

# Module: XML et les bases de données

## Rappel sur les SGBD: R & RO

***Houda Chabbi Drissi***

houda.chabbi@hefr.ch

- Bases de données de Georges Gardarin Eyrolles
- SQL in a Nutshell by K. Kline, D. Kline, B. Hunt ISBN 0596004818 O'REILLY.

# SGBDR-RO: Système de gestion de base de données

---

- **SGBD** est un logiciel spécialisé pour la gestion de base de données qui offre un **langage de définition** des données (LDD) et **langage de manipulation** des données (LMD comme SQL, OQL)
- Les **données** sont:
  - **fortement structurées**
  - **persistantes**
  - **avec une structure définie dans un schéma**

# Historique

---

- Réseau et hiérarchique 70 - 80
- Relationnel 80 - 90
- Objet et objet-Relationnel 90 - ...
- *Natif XML 2000-...*
- *NoSQL 2004 (bigtable)-...*

## Le who's who

---

- **Réseau, hiérarchique:** CODASYL, IMS,...
- **Relationnel:** Oracle, Ingres, DB2, SQLserver,...
- **Objet:** O2, Object Store, Versant,...
- **Relationnel objet:** Postgres, Oracle8, DB2
- **NatifXML:** eXist, BaseX, *Oracle9i*, *DB2*, SQLserver

---

# Concepts des SGBDs

# Caractéristiques

---

- **Persistance**
- Gestion de la **concurrency**: partage des données
- **Fiabilité**: atomicité des transactions
- **Confidentialité**: sécurité
- **Indépendance Logique/Physique**: organisation physique transparente
- **Langage de requêtes** (ex : SQL, OQL, XQUERY)
- **Optimisation**: minimiser les accès disque / optimiser les requêtes

## Partage des données

---

- Environnement multi utilisateurs: **partage** transparent de données **cohérentes**
- Unité de cohérence (ACID): la **transaction** (commit, abort)
- Concurrence au niveau des données, des schémas et des index.

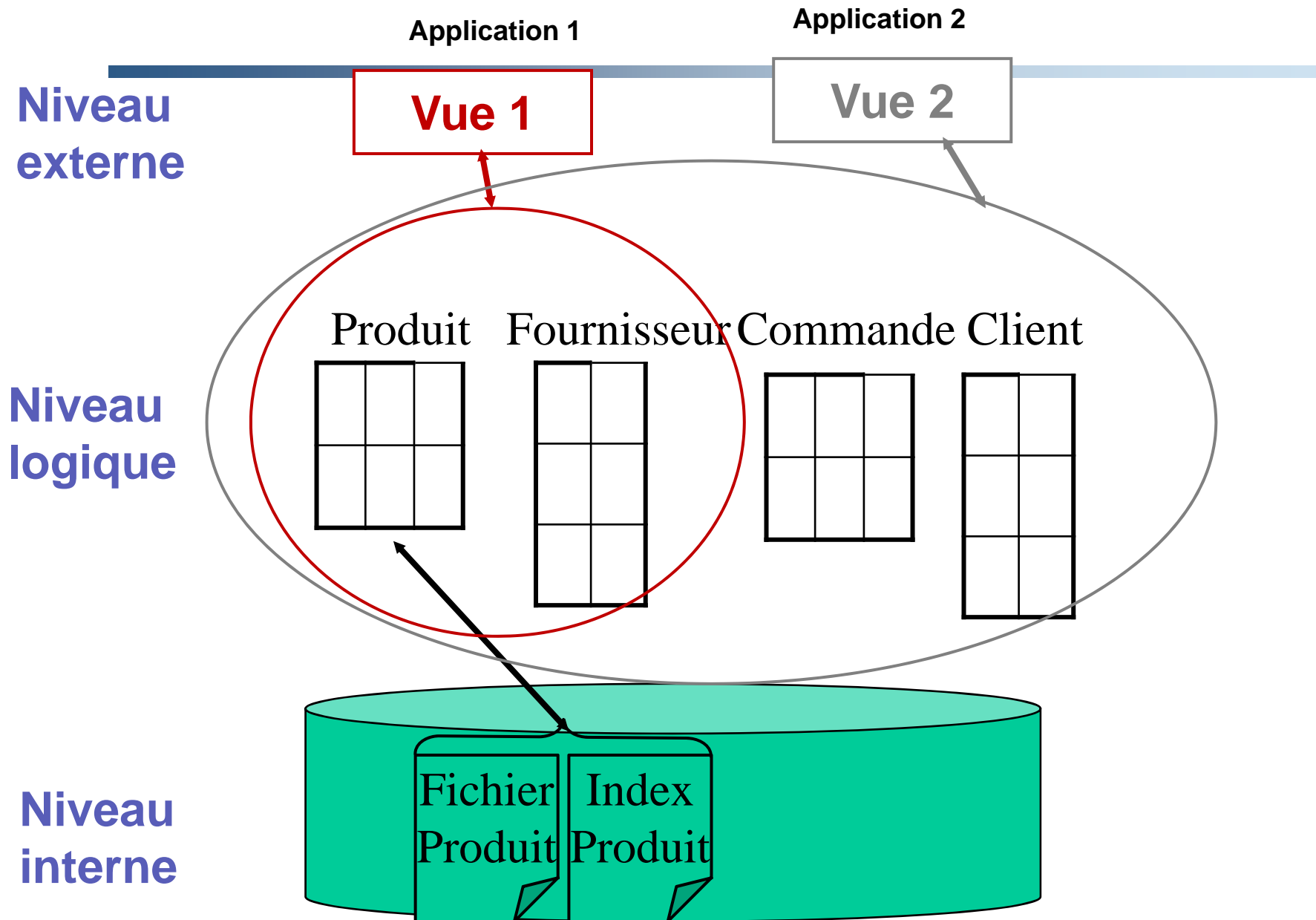
# Indépendance logique / physique

---

Organisation physique de la BD transparente aux programmeurs d'applications:

- Modification possible sans reprise des applications (ajout/retrait d'index, trigger, groupements ou distributions).
- Mais recompilation pour optimiser les requêtes suivant le nouveau schéma physique





# Synthèse sur la conception

<p>Monde réel</p> <p>Pourquoi?</p>				
<p>Modèle conceptuel</p> <p>Quoi?</p>	<ul style="list-style-type: none"> <li>• Indépendant du modèle de données</li> <li>• Indépendant du SGBD</li> </ul>	<pre> graph LR     Malade[Malade] -- "(0,n)" --- Opération((Opération))     Opération -- "(1,n)" --- Médecin[Médecin]         </pre>		
<p>Modèle logique</p> <p>Comment?</p>	<ul style="list-style-type: none"> <li>• dépendant du modèle de données</li> <li>• Indépendant du SGBD</li> </ul>	Relationnel	XML	Objet...
<p>Modèle physique</p> <p>Techno?</p>	<ul style="list-style-type: none"> <li>• dépendant du modèle de données</li> <li>• dépendant du SGBD</li> </ul>	<ul style="list-style-type: none"> <li>• Organisation physique des données</li> <li>• Structures de stockage des données</li> <li>• Structures accélératrices (index)</li> </ul>		

# Les requêtes

---

- Simples à écrire
- **Déclaratives** : indiquer ce que l'on veut (le **quoi**) sans préciser les algorithmes de recherche (le **comment**) (indépendance du physique/logique)
- **Optimisées** automatiquement par le processeur de requêtes.

# Optimisation

---

- Gestion du disque / mémoire (buffers / cache)
- Optimisation des requêtes
- Dénormalisation
- Indexation

Index: Permet d'accéder aux données (ligne, mot) quasi instantanément au lieu de réaliser un balayage séquentiel des données (table, texte)

# Synthèse

---

- Modèle de donnée logique: EA? R? RO?XSD?
- LMD, LDD: structurelle? Contenu?
- Granularité: sécurité
- Optimisation: indexation?

Transactionnel + fiabilité + évolutivité

---

# SQL: Structured Query Language

## Standard ANSI/ISO

---

- SQL86: version préliminaire
- SQL89 (SQL, SQL1) niveau minimal supporté
- SQL92 (SQL2): support accru de l'intégrité
  - Le plus répandu
- SQL1999 (SQL3): extensions objet, TRIGGER, ROLE, SQL/PSM, ...
- SQL2003: standardise les séquences et les colonnes auto-générée, support de XML (SQL/XML)...

# Introduction SQL

---

Norme établie pour *SGBD* relationnel:

- *Partie LDD*

- *Conceptuel* : CREATE SCHEMA, TABLE, PRIMARY KEY, FOREIGN KEY, CHECK, TRIGGER...
- *Externe* : CREATE VIEW, GRANT,...
- *Interne* : CREATE INDEX, CLUSTER,...

- *Partie LMD*

- SELECT, INSERT, DELETE, UPDATE

- *Partie procédurale*: PL/SQL



## Certains type de données (1)

---

### ▪ Type concernant les caractères:

- Char(longueur) (max. 2000 carac)
- Varchar2(longueur) (max. 4000 carac)
- Long (max. 2Go)

### ▪ Type gros objets: LOB

- LOB interne:
  - ✓ CLOB documents (max. 4Go)
  - ✓ BLOB vidéo, son (max. 4Go)
- LOB externe:
  - ✓ BFILE pour stocker une liste de pointeurs vers des fichiers extérieurs à la BD.  
(Fichier de taille max. 4Go)

Ordres SQL incapables de gérer directement les colonnes de types LOB utilisation de package spécifiques.

## Certains type de données (2)

---

### SQL2003: SQL/XML (Part. 14) définit:

- Un nouveau type: XML basé sur le modèle infoset.

## LDD: Exemple LOB

---

Create Table image (

Code Char(3),

Description CLOB,

Photo BLOB,

Plan BFILE);

```
Create Table produits (  
  Code                Char(3),  
  Description         XML);
```

## LMD: SELECT

- Syntaxe de *requêteSQL*

```
selectSQL |
(requêteSQL) {UNION|INTERSECT|EXCEPT} (requêteSQL)
```

- Syntaxe du *selectSQL*

```
SELECT      {[ALL|DISTINCT] expression [AS nomColonne]
              [,expression [AS nomColonne]]...} | *
FROM        table [AS nomTable [(nomColonne[,nomColonne])]]
              [,table [AS nomTable [(nomColonne[,nomColonne])]]]]...
[WHERE      conditionSQL]
[GROUP BY   nomColonne [,nomColonne]...]
[HAVING     conditionSQL]
[ORDER BY   nomColonne [ASC|DESC] [,nomColonne[ASC|DESC]]...]
```

# SelectSQL fonctionnement

	A1	A2		An	
T	1	10		1	Group 1
	2	21		1	
	3	11		1	
	4	21		1	
	5	32		2	Group 2
	6	2		2	
	7	22		2	
	8	23		3	Group 3
	9	14		4	Group 4

```

SELECT An,MAX(A2) FROM T
WHERE A1 > 2
GROUP BY An
HAVING Count(*) > 2
    
```

## LMD: Exemples

```
Create Table image ( Code char(3),
Description      CLOB,
Photo            BLOB,
Plan             BFILE);
```

**Insert** into image values (' A12' , ' <Photo> EIF - Bloc C </Photo>' , EMPTY\_BLOB(),NULL)

**Update** image set Plan =  
BFILENAME(' C:\Mesplans' , 'PI1' )

**Select** Code, Description FROM image;

Retourne: A12   <Photo> EIF - Bloc C </Photo>,  
*N'affiche que les 4000 premiers carac.*

## PL/SQL: Exemple CLOB

```
Create Table image ( Code char(3),
Description      CLOB,
Photo            BLOB,
Plan             BFILE);
```

```
CREATE PROCEDURE insert_clob (code char(3),
                             p_text VARCHAR2) IS
v_clob CLOB; --variable
BEGIN
-- On insère la ligne avec un CLOB vide
INSERT INTO image (code, description) VALUES (code,
empty_clob()) returning description into v_clob;

-- On le remplit avec son contenu
DBMS_LOB.WRITE(v_clob, 1, length(p_text), p_text);
END;
```



## PL/SQL: Exemple BLOB(1)

```
Create Table image ( Code char(3),
Description      CLOB,
Photo           BLOB,
Plan            BFILE);
```

```
CREATE PROCEDURE insert_blob (code char(3), p_dir
varchar2, p_file VARCHAR2) IS v_blob BLOB; v_bfile
BFILE;
```

```
BEGIN
```

```
-- On insère la ligne avec un BLOB vide
```

```
INSERT INTO image (code, Photo) VALUES (code,
empty_blob()) returning image into v_blob;
```

```
-- On déclare un pointeur vers le fichier
```

```
v_bfile := bfilename(p_dir, p_file);
```

## PL/SQL: Exemple BLOB(2)

```
Create Table image ( Code char(3),
Description      CLOB,
Photo            BLOB,
Plan             BFILE);
```

*-- On ouvre ce fichier*

```
dbms_lob.fileopen(v_bfile);
```

*-- On remplit l'emplacement du BLOB vide dans la*

*-- table avec le contenu du fichier*

```
dbms_lob.loadfromfile(v_blob, v_bfile,
dbms_lob.getlength(v_bfile));
```

*-- On ferme le fichier*

```
dbms_lob.fileclose(v_bfile);
```

```
END;
```

# Normalisation

## Normalisation

- **But:** Résolution de problèmes de redondance de données
- **Technique:** Par décomposition en utilisant les dépendances fonctionnelles, multivaluées,...

ProdFournisseur

Id	nomP	NomF	Tel

Fournisseur

Id	nomF	Tel

Id	nomP	Fourn.

Prod

## Normalisation vs Performance

ProdFournisseur

Id	nomP	NomF	Tel

Select \* from ProdFournisseur

**JOINTURE**

Select \* from (Prod, Fournisseur)

Where Fourn = Fournisseur.Id

Fournisseur

Id	nomF	Tel

Prod

Id	nomP	Fourn.

# Les formes normales 1,2,3...

---

- **FN1**: valeurs atomiques (ni multivaluées ni composées)
- **FN2**: Tout attribut non clé dépend de la clé
- **FN3**: Tout attribut non clé dépend directement de la clé
- ....

---

## Conclusion sur les SGBDR

## Normalisation?

---

- **Normalisé** : pas toujours satisfaisant
  
- **Dénormalisation** pour tenir compte des
  - colonnes dérivées
  - choix des colonnes
  - performance en lecture
  - ...



## Les plus

---

- Fondements **théoriques solides** (algèbre relationnelle)
- **Standard SQL**
- Modèle **simple**
- Support solide de l'**intégrité**
- **Indexation** efficace
- Outils de développement
- **Maturité de la technologie**  $\Rightarrow$  Technologie la plus répandue

## Les moins

---

- Structure de données **trop simple**
- Ecriture d'application se fait hors SGBD avec une nécessité de **convertir les types entre les deux mondes**
- LMD déclaratif orienté ensemble entraine l'utilisation **de curseurs**
- Ne permet **pas de gestion de fonctions spécifiques** à certaines données (multimédia)

## Les performances

---

- Sur un SGBDR deux opérations
  - Join (lent)
  - Select (rapide)
- Cas des objets complexes *ou de XML*
  - répartis sur plusieurs tables avec une traduction (schéma et requêtes) source d'erreur
  - beaucoup de jointures  $\Rightarrow$  mauvaises performances

# Les SGBDs Relationnel-Objet

***Houda Chabbi Drissi***

houda.chabbi@hefr.ch

# SGBDRO utilisé pour les exemples: Oracle

---

## Concepts OO

Concepts  
SGBDR

+

- Objet
- Encapsulation
- Objet complexe
- Oid: identité d'objet
- ~~Classe~~
- Héritage
- ~~Polymorphisme~~

## Objets complexes

---

- Attributs multivalués
- Introduction de collections prédéfinies telles liste, ensemble, tableau, ...
- Imbrication des collections

Exemple:


Type Employe

{Nom,list{prénom}, list {Diplôme},photo}

Type Departement

{id,intitulé,list (Employes)}

## Exemple de table et objet

Id	Intitulé	Employé			
1	TIC				
		Nom	Prénoms	Diplômes	photo
			<div><div></div><div></div></div>	<div><div></div><div></div></div>	

Objet Département

## SQL3

---

- Un langage de **définition de types**:
  - Définition de **types abstraits utilisateurs**
  - Possibilité de types avec ou sans OID / Utilisation de référence (OID)
  - **Constructeurs de types** (tuples, set, list, ...)
  - Définition de sous-types/ sous-tables (héritage)
- Un langage de **programmation**
- Un langage de **requêtes**



## Type abstrait

---

**CREATE TYPE <nom ADT> <corps de l'ADT>**

SQL3 les nomme types abstraits (ADT). Ces types sont alors utilisables comme type:

- ✓ D'une colonne d'une table ordinaire,
- ✓ D'un attribut d'un autre type,
- ✓ D'un type d'objet d'une table d'objets

Une table en 2FN peut contenir un attribut composé d'une liste de valeurs et/ou un attribut composé de plusieurs attributs.

## Exemple de création d'objet / table d'objet (oracle)

---

Create type **typeAdresse** as object(  
 Rue Varchar(50),  
 Ville Varchar(30),  
 Pays Varchar(30))

Create type typePersonne as object(  
 Nom Varchar(50),  
 Adr **TypeAdresse**)

Create table **Personnel** OF **typePersonne**(  
 Nom Primary Key)

## Exemple de création d'objet avec méthode

```
Create or Replace type typePersonne as object(  
  Nom      Varchar(50), Adr TypeAdresse,  
  Salaire  Number(8,2)  
  Member function revenu return Number)
```

```
Create type body typePersonne is  
  Member function revenu return Number is  
    revenu number (8,2);  
  
  Begin ....  
  End revenu;  
End
```

# Référence d'objet

**OID**: adresse invariante d'un objet (**OID**).

**Référence** = permet les jointures et implémente le concept de foreign key

Create type typePersonne as object(

Nom

Varchar(50),

Departement

REF typeDepartement

Adr

TypeAdresse)

Create table Personnel OF typePersonne(

Nom Primary Key)

Select Nom, P.department.intitule from Personnel P

Insert into Personnel values select 'Martin', ref(D), ...  
from Departement D where code = 12;

## Collection

SQL3 permet de supporter des attributs multivalués via les constructeurs de base: **SET(T)**, **MULTISET(T)** et **LIST(T)**. Oracle propose les tableaux et les tables imbriquées

### Oracle: Tableau

```
Create Type typeStock AS VARRAY (12) OF Number(6,2)
```

```
Create table Recap  
(produit number(4),  
Recap_stock typeStock)--récapitulatif des stocks par mois
```

```
Insert into Recap values (12,typeStock(1234,43))
```

Les valeurs des deux premiers mois

Manipulation via PL/SQL et une variable de type tableau

## Conclusions

---

- **Standard** SQL3: variation dans les implémentations
- Étend les possibilités du relationnel  $\Rightarrow$  **Prise en compte des nouveaux types (XML 😊)** plus aisée

## Standard SQL ET XML?

---

- SQL92
  - Mauvais support de l'imbrication
  - GROUP BY limités
- SQL3
  - Requêtes imbriquées difficiles
  - Références pas très claires
- XQUERY
  - Parcours arbres
  - Recherche textuelles

SQL2003 la solution?

## SQL ET XML? (1)

Nos premières requêtes basiques SQL qui génèrent du XML?

T\_region

Id	Nom	...

```
<?xml version="1.0" ?>
<regions>
  <region>
    <nom>... </nom>
  </region>
  ....
</regions>
```



## SQL ET XML? (2)

Nos premières requêtes basiques SQL qui génèrent du XML?

```
Select '<?xml version=« 1.0 » ?><regions>' from dual;
```

```
Select '<region><nom>' || nom || '</nom> </region>'
From T_region where id = 1;
```

...

```
Select '<region><nom>' || nom || '</nom> </region>'
From T_region where id = 100;
```

```
Select '</regions>' from dual;
```

```
<?xml version=« 1.0 » ?>
<regions>
```

```
<region>
    <nom>... </nom>
</region>
```

.....

```
</regions>
```

Pas tout à fait  
un document!

## SQL ET XML? (3)

Encore mieux: SQL procédural!

Pseudo-code:

Begin

Print ('<?xml version=« 1.0 » ?><regions>');

Curseur Regions as select \* from T\_region;

For R In Regions

Print ('<region><nom>' || R.nom || '</nom> </region>');

Print ('</regions>');

End

```
<?xml version=« 1.0 » ?> <regions>
  <region>
    <nom>... </nom>
  </region>
  ....
</regions>
```



## SQL ET XML? (4)

---

- C'est bien le maximum que l'on puisse faire avec SQL92!
- C'est très limité:
  - Structure hiérarchique = multitude de selects complexes (annexe: [SQL hiérarchique](#))
  - Les conversions en entités interne ne sont pas assurées = XML pas bien fondé ☹