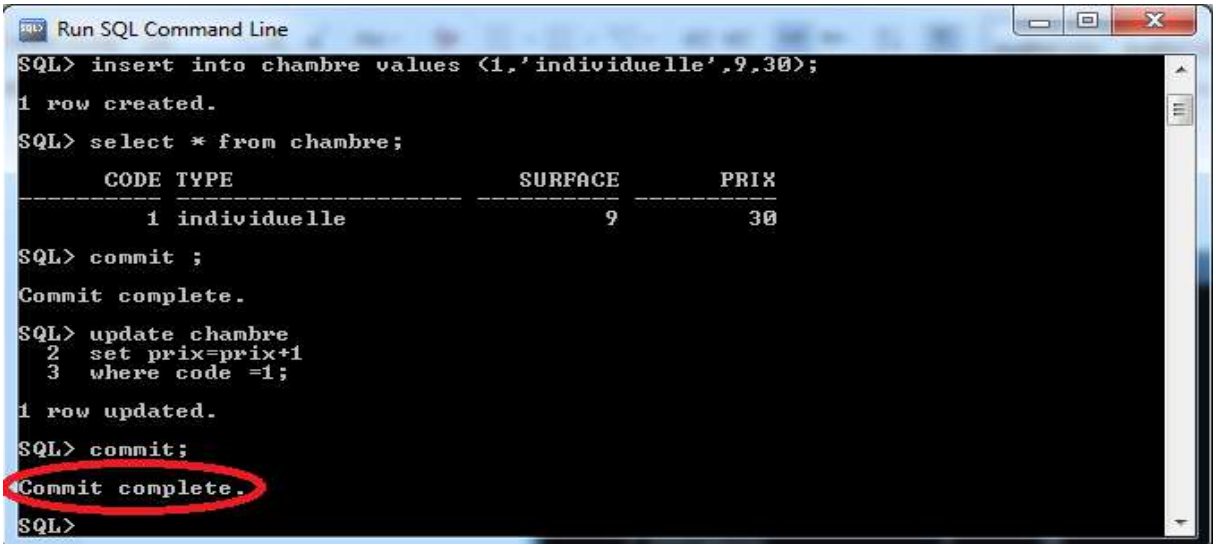


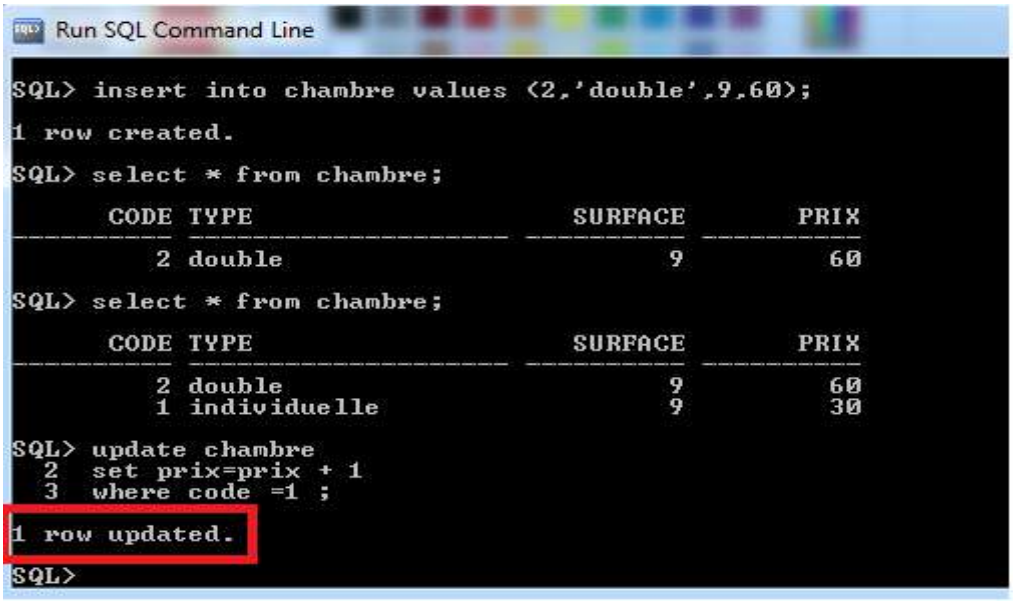


→Modifiez le prix d'une **même chambre** dans les **deux sessions** (avec des valeurs différentes). Que se passe-t-il ?

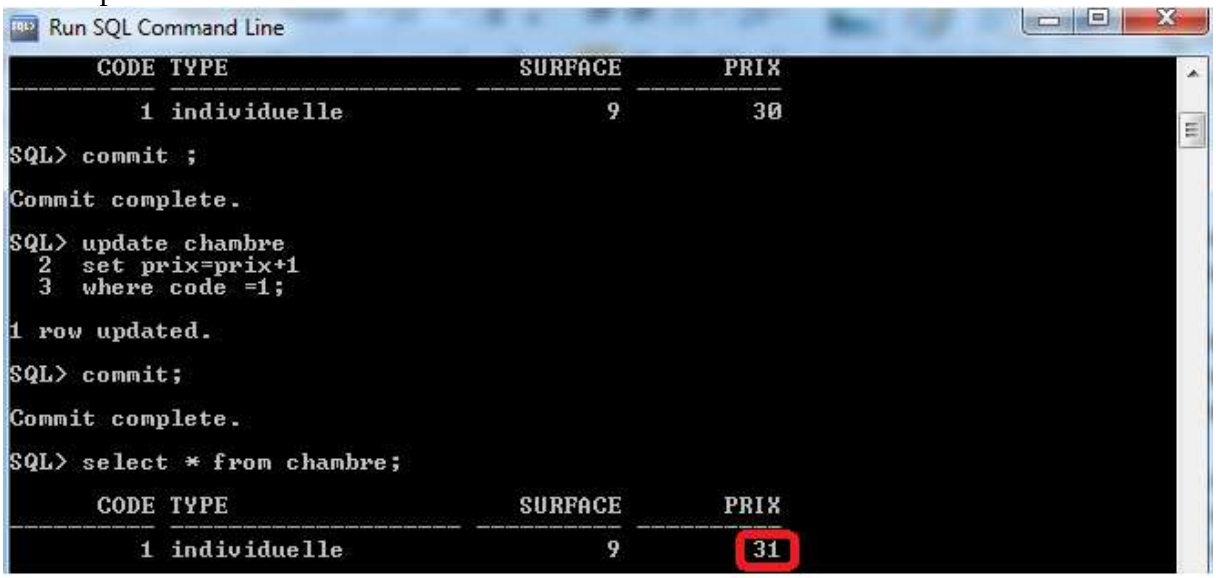


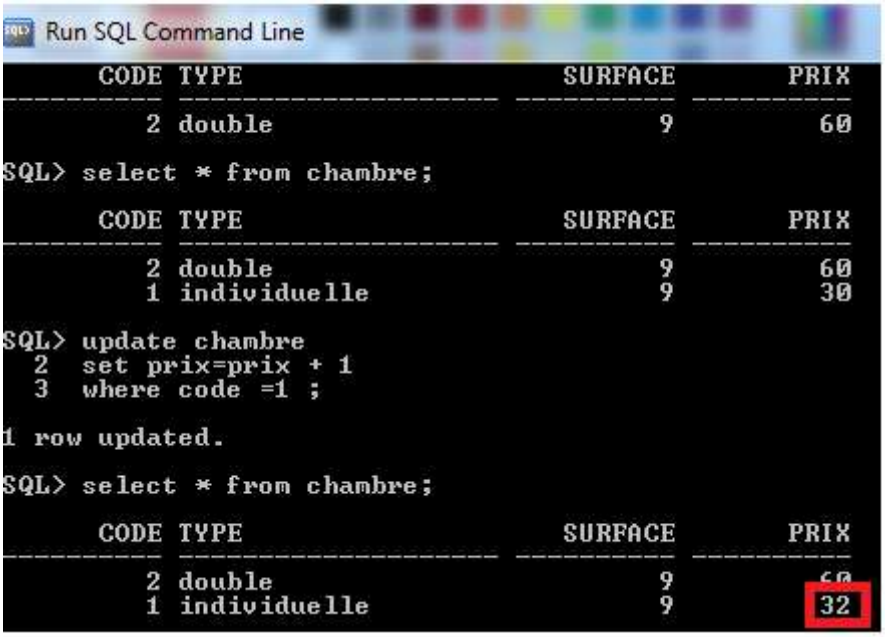
→Faites un **COMMIT** sur **la session** qui a fait les modifications **en premier**. Que se passe-t-il ? Faites un select dans les 2 sessions pour voir la modification.

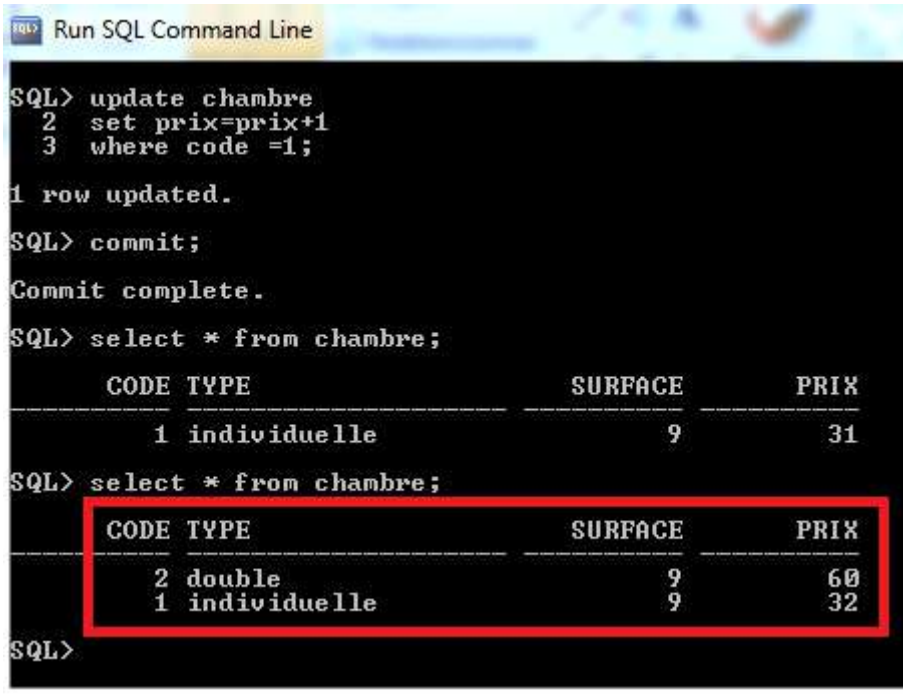


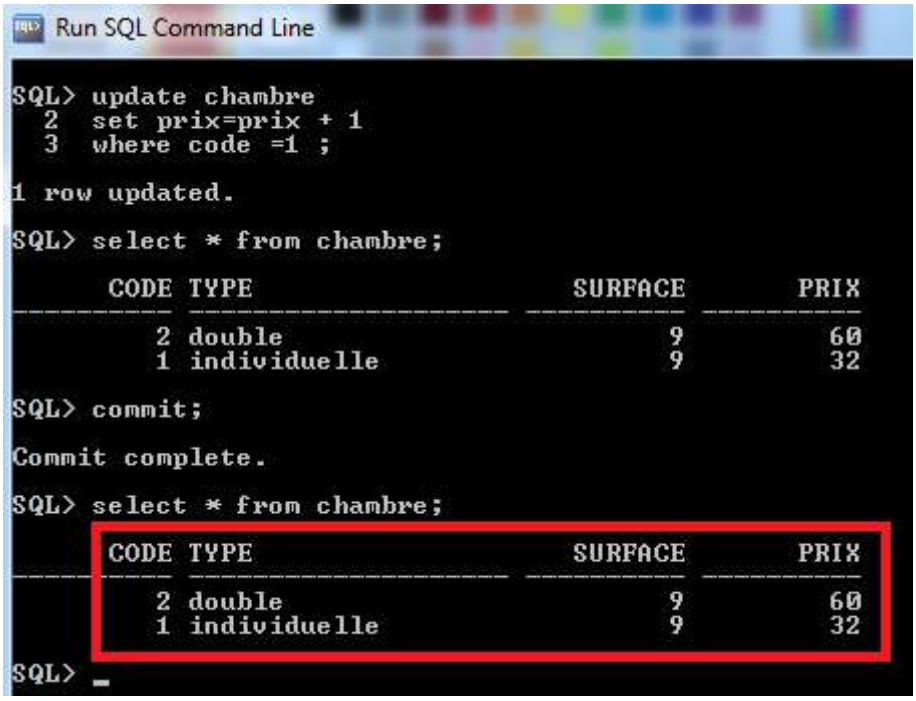


→Faites un **COMMIT** dans **la deuxième session**. Faites un select dans les 2 sessions pour voir la modification.

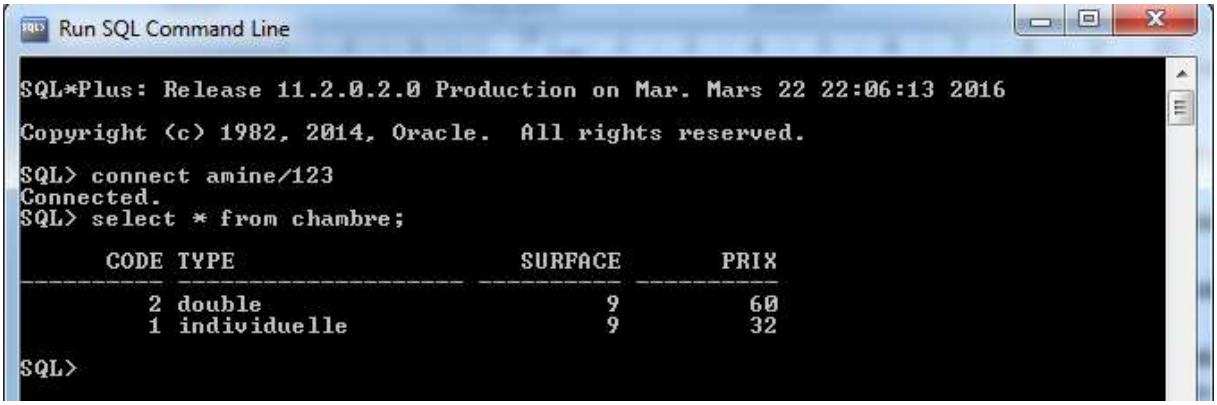



Run SQL Command Line
CODE TYPE SURFACE PRIX
2 double 9 60
SQL> select * from chambre;
CODE TYPE SURFACE PRIX
2 double 9 60
1 individuelle 9 30
SQL> update chambre
2 set prix=prix + 1
3 where code =1 ;
1 row updated.
SQL> select * from chambre;
CODE TYPE SURFACE PRIX
2 double 9
1 individuelle 9 32


Run SQL Command Line
SQL> update chambre
2 set prix=prix+1
3 where code =1;
1 row updated.
SQL> commit;
Commit complete.
SQL> select * from chambre;
CODE TYPE SURFACE PRIX
1 individuelle 9 31
SQL> select * from chambre;
CODE TYPE SURFACE PRIX
2 double 9 60
1 individuelle 9 32
SQL>



→ Déconnectez-vous des deux sessions puis reconnectez-vous avec une des sessions et faites un select.



Accès e-learning au module

Dans ce qui suit, nous présentons les instructions qui permettent aux apprenants d'accéder aux ressources et aux activités relatives au module « base de données avancées »

1. Accédez au site de l'université http://elearning.univ-chlef.dz/fr/login/

2. Introduire votre nom utilisateur et votre mot de passe

3. Après identification, l'interface suivante vous permet de choisir le cours BDA.

4. L'interface du cours BDA apparait.

Aperçu des sections

Base de données avancées

Objectif général

Le module vise à préparer les apprenants à la conception et l'administration de base de données relationnelle ou post-relationnelles.

Objectifs spécifiques

A l'issue de ce cours, vous allez être capable de :

- **Utiliser une méthodologie de conception de base de données.**
- **Maîtriser des éléments d'architecture logique et physique d'une base de données relationnelles ou post-relationnelles.**
- **Gerer les accès concurrents.**

5. L'interface du cours BDA contient les ressources et les activités suivantes :

Contenu du programme du cours

Contenu du cours multimédia :

Les cours à préparer pour l'examen

Exposés à préparer pour l'examen

Travail demandé

Chaque binome doit choisir un sujet qu'il a traité en 3LMD c'est à dire il dispose de son diagramme de classe UML et son diagramme de cas d'utilisation.

Pendant votre projet de licence, vous avez fait une implémentation de type relationnel c'est à dire vous avez utilisé un SGBD relationnel (Oracle,mysql,SQL server,...etc.)

Notre travail relatif à ce module consiste à traiter les objectifs suivants:

1. Transformer le modèle relationnel en objet relationnel afin d'intégrer des notions de NF2, REF,héritage, ..etc.
2. Si on suppose que votre PFE va durer 10 ans, penser à intégrer un système de pilotage grace à l'utilisation des schéma OLAP (Modèle en flocon, en étoile,...)

Questions autour du COURS

Vous pouvez déposer votre travail ICI

Cliquez ici pour voir les notes du Module BDDA pour ISIA

Cliquez ici pour voir les notes du Module BDDA pour IL

Cliquez ici pour voir l'examen

Cliquez ici pour voir la correction de l'examen

Cliquez ici pour voir les notes de rattrapage

Bibliographie

AMANN, B., & SCHOLL, M. (2016). *SYSTÈMES INFORMATIQUES - Systèmes de gestion de bases de données.* Encyclopædia Universalis.

Andreas, L., & Lars, M. (2000). *chapitre_bdxml*. Récupéré sur http://miage.univ-nantes.fr/miage/D2X1/chapitre_bdxml/section_principes.htm#contexte.

campioni, c. (2009). *TP-acces-concurrents.htm.* Récupéré sur http://pageperso.lif.univ-mrs.fr/: http://pageperso.lif.univ-mrs.fr/~christine.campioni/documents/BD/TP-acces-concurrents.htm

Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., & Papakonstantinou, Y. (1994). The TSIMMIS Project: Integration of heterogeneous. *in 16th Meeting of the Information Processing Society of Japan, Tokyo*.

Connolly, T., & Begg, C. (2005). *Systèmes de bases de données : approche pratique de la conception, de l'implementation et de l'administration.* Eyrolles .

Connolly, T., Begg, C., & Strachan, A. (2004). *chapitre 29: Database Systems A Practical Approach to Design, Implementation and Management.* Addison Wesley.

CROZAT, S. (2014). *Le modèle logique relationnel-objet et son implémentation sous Oracle.* http://bdd.crzt.fr.

DAL ZILIO, S., & LUGIEZ, D. (2002). *Fondements de l'Interrogation des Données Semi-Structurée.* Récupéré sur http://homepages.laas.fr/dalzilio/atipexml.html

Delobel, C., Lécluse, C., & Richard, P. (1991). *Bases de données des systèmes relationnelles aux systèmes objets.* intereditions.

Gardarin, G. (2003). *chapitre 13:Bases de Données Objet et Relationnel.* Eyrolles.

Georges, G. (1986). *Bases de Données, les Systèmes et leurs Langages* . Eyrolles.

Godin, R. (2006). *Chapitre 7:Systèmes de gestion de bases de données par l'exemple.* Montréal: Loze-Dion.

Halevy. (2001). *Answering queries using views : a survey*. Récupéré sur VLDB Journal: http://www.cs.washington.edu/homes/alon/

JUGANARU-MATHIEU, M. (2012). XML et les bases de données. École Nationale Supérieure des Mines de St Etienne.

Kaabi, R. S. (2011). *Lacunes du modèle relationnel.*

Les accès concurents dans les bases de données . (s.d.). Récupéré sur http://perso.telecom-paristech.fr/~talel/cours/inf225/wwwbd/Cours/transactions/Cours/Cours_BD.html

Manolescu, I., Florescu, D., & Kossman, D. (2001). Answering XML queries over heterogeneous data sources, BDA .

Melton, J., & Simon, A. (2001). *SQL 1999: Understanding Relational Language Components.* Morgan Kaufmann Publishers.

Mirand, S., & Busta, J.-M. (1986). *L'art des Bases de Données-Les bases de données relationnelles* . Eyrolles Tome 2.

Moussa, R. (2006). *Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle.* Carthage: Ecole Supérieure de Technologie et d'Informatique à Carthage.

Navathe, E. (1994). *Fundamentals of Database Systems The Benjamin* . Cummings Publishing Company.

Omran, A., Bukhres, A., Elmagarmid, & K. (1996). *Object Oriented Multidatabase Systems: A solution for advanced applications* . Prentice Hall .

Rob, P., & Coronel, C. (1993). Database Systems. *Publishing Company* .

Sans, V. (2000). *BASE DE DONNÉES OBJET.* Université de RENNE 1.

telecom-paris. (2011). *Les accès concurents dans les bases de données*. Récupéré sur http://perso.telecom-paristech.fr/: http://perso.telecom-paristech.fr/~talel/cours/inf225/wwwbd/Cours/transactions/Cours/Cours_BD.html

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer, 25(3),* 38-49.

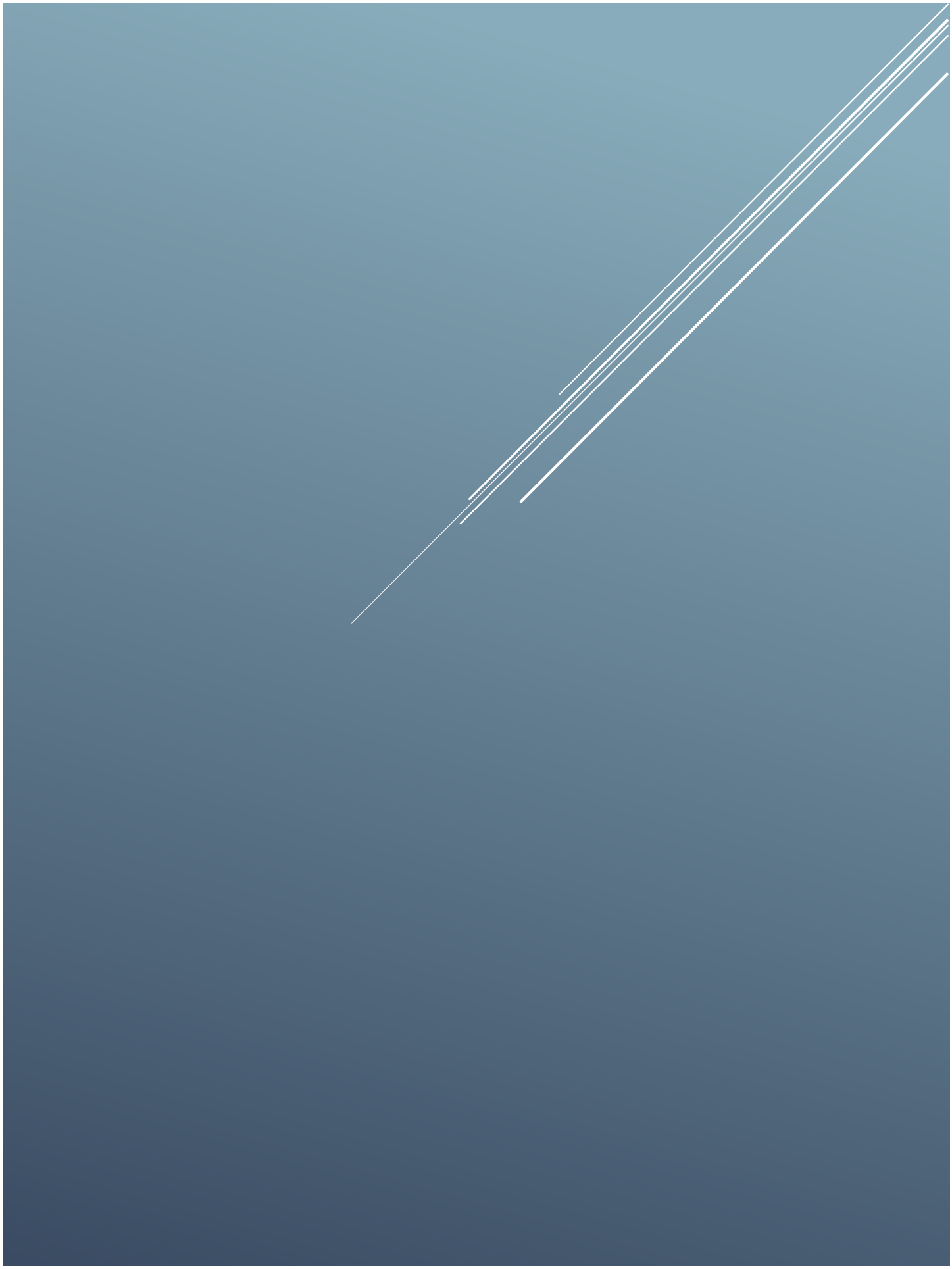
Wiederhold, G. (1995). Mediation in information systems. *ACM Computing Surveys*, 27(2).

Wikipédia. (2016). *Base de données relationnelle*. Récupéré sur http://fr.wikipedia.org/w/index.php?title=Base_de_donn%C3%A9es_relationnelle&action=history: http://fr.wikipedia.org/w/index.php?title=Base_de_donn%C3%A9es_relationnelle&oldid=123523239

Wikipédia. (2016). *Extensible Markup Language*. Récupéré sur Extensible Markup Language,Wikipédia: http://fr.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=123142791

Wikipédia. (2016). *Standard Generalized Markup Language*. Récupéré sur Wikipédia, l'encyclopédie libre: http://fr.wikipedia.org/w/index.php?title=Standard_Generalized_Markup_Language&oldid=122494051

Wiley, K. (2006). . Interscience mobile database Systems.



View publication stats