Université Henri Poincaré

Bases de données Relationnelles

Hala Skaf-Molli

Loria

bureau B032

e-mail: skaf@loria.fr

www.loria.fr/~skaf

Plan de cours

- Introduction
- Fonctionnalités d'un SGBD
- Développement d'une base de données
- Modèle relationnel...
  - Algèbre relationnel
  - Langage de requête SQL
- Programmer une application: JSQL, JDBC
- Trigger ..
- Transactions ..

BIBLIOGRAPHIE

- C. Date , "An Introduction to Data Base System", Addison Wesley, 1994.
- C. Delobel, M. Adiba "Bases de données et systèmes relationnels", Dunod Informatique, 1982 .
- G. Gardarin "Bases de données : les systèmes et leurs langages", Eyrolles, 1983
- J. Ullman "Principles of database systems", Computer Science Press, 1982
- **N. Boudjlida. Bases de données relationnelles et systèmes d'informations : Langages, systèmes et méthodes. Cours et exercices corrigés. Dunod, Paris, 1999, Séries Sciences-Sup.**
- **H.G. Molina, J. Ullman, J. Widom. Database Systems the complete Book, Prentice Hall, 2002.**
- http://www.javasoft.com/products/jdbc/index.html
- et d'autres références sur le web..

Introduction

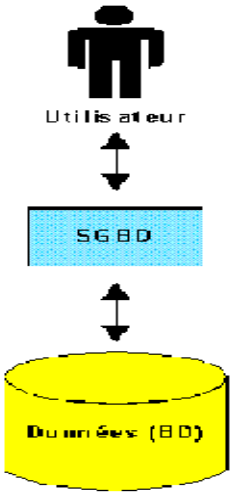
- Pourquoi un SGBD ?
  - Limite de SGF à gérer une grande masse volume de données et les liens entre ces données
  - données reparties entre plusieurs fichiers
  - redondances de données et problème du maintien de la cohérence…
  - effort de programmation important pour exploiter les données (séquentiel, séquentielle indexé, ..)
  - Programme dépend de données..
- Un SGBD permet de répondre à ces besoins..

Base de Données et Système de Gestion de Bases de Données

- BD
  - un ensemble de données bien structuré
  - relatives à un sujet global et accessible par **plusieurs utilisateurs** à la fois.
- Exemples
  - Une banque stocke les informations sur les clients et leurs dépôts d'épargne dans une BD.
  - BD de réservation de tickets du SNCF.
  - Un service de scolarité stocke les informations relatives aux étudiants inscrits dans une BD...

Système de gestion de bases de données

- **SGBD** : un ensemble de programmes permettant à des utilisateurs de créer et d'utiliser de BDs. Les activités supportées sont:
  - la définition d'une base de données (spécification des types de données à stocker)
  - la construction d'une base de données, (stockage des données proprement dites)
  - et la manipulation des données (principalement ajouter, supprimer, retrouver des données).



SGBD

- Sépare la description de données, des données elles mêmes...
- Cette description est stockée dans un **catalogue (dictionnaire)** également géré dans le SGBD et peut-êe consultée par les utilisateurs...
- Les SGBD commerciaux les plus connus sont

  Oracle, Sybase, Ingres, Informix, Access et DB2,
- Les SGBD non commerciaux (free !) les plus connus sont

  Postgres, Sybase sous LINUX, Oracle sous LINUX...

Fonctionnalités d'un SGBD

- Définition de données
- Manipulation des données
- Contrôler la redondance d'informations
- Intégrité des données
- Partage des données
- Confidentialité des données
- Sécurité de fonctionnement
- Support de persistance

Définition de données

- LDD permet de décrire:
  - des **objets** (personnes, étudiants, voitures..)
  - des **attributs** sur les objets (nom, numéro,..)
  - des **liens** entre les objets (une personne *possède* des voitures)
  - des **contraintes** sur les objets, les attributs, les liens (une voiture n'a qu'un propriétaire)

- **Schéma ..**

Manipulation de données

- LMD
  - Création, Recherche, Suppression, Modification des données
- Interfaces d'accès multiples :
  - Interfaces orientées utilisateur final
    - langages de requêtes déclaratifs comme **SQL** avec mise en œuvre graphique, interface de type formulaire, ...
  - Interfaces orientées programmeurs d'applications:
    - interface avec des langages de programmation classiques : C, Cobol, Pascal, C++, JAVA ou "embedded SQL"

Redondance de données

- Problèmes
  - coût en temps,
  - coût en volume
  - et risque d'incohérence entre les différentes copies…
- Supprimer la redondance

Contraintes d'intégrité

- Un schéma BD se compose de:
  - une **description des données**, de leurs **relations**, ainsi que d'un ensemble de **contraintes d'intégrité**.

  Une CI est une propriété de l'application à modéliser.

  Les données stockés dans une BD doivent vérifier ces CI..

- 2 types des contraintes:
  - Contraintes **structurelles** : *un employé a un et un seul chef*
  - Contraintes **dynamique**: *un salaire ne peut pas diminuer*
  - les SGBD commerciaux supportent automatiquement un certain nombre de contraintes **structurelles**, mais ne prennent pas en compte les contraintes dynamiques (elles doivent être codées dans les programmes d'application)

Les accès concurrents

- Plusieurs utilisateurs peuvent accéder à la même information en même temps.
  - Contrôler les accès concurrents:
    - des techniques de **verrouillage** des données (pour éviter par exemple qu'on puisse lire une information qu'on est en train de mettre à jour).
  - Vue utilisateur

Confidentialité des données

- Multi-utilisateurs ..
- Problème de la confidentialité des données
- Gérer des droits d'accès sur les données:
  - droits de lecture,
  - de mise à jour,
  - mots de passe

Sécurité de fonctionnement

- Remettre rapidement une BD dans un état **opérationnel** après un incident hardware ou software
- J**ournalisation** des opérations réalisées sur la BD;
- Ré-exécution automatique en cas de besoin.

Architecture Fonctionnelle d'un SGBD

Interface 4ème génération

S G B D

LDD | LMD | Confidentialié | Securité | concurrence | partage | Intégrité | Interface

Pascal
Cobol
C
Java

Système d'exploitation

Base de données

Base de données

Développement d'une base de données

- Méthodes de conception classiques pas adaptées
  - Nouvelles méthodes de conception (MERISE, REMORA,…).
  - Trois niveaux d'abstraction de représentation des données

Niveau conceptuel

Description abstraite et globale du monde réel

- aspect statique (données)
- aspects dynamique (traitements)

=> le schéma conceptuel

Le schéma conceptuel décrit la structure de la base **indépendamment de son implantation**

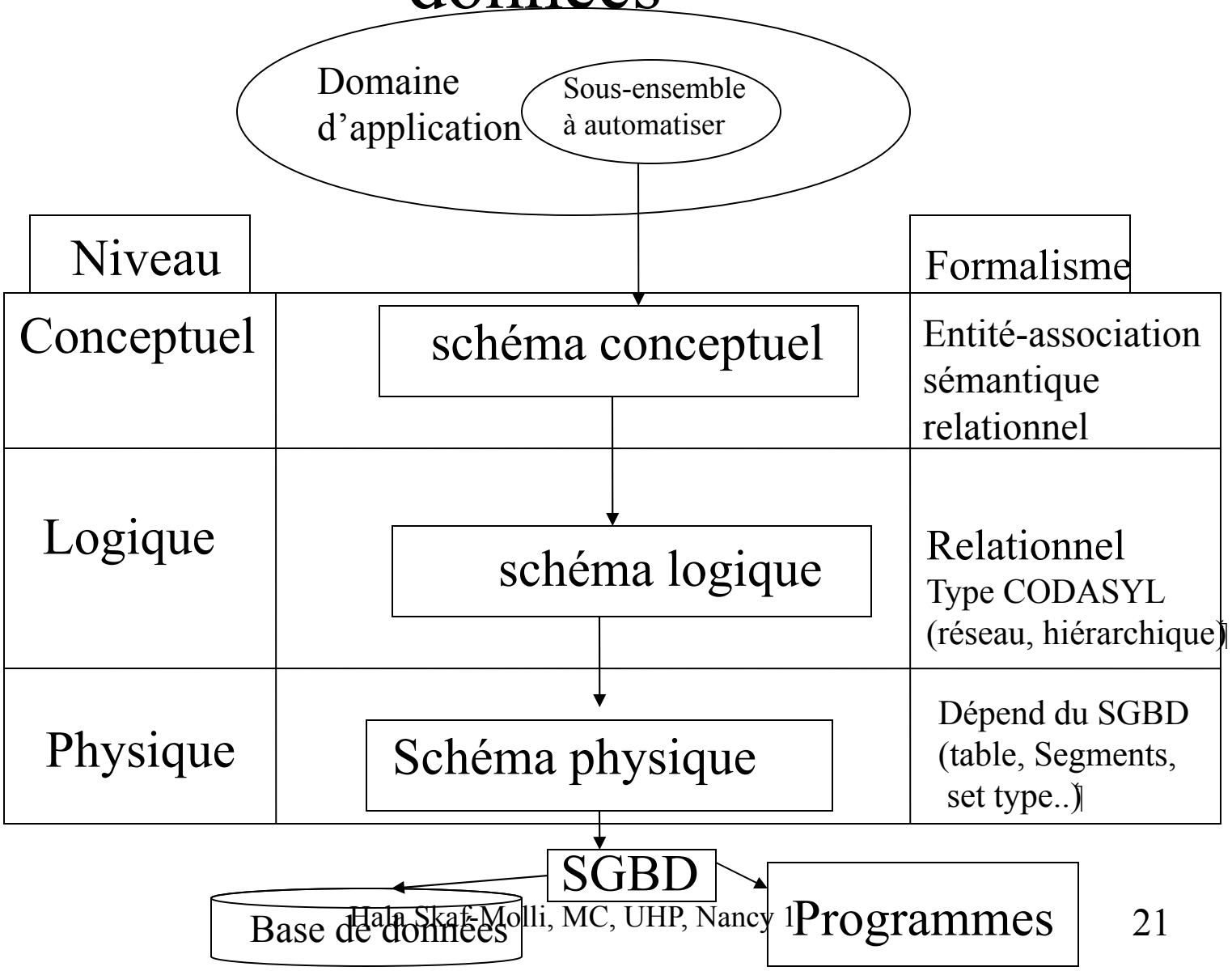
Niveau logique

- Prise en compte de facteurs quantitatifs
  - évaluer le volume de données
  - évaluer les coûts de mise en place de la BD
  - Le schéma logique est issu du schéma conceptuel + [transformations]

Niveau physique

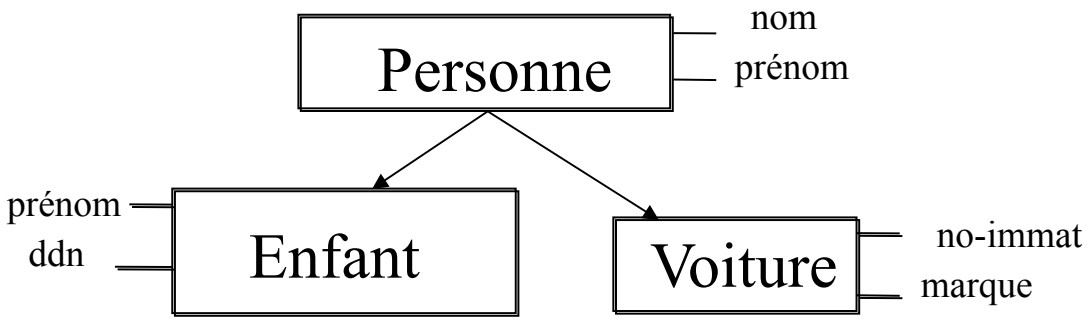
- Contraintes liées au matériel et logiciel
- Adaptation du schéma logique
- Structures de données décrites dans le LDD
- Traitements exprimés avec les outils de manipulation du SGBD

Cycle de développement des bases de données



Modèle de représentation de données: Typologie des SGBD

- Modèle hiérarchique:



- Modèle réseau
- **Modèle relationnel**
- Modèle entité-association
- Modèle Orienté objet

Métiers du domaine

- Utilisaturs de bases de données
  - Les utilisateurs **occasionnels** : technicité moyenne, LMD
  - Les utilisateurs **naïfs**: "presse-bouton "
  - Les utilisateurs **plus spécialisés**: mettre en oeuvre les différents outils du SGBD

Métiers du domaine

- Concepteurs et développeurs:
  - **Concepteur de bases de données**: il identife et strusture les types de données, et traitements (compétence en BD + conception de logiciel)
  - **Les développeurs d'applications**: déterminer les besoin des utilisateurs, implanter les transactions et programmes nécessaires (compéténce algorithmique et programmation + LMD)

Métiers du domaine

- Administrateur des bases et des systèmes (personne ou équipe de personnes)
  - Droit d'accès à la base
  - Problème de performance
  - Sécurité de fonctionnement

Compétence au-déla de concepteurs de la base de données et de développeurs d'applications

Métiers du domaine

- Réalisateur de logiciel de gestion et de développement de bases de données
  - **Les développeur d'outils**: logiciels facilitant la conception, l'implémentation et l'utilisation de bases de données ou SGBD
  - **Les concepteurs et implémenteurs de SGBD:** compétence en BD, en compilation, en systèmes d'exploitation, en réseaux, ...

Le modèle relationnel de données

- Introduction
- Définition formelle
- Caractéristiques des relations
- Contraintes d'intégrité
- Algèbre relationnelle

Introduction

- 1970 par Codd
- 1ères réalisations (System-R IBM, Ingres Berkeley), vers 1976.
- Les premiers systèmes commerciaux, au début des années 80.
- Le modèle relationnel est **simple**, facile à appréhender, même pour un non spécialiste.
- Solides bases théoriques: définir de façon formelle les langages de manipulation associés.
- Le **modèle relationnel** représente l'information dans une collection de relations.
  - Une **relation** comme une **table** à double entrée, voir même comme un fichier.
  - Chaque **ligne** de la table (appelée **nuplet** ou **tuple**) peut être vue comme un fait décrivant une entité du monde.
  - Une **colonne** de la table est appelée un **attribut**.

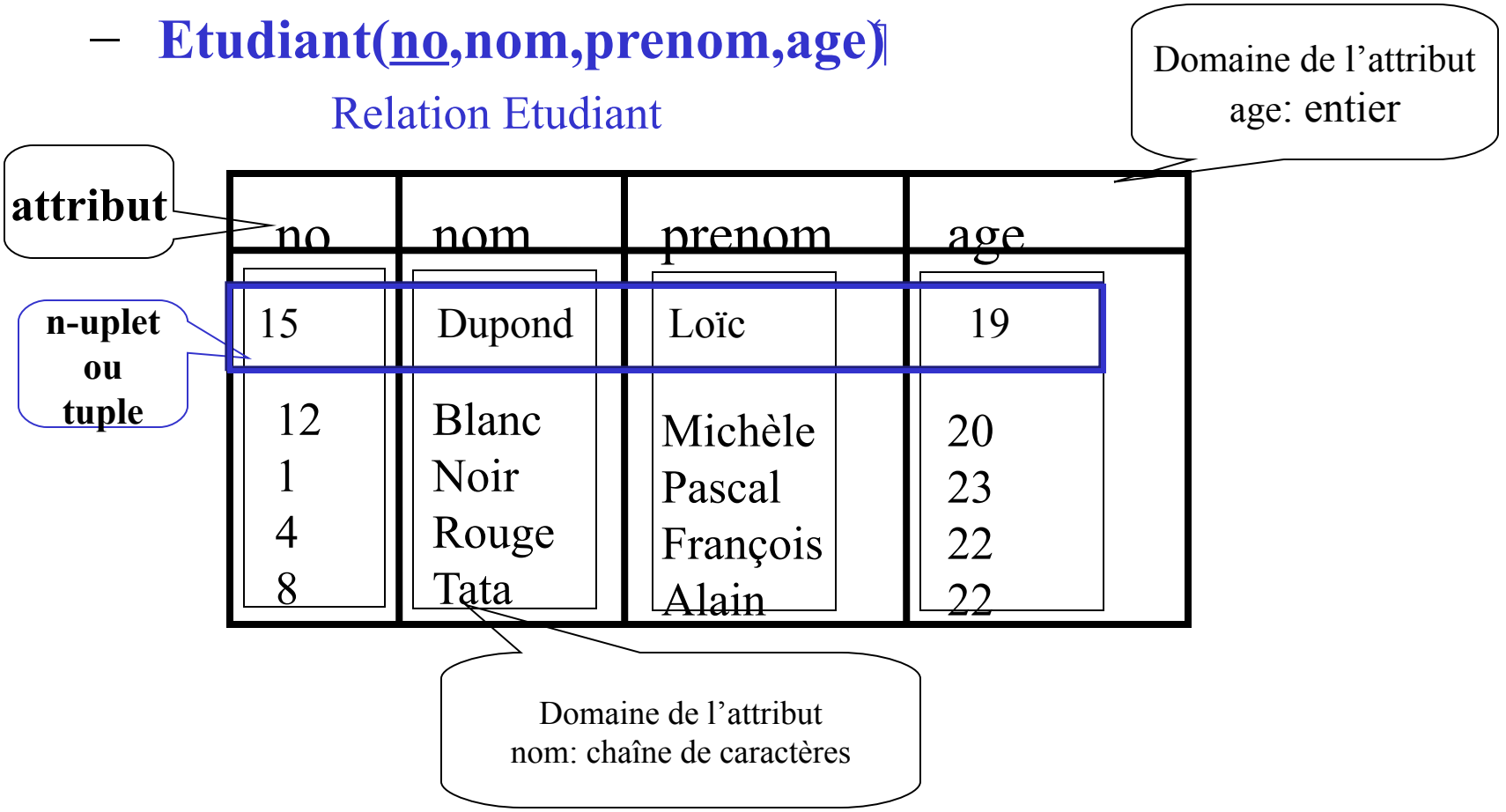
Relation et schéma de relation

- **Schéma de relation R**
  - R(A1, A2, ..., An) est un ensemble d'attributs R = {A1, A2, ..., An}; chaque **attribut** Ai est associé un domaine Di
- **Domaine** est un ensemble de valeurs atomiques.

- Une **relation** r sur le schéma de relation R(A1, A2, ..., An) est un ensemble de n-uplets r= {t1, t2, ..., tn}. r est souvent appelée **extension** (ou instance) du schéma R.

Schéma de la relation Etudiant:

- **Etudiant(no,nom,prenom,age)**



Extension d'une relation (ou instance): ensemble de ses n-uplets

DB relationnelle: ensemble de relations (variables dans le temps)

Schéma relationnel: ensemble des schémas des relation de la BD

Caractéristiques des relations

- Une **relation** est un ensemble de n-uplets: pas d'ordre sur les n-uplets,
- Un **n-uplet** est un **séquence ordonnée** d'attributs...
- Une **valeur d'attribut** est **atomique** mais peut être éventuellement nulle (valeur particulière qui indique que la valeur est manquante).

Contraintes d'intégrité

- Schéma de relation R(U,P)
  - U: liste des attributs et leur domaines
  - P: liste des contraintes d'intégrité CI

- A chaque n-uplet est associé une **clé** qui l'identifie de manière unique.
  - Etudiant(**no,nom,prenom,age**), l'attribut **no** est une **clé primaire**
  - Unicité des valeurs de clé (un seul nuplet étudiant peut avoir une valeur de **no** donnée).
  - Une clé primaire ne peut contenir de valeur nulle.
- Les n-uplets d'une relation sont **distincts** deux à deux

Contraintes d'intégrité

- **CI référentielle entre deux relations**.
  - vérifier que l'information utilisée dans un n-uplet pour désigner un autre nuplet est valide, notamment si le n-uplet désigné existe bien.

    Employe(<u>no_emp</u>, no_emp, adresse_emp, rôle, no_dept)

    Departement(<u>no_dept</u>, nom_dep)
  - un n-uplet de Employe référence un n-uplet de Departement via l'attribut no-dept (numéro de département) ,

Les langages relationnels

- LDD et LMD
- Deux classes de langage:
  - **langage algébrique**
  - **langage prédicatifs**
- Sur le plan industriel:
  - langage algébrique : **SQL** (structured query language)
  - langage prédicatifs : QBE (Query By Example)

Langage algébrique ou algèbre relationnelle

- Un ensemble d'opérateurs sur des **relations** et produisant de nouvelles **relations**.
- Construire de nouvelles informations à partir des relations de départ et d'une composition séquentielle d'opérateurs.

Algèbre relationnelle

Opérateus unaires	Opérateurs binaires (schémas différents)	Opérateurs binaires (relations de même schéma)
• projection • sélection	• produit cartésien • thêta-jointure • division	• union • intersection • différence

Opérateurs unaires

- Projection

  Signature: relation x liste d'attributs relation

  Notation: proj(R,Y) ou y(R) = S(y) tel que:

  Définition informelle:

  La relation S est formée par les colonnes Y de la relation R.

Base de données utilisées dans les exemples

Produit(prod_id,nom,pu)

Depot(dep_id, adr,volume)

Stock(prod_id,dep_id,qte)

Produit

prod_id	nom	pu
p1	A3	10.0
p2	crayon	9
p3	stylo	15
p4	A4	10.0

Depot

dep_id	adr	volume
1	Nancy	100
2	Laxou	200
3	Vandoeuvre	115
5	Nancy	220
6	Nancy	1000

Stock

prod_id	dep_id	qte
p1	1	0
p3	2	9
p1	3	15
p2	5	20
p3	6	0
p1	5	5
p2	6	2
p3	3	30
p1	2	10

Exemple: Projection

- Donner les noms des produits ?

Sélection

- Signature: relation x expression logique relation
- Notation: *select (R,E) ou E(R) ou (R,E)*
- Le résultat a le même schéma que la relation R et chaque n-uplets du résultat vérifie l'expression E.
- E : **Condition booléenne, simple ou composé**
  - **Simple** *nom_attribut op nom_attribut* | *nom_attribut op constante*
  - Op est un opérateur de comparaison (>,>,=,<>,>=,<=)
  - **Composé** si E1, E2 sont des expressions logiques et est un connecteur logique *and or et not* est une expression logique

Exemple: Sélection

- Les produits dont le pu est inférieur à 10?

Opérateurs binaires (schémas différents)

- Produit cartésien x

  Signature: relation x relation relation

  R(X) x S(Y) = T(X U Y)

  X et Y: ensemble disjoints d'attributs

  Définition informelle (extension T)

  ensemble des tuples obtenus par concaténation de chaque tuple de R avec chaque tuple de S

- Attention le nombre d'éléments du produit cartésien est le produit des cardinalités des relations R et S!

Exemple : Produit cartésien

- Produit x Stock

Opérateurs binaires (schémas différents)

- -jointure

  Signature: relation x relation x -- exp relation

  join -E-exp(R,S), join(R,S, -exp), R -exp S

  -exp porte sur les attributs de R et S **(attributs de jointure)**

- -jointure : est équivalente à un produit cartésien suivi d'une opération de sélection.

  R join -E S = -E (R X S)

Exemple : theta-jointure

- Relation qui associe à chaque produit le dépôt dans lequel il est stocké ...

Opérateurs binaires (schémas différents)

- EQUI-JOINTURE
  - θE-exp porte sur les attributs de jointure est ‘=‘
- jointure naturelle *:
  - Équi-jointure suivie de la suppression des attributs superflus
- Outer jointure
- Inner Jointure ..

Opérateurs binaires (schémas différents)

- Division

  Signature: relation x relation → relation

  div(R,S), R ÷ S

  R(X,Y) ÷ S(Y) = T(X)

- Définition formelle:

  $T(X) = \{<x> \mid \quad y, <y> \in S \Rightarrow <x,y> \in R\}$

- Définition informelle

  T est la projection de R sur X restreinte aux tuples en liaison avec tous les tuples de S

Exemple : Division

- -

Opérateurs binaires (relations de même schéma)

- union □ , intersection □, différence -

  Signature: relation x relation □ relation

- R □ S, R □ S, R - S
- Sémantique identique à celles des opérations ensemblistes
- R, S **ont le même schéma** (même ensemble d'attributs)..
- Exemple:

Exemple

- -

Propriétés des opérateurs

- Union et intersection:

  commutatives: $R \cup S = S \cup R$

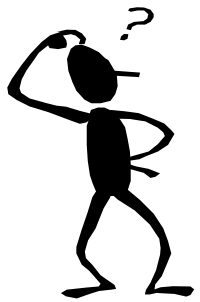
  associatives, $(R \cup S) \cup T = R \cup (S \cup T)$

  idempotentes $R \cup R = R$

- Jointure: commutative, associative
- Optimisation de requêtes ...

Résumons

- 8 opérateurs:
  - sélection, projection
  - jointure, produit cartésien, division
  - union, intersection, différence
- Ensemble minimal d'opérateurs:
  - projection, sélection, produit cartésien,union, différence
- La prochaine épisode, le langage SQL !!



Plan

- Les langages des SGBDR
- De l'algèbre relationnelle à SQL

  select … from… where

Les langages des SGBDR

- SQUARE --->SEQUEL ---> SQL (Structured Query Language)
- SQL est un standard international (ANSI et ISO). Il est connu par tous les SGBDR.
- Désormais la présence de standards internationaux tels que SQL-86, SQL-89, SQL-92 (SQL2), la dernière SQL-99 (SQL3), chaque SGBD sur le marché utilise un peu son propre dialecte du langage SQL.

Les langages des SGBDR SQL

- Des primitives de base de définition, et de manipulation de données.
  - Recherche de données, création des relations temporaires..
  - Fonctions d'agrégation (min, max,….)
  - Définition et Manipulation de schéma de relations (create table, update, insert..)
  - Extensions procédurales (create proc,…)

Exemple

Produit(prod_id,nom,pu)

Depot(dep_id, adr,volume)

Stock(prod_id,dep_id,qte)

Produit

prod_id	nom	pu
p1	A3	10.0
p2	crayon	9
p3	stylo	15
p4	A4	10.0

Depot

dep_id	adr	volume
1	Nancy	100
2	Laxou	200
3	Vandoeuvre	
115		
5	Nancy	220
6	Nancy	1000

Stock

prod_id	dep_id	
qte		
p1	1	0
p3	2	9
p1	3	15
p2	5	20
p3	6	0
p1	5	5
p2	6	2
p3	3	30
p1	2	10

Algèbre relationnelle

- Projection

$\Pi_{\text{liste d'attributs}}(\text{nom de relation})$

SQL

select liste d'attributs

from nom de relation

Exemple:

SQL

Attention: pas d élimination des doubles sauf mention explicite:

select distinct pu
from Produit

ou

select unique pu
from Produit

AR

Sélection

$\sigma_{\text{condition}}$**(nom de relation)**

SQL

select *
from nom de relation
where condition

- * dénote toutes les colonnes de la relation.
- et la condition peut comporter:
  - opérateurs de comparaison: =,! =,>,>=,<,<=, like
  - connecteurs logiques: and, or, not
  - caractères génériques

    % peut se substituer à toute **chaîne** de caractères

    _ peut se substituer tout caractère

Exemple: produit dont le pu est < ? 10

- Quels sont les produits dont le nom commence par un ‘A’ et dont le prix unitaire est inférieur ou égal à 25?
- Réponse (en SQL):

AR

- Composition

  projection-sélection

$\Pi_{\text{l\_attributs}} (\sigma_{\text{cond}(\text{nom de relation})})$

$\sigma_{\text{cond}} (\Pi_{\text{l\_attributs}} (\text{nom de relation}))$

SQL

select l_attributs

from nom de relation

where cond

- Ex: nom et pu des produit dont 9<pu <10 ?

AR

Produit cartésien

jointure sans condition

nom_rel1 x **nom_rel2** x **[…]**

Ex:

Produit x Depot

SQL

select *

from **nom_rel1**, **nom_rel2**, **[…]**

select *

from Produit, Depot

AR

Jointure

nom_rel1 θ **-exp nom_rel2**
[…]

SQL

Forme 1: sans sous-requête

select *
from **nom_rel1**, **nom_rel2**, **[…]**
where θ **-exp**

AR

Jointure

Produit θ **Produit.prod_id= Stocke.prod_id** **Stock**

SQL

Forme 2: Imbrication de select (ou sous-requête)

Outer join

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

```
SELECT A1.store_name, SUM(A2.Sales) SALES
FROM Georgraphy A1, Store_Information A2
WHERE A1.store_name = A2.store_name (+)
GROUP BY A1.store_name
```

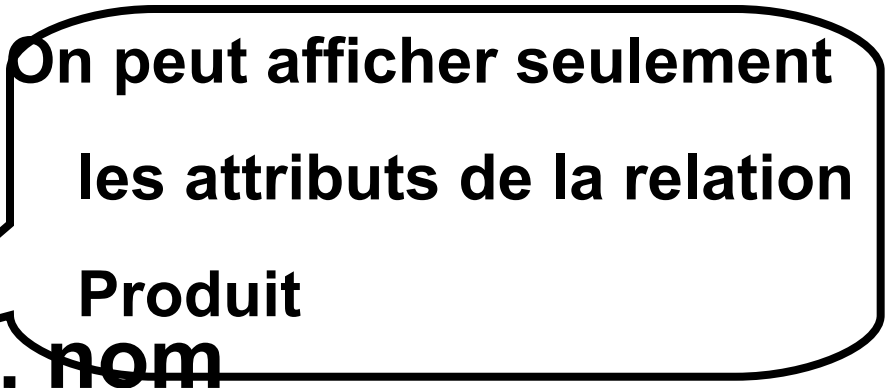
Result:

store_name	SALES
Boston	$700
New York	
Los Angeles	$1800
San Diego	$250

Composition jointure, sélection, projection

- numéros et noms des produits en rupture de stock ?

- En utilisant sous-requête:




```
select prod_id, nom
from Produit
where prod_id in ( select prod_id
                   from Stock
                   where qte <= 0)
```

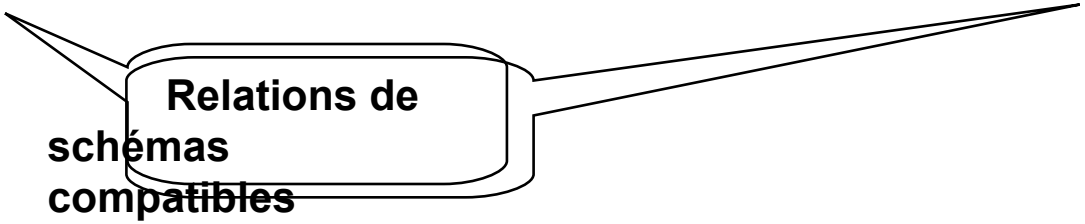
Composition jointure, sélection, projection (suite): *utilisation des alias*

Quels sont les produits en rupture de stock (sortir prod_id, nom, dep_id et adr) ?

Union, Intersection, Différence

Syntaxe:

<clause select> **union** | **intersect** | **minus** <clause select>



Exemple

(select prod_id from Produit

where pu > 100) union (select prod_id from Produit where pu <50)

Ordonnancement des résultat (order by)

```
select dep_id,prod_id, qte
from Stock
order by prod_id ASC

select dep_id,prod_id, qte
from Stock
order by prod_id ASC, qte DESC
```

Fonctions d'agrégation

- count: cardinal ...
- sum : la somme des valeurs..
- avg: moyenne...
- max: la valeur maximale
- min: la valeur minimum

- **Ces fonctions peuvent être utilisées dans une clause select**
- **Le résultat est une valeur (pas un ensemble)**
- **Toutes les fonctions ignorent les valeurs nulles (sauf count(*) voir l 'exemple)**

Fonctions d'agrégation: exemples

```
select count(prod_id)
from Produit
-----
3
select count(nom)
from Produit
-----
2
select count(distinct nom)
from Produit
```

Produit

prod_id	nom
p1	n2
p2	NULL
p3	n2

```
select count(*)   /* donne le nombre des lignes dans une table*/
from  Produit
```

Fonctions d'agrégation: exemples

- Moyenne des pu des produits stockés dans le dépôt 4?

- Le numéro et l 'adresse du dépôt de plus grande capacité ?
  - sinon on peut définir une relation temporaire

Les relations temporaires

- Dans le SGBD Sybase
  - Create table #toto (name type..)
  - **select** liste_attributs **into nom_relation_temporaire**

    from liste_relations

    where condition
- Durée de vie d'une relation temporaire
  - Durée de la session de travail où elle a été créée
- Durée de vie d'une relation
  - Temps écoulé entre sa création et sa destruction
- Toute relation dont le nom débute par # est temporaire: elle est supprimée systématiquement à la fin de session

Les relations temporaires

- Oracle:
  - Table temporaire d'une session:
    - Create global temporary table (nom_col type …..)
  - Table temporaire d'une transaction:
    - Create global temporary table (nom_col type …..) on commit delete rows

Exemples des relations temporaires

- Le numéro et l'adresse du dépôt de plus grand capacité ?

  Avec une relation temporaire:

Synthèse: syntaxe générale de recherche

select <liste d'attributs projetés>

from <liste de relations>

where <liste de critères de restriction et de jointure>

Syntaxe générale de recherche

- La partie from décrit les relations qui sont utilisables dans la requête (c'est à dire l'ensemble des attributs que l'on peut utiliser).
- La partie where exprime la (les conditions) que doivent respecter les attributs d'un tuple pour pouvoir être dans la réponse. Cette partie est optionnelle.
- La partie select indique le sous-ensemble des attributs qui doivent apparaître dans la réponse (c'est le schéma de la relation résultat). Bien entendu ces attributs doivent appartenir aux relations indiquées dans la partie from.

Partitionnement d'une relation (ou groupement)
Clause group by

- ***Syntaxe***

  group by <liste d'attributs>

- ***Principe***
  - partitionnement horizontal d'une relation, selon les valeurs d'un attribut ou d'un groupe d'attributs qui est spécifié dans la clause **group by**
  - la relation est (logiquement) fragmentée en groupes de tuples, où tous les tuples de chaque groupe ont la même valeur pour l'attribut (ou le groupe d'attributs) de partitionnement

Exemple (suite)

- **Q:** *sommes des quantités en stock de chaque produit?*

Attention en **SQL strandard** chaque expression du **select** doit être **monvaluée** par groupe:

- soit l'attribut de groupement
- soit une fonction sur cet attribut
- soit une fonction qui réduit les valeurs d'un groupe à une valeur unique..

Exemple (suite)

- **Attention en SQL standard:**
  - les attributs après le **groupe by** doivent apparaître dans la clause **select**
  - et les attributs dans **select** doivent apparaître dans **group by** ou bien ils doivent être un argument d'une fonction d'agrégation
- Ce n'est pas le cas de Transact-SQL de Sybase, par exemple...

Exemple de requête erronée en SQL standard

```
Select  prod_id, dep_id,
    avg(qte)
from  Stock
group by prod_id
```

Résultat "attendu"

prod_id	dep_id	Avg(qte)
p1	{1,3,5,2}	10.0
p2	{5,6}	11.0
p3	{2,6,3}	13.0
p4	{2}	0

Requête non valide en SQL standard !!!

L'attribut **dep_id** est multivalué par rapport à l'attribut de partitionnement prod_id

La relation résultat n'est pas en première forme normale (à voir l'année prochaine)

Exemple (suite)

Select prod_id, dep_id, avg(qte)

from Stock

group by prod_id

Résultat	prod_id	dep_id	Avg(qte)
p1	1	10.0	
p1	3	10.0	
p1	5	10.0	
p1	2	10.0	
p2	5	11.0	
p2	6	11.0	
p3	2	13.0	
p3	6	13.0	
p3	3	13.0	

Requête valide en Transact-SQL de sybase !!!

Exemple (suite)

- Q: la *somme des quantités en stock de chaque produit dans tous les dépôts sauf le dépôt numéro 5?*

*donc, si la requête comporte une clause where: les tuples ne vérifiant pas la condition sont exclus **AVANT** d'opérer le groupement..*

Exemple (suite)

- *Q: la somme des quantités en stock de chaque produit dans tous les dépôts de Nancy? Trier le résultat sur prod_id?*

*Donc, si le résultat doit être trié, la clause **order by** doit apparaître APRES le group by.*

Partitionnement d'une relation (ou groupement) Clause group by

- **Restriction sur les groupes**
  - application possible d'un critère de restriction sur les groupes obtenus
  - clause **having**
  - l'expression suivant le **having** doit être monovaluée pour chaque groupe

Exemple

- *Quels sont les produits stockés dans plus de 2 dépôts ?*

Exemple (suite)

- *quels sont les dépôts ayant plus de trois produits en rupture de stock ?*

```
Select dep_id, adr
    from Depot
    where dep_id in ( select  dep_id
                from Stock
                where qte <= 0
                group by dep_id
                having count(*) >3)
```

➔*éliminer des dépôts ayant qte >0*

➔*groupement sur dep_id*

➔*comptage des membres de chaque groupe*

➔*éliminer des groupes ayant moins de 4 membres*

Restriction

Clause where restriction sur les tuples d'une relation (pas de fonction d'agrégation)

Clause having restriction sur les groupes d'une relation obtenus par la clause group by (on peut utiliser une fonction d'agrégation)

Donner les produits et les sommes des quantités associés, si aucun produit considéré n'a pas de quantité supérieur ou égal à 12" ?

Exemple (suite)

- On peut comparer à la requête suivante :

```
SELECT prod_id, sum(qte)
FROM Stock
WHERE qte <12
GROUP BY prod_id
```

- qui donne pour chaque **produit** la somme des quantités si la quantité est inférieur à 12 (elle correspond à la requête initiale mais travaillant non pas sur la relation **Stock** toute entière, mais seulement les tuples de **quantité** inférieur à 12).

Prédicat d'existence " exists "

- Syntaxe **exists** <sous-requête>
  - Tester si la réponse à une sous-question est *true* ou *false*:
    - *false* si la réponse est l'ensemble **vide**, et *true* sinon
- Q: *donner l'adresse des depots où est stocké le produit «p1» ?*

- Cette requête peut également s'écrire sans utiliser de quantificateur :

Prédicat d'existence "exists "

Adresse des dépôts où n'est pas est stocké le produit «p1» ?

Traduction de la division

- *"Quels sont les numéros et les adresses des dépôts où sont entreposés tous les produits (ceux connus de la base de données)?"*
- PARAPHRASE EN FRANCAIS

  "Un dépôt est sélectionné s'il n'existe aucun produit qui n'ait pas été stocké par ce dépôt"

  => DOUBLE NEGATION

Exemple

- -

Exemple (suite)

- Pour ceux n'ayant aucun attrait particulier pour la logique, voici une solution :

```
SELECT   d. dep_id, d.adr
FROM     Depot d
WHERE    d.dep_id  IN
         (SELECT dep_id FROM Stock
          GROUP BY dep_id
          HAVING COUNT (*)=
                (SELECT COUNT(*) FROM Produit)
         )
```

- Attention, cette écriture n'est correcte que parce que l'on est sûr (par définition) que l'ensemble des produits d'un entrepôt est inclus (ou égal) dans l'ensemble des produits de la base de données.

Synthèse

(6) select <liste attributs Aj et/ou expressions sur attributs Ap>
(1) from <liste relations Ri>
(2) where <condition sur tuples C1>
(3) group by <liste attributs Ak>
(4) having <condition sur groupes C2>
(5) order by <liste attributs Al>

1) Produit cartésien des relations Ri

(2) Sélection des tuples de (1) vérifiant C1

(3) Partitionnement de l'ensemble obtenu en (2) suivant les valeurs des Ak

(4) Sélection des groupes de (3) vérifiant C2

(5) Tri des groupes obtenus en (4) suivant les valeurs des Al

(6) Projection de l'ensemble obtenu en (5) sur les attributs Aj, avec calcul des fonctions appliquées aux groupes (s'il y en a)

Un exemple complet

- *"Donnez par ordre croissant le nom et la somme des quantités des produits, uniquement si chaque quantité stockée est strictement supérieure à 20?"*

- Requête SQL

```
SELECT p.prod_id, p.nom, Sum(s.qte)
FROM Produit p, Stock s
WHERE p.prod_id = s.prod_id
GROUP BY p.prod_id,p.nom
HAVING  min(s.qte) > 20
ORDER BY p.nom
```

Dictionnaire

Une caractéristiques des SGBD est qu'ils gèrent des «méta-données»

i.e. des données qui décrivent les objets de la base: relation, attributs, vues, utilisateurs connus, mots de passe.

Les métas-données sont elles-mêmes gérées comme une base de données relationnelles, de schéma prédéfini et parfois appelée dictionnaire de SGBD ou catalogue

Dictionnaire de Sybase

Dictionnaire de données

contient les informations relatives à une base des données:

nom de la base, nom des relations, nom et types des attributs..

Les SGDB sybase (comme les autres SGBDR) structure le dictionnaire comme une base de données: on peut interroger cette BD

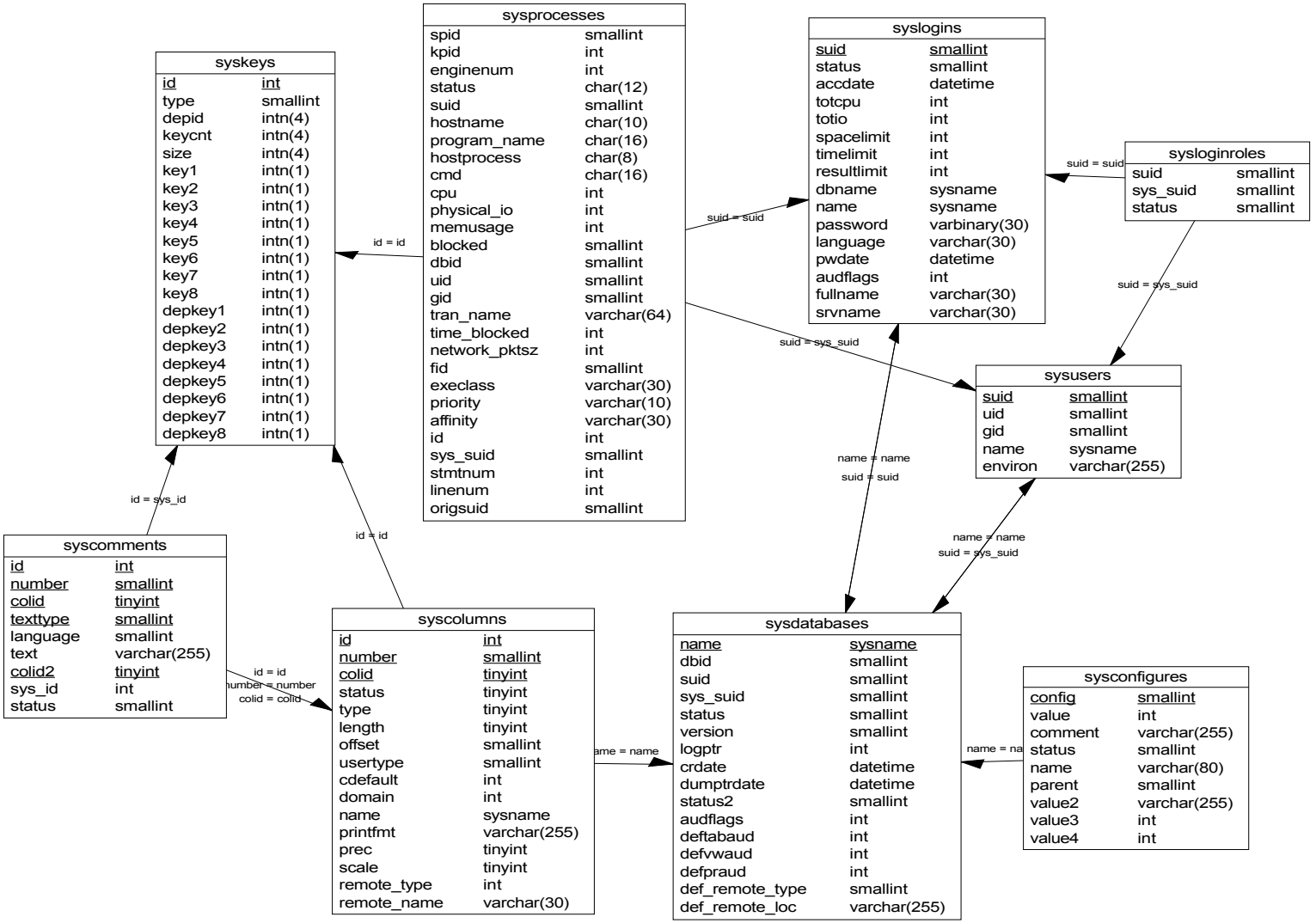
Dictionnaire de Sybase

Sybase gère un dictionnaire fractionné en deux parties:

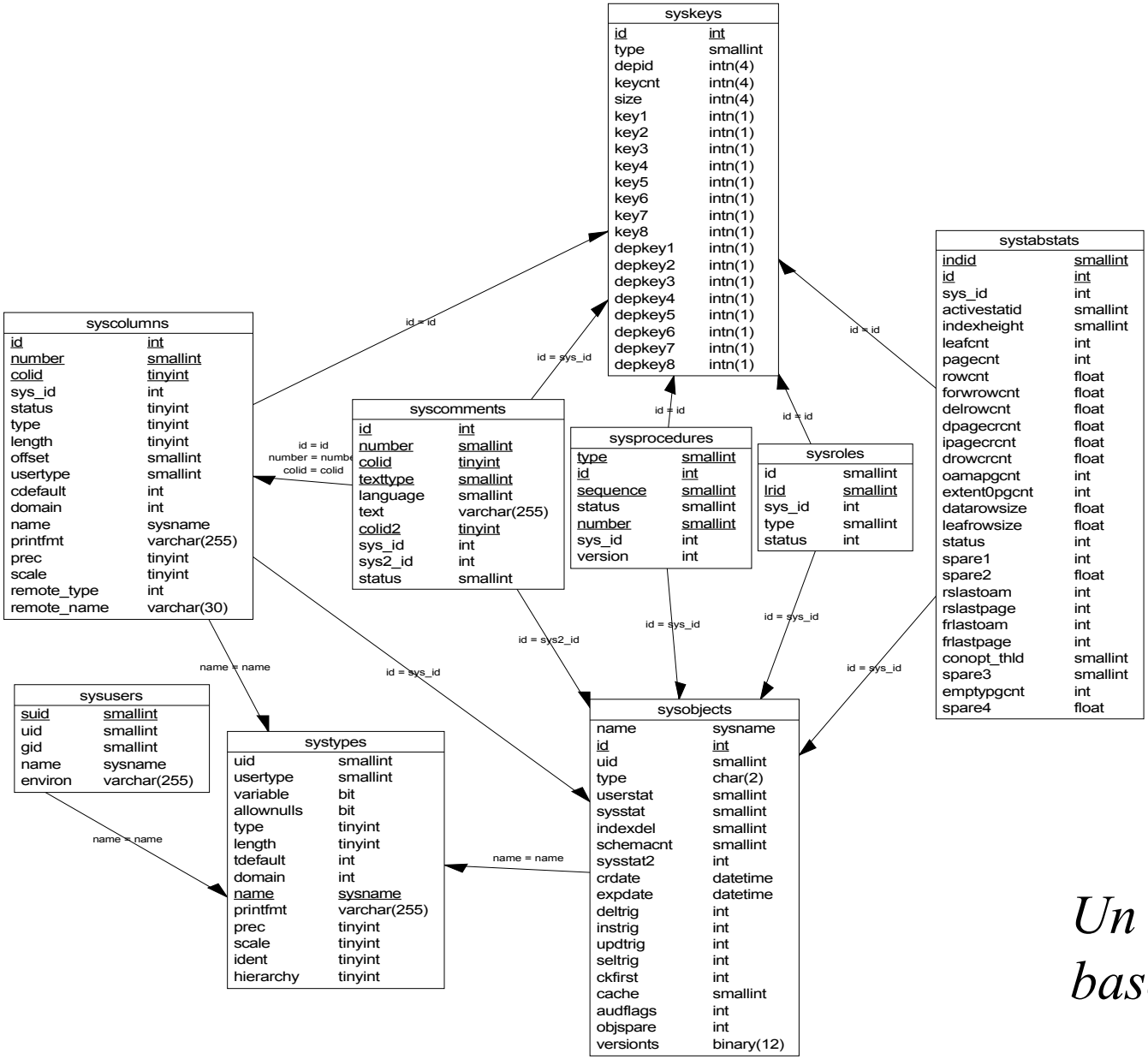
tables communes à toutes les BD (base **master**) : sysservers, sysdatabases, syslogins…)

tables spécifiques à chaque BD (base **Model**):

sysusers, sysobjects, syscolumns, sysprocedures,..



Un extrait de la base master



Un extrait de la base Model

Dictionnaire d'oracle

- Vue utilisateur: USER_xxx
  - Informations sur les objets dont l'utilisateur (courant) est propriétaire
    - USER_TABLES
    - USER_TAB_COLUMNS
    - USER_VIEWS
    - USER_CONSTRAINTS
    - USER_TRIGGER
- Exemples:

```
Select Table_Name  from USER_TABLES;
Select Column_Name, Data_Type
      from USER_TAB_COLUMNS
      where Table_Name='PRDOUIT'
```

Dictionnaire d'oracle

- Vue de la forme ALL_xxx:
  - Informations sur les objets auxquels a accès l'utilisateur.
    - ALL_TABLES
    - ALL_VIEW
- Vues de la forme BDA_xxx
  - Informations dont seul un administrateur peut bénéficier (tous les objets de tous les utilisateurs, config de la base, droits des différents utilisateurs)
- V$: les vues dynamiques de la base
- GV$: les vues de la base en cluster

Dictionnaire d'oracle

- La vue DICTIONARY
  - Ensemble des vues BDA,USER, ALL, V$ du dictionnaire (avant Oracle9i)
  - SQL> desc dictionnary

Nom	NULL?	Type
Table_NAME		varchar(30)
Comments		varchar2(4000)

- Select table_name, comments from dictionnary
- Where table_name like '%FILE%';
- Select * from Dict[IONARY]
  - Toutes les tables et vues accessibles par l'utilisateur.
  - Select * from TAB;
  - Select * from COL;

Nommage des objets

- Notion de propriétaire: l'utilisateur qui crée l'objet.
  - Le propriétaire d 'un objet peut donner des droits (la commande « grant » sous Sybase..)
- Nommage
  - **Relation (table)**
    - **[[database.]owner.].table_name (ex: iup2.skaf.Produit)**
    - table_name si elle m'appartient (ex: Produit)
    - **owner.table_name** sinon (ex: lui.Produit)

Nommage des objets (suite)

Attribut (colonne)

si pas d ambiguïté: nom de l 'attribut donné lors de la création de la relation (ex: prod_id)

sinon ***table_name***.*column_name* (ex: Produit.prod_id)

ou *aliasTable.* *column_name* (ex: p.prod_id)

Nom des attributs dans la relation résultat

par défaut, ceux de la relation sinon, renommage

select prod_id ``Numero produit ''

from Produit

La définition de schémas de relations en SQL

- **create table**: création de schéma de relation
- **alter table**: modification de schéma de relation
- **drop table**: suppression de schéma de relation

Rappel:

schéma de relation R(U,P)

U: {<attribut, domaine>}

domaine: type de données

P: {contraintes d'intégrité}

ce qu'on peut exprimer dépend du SGBD

Création d'un schéma de relation

create table nom-table (nom-attr1 type-attr1 [……], nom-attr2 type-attr1 [……])

- Le type d'un attribut

  Chaque SGBD possède des domaines qui lui sont propres

En sybase 11.0.x, (voir sybooks, pour plus de détail..)

- Type de base
  - **int** (entier signé sur 4 bytes), **smallint** (entier signé sur 2bytes), **tinyint** entier non signé sur 1byte)
  - **float**(précision) (sur 8 bytes, la précision dépendant de la machine sur laquelle est installé le système)
  - **real** (sur 8 bytes..)
  - **char**(longueur) (chaine de caractères de longueur fixe, au plus 255 caractères)
  - **varchar**(longueur) (chaine de caractères de longueur variable, au plus 255 caractères)
  - **datetime** (la date de base est 1er janvier 1990….)
  - **money….**
- Types définis par l'utilisateur (user defined types, sp_addtype)

Types définis par l'utilisateur

Définition : sp_addtype typename,
phystype[(length)| (precision[,scale])]
[, "identity" |nulltype]

Exemple: sp_addtype tid, "char(6)" , "not null"

info sur un type: sp_help typename

Exemple: sp_help money
sp_help tid

supprimer: sp_droptype typename

Exemple: sp_droptype tid

Contraintes d'intégrité

- Une contrainte d'intégrité est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD

- Quelles contraintes
  - **non nullité** des valeurs d'un attribut
  - **unicité** de la valeur d'un attribut ou d'un groupe d'attributs
  - **check ou validité** restreint les valeurs à insérer dans une table
    - Sur un attribut
    - Sur un ensemble d'attributs …

Contraintes d'intégrité

- **clé primaire** (un attribut ou un groupe)
  - quand : insert or update
  - Quoi faire : rejet de l'opération
- **intégrité référentielle**
  - Dans la 2ème table on référence: unique ou primar key
  - Quand: insert, delete, update
  - Quoi faire:
    - Default policy: rejet des modifications
    - Cascade policy: propager les modifications (delete, update)
    - Set-Null Policy: mettre une valeur nulle lors de (delete, update)
  - On peut choisir entre les différentes stratégies

intégrité référentielle

CONSTRAINT nomContr

FORIGN KEY listeAttributs REFERENCES nomTable(listeAttributs)

[ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]

[ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]

Modifications des constraintes

- Alter table Produit drop constraint Idestcle;
- Alter table Produit add constraint Idestcle primary key(prodid);

assertions

- Contrainte sur le schéma:
  - create assertion <toto> check <condition>
  - Tuple based check: seulement les attr de la relation
    - Create asssertion beaucoup_prod check

```
(not exists (select * from Produit p, Stock s
                where p.prod_id = s.prod_id
            and  qte < 10000 ));
```

assertions

- L’assertion précédente dans la table Stock:

Exemple de création de schémas de relation

- Contraintes d'intégrité:
  - pu est obligatoire dans Produit
  - les numéros de dépôts sont compris entre 1 et 100
  - les numéros des produits commencent par p..
  - CI référentielles existent entre Produit, Stock, Depot

Modification de schémas de relations

- Commande: alter table
  - ajouter de nouveaux attributs, contraintes
  - modifier de définition d ’attributs
- Possibilité de:
  - ajouter d’attributs, de contraintes (add)
  - supprimer de contraintes (drop constraint)
  - modifier de valeurs par défaut d’attributs (replace)
- Restriction:
  - On ne peut pas supprimer un attribut..
  - Un attribut ajouté à l’aide d’alter table n’est pas visible dans les procédures qui font (select *)
    - drop procedure, puis recréer la procédure

Exemples

alter table ***Produit***

add couleur varchar(20) NULL

- Ajout d'une colonne à la relation Produit
- Pour les lignes existantes, on affecte la valeur **NULL**

alter table ***Stock***

add st_id numeric(5,0) identity

- ajout d'une colonne identité à la relation Stock
- Pour les lignes existantes déjà dans la table, le serveur affecte des valeurs en séquence**.**

alter table ***Stock***

drop constraint stock-cons

- suppression d'une contrainte de la relation Stock

alter table ***Stock***

replace qte defaut null

- modification de la valeur par défaut de l'attribut qte

Suppression d'un schéma de relation

- drop table nom-relation [,nom-relation,…]

- Exemple:

  drop table Stock

  drop table Produit, Depot

  – suppression du **schéma** de la relation
  – suppression des **tuples** de la relation
  – suppression des **indexes** de la relation

Mise à jour d'une relation

- Insertion d'un ensemble de tuples
  - un seul tuple
  - plusieurs tuples
- Suppression d'un ensemble de tuples
- Modification d'un ensemble de tuples

Insertion de tuples dans une relation

- Insertion d'un seul tuple

insert into Produit values (100, 'vis', 23)

insert into Produit (prod_id, nom) VALUES (100, 'vis')

- Insertion d'un ensemble de tuples

Ex: **create table** Vis
(prod# int, pu float)

insert into Vis
select prod#, pu
from produit
where nom = 'vis '

➡

on peut insérer dans une table le résultat d'une clause **select**

Modification de tuples dans une relation

```
update  Produit
set  pu = 88
where prod_id =150

update  Produit
set  pu = pu * 1.1
where prod_id =150

update Stock
set qte = qte + 10
where prod_id in
    (select prod_id
     from Produit
     where nom = ‘ vis ’)
```

Suppression de tuples dans une relation

Delete [from] *nom-relation [,nom-relation,..]*

[where] *qualification*

- *Supprimer tous les tuples de Produit?*

  delete Produit

  on supprime tous les tuples mais le schéma de la relation existe toujours..
- *Supprimer le produit de numéro 150?*

  delete from Produit
  where prod_id= 150
- *Supprimer les produits de numéro <9 ou >12?*

  delete from Produit
  where prod_id < 9 or prod_id > 12

- *Supprimer les produits stockés au dépôt numéro 5 ?*

Synthèse

- SQL est un langage standard de définition de données LDD (create table,..)
- SQL est un langage standard de manipulation de données LMD (select…)
- Chaque SGBD implante sa propre «version» de SQL
- à suivre !!!

Plan

- Création d'une base de données
- Destruction d'une BD
- Vues et relations temporaires
- Les indexes et les clusters
- Les extensions procédurales de SQL

Création d'une BD

- Avoir le droit (l'administrateur système peut donner la permission: grant ..)..
- utilisateur de la base **master**

- **Syntaxe (simple)**:

  create database base_name

  – Ex: - use master
    – create database newpubs
- En général, l'administrateur crée les bases puis il transfère la propriété à d'autres (sp_changedbowner)

Create database

Create database databae_name

[**on** {default | database_device} [=size]

[,database_device [=size]..]

[**log on** database_device [=size], [database_device[=size] …]

[**with** override]

[**for load**]

On: permet de spécifier la localisation et éventuellement la taille des données..

Journal ou log d'une BD = trace des opérations effectuées sur la BD (pour la reprise)

Create database

- Exemple:

  create database newpubs /* unités et volume par défault */

  create database newpubs

  on pubsdata = 3

  log on pubslog = 1

Create database

- Lors de la création (dans la base master), il y a:
  - vérification de l'unicité du nom;
  - vérification de la disponibilité de l'unité (device) où sera stockée la base (nom de *device* ou fichier UNIX);
  - attribution d'un numéro d'identification;
  - enregistrer la nouvelle base dans la table: **sysusages**
  - ajout d'un n-uplets dans la table **sysdatabases**
  - copie de la base **modèle** dans l'espace réservé de la nouvelle base..

Destruction d'une BD

Syntaxe: drop database database_name

Destruction d'une base de données et de tous ses objets

Extension de « sizes » de BD

alter database databae_name
[**on** {default | database_device} [=size]
[,database_device [=size]..]
[**log on** database_device [=size], [database_device[=size]…]
[**with** override]
[**for load**]

Ex: alter database newpubs

on pubsdata = 2, /* augmenter l 'espace réservé à la base newpubs de 2MB sur le device pubsdata */

Les vues

- Une **vue**: c'est une relation **virtuelle.**
  - ses instances ne sont pas stockés physiquement;
  - ils sont calculé ...
- Définie par:
  - une requête portant sur des relations ou des vues existantes
  - le schéma de la vue est la liste des projections
- Pourquoi ?
  - Répéter un ensemble d'opérations sur un ensemble de données
  - Visualiser la BD sous différents angles...
  - Fournir un mécanisme de sécurité..

Gestion des vues

- **Définition d'une vue : create view** < nom vue>

  as < requête d'interrogation select >

- **Suppression d'une vue: drop view** < nom vue >
- **Interrogation d'une vue**: toujours possible
  - identique à l'interrogation d'une relation de base
- **Mise à jour d'une vue**
  - Non définie dans de nombreux cas
  - Syntaxe identique à la mise à jour d'une relation de base (*si la mise à jour sur la vue est autorisée*)

Les vues: exemples

create view produits-chers
as select * **from** produit **where** pu > 100

create view produit_tres_chers (numero-p,nom-p)
as select prod_id,nom
from produit_chers **where** pu >300

Différence entre vue et relation temporaire (1)

- **Vue**

- **Relation temporaire**

- **Interrogation**

Produit

Prod_id	nom	pu
p1	vis	100
p2	vis	200
p3	xy	300

Différence entre vue et relation temporaire (2)

- **Mise à jour de la relation Produit**

Produit

#prod	nom	pu
p1	vis	100
p2	vis	200
p3	xy	300
p5	***FL***	***400***
p6	***MM***	***99***

- **Interrogation:**

Création de Vue sous ORACLE

Create [or replace] [FORCE | NOFORCE]

View Nom_VUE [(NOM_ALIAS, ..)]

As sous_Requête [with {Check OPTION [CONSTRAINT Nom_CONSTRAINT] | READ OLNY}] ;

- FORCE: Création de la vue même en présence d'une erreur (table n'existe pas)
- NOFORCE: Option par défaut.
- Sous_Reqte toutes les clauses d'un ordre SELECT à l'exception des clause ORDER BY
- Check option: vérifier que les mises à jour ou les insertions ne produisent pas des lignes qui ne feront pas partie de la sélection de la vue.
- READ ONLY: interdit tout modification de données en utilisant le nom de la vue dans un ordre INSERT, UPDAT, ou DELETE.

Mise à jour

- Utiliser INSERT, UPDAT, DELETE dans une vue.
- Restrictions:
  - La vue doit être construite sur une seule table.
  - L'ordre select de la vue ne comporte ni jointure, ni GROUP BY.
  - Les colonnes résultant de l'ordre SELECT doivent être des colonnes réelles d'une table de base et non des expressions.
  - La vue contient toutes les colonnes ayant l'option NOT NULL de la table de base.

Les index et les clusters

- Index sur un attribut *A* d'une relation:
  - c'est une structure de données qui permet de trouver de manière efficace les tuples de la relation qui ont une valeur constante pour *A*.
- Film(titre, annee, nomstudio)

```
Select *
 from Film
where studionom=« Disney » and annee =1990
```

- Dans film, il y a 10 000 tuples, 200 sorti en 1990

Les index et les clusters

- Naïve : parcours séquentielle de la relation et de tester la condition pour chaque tuple ….
- Plus efficace: trouver les 200 films de l'année 1999 et tester si le nom de studio est « Disney »????
  - Create index AnneeIndex on Film(année) ;
- Beaucoup plus efficace: trouver directement les tuples qui vérifient les conditions (multiattributes indexs) .. ..
  - Create index keyindex on Fiml(titre, année) .

Les index et les clusters

- La création d'un index ne fait pas parti de SQL standard mais …

  Create index AnneeIndex on Film(année) ;
- Index sur plusieurs attrs:

  Create index keyindex on Film(titre, année) .
- Ajouter un index permet d'accélérer la sélection mais il ralentit la mise à jour (insert, delete, update)
- An aspect très important lors de la conception de la base de données …Un mauvais choix ???

Explication?

- Les relations sont stockées sur plusieurs «disk blocs»
- Le coût principal d'une requête est le nb de «disk blocs» à mettre dans la mémoire central (buffer ..)
- Les index sont stockés sur le disque ..
- Modifier coûte plus cher que sélectionner. Pourquoi ?

Explication ?

- Acteur(titreF, anneeF, actNom)
- Q1: select titreF, anneeF
  from Acteur
  where actNom=n
- Q2 : select actNom
  from Acteur
  where titreF=t and and anneeF=a
- Q3: insert into Acteur values(t, a, n)

Hypothèses

- **Acteur** stocké sur 10 blocs de disque, examiner toute la relation coûte ??
- En moyenne, un acteur apparaît dans 3 films et un film a 3 acteurs…
- 3 accès disque pour trouver les 3 tuples d'un acteur ou d'un film en cas d'index sur actNom ou sur le couple (titreF, anneF), sans index il faut 10..
- Pour modifier l'index:
  - 1 accès pour lire l'index
  - 1 accès pour écrirel'index (en cas d'insertion)
- Insertion sans index:
  - 1 accès pour lire la bloque ou on va placer les données
  - 1 accès pour écrire

Coût de requêtes

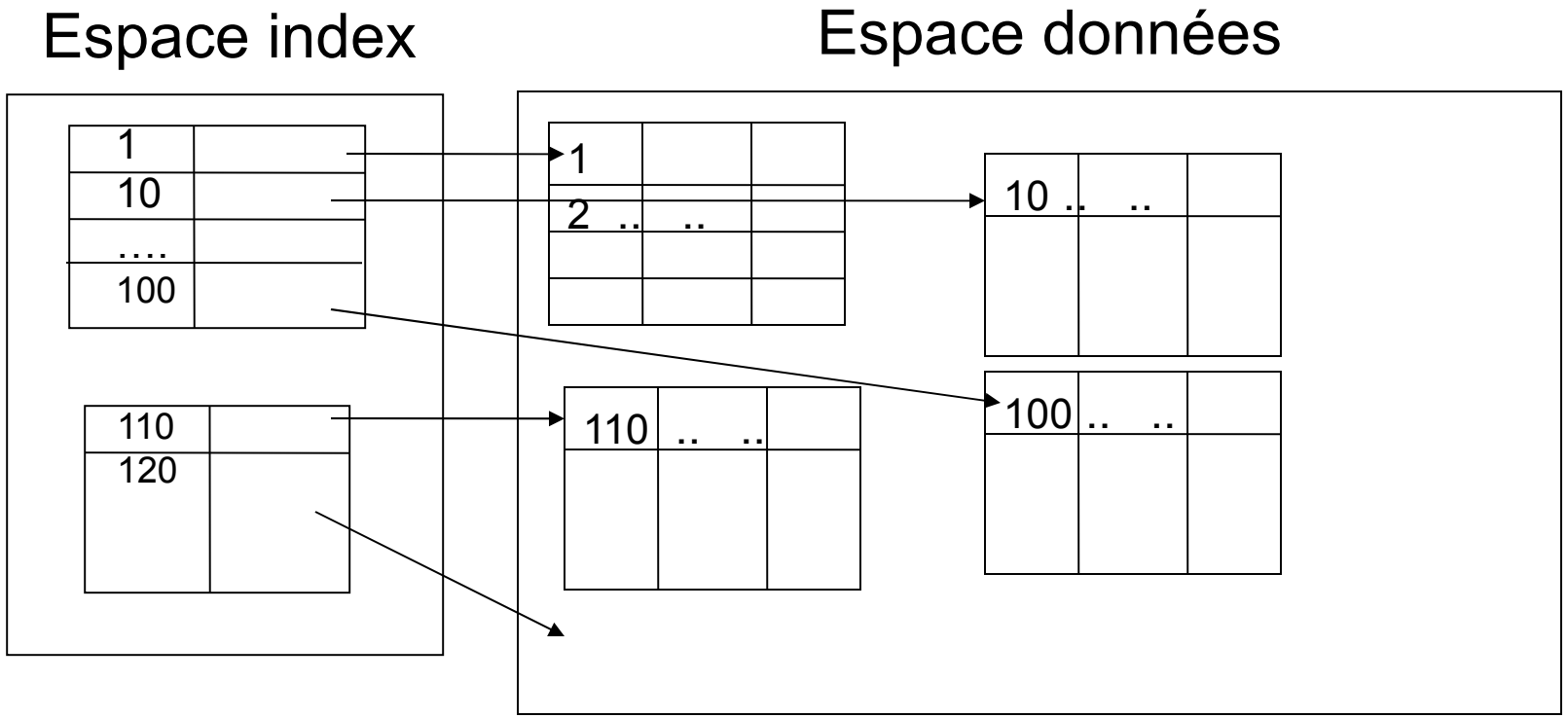
Action	Sans index	Nominde x	titreF ind	Les deux
Q1	10	4	10	4
Q2	10	10	4	4
Q3 (I)	2	4	4	6
	2+8p1+8p2	4+6p2	4+6p1	6-2p1-2p2

Les index et les clusters

- **Résumons:**
  - faciliter la localisation des données par serveur SQL.
  - accélérer l'extraction des données ..
  - Une table peut comporter plusieurs index.
  - Les index sont transparents pour l'utilisateur:
    - aucune syntaxe SQL ne permet d'y faire référence lors d'une requête.
- **Trois types d'index:**
  - index primaire
  - index secondaire
  - index de groupement

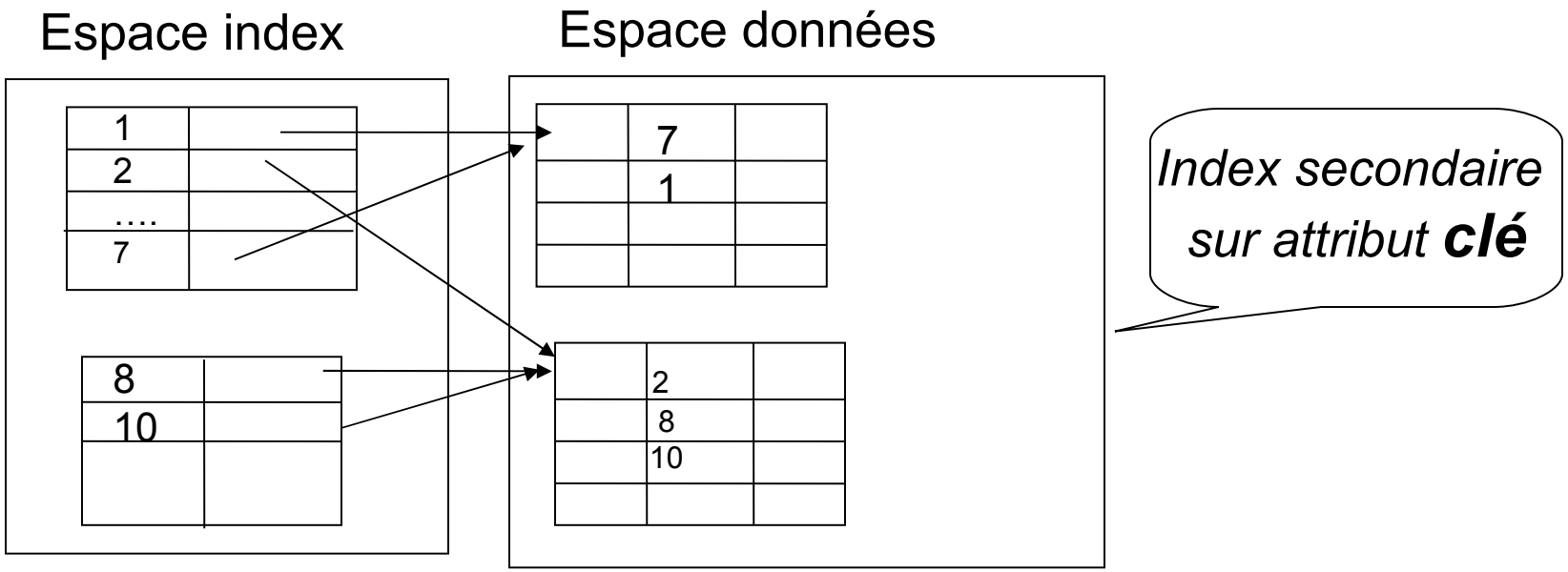
Index primaire

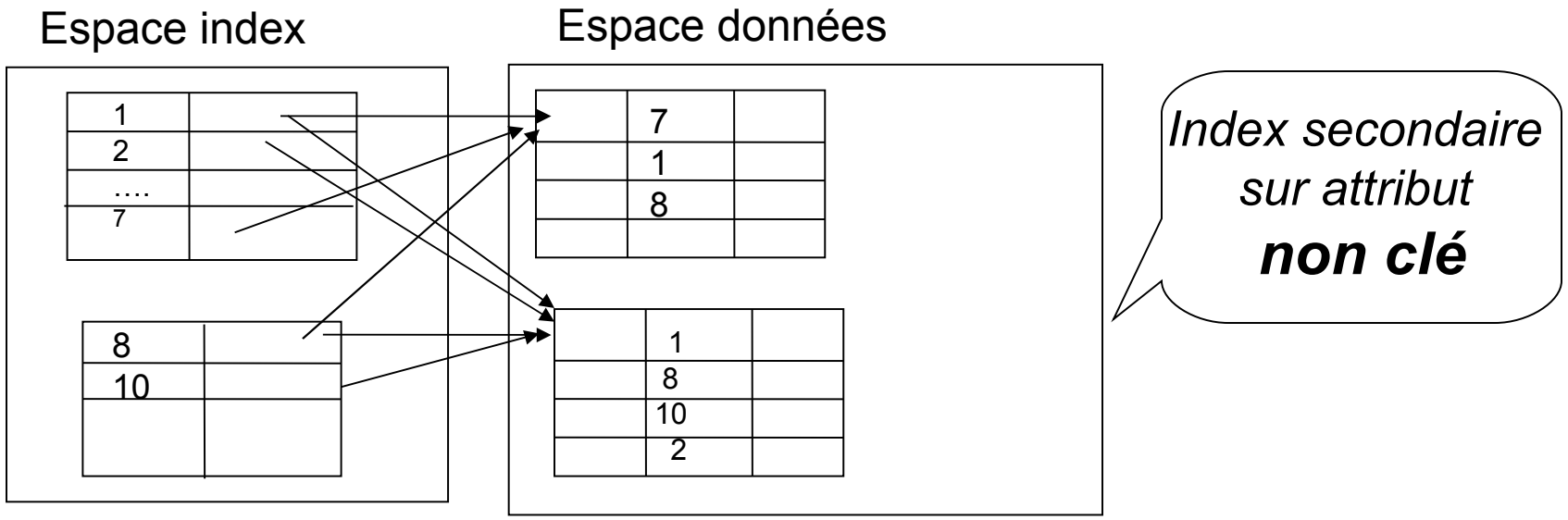
- Construit sur une clé primaire dont les valeurs sont ordonnées dans la relation



Index secondaire

- Construit sur un attribut (non forcement clé) dont les valeurs sont ordonnées dans la relation;
- permet d'établir un ordre logique sur les n-uplets.

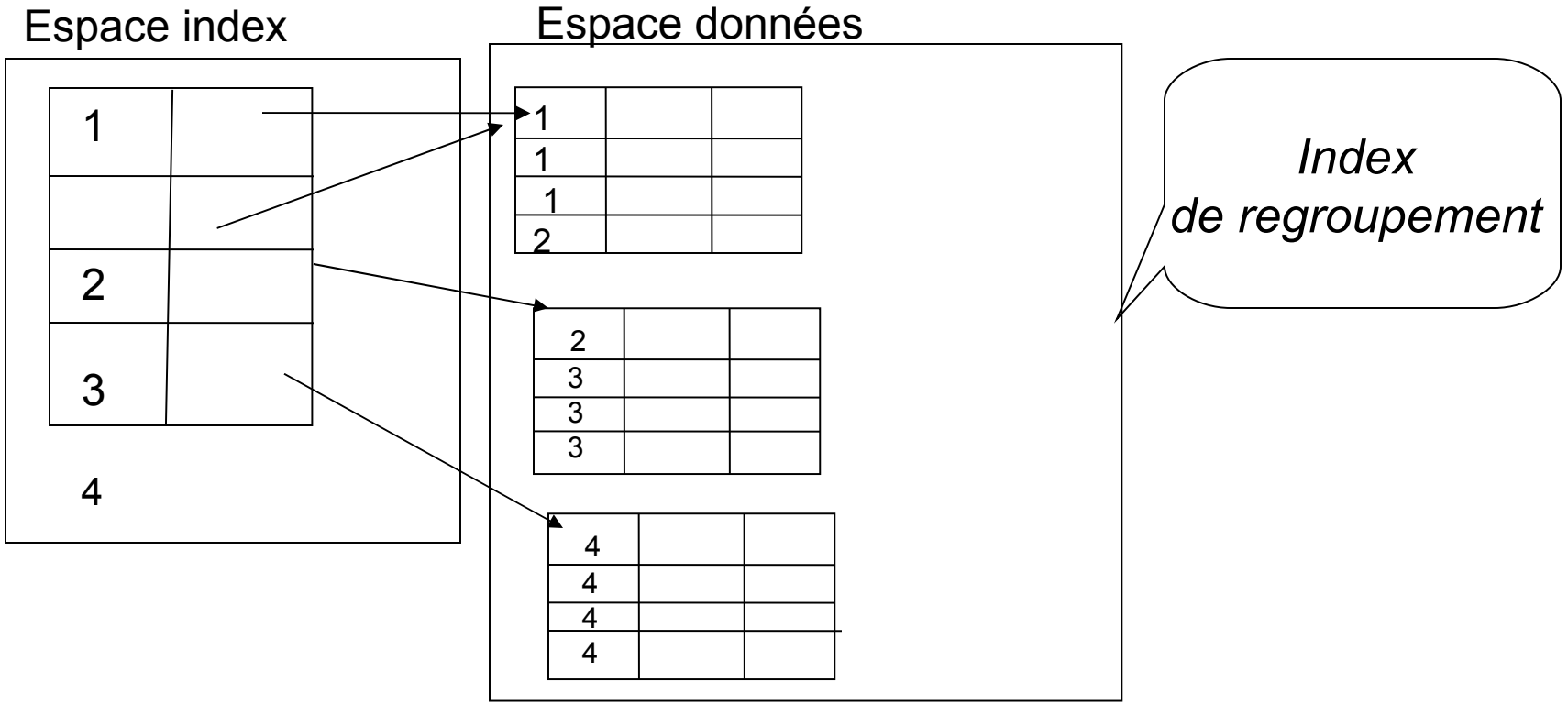

Espace index
Espace données
1
2
….
7
8
10
7
1
2
8
10
Index secondaire sur attribut clé


Espace index
Espace données
1
2
….
7
8
10
7
1
8
1
8
10
2
Index secondaire sur attribut non clé

Index de regroupement physique (index de clustering)

- Construit sur un attribut dont les valeurs sont ordonnées et qui n'est pas forcement clé de la relation.
- Les n-uplets ayant la même valeur pour l'attribut de clustering se retrouvent rangés de façon contiguë.

Index de regroupement



- Ce type d'index est approprié sur les attributs de jointure ! Pourquoi ?
  - « Rapproche » physiquement les n-uplets à joindre et ce en vue de minimiser les transferts entre la base de données et la mémoire central..

Les indexes et les clusters sous Sybase

create [unique] [clustered | nonclustered] index *nom_index* on [[*basededonnées* .] *propriétaire* .] *nom_table* (*nom_colonne* [, *nom_colonne*]...)[with {{fillfactor | max_rows_per_page}= *x,* ignore_dup_key, sorted_data, [ignore_dup_row | allow_dup_row]}] [on *nom_segment*]

- Imposer des contraintes d'unicité ou non:
  - sur la valeur de l'index (unique, ignore_dup_key)
  - sur les n-uplets de la relation (ignore_dup_row | allow_dup_row)
  - clustered: faire correspondre l 'ordre des clés de l 'index avec l 'ordre de stockage physique des n-uplets de la relation..
- Ex: un index sur la colonne *prod_id* de la table *Produit*

create index prod_ind on Produit(prod_id)

Les extensions procédurales de SQL

Dans Sybase TransactSQL

Oracle:PL/SQL «Procedural Language extensions to SQL»

Pourquoi ?

SQL Langage ensembliste ..

langage procédurale: permet de traiter de manière conditionnelle les données retournées par une instruction SQL

Extension procédurale

- Variables et les types
- Les structures de contrôle et les boucles
- Les procédures et les fonctions.
- Curseurs ..
- Exceptions ..

Extension procédurale dans sybase

déclaration de variable

declare @v1 int, @v2 int

Affectation de valeurs à des variables

select @v1 = 100

select @v2 = (select count(*) from produit)

structures de contrôle (conditionnelle, itération)

Sybase: Transact-SQL

- If .. else
- Syntaxe :

  if *expression_booléenne*
  instruction
  [else
  [if *expression_booléenne*]
  instruction]

- Exemple:

while

- Syntaxe

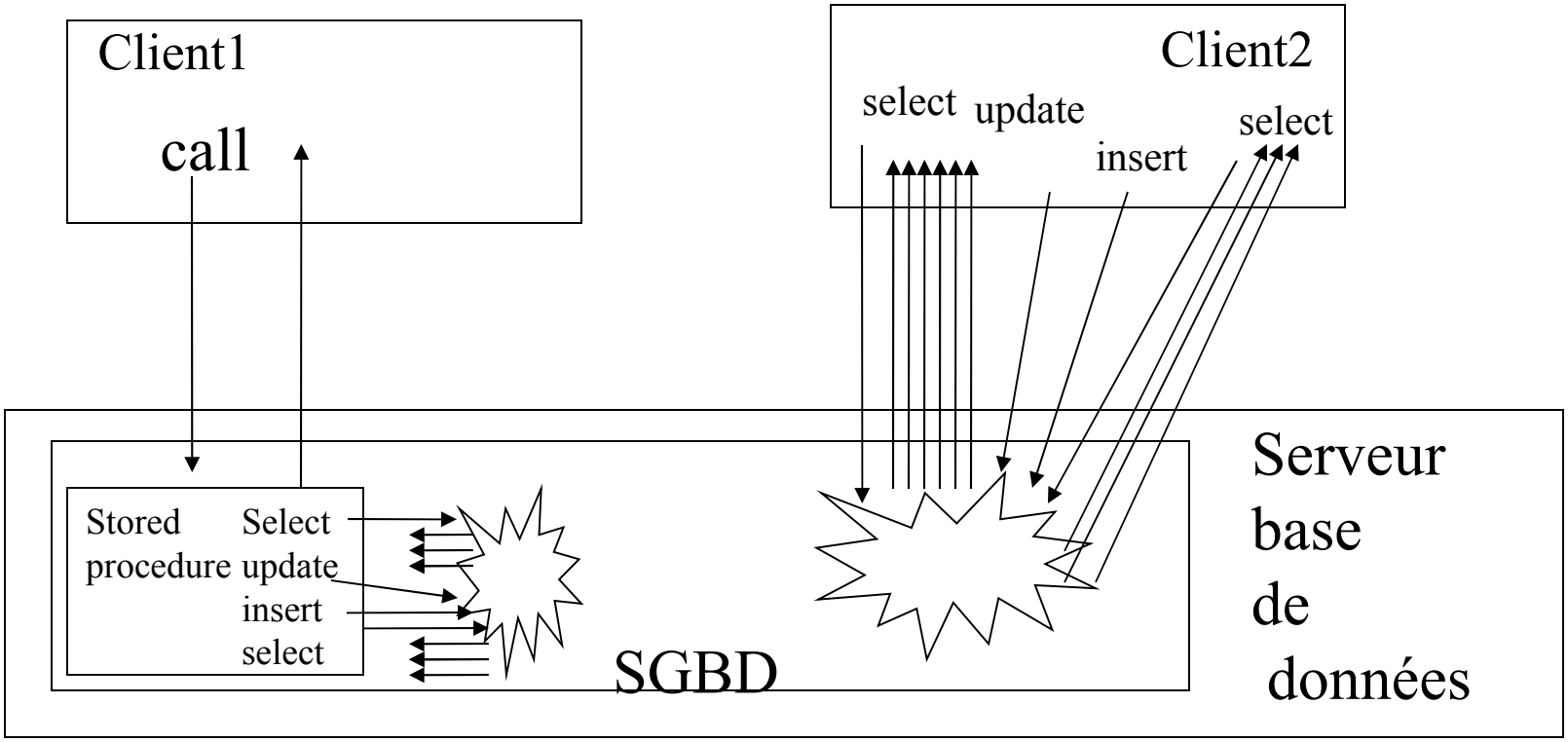
  while *expression_booléenne*
  instruction

- Exemple:

Procédures stockées

- Instructions SQL + le langage de contrôle de flux dans une procédure stockée afin d'améliorer les performances de SQL Server.
- Les procédures stockées sont précompilées.
  - 1ère fois:
    - le processeur de requêtes de SQL Server l'analyse et prépare un plan d'exécution
    - le nom est stocké dans **sysobjects,** text dans **syscomments**
  - La procédure est exécutée selon ce plan stocké: exécution presque instantanément.

Procédures stockées et trafic réseau



Procédures stockées

Déclaration

creat proc[edure] nom_procedure [(@param1 type1, @param2 type2,..)] as requête_SQL

Suppression

drop proc nom_procédure

Exemple:

create procedure namelist

as select name from sysusers

Exécution de procédure:

namelist

execute namelist

exec namelist

Exemple 1

```
create proc prod_info (@id varchar(30) )
     as  select  *
    from Produit
    where prod_id = @id
```

Exécution

```
    prod_info  p1
    ou bien: declare @id varchar(30)
    select @id= « p1 »
    prod_info  @id
```

Display le texte du prod_info :

```
sp_helptext prod_info
```

Exemple 2

```
create proc prod_info (@id varchar(30) = null)
    as
    if (@id is null )
            print « donner un id »
    return
    else select  *
    from Produit
    where prod_id = @id
```

Procédures systèmes

- SQL Server offre une variété de procédures stockées qui sont autant d'outils pratiques pour l'utilisateur. On les appelle des procédures système.
- Les procédures système permettent
  - d'extraire des informations des tables système
  - d'effectuer **l'administration** des bases de données ainsi que d'autres tâches impliquant la mise à jour des tables système

– ***Ex:*** sp_help

sp_helpdb

Remarque : Tous les noms de procédures système commencent par "sp_".

Les curseurs de Sybase

- Manipuler chaque ligne individuellement d ’un **select**
- Qu’est-ce que c’est un curseur?
  - Un **curseur** est un nom symbolique associé à une instruction **select** Transact-SQL.
  - Deux informations importantes:
    - Un ensemble de lignes résultant de l'exécution de la requête associée au curseur.
    - la position du curseur : pointeur sur une ligne.

Curseur de Sybase

- A quoi ça sert?
  - Accès aux donner ligne par ligne.
- Comment l'utiliser?

  Déclare, ouvrir, parcourir et fermer.

  - Déclaration du curseur

  declare *nom_curseur* cursor for *instruction_select*

Exemple

```
declare prod_crsr cursor for
    select  prod_id, nom
    from Produit
   order by nom

open prod_crsr
fetch prod_crsr
set cursor rows 3 for prod_crsr
fetch prod_crsr
close newauthors_crsr

deallocate cursor newauthors_crsr
```

Extraire des lignes

- Exemple:
  - fetch produit_crsr into @id, @name, @price

  Attention:

  - les variables @id, @name et @price doivent être déclarées avant :-)
  - et une compatibilité de type entre les variables et les colonnes du jeu de résultats du curseur.

Variables globales

Variables globales utiles:

- pour connaître l'état du curseur: *@@sqlstatus*

  0 : indique que l'instruction **fetch** a abouti.

  1: indique que l'instruction **fetch** a généré une erreur

  2 : Indique que le jeu de résultats ne comporte plus de données (le curseur est positionné sur la dernière ligne du jeu de résultats).

- Exemple

```
select @@sqlstatus
```

Variables globales

– Pour contrôler le nombre de lignes parcourues par le curseur à un moment donné : *@@rowcount*

Exemple

```
select @@rowcount
```

– Obtention de plusieurs lignes avec chaque commande *fetch*

set cursor rows *nombre* for *nom_curseur*

Exemple:

```
set cursor rows 3 for produit_crs
```

Utilisation de curseurs dans des procédures stockées

titres(title_id,tite,total_sales,...)

titreauteur(au_id,title_id,...)

Question:

pour un auteur donné, donner l'ensemble des titres des livres qu'il a écrits ainsi que le nombre total d'exemplaires vendus pour chaque titre ?

Comment faire?

```
create procedure au_sales (@author_id id)  as
     declare @title_id tid
     declare @title varchar(80)
     declare @ytd_sales int
     declare @msg varchar(120)
     declare author_sales cursor for
          select ta.title_id, t.title, t.total_sales
          from titleauthor ta, titles t
          where ta.title_id = t.title_id  and ta.au_id = @author_id
     open author_sales
     fetch author_sales    into @title_id, @title, @ytd_sales
     if (@@sqlstatus = 2)
           begin
             print "Nous ne vendons pas les livres de cet auteur."
              close author_sales
            return
          end
     while (@@sqlstatus = 0)
          begin
            if (@ytd_sales = NULL)
               begin
                 select @msg = @title +  " pas de ventes cette année."
                 print @msg
                 end
          else if (@ytd_sales < 500)
            begin
               select @msg = @title + " peu de ventes cette année."
               print @msg
            end
           else if (@ytd_sales < 1000)
               begin
                select @msg = @title + " ventes médiocres cette année."
                print @msg
               end
          else     begin
               select @msg = @title +" ventes importantes cette année."
               print @msg
          end
               fetch author_sales into @title_id, @title, @ytd_sales
       end
     if (@@sqlstatus = 1) exec error_handle
     close author_sales
      deallocate cursor author_sales
     return
```

Les extensions procédurales de SQL

Dans Sybase TransactSQL

Oracle:PL/SQL «Procedural Language extensions to SQL»

Pourquoi ?

SQL Langage ensembliste ..

langage procédurale: permet de traiter de manière conditionnelle les données retournées par une instruction SQL

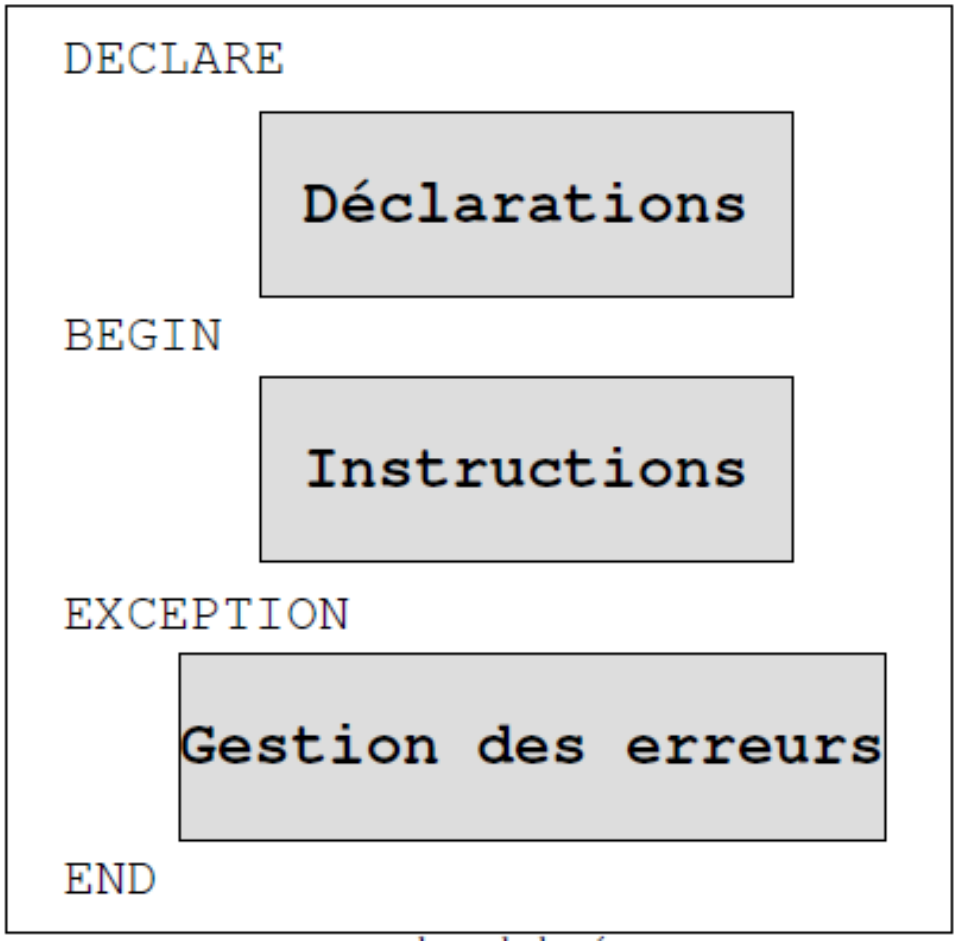
Extension procédurale

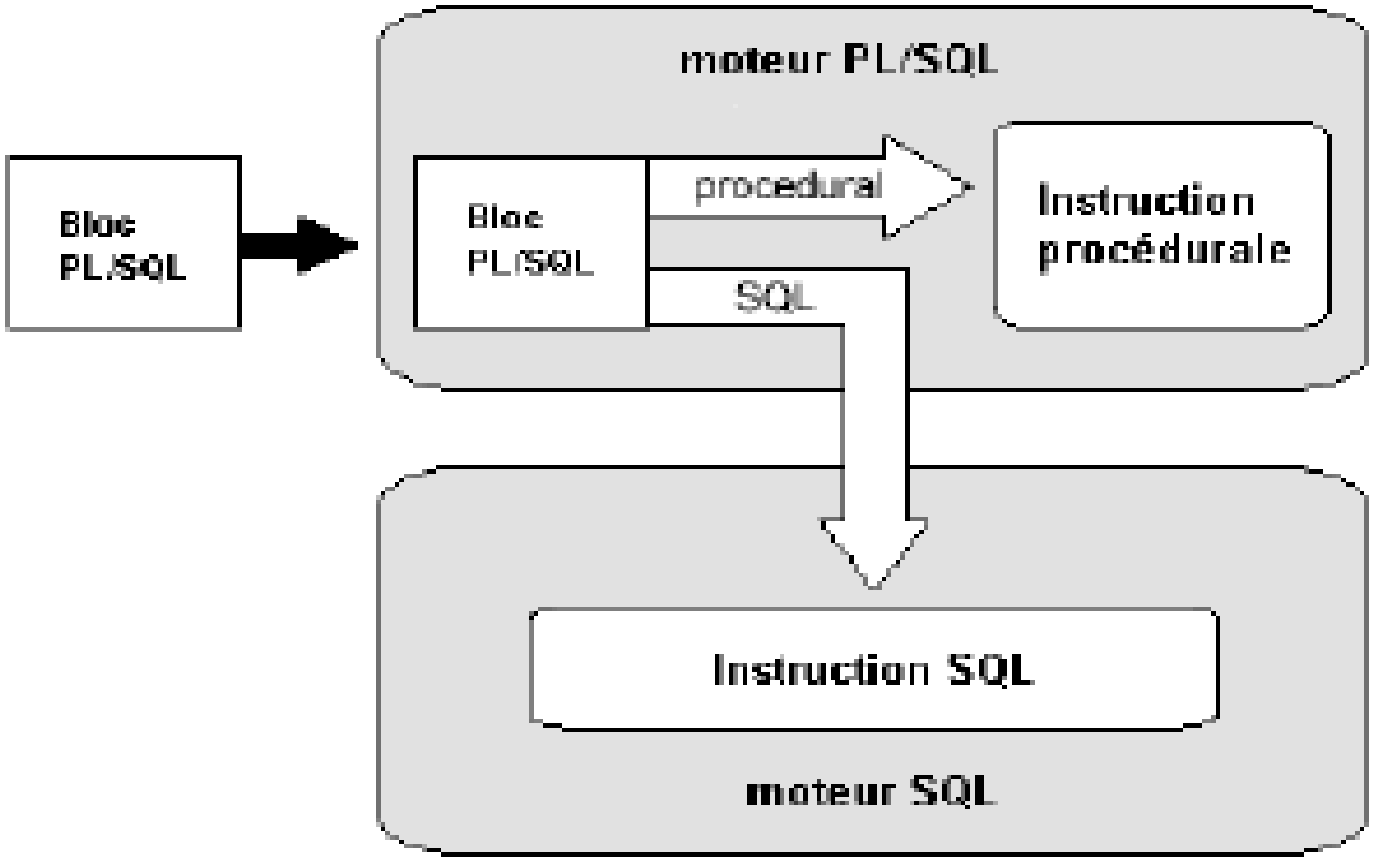
- Variables et les types
- Les structures de contrôle et les boucles
- Les procédures et les fonctions.
- Curseurs ..
- Exceptions ..

Extension procédurale dans Oracle

PL/SQL version 10g

Structure d'un Bloc d'instructions PL/SQL





Syntaxe d'un bloc PL/SQL

Exemple

```
SQL >
begin
delete  produit where prod_id ='p1';
end;

PL/SQL procedure successfully completed.
```

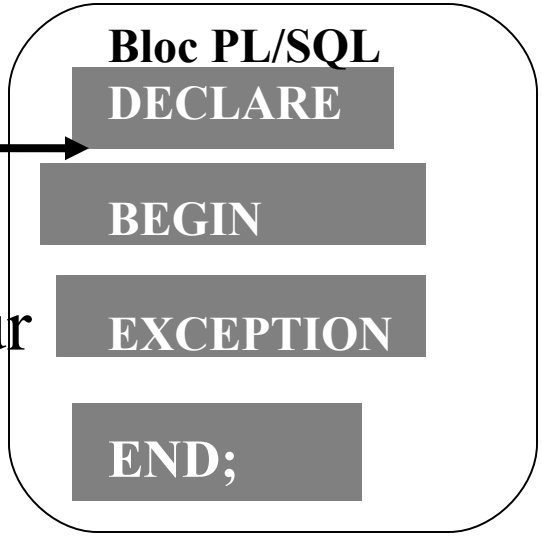
Sortie à l'écran

```
> SQL
set serveroutput on
begin
dbms_output.put_line('Bonjour utilisateur id :  '|| UID  || 'nom est:  ' ||
    USER || 'on est:  ' || to_char(sysdate,'dd month yyyy'));
end;
```

- Bonjour utilisateur id : 69nom est: SKAFon est: 11 january 2006
  PL/SQL procedure successfully completed.

Déclarations

- Variables et constantes
  - Types de base
  - types définis par l'utilisateur
- Curseurs
- Exceptions



- Pas de différences entre minuscules et majuscules pour les identifiant

Types de base

<type>::= CHAR | BOOLEAN | DATE | NUMBER(t,d) | VARCHAR2(s)

avec

- **t** nombre total de chiffres
- **d** nombre de chiffres après la virgule
- **s** nombre maximum de caractères dans la chaîne

Syntaxe déclaration

- Nom_variable [CONSTANTE] TYPE {NOT NULL}[{DEFAULT | :=} VALEUR];

```
SQL> set serveroutput on
declare
User_id  number  := UID;
nom Varchar2(30) :=USER;
trouve boolean;
begin
dbms_output.put_line('Bonjour utilisateur id:  '|| User_id  || 'nom est:' || nom || 'on est' || to_char(sysdate,'dd month yyyy'));
end;

Bonjour utilisateur id: 69nom est:SKAFon est11 january 2006
PL/SQL procedure successfully completed.
```

Exemple

```
SQL>
declare
tcon constant number := 5;
begin
dbms_output.put_line('CONSTANT:  ' || tcon );
end;

CONSTANT: 5
    PL/SQL procedure successfully completed.
```

Exemple

```
SQL> declare
tcon constant number := 5;
begin
tcon:= 10;
dbms_output.put_line('CONSTANT:  ' || tcon );
end;
ERROR at line 4:
    ORA-06550: line 4, column 1:
    PLS-00363: expression 'TCON' cannot be used as an
    assignment target
    ORA-06550: line 4, column 1:
    PL/SQL: Statement ignored
```

Variables basées

- Faire référence à une entité existante
  - %TYPE: référence une colonne d'une table ou une variable

Nom-variable {nom_table.colonne |nom_varaible}%TYPE

Exemple:

```
Sql> Declare
    date_commande Commande.date_com%TYPE;
…..
```

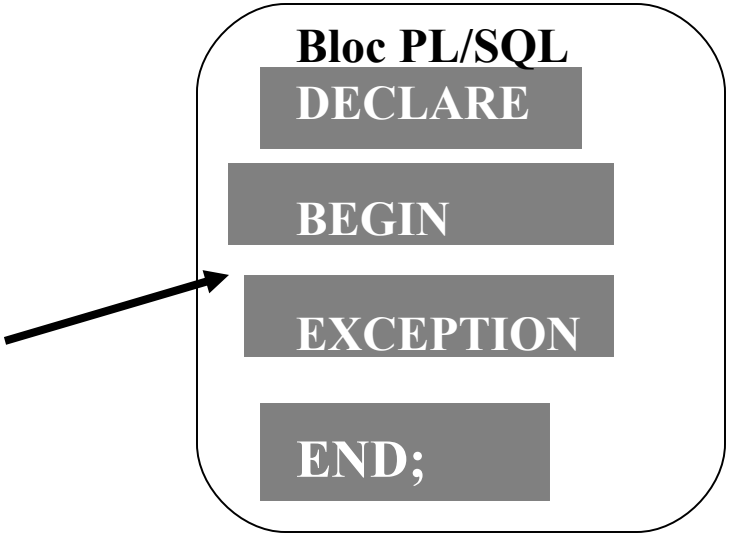
Variables basées

%ROWTYPE:

Nom_variable {nom_table |nom-varaible}%ROWTYPE;

```
Sql> declare
      unProduit Produit%ROWTYPE;
Begin
    unProduit.prod_id := 'p1';
    unProduit.nom := 'cryon';
    unProduit.pu := 200;
dbms_output.put_line(unProduit.prod_id  ||  unProduit.nom   ||
    unProduit.pu);
End;
p1cryon200
    PL/SQL procedure successfully completed.
```

Instructions



- Requête SQL
- Affectation
  - Deux façons
- Instruction de contrôle conditionnel
- Instruction de contrôle itératif
- Gestion des curseurs

Syntaxe

- Les **instructions** sont séparées par des ;
- Les commentaires se mettent entre /* mes commentaires */ ou après **--** mes commentaires
- **Affectation** : := …. Ou bien **select into** ..
- Exemple:

```
declare
unProduit Produit%ROWTYPE;   /*Une variable */
Begin
Select * into unProduit from produit where prod_id ='p2';
dbms_output.put_line(unProduit.prod_id  ||  unProduit.nom   ||  unProduit.pu);
End;
```

```
p2toto200
    PL/SQL procedure successfully completed.
```

Conditionnelle

if condition **then** instructions;
[elseif condition **then** instructions; ..]
[else
instructions; ..]
END IF;

Expressions de comparaison

- AND, OR, NOT
- = (égalité), <, >, <=, >=, !=(différent)
- is NULL

Exemples conditionnelles

```
Sql > begin
if  MOD(UID,2) <> 0  then
dbms_output.put_line(UID || 'est un nombre impaire');
end if;
end;
69est un nombre impaire
     PL/SQL procedure successfully completed.

Sql>
begin
if  MOD(UID,2) <> 0  then
dbms_output.put_line(UID || 'est un nombre impaire');
else
dbms_output.put_line(UID || 'est un nombre pair');
end if;
end;
```

Traitements itératifs

- Loop
- while loop
- For loop

LOOP

[<<NOM_BOUCLE>>]
 LOOP <instructions> END LOOP ;

```
Sql>
declare
 x number := 0;
Begin
loop
x:=x+1;
end loop;
End;
```

- Boucle infinie !

LOOP

[<<NOM_BOUCLE>>]
LOOP
  <instructions>
EXIT [<<NOM_BOUCLE>>] WHEN
  <condition>;
<instructions>
END LOOP ;

```
declare
 x number := 0;
Begin
<<incrt_boucle>>
Loop
      x:=x+1;
        dbms_output.put_line('la valeur de x est dans la boucle:' || x);
      exit incrt_boucle when x>5;
 end loop;
dbms_output.put_line('la valeur de x à la fin est :' || x);
```

```
la valeur de x est dans la boucle:1
      la valeur de x est dans la boucle:2
      la valeur de x est dans la boucle:3
      la valeur de x est dans la boucle:4
      la valeur de x est dans la boucle:5
      la valeur de x est dans la boucle:6
      la valeur de x à la fin est :6
      PL/SQL procedure successfully completed.
```

WHILE

[<<NOM_BOUCLE>>]
WHILE <condition> LOO P
<instructions>
END LOOP [NOM_BOUCLE]

```
declare
 x number := 0;
Begin
<<boucle_while>>   /* pas obligatoire */
while x <= 5 loop
     x:=x+1;
       dbms_output.put_line('la valeur de x est dans la boucle:' || x);
      end loop boucle_while;
dbms_output.put_line('la valeur de x à la fin est :' || x);
end;
la valeur de x est dans la boucle:1
     la valeur de x est dans la boucle:2
     la valeur de x est dans la boucle:3
     la valeur de x est dans la boucle:4
     la valeur de x est dans la boucle:5
     la valeur de x est dans la boucle:6
     la valeur de x à la fin est :6
     PL/SQL procedure successfully completed.
```

FOR LOOP

[<<NOM_BOUCLE>>]

FOR indice IN [REVERSE] inf..sup LOOP

<instructions>;

END LOOP [NOM_BOUCLE];

Exemple

```
declare
 x number := 0;
Begin
for  x in 1..6 loop
       dbms_output.put_line('la valeur de x est dans la boucle:' || x);
      end loop;
dbms_output.put_line('la valeur de x à la fin est :' || x);
end;
```

```
la valeur de x est dans la boucle:1
     la valeur de x est dans la boucle:2
     la valeur de x est dans la boucle:3
     la valeur de x est dans la boucle:4
     la valeur de x est dans la boucle:5
     la valeur de x est dans la boucle:6
     la valeur de x à la fin est :0
     PL/SQL procedure successfully completed.
```

Intégration des requêtes

- Requêtes retournant une seule ligne
- Requêtes retournant plusieurs lignes (curseurs)

Requête à ligne unique

Mot clé **into**

- Exemple:

```
declare
unProduit Produit%ROWTYPE;   /*Une variable */
Begin
Select * into unProduit from produit where prod_id ='p2';
dbms_output.put_line(unProduit.prod_id  ||  unProduit.nom   ||
unProduit.pu);
End;
```

Requêtes à lignes multiples

- Utilisation d'une structure de données appelée **CURSEUR**
- Qu'est-ce que c'est un curseur?
  - Un **curseur** est un nom symbolique associé à une instruction **select**
- A quoi ça sert?
  - Manipuler chaque ligne individuellement d 'un **select**
  - Accès aux donner ligne par ligne.

Les curseurs

- Comment l'utiliser?

Déclare, ouvrir, parcourir et fermer.

- Déclaration du curseur

CURSOR nom_cursor [[nom_arg type
:=valeur_defalut[……])] is reqte;

Exemple

- Sql> declare
  - Cursor c_produit_sans is

    select * from produit order by nom;
- Sql> declare
  - Cursor c_produit_avec(p_nom Produit.nom%TYPE := ‘Titi’) is

    select * from Produit where nom =p_nom;

Ouverture d'un curseur

- Open nom_curseur [(valeur_arg[..,..])
- Exemple
  - Open c_produit_sans ;
  - Open c_produit_avec('crayon') ; -- la valeur de param p_nom est affectéé à 'crayon'
  - Open c_produit_avec(p_nom =>'crayon') ;

Traitement des lignes d'un curseur

- Fetch nom_curseur into nom_enregistrement | nom_variable[..]};
- Status d'un curseur:
  - %FOUND: attr de type booléen…vari si exécution correcte de la reqte SQL
  - %NOTFOUND
  - %ISOPEN
  - %ROWCOUNT
- La syntaxe de consultation:
  - Nom_curseur%ATTRIBUT
- Exemple : c_produit_sans%ISOPEN

```
Sql> Declare
unProduit Produit%ROWTYPE;
Cursor  c_prod is select  prod_id, nom,pu  from Produit
      order by nom;
begin
open c_prod;
if c_prod%ISOPEN then
 dbms_output.put_line('la valeur de rowcount: ' || c_prod%ROWCOUNT);
loop
 FETCH c_prod  INTO unProduit;
 EXIT WHEN c_prod%NOTFOUND;
  IF unProduit.pu > 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || ' cher');
end if;
dbms_output.put_line('la valeur de rowcount: ' || c_prod%ROWCOUNT);
End loop;
end if;
close c_prod;
End;


la valeur de rowcount: 0
      la valeur de rowcount: 1
      toto200 pas cher
      la valeur de rowcount: 2
      PL/SQL procedure successfully completed.
```

Avec while:

```
Set serveroutput on;
Declare
unProduit Produitt%ROWTYPE;
Cursor  c_prod is select  prod_id, nom,pu  from Produit   order by nom;
begin
open c_prod;
 dbms_output.put_line('la valeur de rowcount  ' || c_prod%ROWCOUNT);
 FETCH c_prod  INTO unProduit;
while c_prod%FOUND loop
  IF unProduit.pu> 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || ' cher');
end if;
dbms_output.put_line('la valeur de rowcount: ' || c_prod%ROWCOUNT);
FETCH c_prod  INTO unProduit;
end loop;
close c_prod;
End;
```

Les boucles FOR et les curseurs

- Curseur est ouvert et fermé automatiquement
- La variable unProduit est définie automatiquement comme une variable de type enregistrement en lecture seule.

```
 Sql> Declare
Cursor  c_prod is select  prod_id, nom,pu  from Produit
      order by nom;
begin
for unProduit in c_prod loop
  if  unProduit.pu > 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || '  cher');
  end if;
End loop;
End;
```

Les boucles FOR et les curseurs

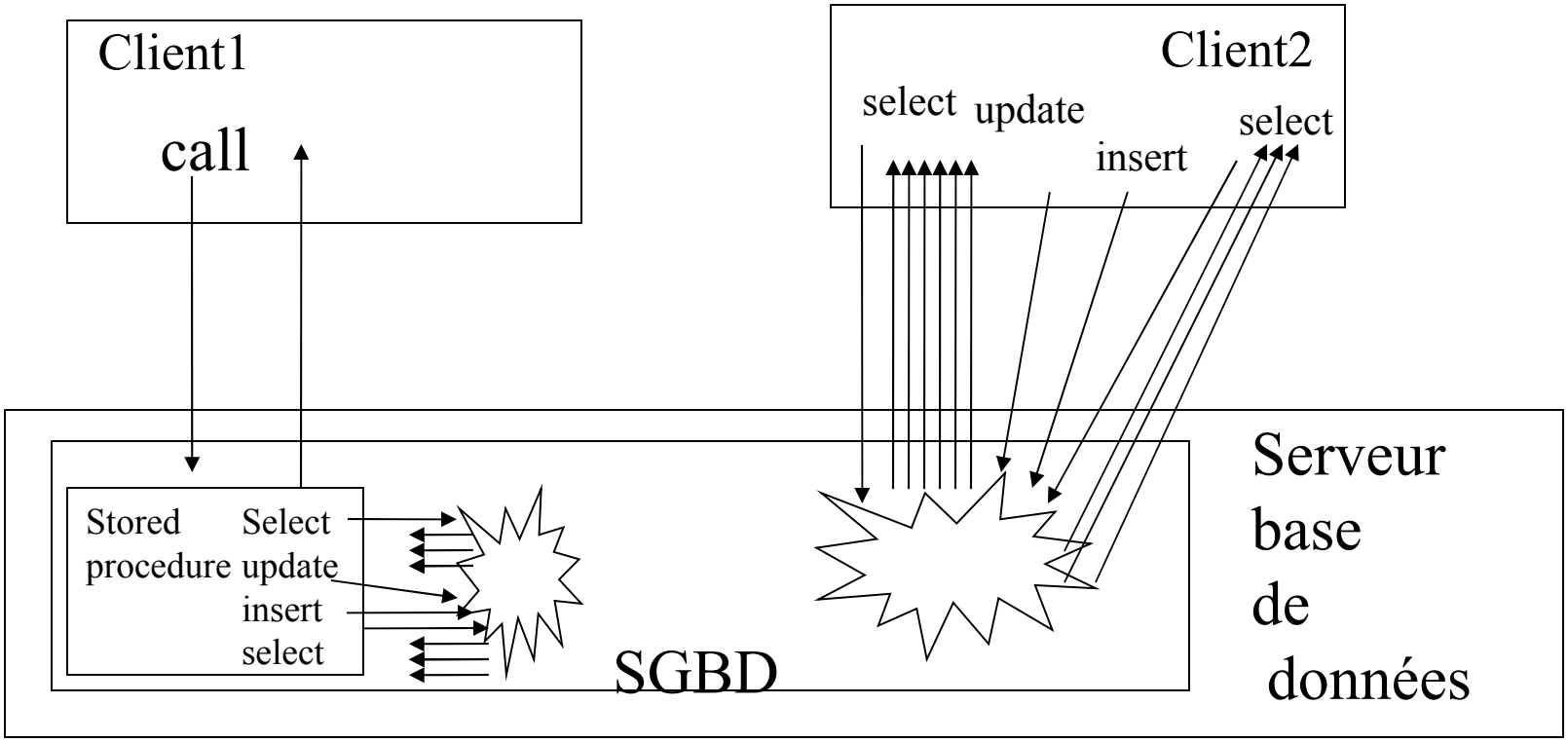
- Sans déclarer le curseur:

```
begin
for unProduit in (select prod_id, nom,pu  from Produit  order by nom)  loop
  if  unProduit.pu > 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || '  cher');
  end if;
End loop;
End;
```

Procédures stockées

- Instructions SQL + le langage de contrôle de flux dans une procédure stockée afin d'améliorer les performances de SQL Server.
- Les procédures stockées sont précompilées.
  - 1ère fois:
    - le processeur de requêtes de SQL Server l'analyse et prépare un plan d'exécution
    - le nom, text sont stockés dans le dictionnaire
    - La procédure est exécutée selon ce plan stocké: exécution presque instantanément.

Procédures stockées et trafic réseau



Procédures stockées

- Déclaration

Create or replace <nom procedure>(paramètres) {is |as}

<bloc pl/sql>

End [nom_procedure] ;

- Suppression
  - drop procedure nom_procédure ;

```
select * from produit
        p2  toto  200
        p1  A3  10
show errors;
create or replace procedure addProduit
is
Begin
 insert into Produit values('p3','toto',100);
end addProduit;
No errors.
     Procedure created.
```

Appel :

```
SQL> call addProduit();
        p2  toto  200
        p1  A3  10
        p3 toto 100
Sql >
begin
addProduit();
end;
```

En cas de problème

« Attention : procédure créée avec erreurs de compilation »

> show errors

LINE/COL ERRO R

XX/YY m essage d ’erreur

Procédure avec paramètres formels

- Les paramètres sont IN, OUT ou IN OUT
- Exemple:

```
create or replace procedure addProduit ( prod_id in char(2),
    nomp in  varchar2(20) Produit.nom%type , pup float)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
```

Warning: Procedure created with compilation errors.

- Attention: dans la déclaration de procédure, une contrainte de longueur ne peut pas être appliquée aux arguments char, varchar.. De même qu'une contrainte de précision aux arguments number

Exemple

```
create or replace procedure addProduit
   ( prod_id in Produit.prod_id%type, nomp in
   Produit.nom%type , pup in Produit.pu
   %type)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
```

Exemple

```
create or replace procedure
    updatePuProduit( montant in produit.pu%type)  is
begin
Update  Produit set pu = pu +montant ;
Montant := 200 ;
end updatePuProduit;
```

- Errors .. Pourquoi ?

Exemple

```
create or replace procedure
   updatePuProduit( montant in Produit.pu
   %type)  is
begin
Update  Produit set pu = pu +montant ;
end updatePuProduit;
```

Exemple

```
create or replace procedure nombreProduits(nb_produits out  number)  is
begin
Dbms_output.put_line('la valeur est :' || nb_produits);
Select count(*) into nb_produits from Produit;
end nombreProduits;
```

Appel:

```
declare
res number;
begin
nombreProduits(res);
dbms_output.put_line(res);
end;
```

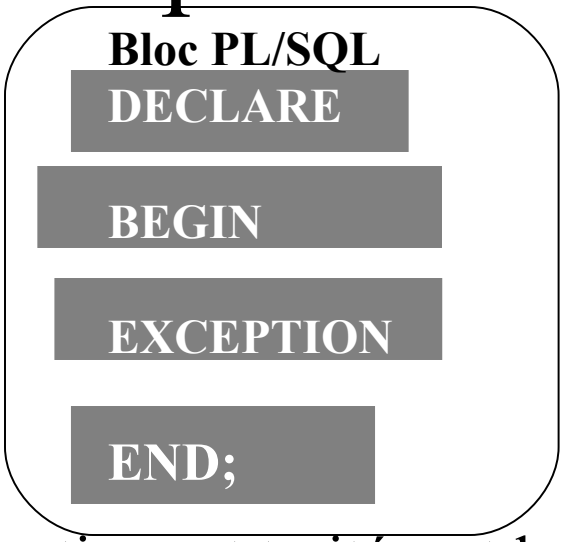
Functions

```
create or replace function nbProduits return number is
vnb number;
Begin
select count(*) into vnb from produit;
return vnb;
end nbProduits;
```

Appel:

```
begin
dbms_output.put_line('nombre de produits:'|| nbProduits());
end;
```

Gestion des exceptions



- Les exceptions provoquent

l'arrêt du déroulement normal

du bloc PL/SQ L.

- Si un gestionnaire

d'exception a été mis en place l'exception est traitée, et le programme peut éventuellement reprendre son cours

- Si aucun gestionnaire n'est mis en place l'exception provoque l'affichage d'un message d'erreur (SQ L*PLUS) et l'arrêt de l'exécution du bloc.

Les types d'exceptions

- 4 types:
  - Les exceptions systèmes nommées :suite à une erreur PL/SQL ou oracle.
  - Exceptions systèmes anonymes: suite à une erreur PL/SQL ou oracle. Il est possible de donner un nom (PRAGMA EXCEPTION_INIT)
  - Les exceptions utilisateur nommées: suite à une erreur dans le code applicatif
  - Exceptions utilisateur anonymes : code compris entre –20 000 et –20 999, déclenche avec RAISE_APPLICATION_ERROR ..

Quelques exceptions prédéfinies

- Dup_val_on_index:
  - essai d'insertion d'une ligne dont la clé existe déjà dans la table
- Invalid_number:
  - conversion d'une chaîne en nombre qui échoue
- no_data_found:
  - aucune ligne retournée par la dernière requête
- Too_many_rows:
  - le dernier select a retourné plus d'une ligne
- Zero_divide:
  - division par 0
- Others:
  - pour récupérer toutes les exceptions non traitées explicitement

Exception

Exception
 when nom_exception [OR nom_exception ..] then
 instructions;
[when others then instructions ;]
End;

Exemple

```
declare
    unProduit Produit%ROWTYPE;   /*Une variable */
    Begin
    Select * into unProduit from produit where prod_id ='p9';
    dbms_output.put_line(unProduit.prod_id  ||  unProduit.nom   ||
    unProduit.pu);
    End;
```

```
declare * ERROR at line 1:
    ORA-01403: no data found
    ORA-06512: at line 4
```

Exemple

```
declare
    unProduit Produit%ROWTYPE;   /*Une variable */
Begin
    Select * into unProduit from produit where prod_id ='p9';
    dbms_output.put_line('Vous ne verrez pas cette ligne ');
 exception
    When NO_DATA_FOUND then
    dbms_output.put_line('pas de produit avec ce numero ');
end;
```

pas de produit avec ce numero
PL/SQL procedure successfully completed.

Packages

- Un package est un regroupement d'un ensemble d'objets (curseurs, variables, procédures et de fonctions) logiquement associés.
- Deux parties distinctes :
  - le corps contient le code et déclarations privées
  - la spécification: interface publique

Avantages

- Protection de données:
  - Public: peut-être référencé à l'extérieur du package
  - Privé: dans le package lui-même
  - Accès aux spécification et masquer le code
- Conception descendante
  - Écrire l'interface avant de coder le corps
  - Compiler l'interface seules ..

Avantages

- Données globales:
  - Ex: une procédure packagée ouvre un curseur, celui-ci reste ouvert et utilisable par d'autres routines packagées durant toute la session…
  - Les variables de packages sont liées à la session et non pas à une transaction
- Amélioration des performances:
  - Appel d'un objet 1ère fois, tout le package est chargé en mémoire…..

Spécification de package

```
Create or replace package gererProduit
As
v_produit   Produit%ROWTYPE;
e_pasDeproduit  EXCEPTION;
function nbProduits return number;
procedure addProduit (prod_id in Produit.prod_id%type,
    nomp in Produit.nom%type , pup in Produit.pu%type);
end gererProduit;
```

Attention: Dans l'interface c'est **public** dans le corps est **privé**

Corps de package

```
Create or replace package body gererProduit
Is
function nbProduits return number is
vnb number;
Begin
select count(*) into vnb from produit;
return vnb;
end nbProduits;
procedure addProduit ( prod_id in Produit.prod_id%type, nomp in
    Produit.nom%type, pup in Produit.pu%type)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
End gererProduit;
```

Exécution: call gererProduit.addProduit('p7', 'titi',300);

Corps de package

```
Create or replace package body gererProduit
Is
function nbProduits return number is
vnb number;
Begin
select count(*) into vnb from produit;
return vnb;
end nbProduits;
procedure addProduit ( prod_id in Produit.prod_id%type, nomp in Produit.nom%type, pup in Produit.pu%type)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
```

Begin

Exception

when Dup_VAL_ON_INDEX then

dbms_output.put_line(‘Erreur le produit est là ’);

End gererProduit;

Exécution: call gererProduit.addProduit('p7', 'titi',300);

Les déclencheurs (Triggers)

- Les contraintes statiques ne permettent pas de exprimer toutes les règles
  - Le prix d'un produit ne peut pas augmenter
- Les triggers sont plus généraux que les contraintes
  - À partir de SQL3 ..
  - Règles : ECA

Trigger (ECA)

- Déclencheur (Trigger)
  - When <Événement> IF <Condition sur BD> THEN <Action sur BD>
- Événement
  - cause l'activation de trigger
- Condition
  - vérifier quand la règle est déclenchée
- Action
  - est exécutée lorsque le trigger est activé et sa condition est vraie.

ECA

- Événement
  - LMD: une modification dans les bases de données: insert, delete, update ..
  - Événement système :
    - lancement d'une instance de la base, …
- Condition
  - Une requête SQL avec une valeur booléenne ..
- Action
  - Une requête SQL

Syntaxe en SQL 3

CREATE [OR REPLACE] TRIGGER NOM_TRIGGER
{ BEFORE | AFTER | INSTEAD OF } EVENEMET
<clause REFERENCING> // optional
WHEN (<condition>) // optional
[FOR EACH ROW] //optional
begin
<action>
End [NOM_TRIGGER]

Définition de Trigger

- Moment d'exécution:
  - Avant l'exécution, après l'exécution ou à la place de l'instruction qui a déclenchée le trigger..

Niveau d'activation

- L'activation au niveau ligne se produit une fois pour chaque ligne manipulée par l'instruction déclenchant
  - FOR EACH ROW: Exécuter l'action pour chaque lignes modifiée .. ("row-level")
- L'activation au niveau instruction se produit une fois soit avant, soit après l'exécution de l'instruction ("statement-level")

L'utilisation de :OLD et :New

- A l'intérieur **d'un déclencheur de niveau ligne**, il es possible d'accéder:
  - :New : Valeurs qui seront insérées ou remplaceront celles d'origine au terme de l'instruction
  - :OLD : Valeurs d'origine de la ligne avant le traitement

- Attention : :OLD pas défini pour «insert» et :OLD pas défini pour « delete »

Clause REFERENCING

- Sert à spécifier un nom différent pour :NEW et :OLD (à personnaliser)

REFERENCING [OLD AS NOM_ANCIEN]
[NEW as NOM_NOUVEAU]

CLAUSE WHEN

- Le corps de déclencheur sera exécutée uniquement pour les lignes qui répondent à la condition spécifiée.
- Seulement pour « Row level » trigger.
  - When CONDITION_DECLENCHEUR

Exemple (1)

- Maintenir une liste de produits dont le prix a augmenté
  - Create table PrixAug(prod_id char(2));

```
CREATE or replace TRIGGER PrixTrigg
after update of pu on Produit
For each ROW
When (NEW.pu > OLD.pu )
    begin
    insert into PrixAug values(:new.prod_id);
    End;
update produit set pu =300 where prod_id ='p2'
```

Suite (1)

```
CREATE or replace TRIGGER PrixTrigg
after update of pu on Produit
Referencing new as newProd old as oldProd
For each ROW
When (newProd.pu > oldProd.pu )
   begin
   insert into PrixAug values(:newProd.prod_id);
   End;
```

Exemple (2)

- Le prix d'un produit ne peut pas augmenter

```
show errors;
CREATE or replace TRIGGER PrixDecroissant
 before update on Produit
For each ROW
When (NEW.pu > OLD.pu )
    begin
    RAISE_APPLICATION_ERROR (num => -20000,
msg => 'pas possible d augmenter le prix');
    End;
```

STOCK(PROD_I, DEP_ID, QTE)

p1 1 0

Produit(prod_id, nom, pu)

p2 toto 200

p3 toto 100

p3 toto 100

p1 A3 10

insert into stock values('P6',2,100)

STOCK(PROD_I, DEP_ID, QTE)

P6 2 100

p1 1 0

Suite (2)

- Produit(prod_id, nom, pu)

p2	toto	200
p3	toto	100
p3	toto	100
p1	A3	10

- update produit set pu =300 where prod_id = 'p2'

  ERROR at line 1:
  ORA-20000: pas possible d augmenter le prix
  ORA-06512: at "SKAF.PRIXDECROISSANT", line 2
  ORA-04088: error during execution of trigger
  'SKAF.PRIXDECROISSANT'

Pas possible !!

```
show errors;
CREATE or replace TRIGGER PrixDecroissant
 after update on Produit
For each ROW
When (NEW.pu > OLD.pu )
   begin
   update produit set pu= :OLD.pu  where prod_id
   =:OLD.prod_id ;
     End;
```

Suite (1)

- update produit set pu =300 where prod_id = 'p2'
- ERROR at line 1:
  ORA-04091: table SKAF.PRODUIT is mutating, trigger/function may not see it
  ORA-06512: at "SKAF.PRIXDECROISSANT", line 2
  ORA-04088: error during execution of trigger 'SKAF.PRIXDECROISSANT'

- Trigger interactions: significant restrictions
- E.g., a trigger action cannot modify the triggering table

Exemple (2)

Au lieu d'utiliser les contraintes « foreign-key » et d'annuler l'insertion dans la table stock pour un produit inconnu, il est possible d'ajouter le produit dans la table produit avec une valeur null comme nom..

```
CREATE or replace TRIGGER TriggStock
 after insert on Stock
For each ROW
When (NEW.prod_id not in (select prod_id from produit ))
    begin
    insert into Produit values (:NEW.prod_id, null);
      End;
```

```
ERROR at line 4:
   ORA-02251: subquery not allowed here
```

Exemples (Oracle: 5)

si je supprime un produit de stock, je dois le supprimer de la table Produit (à condition que ce produit n'existe plus dans le stock)…

```
CREATE  or replace TRIGGER DeleteProd
 after delete on Stock
for each row
begin
delete from produit where prod_id = :OLD.prod_id ;
end;
```

Les prédicats

- INSERTING
- UPDATING
- DELETING

```
CREATE TRIGGER trigger_name
BEFORE DELETE OR INSERT OR UPDATE
ON schema.table_name
Begin
   Case
    when UPDATING then ……..;
   When DELETING then …….
   When INSERTING then ……;
   End case;
```

Exemple (Sybase: 6)

Employe (ID Int, Nom Varchar, Salaire Float)

```
CREATE TRIGGER SalaireCroissant
 ON Employe
For update as
If update(Salaire)
begin
If  (exsits(select * from Employe e, inserted i where i.ID = e.ID
    and i.salaire < e.salaire)
 print "Les salaires ne peuvent décroître »
rollback transaction
```

Plan de cours

- Fonctionnalités d'un SGBD
- Modèle relationnel, SQL
- Optimisation de requêtes
- **Programmer une application: JSQL, JDBC**
- **PLSQL ??**
- **Développement d'une base de données: Projet**
- **Triggers ..**
- **Transactions ..**
- **Bases de données Objet relationnels**

Interface SQL-Langages hôtes

- Pourquoi ?
- Comment ?

Interface SQL-Langages hôtes

- Pourquoi?
  - SQL tout seul est insuffisant pour écrire des applications complètes.
  - Ex: application pour produire les factures, gérer les clients d'une entreprise..
- Problèmes:
  - *Impedance mismatch*
  - «Mariage» des paradigmes

Problèmes

- «Impedance mismatch»:
  - SQL a ses propres types de données;
  - le langage de programmation considéré (C, COBOL, …) a ses propres types;
  - faire correspondre les deux ensembles de types.
- «Mariage» des paradigmes:
  - SQL est un langage déclaratif orienté ensemble (le résultat d'une requête SQL est un ensemble)
  - les langages de programmation classiques sont de nature procédurale c.à.d. Orientés «tuples» ou «un à la fois»
  - faire appel à des structures itératives pour manipuler les ensembles résultants d'une évaluation de requête.

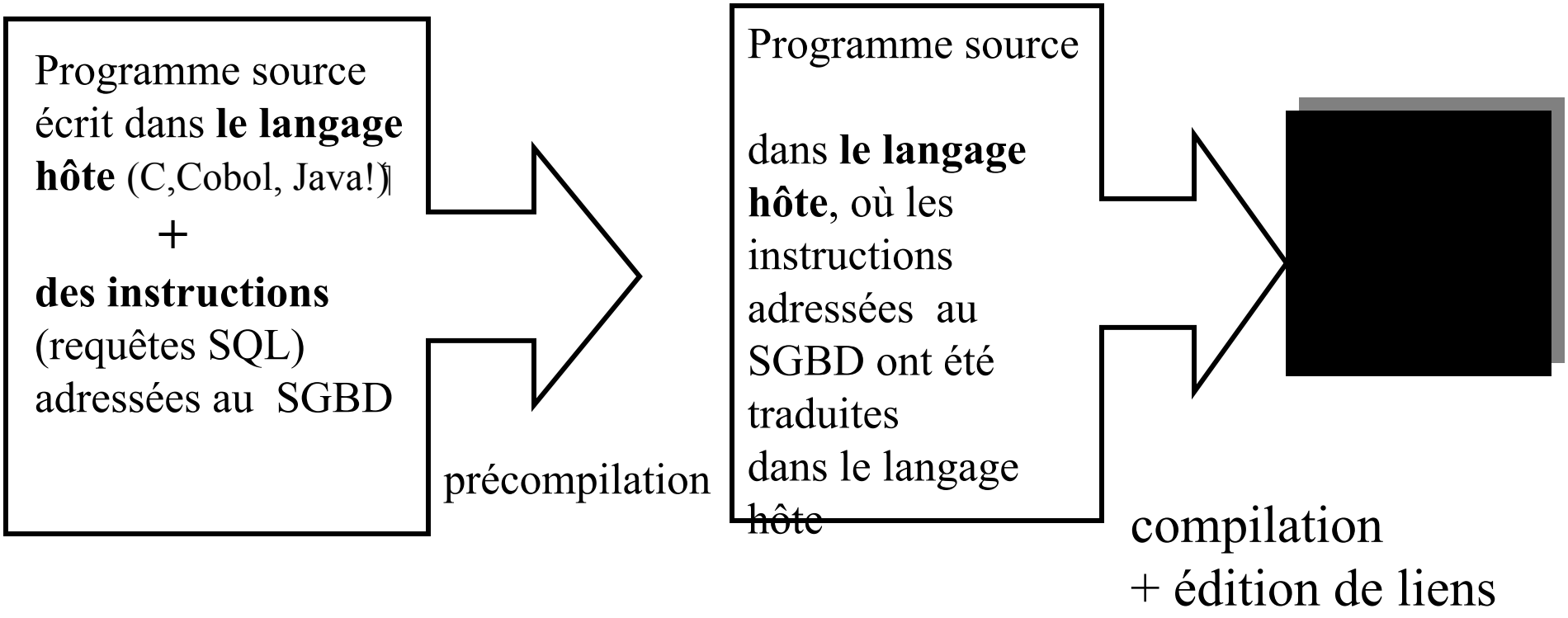
Interface SQL-Langages hôtes

- Deux approches:
  - Approche pré-compilée
  - Approche compilée

Approche pré-compilée

- Embedded SQL,
  - En C ESQL/C,
  - En Java SQLJ..

Approche pré-compilée



- Exemples:
  - dans Oracle V6, PRO*C/C++
  - dans Sybase V11, Embedded SQL (ESQL)

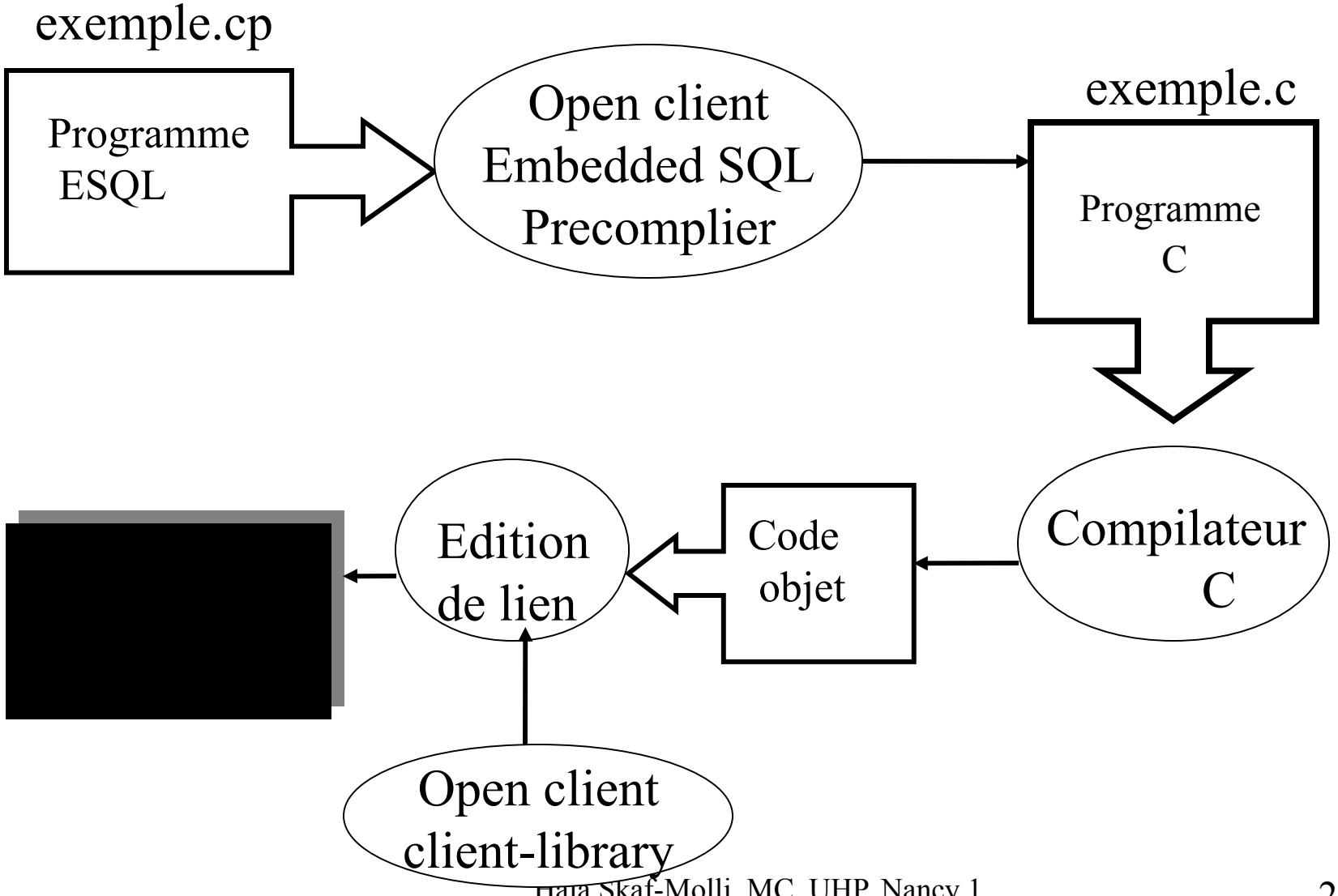
Approche pré-compilée Embedded SQL de sybase

- Sybase Open Client Embedded SQL est un produit permettant à un programmeur d'utiliser de commandes Transact SQL dans un programme écrit dans un langage hôte tel que C ou Cobol.
- Chaque commande Transact SQL est précédée par le mot clé exec sql et termine par «;»
  - Exemple: la table Livre(ID,titre,prix)
    - exec sql update Livre set prix=prix*1.10; // ESQL/C

// **SQLJ**

```
int n;
#sql { INSERT INTO emp VALUES (:n)};
```

ESQL/C

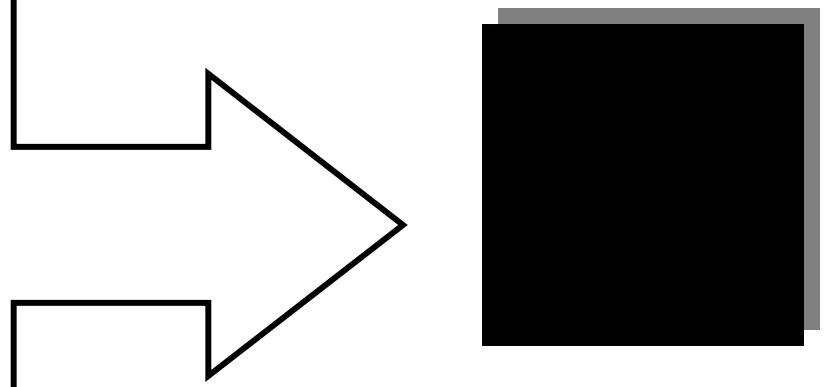

exemple.cp
Programme
ESQL
Open client
Embedded SQL
Precomplier
exemple.c
Programme
C
Compilateur
C
Code
objet
Edition
de lien
Open client
client-library

Approche compilée

Programme source écrit dans **le langage hôte** (C,Cobol, Java!)

+

des fonctions d'une bibliothèques permettant d'écrire des instructions (requêtes SQL) adressées au SGBD



compilation
+ édition de liens

- Exemples:
  - JDBC, dans Sybase V11, DB_library\C

Approche compilée

- JDBC: Java DataBase Connectivity.
- API Java: accéder à n'importe quelle BD à partir d'un programme Java
- Avantages
  - Portabilité
  - Indépendance des constructeurs de SGBD

JDBC

- Etablir une connexions avec une base de données.
- Envoyer des requêtes SQL à cette base
- Traiter les résultats des requêtes

Programming With Database

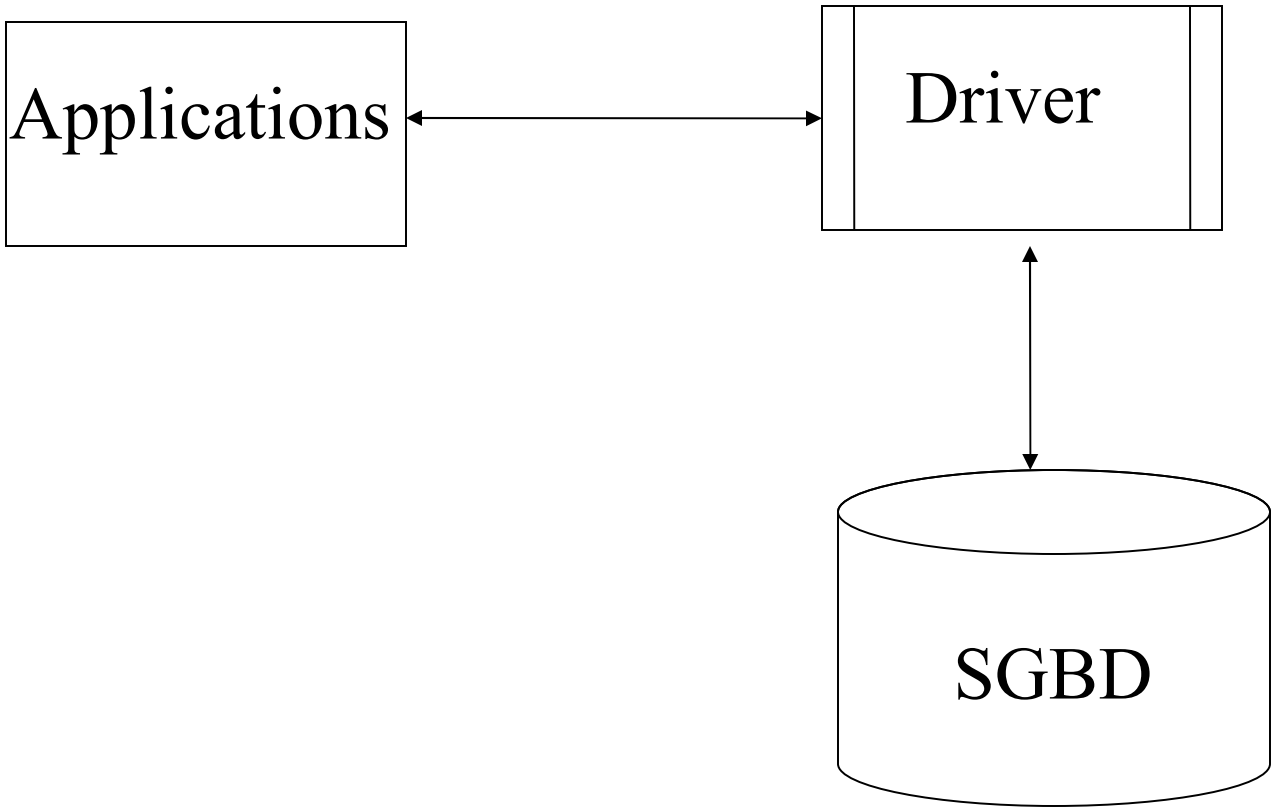


Figure 1.1: illustrates a two-tier architecture for data access.

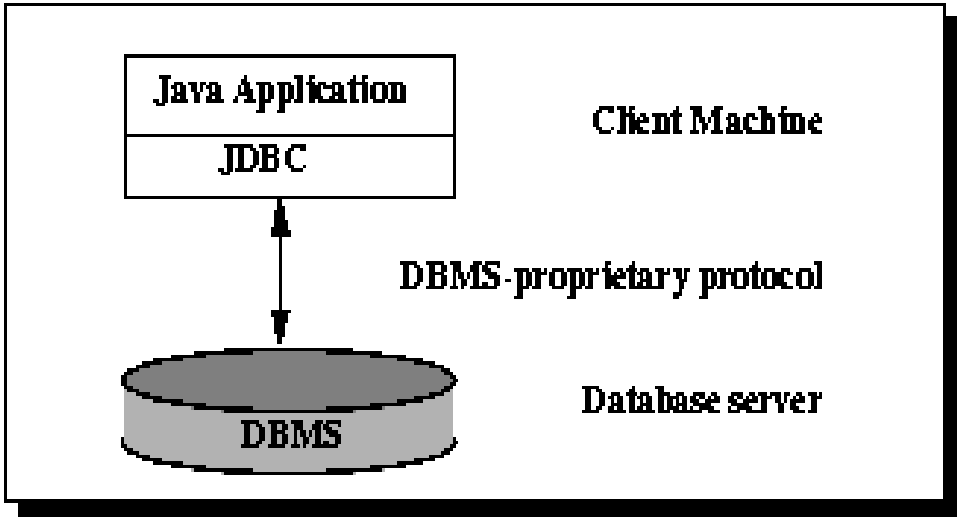
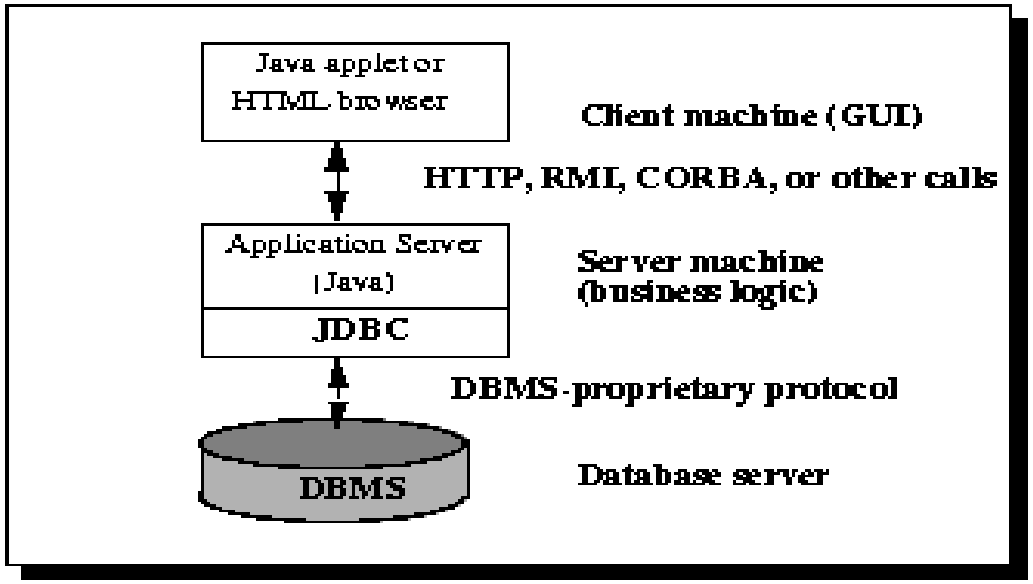


Figure 1.2: illustrates a three-tier architecture for database access.



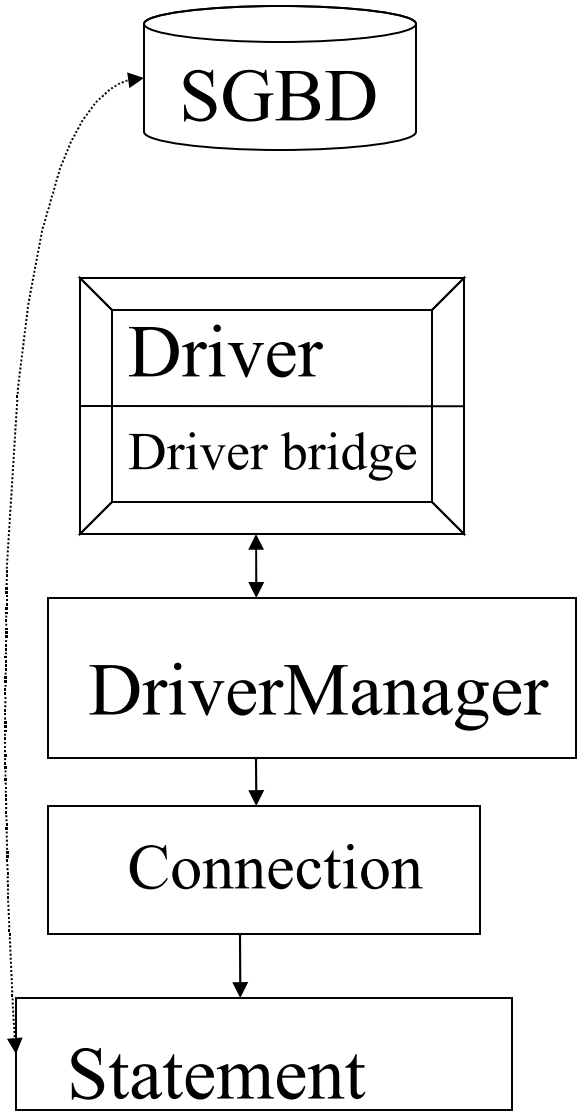
JDBC API

- Package: java.SQL
- java.SQL contient 8 interfaces permettent:
  - donner le nom de pilote jdbc utilisé: **Diver**
  - se connecter à une base de données: **Connection**
  - Création et exécution des requêtes**: Statment**, **PreparedStatement**, **CallableStatement**
  - Traitement des résultats: **ResultStatement**
- Meta-données sur la connexion et sur les résultats **DatabaseMetaData**, **ResultSetMetaData**

Interface	Description
java.sql.Driver	Locates the driver for a database URL
java.sql.Connection	Connection to a specific database
java.sql.Statement	Executes SQL statements
java.sql.PreparedStatement	Handles parameterized SQL statements
java.sql.CallableStatement	Handles database stored procedure calls
java.sql.ResultSet	Gets the results of SQL statements
java.sql.DatabaseMetaData	Used to access a variety of information about connection's DBMS and database
java.sql.ResultSetMetaData	Used to access a variety of information describing a ResultSet's attributes

JDBC API

- **java.sql package**
- **java.sql.DriverManager**
  - Obtain a DB connection
- **java.sql.Connection**
  - A connection with one DB
- **java.sql.Statement**
  - A SQL query



Connexion : URL

<protocol>:<subprotocol>:<datasource>

- <protocol> : jdbc
- <subprotocol> : le nom de driver ou le nom de mécanisme de connexion utilisé par la base de données..
- <datasource>: le nom de la base de données ou l'adresse pour identifier la base de données...

Exemple

- URL Sybase

jdbc:sybase:Tds:serpent.atela.uhp-nancy.fr:4100/rnc3

- URL Oracle

jdbc:oracle:thin:@panoramix:1521:depinfo

- Le driver de sybase:

com.sybase.jdbc.SybDriver

- Le driver d'oracle
  oracle.jdbc.driver.OracleDriver"

```
// Creation d'une table
import java.sql.*;
    public class Create {
      public static void main (String args[]) {
     String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;
       Statement stmt;
        String login="",  passwd="";
        try {
     login=args[0];
     passwd=args[1];
        } catch (Exception e) {
     System.err.println("usage : Create login password");
     System.exit(1);}
        String  query ="create table client(nom varchar(20),prenom varchar(20), age int)";
            try {// enregistrement du driver
     Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch(ClassNotFoundException e){
     System.err.print("ClassNotFoundException");
     System.err.println(e.getMessage());
        } try { // connexion et execution de la requete
     con = DriverManager.getConnection (url,login,passwd);
     stmt = con.createStatement ();
     stmt.executeUpdate(query);
     stmt.close();
     con.close();
            } catch (SQLException ex) {
     System.err.println (" SQLException caught: "+ ex.getMessage());}  }}
```

```
// Insertion des données dans une table

import java.lang.*;
import java.sql.*;

public class InsertTable {
  public static void main (String args[]) {
        String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;
       Statement stmt;
        String login="",  passwd="";
        try {
     login=args[0];
     passwd=args[1];
        } catch (Exception e) {
     System.err.println("usage : InsertTable login password");
     System.exit(1);}
        try { // enregistrement du driver
              Class.forName("oracle.jdbc.driver.OracleDriver");
             } catch(ClassNotFoundException e) {
                   System.err.print("ClassNotFoundException");
                   System.err.println(e.getMessage());  }
         try {
           con = DriverManager.getConnection (url, login, password);
           stmt = con.createStatement ();
      stmt.executeUpdate("insert into client values('Titi', 'lala',20)");
      stmt.executeUpdate("insert into client values('Dupond', 'Paul',30)");
      stmt.executeUpdate("insert into client values('Momo', 'toto',10)");
           stmt.close();
           con.close();
              } catch (SQLException ex) {
           System.err.println (" SQLException caught: "+ ex.getMessage());  }}}
```

```
// selection
import java.sql.*;
  public class Select{
  public static void main (String args[]) {
        String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;
       Statement stmt;
       String query= "select nom, prenom, age from client";
       String login, password;
       String n,p;
       Int a;
           try { // enregistrement du driver
               Class.forName("oracle.jdbc.driver.OracleDriver");
                } catch(ClassNotFoundException e){
                   System.err.print("ClassNotFoundException");
                   System.err.println(e.getMessage());  }
           try { // connexion et execution de la requete
                 con = DriverManager.getConnection (url,login,password);
                   // Create a Statement object so we can submit SQL statements to the driver
                    stmt = con.createStatement ();
                   ResultSet rs= stmt.executeQuery(query);  // Submit a query
        while ( rs.next()) {
            n = rs.getString(1); // nom
            p = rs.getString(2); // prenom
            a = rs.getInt(3); // age
            System.out.println("nom: "+n+ " prenom: "+p+ « age: "+a);}
         stmt.close();    // Close the statement
              con.close();  // Close the connection
              } catch (SQLException ex) { System.err.println (" SQLException caught: "+
   ex.getMessage());  } }}
```

Java.sql.ResultSet

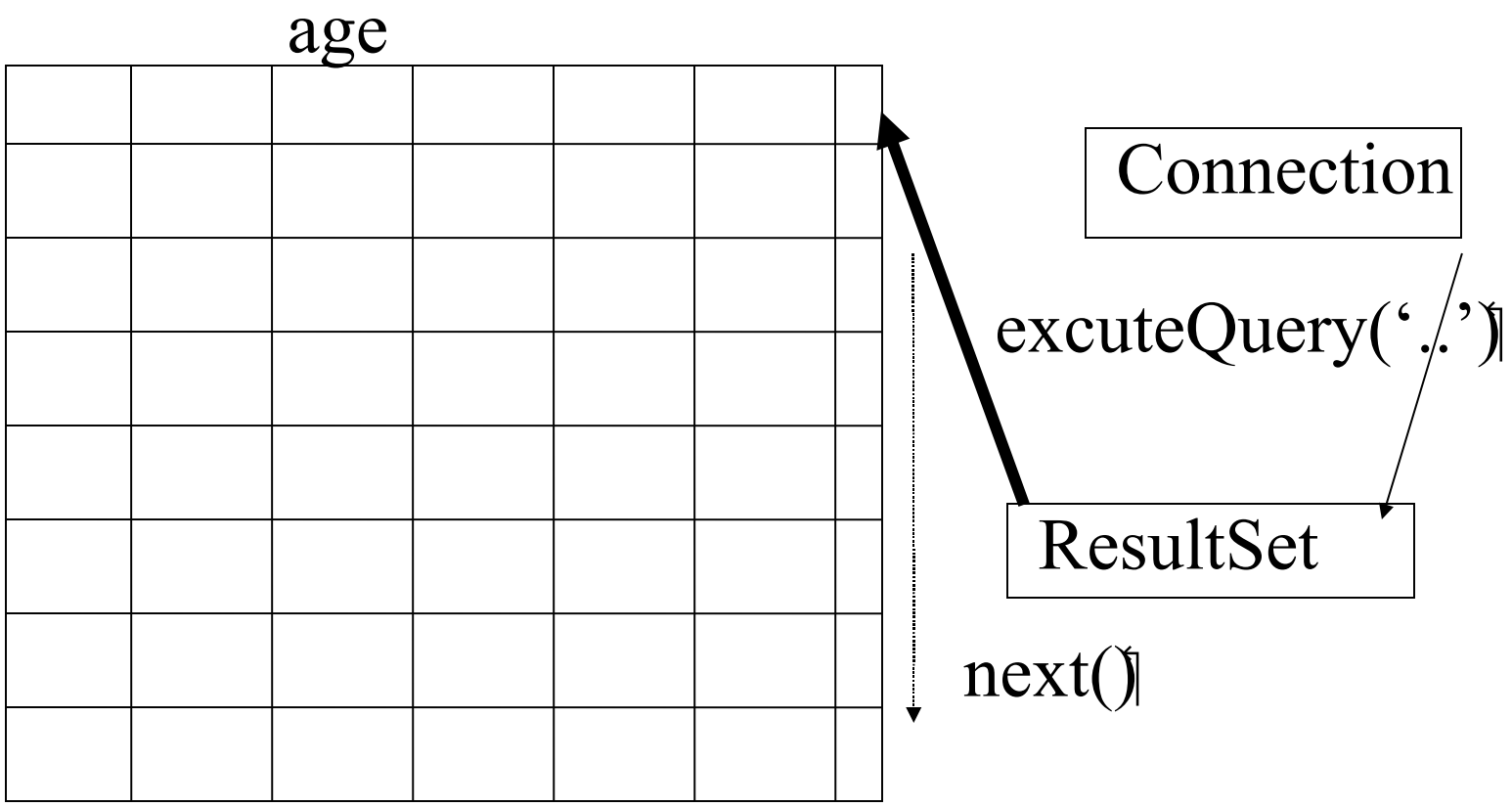


Table 2: Standard mapping from SQL types to Java types.

SQL type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Table 3: Standard mapping from Java types to SQL types.

The mapping for String will normally be VARCHAR but will turn into LONGVARCHAR if the given value exceeds the drivers limit on VARCHAR values. Similarly for byte[] and VARBINARY and LONGVARBINARY.

Java Type	SQL type
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

		SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	X	X	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	x	x	X	x	x	x						
getString	x	x	x	x	x	x	x	x	x	x	X	X	x	x	x	x	x	x	x
getBytes														X	X	x			
getDate											x	x	x				X		x
getTime											x	x	x					X	x
getTimestamp											x	x	x				x		X
getAsciiStream											x	x	X	x	x	x			
getUnicodeStream											x	x	X	x	x	x			
getBinaryStream														x	x	X			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 1: Use of ResultSet.getXXX methods to retrieve common SQL data types
An "x" means that the given getXXX method can be used to retrieve the given SQL type.
An "X" means that the given getXXX method is recommended for retrieving the given SQL type

```
// une requête dynamique ...
import java.lang.*;
import java.sql.*;
public class Dynamique {
  public static void main (String args[]) {
        String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;


    try { // enregistrement du driver
    Class.forName("oracle.jdbc.driver.OracleDriver");
            } catch(ClassNotFoundException e) {
                  System.err.print("ClassNotFoundException"+e.getMessage()); }
       try {   con = DriverManager.getConnection (url, login, password);
          // Precompile SQL statement
        PreparedStatement stmt = con.prepareStatement("select from client
  where  nom = ? and prenom = ?");
       stmt.setString(1,"Titi");
       stmt.setString(2,"Lala");
     ResultSet rs= stmt.executeQuery();  // Submit a query
        while ( rs.next())  {
            String n = rs.getString(1); // nom
             String p = rs.getString(2); // prenom
              int  a = rs.getInt(3); // age
              System.out.println("nom: "+n+ " prenom: "+p+ "ge: "+a);   }
                   stmt.close();
                  con.close();
                } catch (SQLException ex) {
        System.err.println (" SQLException caught: "+ ex.getMessage());  }
       }}
```

SQLJ et JDBC

// **SQLJ**

```
int n;
#sql { INSERT INTO emp VALUES (:n)};
```

// **JDBC**

```
int n;
Statement stmt = conn.prepareStatement  (“INSERT INTO emp VALUES (?)”);
stmt.setInt(1,n);
stmt.execute ();
stmt.close();
```

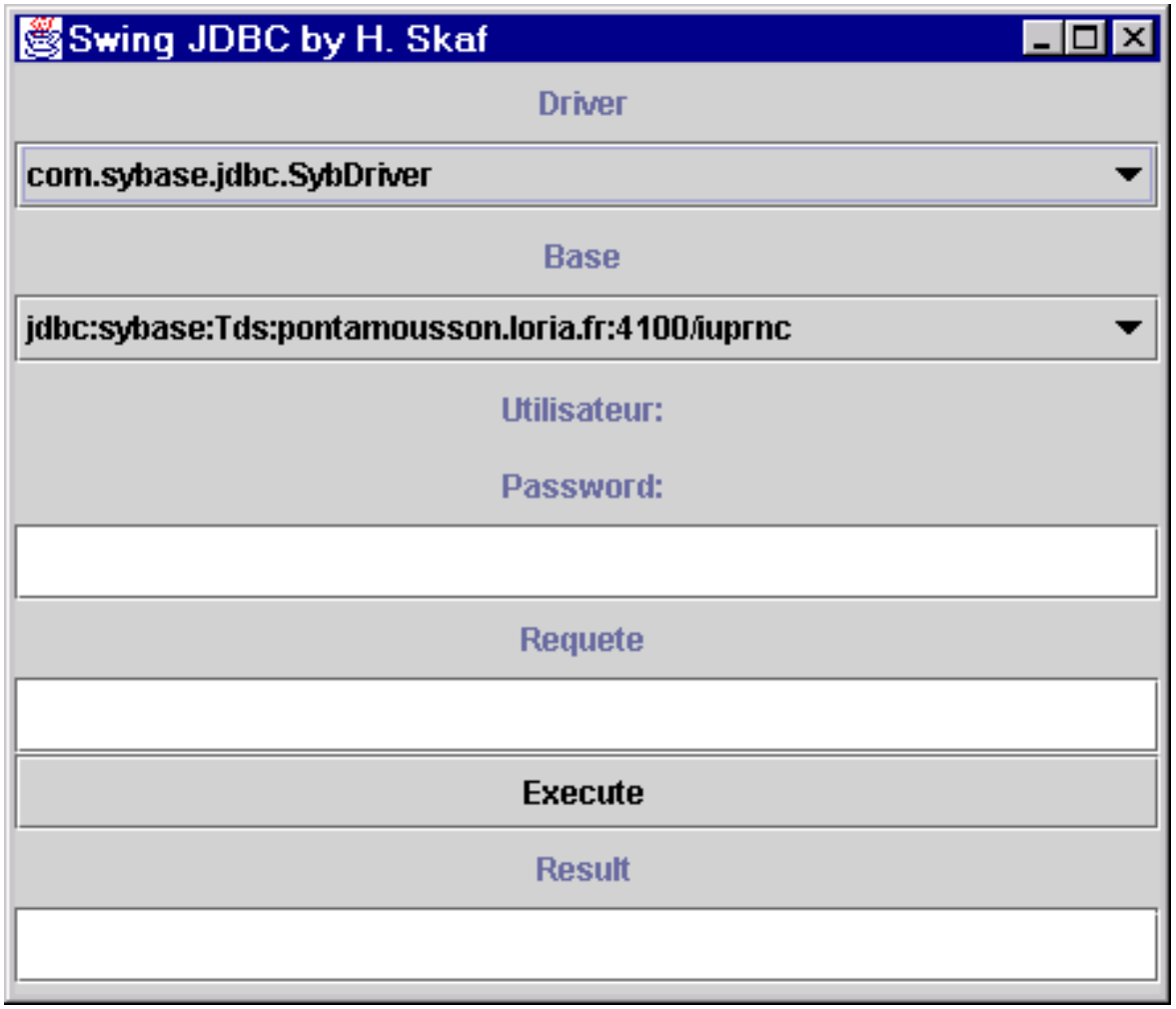
Accès aux Méta données

- **ResultSetMetaData** contient des informations sur le nombre, le type et certaines propriétés des colonnes d'un **ResultSet**

- Consultation des méta-données par les méthodes
  - **getColumnCount** ( ) : nombre de colonnes
  - **getColumnTypeName** (num_col) : nom du type de la colonne
  - **getColumnName** (num_col) : nom de la colonne
  - **getColumnLabel** (num_col) : label de la colonne
  - ...

```
import java.lang.*;
import java.sql.*;
  public class Meta{
  public static void main (String args[]) {
  String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;

          Connection con;
          Statement stmt;
          String colnom;
          String query= "select nom, prenom, age from client";
          String login="",  passwd="";
          try {
            login=args[0];
            passwd=args[1];
          } catch (Exception e) {
            System.err.println("usage : Meta login password");
            System.exit(1);
          }
          try { // enregistrement du driver
                  Class.forName("oracle.jdbc.driver.OracleDriver");

              //   Class.forName ("com.sybase.jdbc.SybDriver");
                 } catch(ClassNotFoundException e){
                    System.err.print("ClassNotFoundException");
                    System.err.println(e.getMessage());
                    }
            try { // connexion et execution de la requete
                  con = DriverManager.getConnection (url,login,passwd);
                    // Create a Statement object so we can submit
                    // SQL statements to the driver
                     stmt = con.createStatement ();

                    ResultSet rs= stmt.executeQuery(query);  // Submit a query
                 ResultSetMetaData rsmd = rs.getMetaData();
                 int nbcol = rsmd.getColumnCount();
                 for (int i=1; i <= nbcol; i++)
                   {
                  System.out.println("Colonne : "+rsmd.getColumnName(i));
                   System.out.println("Type : "+rsmd.getColumnTypeName(i) );
                   }
                 System.out.println(" Les donnees");
                 while ( rs.next())
                   {
                       String n = rs.getString(1); // nom
                        String p = rs.getString(2); // prenom
                         int  a = rs.getInt(3); // age
                         System.out.println(" "+n+ " "+p+ "  "+a);
                   }
                 stmt.close();    // Close the statement
                     con.close();  // Close the connection
                   } catch (SQLException ex) {
                   System.err.println (" SQLException caught: "+ ex.getMessage());

                     }
          }
}
```

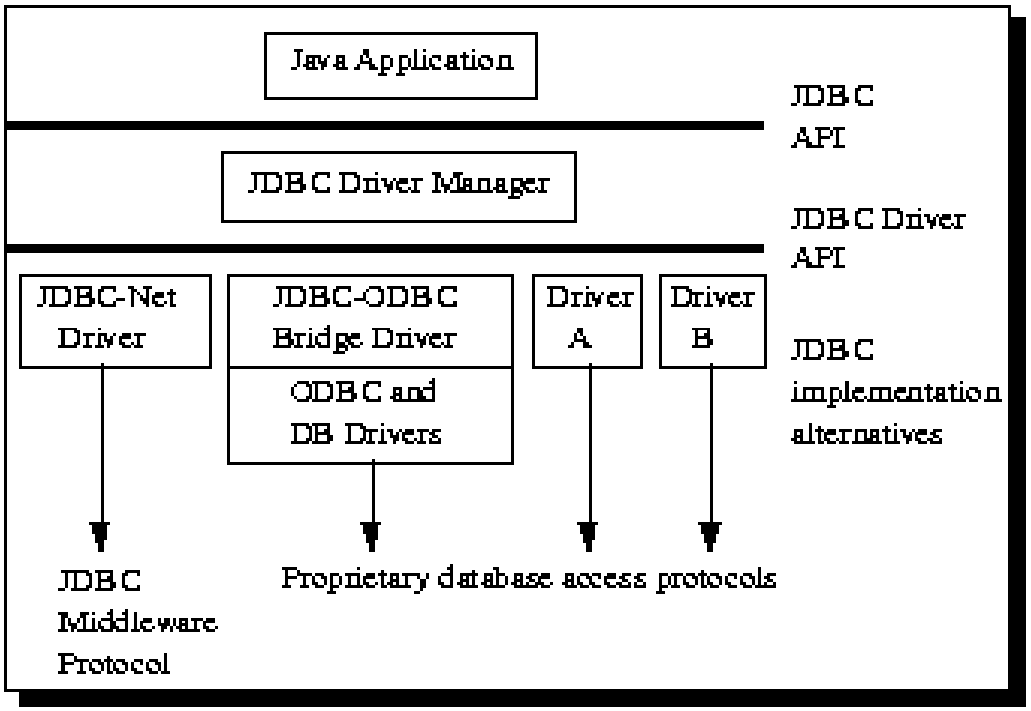
Une application avec Swing et JDBC



JDBC drivers

Selon JavaSoft: 4 types…

Figure 1.3: shows various driver implementation possibilities.



Source: http://java.sun.com/products/jdk/1.3/docs/guide/jdbc/

Type 1 JDBC-ODBC bridge

- Translates JDBC calls into ODBC calls and passes them to an ODBC driver.
- Some ODBC software must be resident on the client machine.
- Some client database code may also reside on the client machine.

ODBC: Open DataBase Connection (accès aux bases de données dans le monde Microsoft..)

Type 2 native-API partly-Java driver

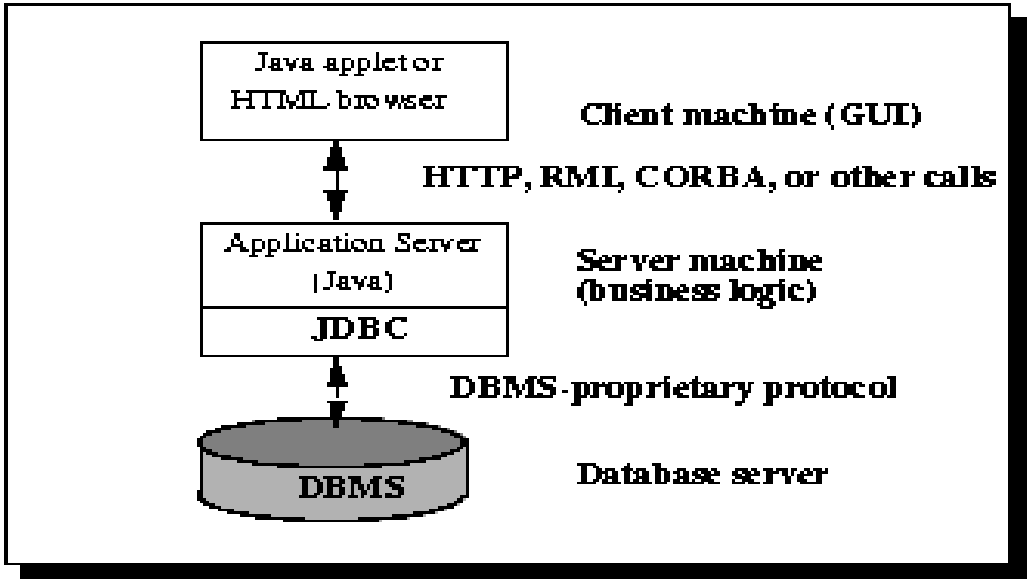
- Converts JDBC calls into database-specific calls.
- This driver, which communicates directly with the database server, also requires some binary code on the client machine...

Type 3 net-protocol all-Java driver

- Communicates to a middle-tier server using a DBMS-independent net protocol.
- A middle-tier gateway then converts the request to a vendor-specific protocol.

Type 3 : Exemple

Figure 1.2: illustrates a three-tier architecture for database access.

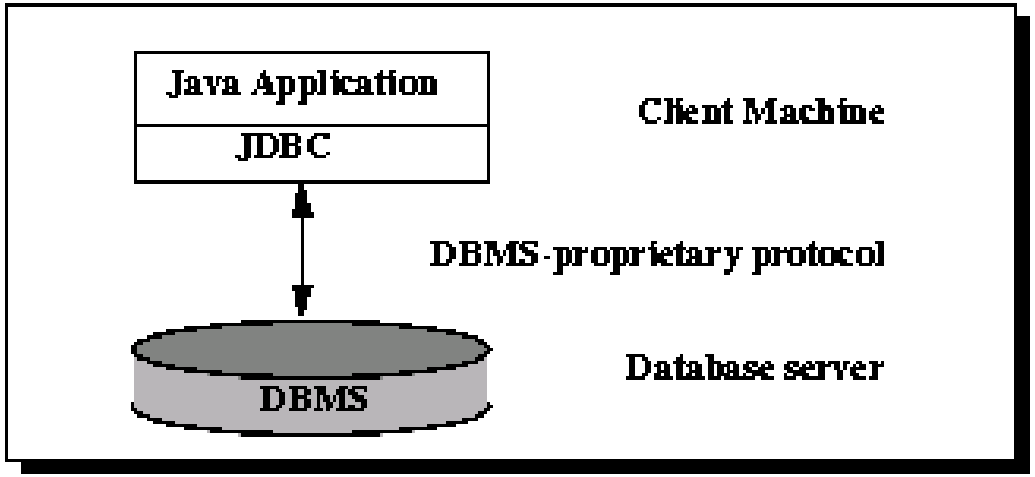


Type 4 native-protocol all-Java driver

- Converts JDBC calls to the vendor-specific DBMS protocol, allowing client applications direct communication with the database server.
- Le driver utilisé dans les exemples est de ce type...

Type 4 native-protocol all-Java driver

Figure 1.1: illustrates a two-tier architecture for data access.



Source:
http://java.sun.com/products/jdk/1.3/docs/guide/jdbc/

Synthèse

DRIVER CATEGORY	ALL JAVA	NETWORK CONNECTION
1 - JDBC-ODBC Bridge	No	Direct
2 - Native API as basis	No	Direct
3 - JDBC-Net	client Yes server Maybe	Indirect
4 - Native protocol as basis	Yes	Direct

Conclusion

- JDBC offre une couche d'abstraction indépendante de:
  - base de données...
  - plate-forme système…
  - une base pour une utilisation plus 'user-friendly'
    - embedded SQL for Java : SQLJ
    - "Object/relational" mapping ..

Sécurité de fonctionnement et gestion des accès concurrents

Hala Skaf-Molli

Références

- Database Transaction Models for Advanced Applications, edited by Ahmed K. Elmagarmid, 1992.
- Environnements de Développement Coopératifs, thèse en Informatique, Pascal Molli, 1997..
- Sybase SQL Server, Transact-SQL User's Guide
- + sur le WEB ...

Architecture Fonctionnelle d'un SGBD

Interface 4ème génération

S G B D

LDD | LMD | Confidentialié | Securité | concurrence | partage | Intégrité | Interface

Pascal
Cobol
C
Java

Système d'exploitation

Base de données

Base de données

Fonctionnalités de SGBD

- Multi-utilisateurs:
  - plusieurs utilisateurs peuvent interagir avec une DB simultanément…
  - régler les problèmes dûs au travail à plusieurs en même temps sur les mêmes objets…
  - assurer la cohérence de la base même en présence des accès concurrents (même en cas dysfonctionnement!!)

  => Transaction…

Définition: Transaction

- Une transaction : un ensemble d'actions permettant de prendre une base données dans état **cohérent** et elle la rend dans un autre état cohérent..

- Elle donne l'illusion à l'utilisateur d'être tous seul

Exemple

- Retrait 50 Fr du compte x; valeur initiale, x= 100Fr; CI: x>=0
- *Programme:*

```
k ← lire(x)‖
if k >= 50 then
        k = k-50
        écrire(x,k)
else
        print(« pas assez d ’argent sur le
            compte »)‖
endif
```

Transaction

T1:Retrait(x,50)‖

---

lire(x, 100)‖

écrire(x,50)‖

Transaction est une séquence d’actions de lire et écrire...

Terminaison de transaction

- L'exécution de transaction se termine : commit ou abort (rollback).
- **Commit**: la transaction est réussie, ses mises à jour doivent être incorporées dans la BD.
- **Abort**: la transaction a échoué, la BD doit annuler ou supprimer tous les effets de la transaction sur la BD

Transaction et cohérence

- ***Par définition**, une transaction qui modifie les objets d'une BD doit préserver les contraintes d'intégrité de la base…*
- Deux approches:
  - l'utilisateur connaît toutes les CI, il écrit que des programmes corrects (qui respecte les CI)…
  2. pas d'hypothèse sur la correction individuelle des transactions : la BD doit vérifier statiquement et dynamiquement les CI…

Transaction et Cohérence

- La première approche est attractive, pourquoi !!!
- Deuxième approche:
  - sur coût «*overhead*» sur le SGBD..
  - Réduire la disponibilité de la base:
    - l'accès aux données est bloqué: assurer dynamiquement la cohérence de la base après chaque transaction implique que l'état de la base ne peut pas être touché à la fin de la transaction par un autre utilisateur..
  - Le SGBG utilisent approche mixte….

Transaction

- L'hypothèse standard

si une transaction s'exécute toute seule, dans une base de données cohérente, alors elle va laisser la base dans un autre état cohérent..

- Attention: pendant son exécution une transaction peut violer les CI mais lorsqu'elle termine (commit), elle met la base dans un état cohérent..

Propriétés de transactions

Chaque transaction doit être **ACID**:

- **A**tomicité: toute ou rien. Soit toutes les opérations de la transaction sont exécutées soit aucune..
- **C**ohérence: une transaction prise individuellement doit faire passer la base d'un état cohérent dans un autre état cohérent.
  Isolation: une transaction ne doit observer que des états cohérents de la base (pas de résultats intermédiaires d'une autre transaction)
- **D**urabilité: lorsqu'une transaction termine, ses résultats deviennent permanents et ne peuvent plus être remis en cause, ni par une panne du système, ni par une autre transaction (remettre en cause la durabilité renvoie aux problème d'exécution partielles)...

Propriétés de transactions

Deux classes d'algorithmes ou protocoles pour assurer l'ACID de transactions:

- Correction des exécutions concurrentes (Execution Atomicity): **protocoles des accès concurrents**
  - maintenir la cohérence des transactions même si elles sont exécutées de manière concurrente : garantir que l'exécution concurrente d 'un ensemble de transactions individuellement correcte fait passer la base dans un état cohérent…
- Prennent en compte les exécutions partielles des transactions (Failure Atomicity) : **protocoles de recouvrement :**
  - maintenir la loi de toute ou rien, isolation et durabilité.

Correction des exécutions concurrentes

- Le système garantit que l'exécution concurrente d 'un ensemble de transactions individuellement correcte fait passer la base dans un état cohérent…

- Pas de violation de cohérence à cause des exécution concurrente des transactions..

Quelles sont les problèmes liés aux exécutions concurrentes ?

Recouvrement

- Trouver un état cohérent de la base à la suite d'une exécution partielle...
- 2 types d 'échecs
  - échec transaction: l'utilisateur veut annuler la transaction, division par zéro, erreur !!
  - Échec système: coupure d'électricité, crash disque !!!
- N'oubliez pas une transaction est atomique (tout ou rien..)

Exécution concurrente

- 3 problèmes :
  - mise à jour perdue
  - lectures impropres
  - lecture non reproductibles

Mise à jour perdue

Etat initial de s:
s=300 ; CI: s>= 0

t_1: programme user1 Retrait(200F, s)∥	t_2: programme user2 Retrait(300F, s)∥
temp1 ← **lire(s)**∥	
temp1 ← temp1 -200	
	temp2 ← **lire(s)**∥
	temp2 ← temp2 -300
	écrire(s,temp2)∥
écrire(s,temp1)∥	

Ordonnancement de t1 et t2

$lire_1(s,300)$∥
$lire_2(s,300)$∥
$écrire_2(s,0)$∥
$écrire_1(s,100)$∥

t1 toute seule est correcte; t2 toute seule est correcte
Après l 'exécution concurrente : s= 100 !!!

La mise à jour de Retrait(300F, s) est **perdue**!!

Lectures impropres

CI:balance= c + s = 400

t1:programme user1 Virement(100F, s, c)	t2:programme user2 Balance()
temp1 ← **lire(s)** temp2 ← **lire(c)** **écrire(s, temp1-100)**	
	temp3 ← **lire(s)** temp4 ← lire(c)
écrire(c,temp2+100)	
	imprimer(temp3+temp4)

Ordonnancement de t1 et t2

lire1(s,200)
lire1(c,200)
écrire1(s,100)
lire2(s,100)
lire2(c,200)
écrire1(c,300)

Après l exécution:
balance= 300 !!!
Balance() a fait de lectures impropres

Lectures non reproductibles

t1:programme user1	t2:programme user2
lire(s, 100)‖	
imprimer(s)‖	
	lire(s, 100)‖
	écrire(s, 300)‖
lire(s, 300)‖	
imprimer(s)‖	

Les 2 valeurs affichées ne sont pas identiques !!

Conclusion sur les exécutions concurrentes

- Imposer que chaque transaction soit individuellement correcte (respecte les CI) ne garantit pas que l'exécution concurrente de ces transactions soit correcte..

i.e. Une exécution concurrente de deux transactions bien écrites peut laisser la base dans un état incohérent..

- Solution possible:
  - SGBD exécute les transactions l'une après l'autre de manière séquentielle (ou en série).
  - **Une exécution séquentielle est correcte**.. Pourquoi??

Inconvénients

- De manière générale, l'entrelacement des opérations permet une meilleure utilisation de CPU..
- De plus, deux transactions n'accédant pas à des données partagées doivent aussi s'exécuter de manière séquentielle !!!

- **Existe-il des exécutions non séquentielles correctes?**

  En d'autre terme:

  Deux transactions accédant à des données partagées peuvent avoir un entrelacement correct de leur opérations !!!

Exemple

t1:programme user1	t2:programme user2
Virement(100F, s, c)¶	Balance()¶
temp1 ← **lire(s)**¶	
écrire(s, temp1-100)¶	
	temp3 ← **lire(s)**¶
temp2 ← **lire(c)**¶	
écrire(c,temp2+100)¶	
	temp4 ← lire(c)¶
	imprimer(temp3+temp4)¶

Ordonnancement de t1 et t2

lire1(s,200)¶
écrire1(s,100)¶
lire2(s,100)
lire1(c,200)¶
écrire1(c,300)
lire2(c,200)

- Exécution correcte..

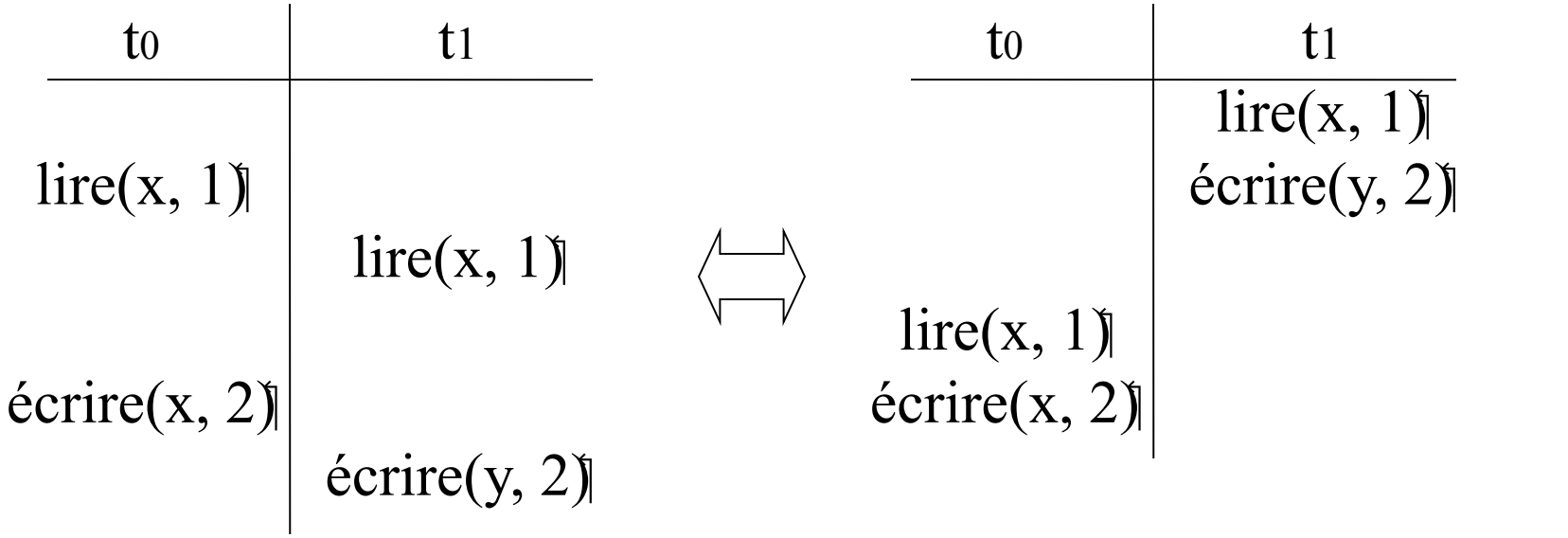
Pour l'utilisateurs : c'est équivalent à l'exécution de transfert suivie par Balance ..

=> accepter les exécutions dont leur comportement n'est pas différent d'une exécution en série ...

Sérialisabilité

- Critère de correction des exécutions concurrentes..
- **Une exécution concurrente d'un ensemble de transactions est correcte s'il existe une exécution en série telle que dans les deux exécutions:**
  - **chaque transaction lit les mêmes valeurs**
  - **et les valeurs finales sur la base sont les mêmes..**
- Une exécution concurrente respectant ce critère est dite sérialisable (View sérialisable)...

Exemple: exécution view sérialisable



Exécution concurrente de : $t_0 \parallel t_1$

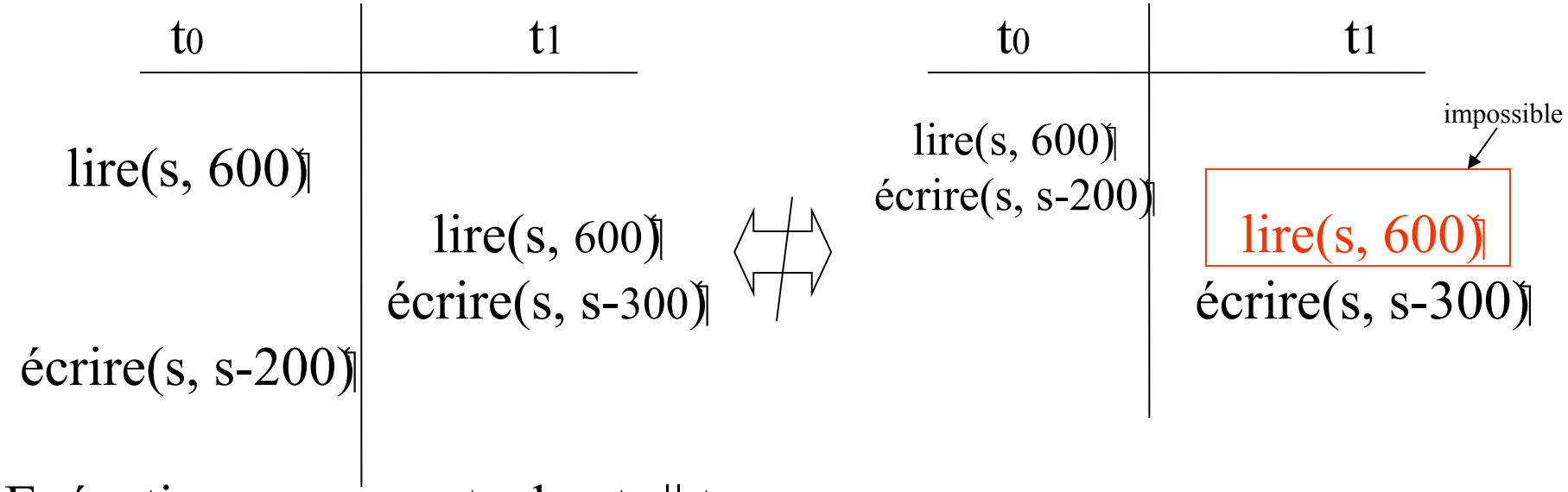
Exécution en série de $t_1;t_0$

Dans les deux cas:

les transactions lisent les mêmes valeurs, ReadSet??

et produisent le même état de la base, WriteSet??

Exemple: exécution non view sérialisable

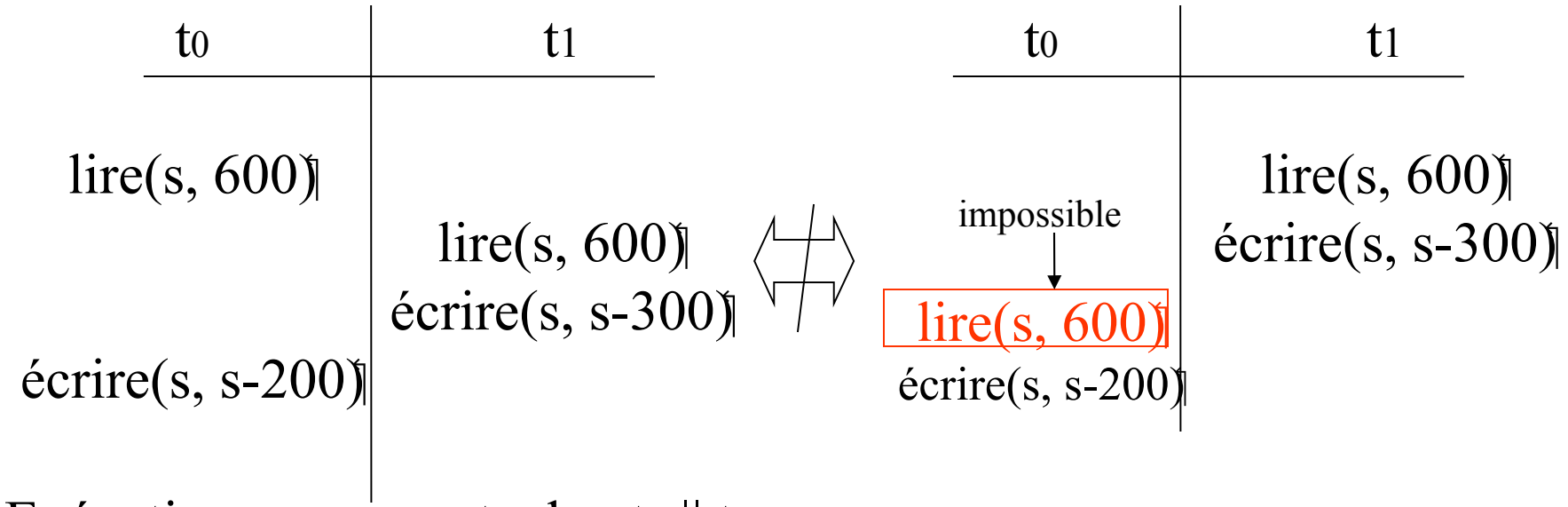


Exécution concurrente de : $t_0 \parallel t_1$

Exécution en série de t_0; t_1

Exécution non view sérialisable ??

Exemples



Exécution concurrente de : t0 || t1

Exécution en série de t1; t0

Exécution non view sérialisable

Est-il possible d'implanter le critère de view serialisabilité ?

- Trouver si une exécution est view sérialisable est un problème NP-complet
- Exemple:
  - Tj lit une valeur x écrite par Ti => toutes les opérations de Tj suivent les opérations de Ti (il y a une contrainte: Ti suivie par Tj)
  - Tk écrit x alors Tk avant Ti ou après Tj (sinon, Tj doit lire la valeur écrite par tk) (pas de contrainte..)
  - Deux choix possibles..
  - Le nb de choix ????
- Plusieurs choix: explosion combinatoire..

Conflict sérailisabilité

- Imposer un ordre d'exécution entre les transactions:
  - pas seulement dans le cas ou une transaction lit une valeur écrite par une autre transaction..
  - Mais aussi lorsqu'il y a une écriture concurrentes
  - ou lecture et écriture…
- Idée: opérations en conflit ou commutativité entre les opérations..

Opérations en conflit

- Deux opérations sont en conflit :
  - si leur résultats dépend de leur ordre d'exécution (cas opérations de lecture/écriture)..
  - ou la valeur finale de la base dépend de leur ordre d'exécution (opérations d'écriture..)
- Pour deux opérations op1 et op2 issues de deux transactions t1 et t2, on a 5 cas possibles..

Opérations en conflit

- op1 et op2 sur des objets différents: pas de conflit…
- op1 et op2 sont des opérations de lectures: pas de conflit..
- écriture précède une lecture: conflit
- lecture précède une écriture: conflit..
- op1 et op2 sont d'écritures sur le même objet: conflit

Résumons

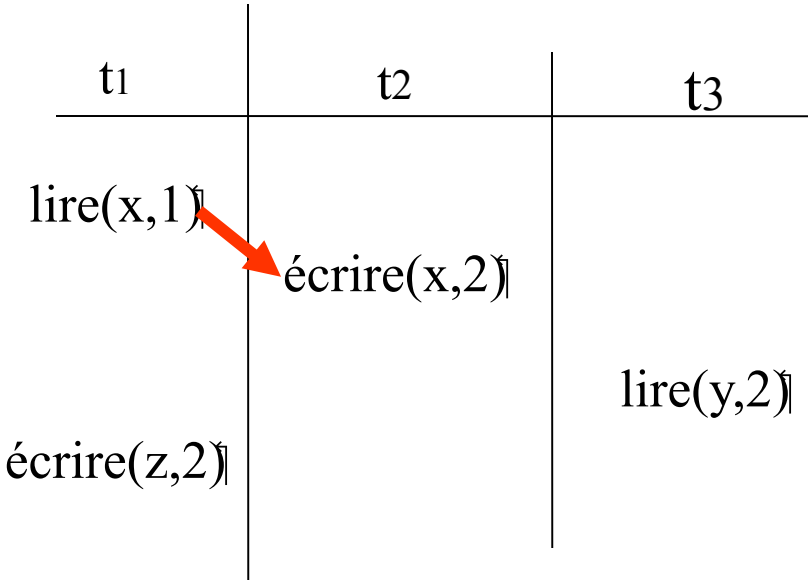
	Lire	écrire
Lire	NO	Yes
écrire	Yes	Yes

Deux opérations sont en conflits si elles opèrent sur le même objet et une parmi elles est une opération d ’écriture

Conflict sérailisabilité

- Une exécution correcte pour un ensemble des transactions est une exécution pour laquelle :
  - il existe une exécution en série pour le même ensemble des transactions dans ces deux exécutions l'ordre des opérations en conflits est le même..

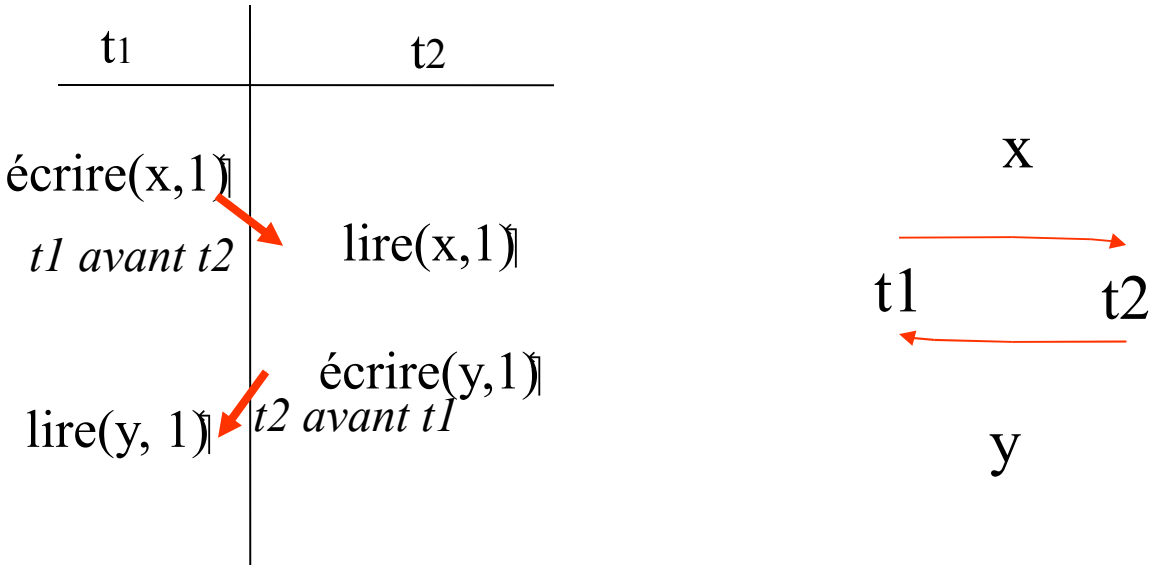
Exemple



T1||t2||t3 <=> t1;t3;t2

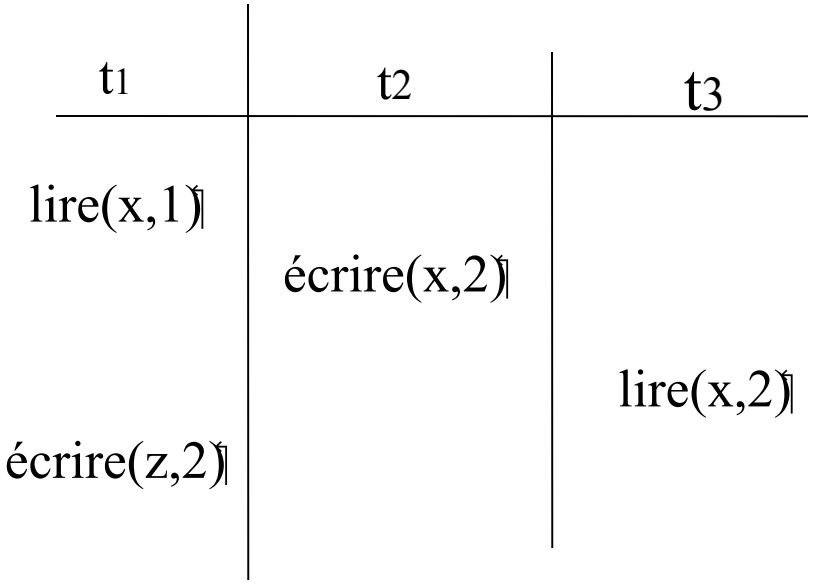
Graphe de dépendance (graphe de précédence)

- Une exécution est conflict sérialisable si son graphe de dépendance est acyclique



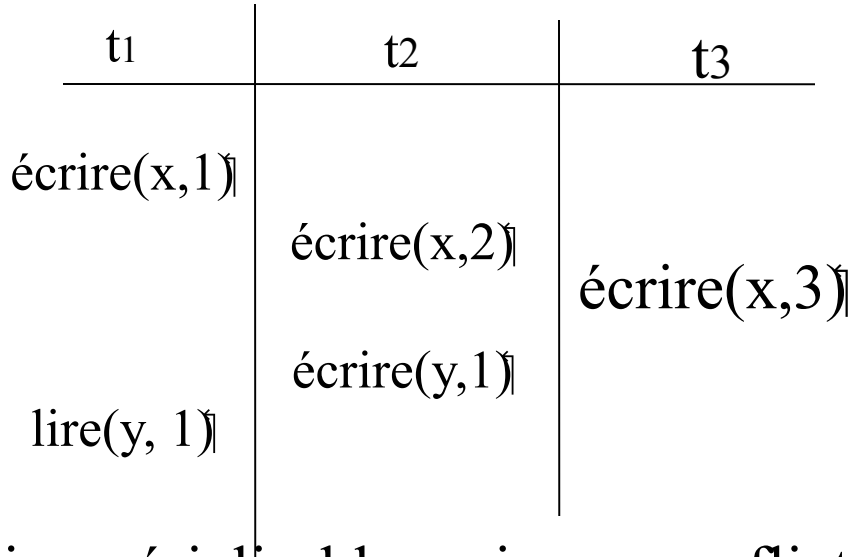
Il y a un cycle donc pas conflict sérialisable

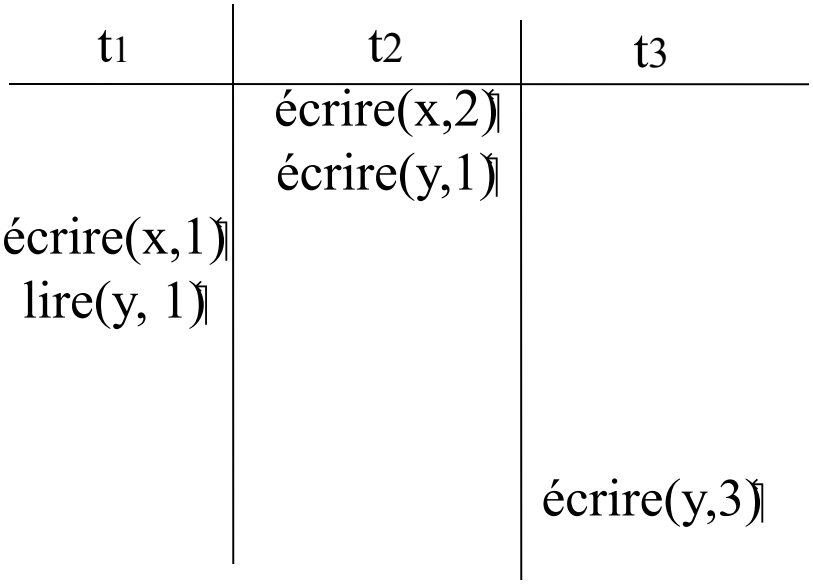
Exemple: exécution conflict sérialisable



Conflict sérialisabilité

- Une exécution Conflict sérialisable est aussi view sérialisable mais le contraire n'est pas vrai





view sérialisable mais pas conflict sérialisable:

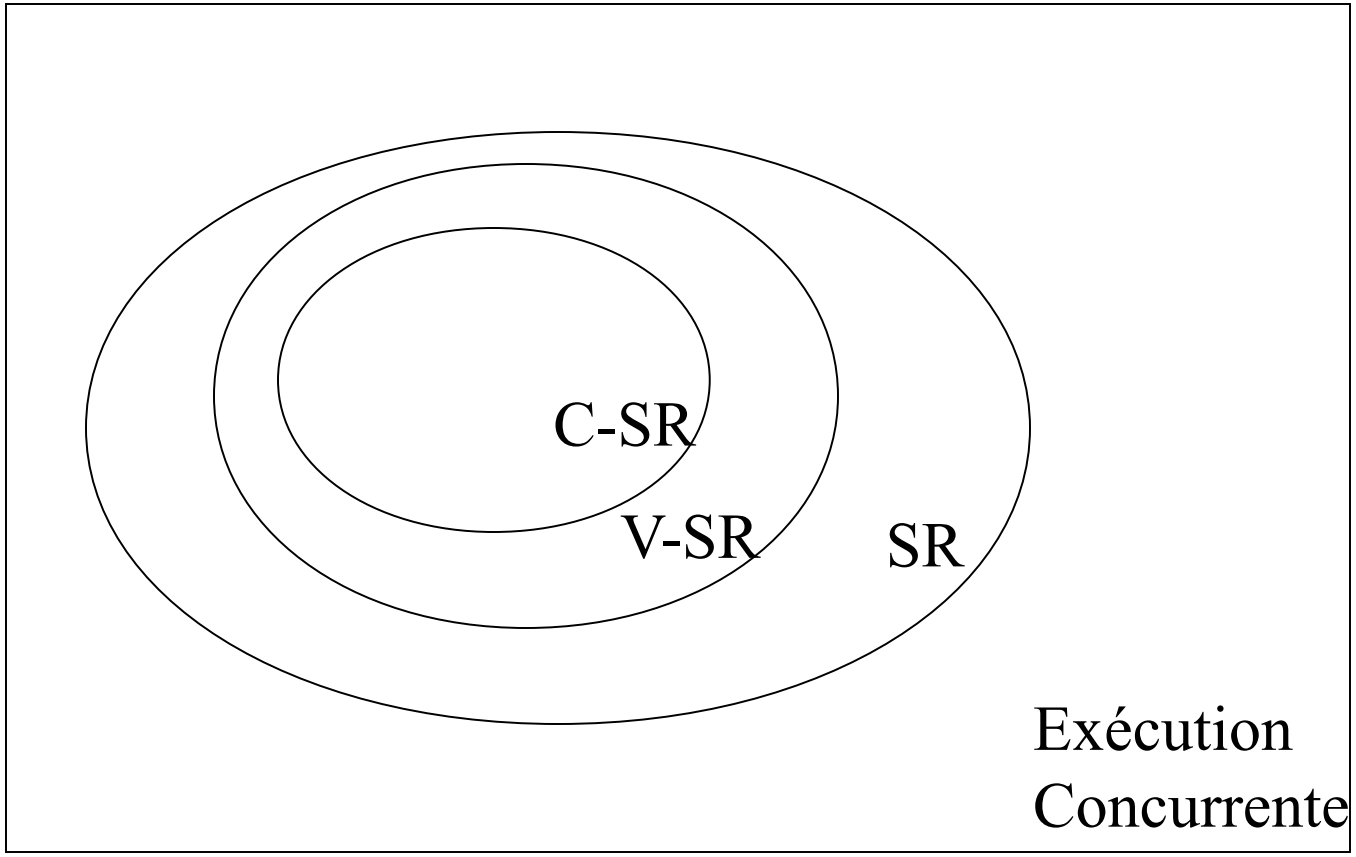
Exécution en série

dans exécution en série équivalente: t1 doit précéder t2 et t2 doit précéder t1

Conclusion

- Une exécution concurrente est correcte si elle est équivalente à une exécution en série..
- Critères de correction:
  - Sérialisabilité (View séralisable) : NP-Complet
  - Conflict Sérialisable: graphe de dépendance..

Conclusion



Conflict sérialisabilité

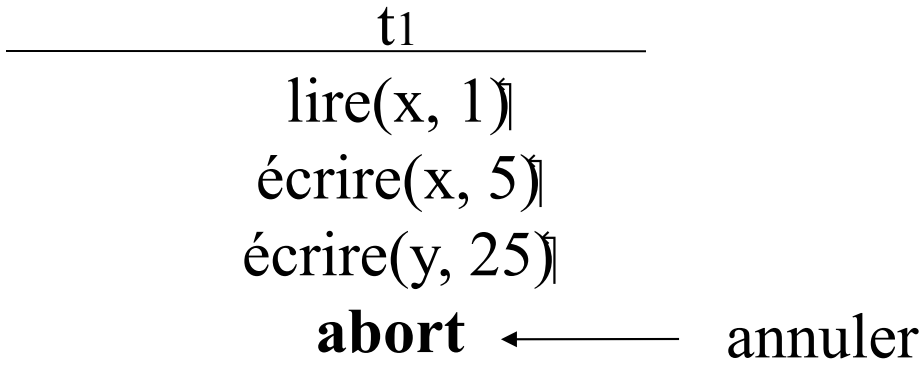
- Protocoles (algorithmes) pour contrôler les accès concurrents et assurer les conflicts sérialisabilité des transactions:
  - **Protocole de verrouillage à deux phases: 2PL (two phases locking protocol).**
  - **Timestamp ordering protocole..**
  - Optimistic protocole...
  - serialization graph testing..

Sécurité de fonctionnement: recouvrement (Failure Atomicity)

- 2 types d'échecs
  - échec transaction: l'utilisateur veut annuler la transaction, division par zéro, erreur !!
  - Echec système: coupure d'électricité, crash disque !!!

N'oubliez pas une transaction est atomique (toute ou rien..)

Echec de transaction



- **Abort**: annuler les effets de la transaction, comme si la transaction n'a jamais existait !!
- Pour l'opération lire: rien à faire..
- Pour l 'opération écrire: il faut l'annuler «undo», pour retrouver l'état de la base avant la transaction (*before-image*)
- La technique utilisée consiste à tenir le journal de modification *log*

Journalisation des transactions

- Pour permettre de **défaire** et \ou **refaire** il faut connaître les actions effectuées sur la BD et conserver les anciennes et les nouvelles valeurs des objets
- Tenir un JOURNAL de toutes les opérations sur la BD (le « LOG »)..
- Synchroniser les E/S sur ce journal avec les E/S sur la base..
- Le JOURNAL est un fichier, il est supposé survivre à une panne (copie, etc..)

Journalisation des transactions

- Un élément du journal contient:
  - id de transaction,
  - début de transaction,
  - opération sur la bd (écrire !!!!)
  - ancienne valeur de l ’objet (image « avant »)
  - nouvelle valeur de l ’objet (image « après »)
  - validation d ’une transaction
  - point de sauvegarde (save point)
- **Principe**: «C’est le journal qui fait foi »

(« WRITE AHEAD LOG »: on écrit sur le journal)

Exemple

t_1
begin
lire(x, 1)‖
écrire(x, 5)‖
écrire(y, 25)‖
abort

Un extrait du log:

t1

begin

écrire, 1, 5

écrire, 3, 25

rollback

Exemple

t_1
lire(x, 1)‖
écrire(x, 5)‖
save point p1
écrire(y, 25)‖
abort

Un extrait du log:

t1

begin

écrire, 1, 5

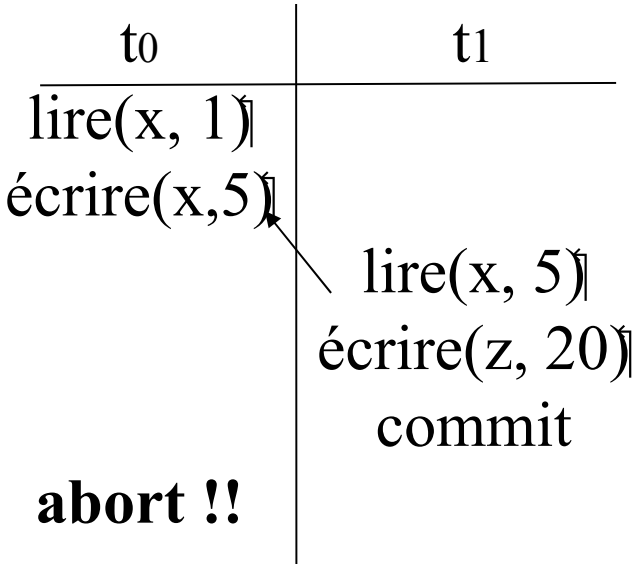
save transction p1

écrire, 3, 25

rollback

Echec de transaction

- Problème est un peu plus complexe !!!
- Scénario:



Annuler t1 => il faut annuler ses effets de la base :
(prendre *before-image* de t1)

Echec de transaction

- Problèmes:
  - t2 observe une valeur de x qui n'existe plus!!
  - t2 est terminée, il ne plus possible de défaire quoi que soit (une transaction est durable !!!!)
- Il faut fixer la règle suivante:

***Règle 1*: si une transaction t2 lit un objet modifié (un résultat intermédiaire) par une autre transaction t1 alors t2 ne peut pas se terminer terminer avant t1..**

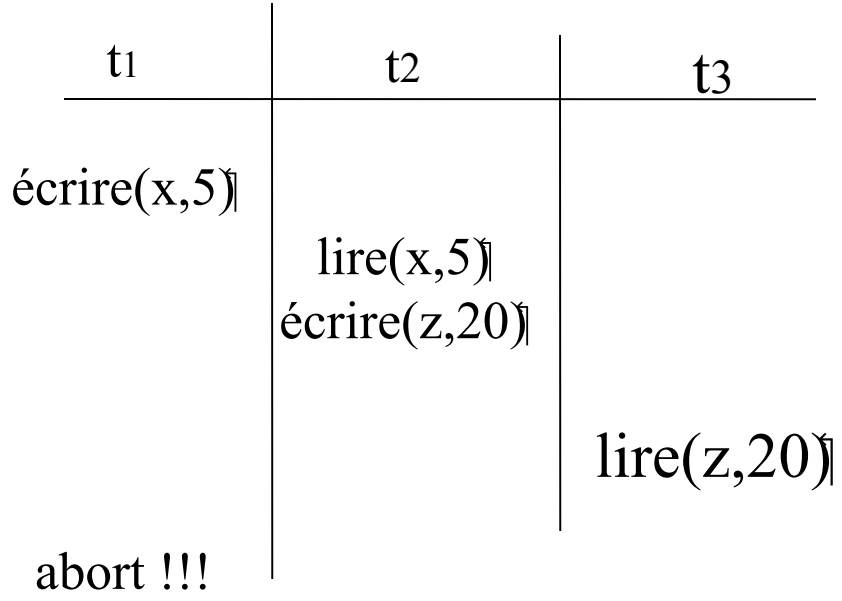
Les exécutions qui vérifient cette règles sont dites ***recouvrable***...

Synthèse

- Transaction est annulée=>
  - ses effets sur la base sont annulés en restaurant *before-image* pour les objets modifiés par la transaction..
  - Ses effets sur les autres transactions (celles qui ont lu des objets modifiés par la transaction) en les annulant *abort*

  => *cascade aborts !!*

Annulation en cascade



t1 est annulée => t2 doit être annulée => doit être annulée

Pour éviter ce phénomène utiliser l'**isolation** de transaction

Eviter les annulations en cascade

- Les transactions sont isolées:
  - une transaction ne peut pas voir les résultats intermédiaires des autres transactions..

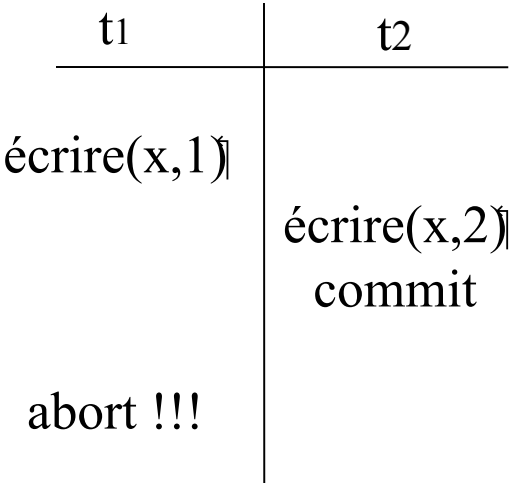
Règle 2:

une transaction t_2 peut lire un résultat d'une autre transaction t_1 seulement si t_1 est terminée (commit)

Les exécutions qui respectent cette règle sont «avoid cascading aborts»

Echec de transaction

- Les deux règles précédentes + *before-image* ne permettent pas d'établir la cohérence de la base en cas d 'échec..
- Exemple: valeur initiale de x est 0



Le before-image de t1 entraîne une mise à jour perdue:
les modifications effectuées par t2 sont perdues…
Et si j 'annule t2 puis t1??

Echec de transaction

- **Règle 3** :
  - Si une transaction t1 écrit une valeur x alors aucune autre transaction ne peut écrire x tant que t1 n ’est pas terminée ou annulée..

Résumons

On peut résumer les trois règles par:

si une transaction t1 écrit une valeur x alors aucune autre transaction ne peut lire ou écrire x tant que t1 n 'est pas terminés ou annulée

- Les exécutions qui respectent cette règle sont ***exécution stricte***
- Le protocole utilisé pour assurer ce type d'exécution est : 2PL strict

Echec système

- Une base de données utilise:
  - Mémoire secondaire, non volatile (disque)
  - Mémoire centrale, volatile (*database cache*)..
- Idée: pendant son exécution, une transaction lit et modifie les données dans la cache et à sa terminaison, l'idéal est que la transaction écrit ses modifications sur le disque..
- Problème: que se passe t-il en cas du panne système?
  - assurer l'atomicité de transactions ..

Echec système

- Solution: Utiliser le journal *log*
  - les effets de transactions qui étaient dans l ’état commit au moment de panne doit être incorporées dans la base (**do** opération)..
  - Les effets des transactions qui étaient dans l ’état abort au moment de défaillance doit être éliminer de la base (**undo** opération)

Echec système

- En cas d'arrêt du système
  - log contient *begin* sans commit: annuler la transaction. La commande **undo** restaure les objets modifié par la transaction à leur valeur initiale.;
  - log contient *begin* et *commit*: valider la transaction. La commande **redo** remet les objets modifié par la transaction à leur nouvelle valeur ..

2PL

- Basé sur le principe de verrouillage..
- Deux types de verrous:
  - partageables en lecture: V-partage(x)
  - exclusif en écriture: V-exclusif(x)
- Granularité (taille de données verrouillée) de verrou: tuple, table, base, page...
- **Table de comptabilité:**

	V-partage	V-exclusif
V-partage	Yes	no
V-exclusif	no	no

2PL

- Transaction bien formée si:
  - Avant de LIRE x elle a au moins un V-partage(x)
  - Avant d'ECRIRE x elle a un V-exclusif(x)
  - Aucun objet ne reste verrouillé après la FIN de la transaction
- Transaction à deux phases (2PL):
  - La règle de 2 phases: une transaction a deux phases:
    - *Growing* phase (verrouillage croissant) : elle obtient les locks mais pas de déverrouillage...
    - *shinking* phases (verrouillage décroissant): elle libère les locks mais pas de nouveaux verrouillages
    - n 'effectue aucune LOCK après avoir exécutée un UNLOCL

2PL

- L'exécution concurrente des transactions à 2PL est conflict sérialisable
- Exemple:

t1	t2
V-partage(A)‖	
lire(A, 5)‖	
	V-exclusif(C)‖
V-partage(B)‖	écrire(C,100)‖
lire(B,10)‖	V-partage(B)‖
unlock(A)‖	lire(B,10)‖
unlock(B)‖	unlock(B)‖
	unlock(C)‖

Problème de 2PL: Iterblocage (deadlock)

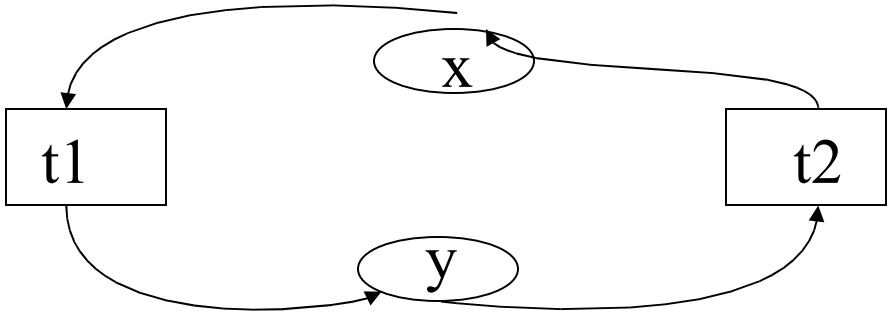
- Le verrouillage entraîne le problème suivant:

t1	t2
V(x) OK	
	V(y) OK
V(x) attendre	
	V(y) attendre

t1 attendre t2 et t2 attend t1 => interblocage

Détection d'interblocage

- Construire le graphe Qui attend Quoi (ou Qui)
- (on peut utiliser aussi le graphe de précédence)
- Nœuds= transactions et objets
- Arcs:
  - de l'objet vers Ti si Ti a verrouillé l 'objet
  - de Ti vers l 'objet si Ti attend



Gestion des interblocages

- 3 méthodes :
  - Temporisation :
  - Détection-guérison:
  - Prévention :

Détection d'interblocage

- Détection des cycles dans le graphes => interblocage..
- Choix d'une transaction à tuer «à sacrifier!» (la plus jeune :-) pour casser le cycle
- Relancer la transaction tuée plus tard..

Problème des objets Fantômes

- Relation EMP(no, dept, sal)

No	dept	sal
e1	d1	50
e2	d1	45
e3	d2	25
e4	d2	30
e5	d2	48

	T1		T2
A1	select max(sal) from EMP where dept= «d1»		
		B1	insert into EMP values(e6,d1,52)
		B2	delete from EMP where ne= e5
A2	select max(sal) from EMP where dept = «d2»		

Problème des objets Fantômes

Exécution 1	Exécution 2	Exécution 3
A1 A2 B1 B2	B1 B2 A1 A2	A1 B1 B2 A2
max d1=50 max d2: 48	max d1 = 52 max d2 = 30	max d1= 50 max d2= 30

- Pas de sérilialisation ….
- T1 ne peut pas verrouiller des n-uplets n'existent pas
- verrou sur **toute** la relation pour éviter insertions et suppressions de n-uplets

SQL et Transactions

- Début de transaction implicite (une instruction SQL)
- Fin de transaction
  - implicit (après chaque instruction, ..)
  - explicite: COMMIT pour valider, ROLLBACK pour annuler
- Type et degré d'isolation d 'une transaction
  - Action explicite de verrouillage et libération (Lock, Unlock)
  - Granule de verrouillage ? Tuple, relation, index, page, segment..

SQL et Transctions

- SQL2: SET TRANSACTION <mode><isolation>
- Mode: READ ONLY: lecture seulement, pas de mise à jour de la BD
- READ WRITE
- Niveau d ’isolation???

Niveaux d'isolation

Niveau d 'isolation	Lecture impropre	Lecture non reproductible	Fantômes
Level 0: *READ UNCOMMITED DATA*	Possible	possible	Possible
Level 1: *READ COMMITED DATA*	NON	Possible	Possible
Level 2: *REPEATABLE READ*	NON	NON	Possible
Level 3: *SERIALIZABLE*	NON	NON	NON

NB: pas de mise à jour perdue

SQL2: Isolation level

- Serializable:
  - T verrouille au début et libère à la fin 2PL strict.
  - Verrous sur ensemble d ’objets pour éviter les fantômes..
- Repeatable Read:
  - T lit seulement les modifications de transactions ayant validé (commit).
  - Aucun objet lu ou modifié par T n’est pas modifié par une autre transaction
  - verrou sur objet individuel et non sur collection, ce qui peut entraîner les fantômes..

SQL2: Isolation level

- Read Committed Data:
  - T lit seulement les mises à jour de transactions ayant validé commit).
  - Aucun objet modifié par T n ’est pas modifié par une autre transition
  - mais un objet lu par T peut avoir été modifié par une autre transaction..
  - Verrous exclusifs pour écrire et ne pas les libérer avant la fin
  - Verrous de lecture libérés de suite..

SQL2: Isolation level

- Read Uncommitted Data:
  - T peut lire des objets modifiés par une autre transaction
  - pas de verrous en lecture et mode «READ ONLY»

Scénario d ’exécution ..

Transactions sous Oracle

- Niveaux 1 et 3
- Undo segment
- Exemple:

Les transactions dans le SGBD sybase

select **@@isolation**

// the current isolation level….

- Modifier le niveau d'isolation

  set transaction isolation level 0

  // allows read uncommitted data

  set transaction isolation level 1

  // allows read uncommitted

  set transaction isolation level 3

  // serializable

Quelques variables utiles

Choix de mode de transaction

set chained on

// begin transaction is executed impicitly before: update,delete, open, fetch select…

select **@@tranchained**

// @@tranchained= 0 -> unchained mode (explicite mode)

// @@tranchained= 1 -> chained mode (implicite mode)

Quelques variables utiles

@@transtate: elle donne l 'état de la transaction

select @@transtate

// @@transtate = 0 -> transaction in progess..

// @@transtate = 1 -> transaction succeded (committed)

// @@transtate = 2 -> statement aborted (only previous state has aborted no effect on transaction)

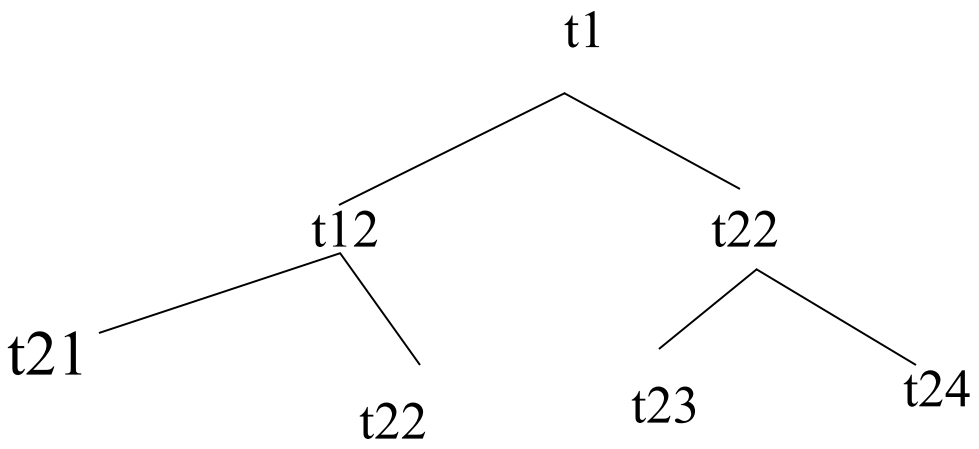
// @@transtate = 3 -> transaction aborted (rolledback)

Conclusion

- La gestion de transaction est assurée par un moniteur transactionnel
- Le moniteur, le journal, le verrou est transparents pour l'utilisateur..
- Tout se passe comme si l'utilisateur était seul avec sa base de données..

Modèle de transactions avancés

- Transactions imbriquées (Nested transactions) (Moss85)
- Avantage ??
  - Plus de parallélisme +
  - limiter l'effets de l 'annulation d 'une transaction..
- si t1 est annulée => toutes ses filles sont annulées
- l'annulation d'une fille n'a pas d 'impact sur la transaction mère …



Transaction imbriquée dans Sybase

```
begin tran
 select @@trancount
 /*trancount = 1*/
     begin transction
     select @@trancount
     /*trancount = 2*/
     a1
      begin transction
                  select @@trancount
                   /*trancount = 3*/
      commit tran
     commit tran
commit tran
                 select @@trancount
                   /*trancount = 0*/
```

Variable globale **@@trancount** numéro de transaction utilise avec les nested transactions

Transaction et procédures stockées

begin tran

update titles set….

insert into tiles…

exec myproc

delete titles where

si myproc est annulée
alors **update** et **insert** sont annulée mais delete est exécutée

create proc myproc
as
begin tran
statements
if … **rollback** tran
else commiy tran