

L2 Informatique, UE INF403

Gestion de données relationnelles et applications

Marie-Christine Fauvet

Université Grenoble Alpes – UFR IM²AG

2016/2017

Equipe pédagogique

- Catherine Berrut (professeur UGA, LIG)
TD et TP groupe 3 Catherine.Berrut@imag.fr
- Michaël Magi (ingénieur d'étude UGA, service informatique de l'UFR IM²AG)
TD et TP groupe 2 (Michael.Magi@univ-grenoble-alpes.fr)
- Seydou Doumbia (doctorant UGA/LIG MRIM)
TD et TP groupe 1 (Seydou.Doumbia@univ-grenoble-alpes.fr)
- Marie-Christine Fauvet (professeur UGA, LIG)
Cours (Marie-Christine.Fauvet@imag.fr)

Evaluation

Moyenne des interrogations hebdomadaires	int
2 comptes rendus de TP	cr1, cr2
CC1	$cc1 = (cr1 + cr2 + int)/3$
CC2 (partiel)	cc2
Examen	ex
Note d'UE	$(cc1 + cc2)/4 + ex/2$

- Le partiel a lieu entre le lu. 6/3/17 et le ve. 10/3/17
- Compte rendu No 1 (Agence) : à déposer¹ au plus tard le ve. 3/3/2017, 17H.
- Compte rendu No 2 : à rendre, sur moodle au plus tard le 12/5/2017, 17H.
- Les TPs sont faits en binôme, un seul compte rendu par binôme
- Les délais sont de rigueur : les CR rendus hors délai ne seront pas acceptés (la note attribuée sera 0).

Objectifs

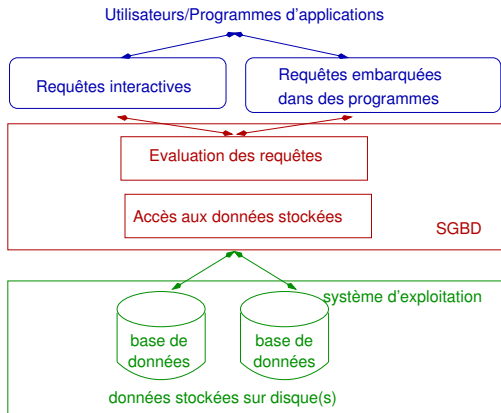
Maîtriser le modèle relationnel de données et SQL.

Etre sensibilisé à la mise en œuvre d'une application web, au dessus d'un Système de Gestion de Bases de Données.

- Le modèle relationnel de données
- Un langage relationnel : SQL
- Les bases de PhP et de HTML
- Les bases de la mise en œuvre d'une application de BD

Systèmes de Gestion de Bases de Données

Un Système de Gestion de Bases de Données (SGBD) est un ensemble de modules logiciels responsable du stockage et de l'accès à des informations.



Notion de Bases de Données

Une base de données contient des informations (d'une fraction) de la réalité, afin qu'elles soient interrogées, modifiées, éventuellement supprimées dans le futur

Une base de données est une collection de données structurées :

- qui modélisent une partie du monde réel
- conçues pour répondre à des besoins particuliers

Exemples

- Mon carnet d'adresses : les noms, prénoms, adresses et téléphones de mes amis, sont enregistrés dans mon carnet d'adresses (quelques kilos octets)
- *World Data Centre for Climate* : 220 terabytes² de données disponibles sur le web 6 petabytes³ d'autres données

²1 tera = 10^{12}

³1 peta = 10^{15}

Un système de gestion de bases de données

Un système logiciel conçu pour être la base des applications de bases de données.

- permettant un accès aux données indépendant de leur implantation (modèle relationnel, SQL)
- offrant un stockage et une recherche efficace (gestion de la mémoire et des disques)
- capable de gérer de nombreux accès simultanés (gestion des privilèges, des accès concurrents)
- capable d'exécuter de très nombreuses opérations (gestion des transactions)
- assurant une gestion fiable des données (sauvegarde, restauration)

Les SGBDs répondent à des problèmes similaires à ceux traités par les systèmes d'exploitation.

Finalement...

Un SGBD permet le stockage et l'accès à des informations, dans un contexte :

- **Partagé** : accès concurrents par plusieurs utilisateurs
- **Instable** : défauts logiciels/matériels toujours possibles

Un SGBD s'appuie sur des techniques qui font croire aux utilisateurs qu'ils sont dans un contexte :

- **Privé** : leur travail n'est pas affecté par les autres
- **Stable** : les données survivent à n'importe quel incident

bien sûr, avec des performances “acceptables”.

Quelques lectures

- P.-C. Scholl, M.-C. Fauvet, F. Lagnier, F. Maraninchi *Cours d'informatique : langages et programmation - chapitre 9* Masson, 1993.
- A. Meier *Introduction pratique aux bases de données relationnelles*, Springer, Collection IRIS., 2002.
- R. Elmasri, S.B. Navathe *Conception et architecture des bases de données relationnelles*, 4e édition - Pearson Education France, 2004. Traduction française de *Fundamentals of Database Systems*, Addison-Wesley

Ces ouvrages sont à la BU !

Un peu de culture

Le Modèle Relationnel de Données...

- A été introduit en 1970 par Ted Codd,
- Est attractif grâce à sa simplicité et ses fondements mathématiques,
- Utilise le concept de relation mathématique pour modéliser l'information,
- Ne peut pas être ignoré à cause de sa popularité.

Comment représenter l'information ?

Une histoire de magasin :

- John, Mary, Tom et Peter sont employés par un magasin,

John			
Mary			
Peter			
Tom			

Comment représenter l'information ?

Une histoire de magasin :

- John, Mary, Tom et Peter sont employés par un magasin,
- Le salaire de John est 120, celui de Mary est 130, etc...

John	120		
Mary	130		
Peter	110		
Tom	120		

Comment représenter l'information ?

Une histoire de magasin :

- John, Mary, Tom et Peter sont employés par un magasin,
- Le salaire de John est 120, celui de Mary est 130, etc...
- Mary habite à Wollongong, John et Peter à Randwick, etc...

John	120	Randwick	
Mary	130	Wollongong	
Peter	110	Randwick	
Tom	120	Botany Bay	

Comment représenter l'information ?

Une histoire de magasin :

- John, Mary, Tom et Peter sont employés par un magasin,
- Le salaire de John est 120, celui de Mary est 130, etc...
- Mary habite à Wollongong, John et Peter à Randwick, etc...
- John et Tom sont employés au rayon Toys, etc...

John	120	Randwick	Toys
Mary	130	Wollongong	Furniture
Peter	110	Randwick	Garden
Tom	120	Botany Bay	Toys

Opérations sur les relations

Interroger des relations

- Projection : Quels sont les noms de rayon ?
- Sélection : Donner les employés qui gagnent plus de 120
- Sélection + projection : Quelle est l'adresse de Tom ?
- Agrégation : Combien d'employés travaillent au rayon Toys ?

Opérations sur les relations

Interroger des relations

- Projection : Quels sont les noms de rayon ?
→ {<Toys>, <Furniture>, <Garden>}
- Sélection : Donner les employés qui gagnent plus de 120
- Sélection + projection : Quelle est l'adresse de Tom ?
- Agrégation : Combien d'employés travaillent au rayon Toys ?

Opérations sur les relations

Interroger des relations

- Projection : Quels sont les noms de rayon ?
→ {<Toys>, <Furniture>, <Garden>}
- Sélection : Donner les employés qui gagnent plus de 120
→ {<John, 120, Randwick, Toys>, <Mary, 130, Wollongong, Furniture>, <Tom, 120, Botany Bay, Toys>}
- Sélection + projection : Quelle est l'adresse de Tom ?
- Agrégation : Combien d'employés travaillent au rayon Toys ?

Opérations sur les relations

Interroger des relations

- Projection : Quels sont les noms de rayon ?
→ {<Toys>, <Furniture>, <Garden>}
- Sélection : Donner les employés qui gagnent plus de 120
→ {<John, 120, Randwick, Toys>, <Mary, 130, Wollongong, Furniture>, <Tom, 120, Botany Bay, Toys>}
- Sélection + projection : Quelle est l'adresse de Tom ?
→ {<Botany Bay>}
- Agrégation : Combien d'employés travaillent au rayon Toys ?

Opérations sur les relations

Interroger des relations

- Projection : Quels sont les noms de rayon ?
→ {<Toys>, <Furniture>, <Garden>}
- Sélection : Donner les employés qui gagnent plus de 120
→ {<John, 120, Randwick, Toys>, <Mary, 130, Wollongong, Furniture>, <Tom, 120, Botany Bay, Toys>}
- Sélection + projection : Quelle est l'adresse de Tom ?
→ {<Botany Bay>}
- Agrégation : Combien d'employés travaillent au rayon Toys ?
→ {<2>}

Mettre à jour une relation

- Le salaire de Mary a augmenté de 10%

John	120	Randwick	Toys
Mary	143	Wollongong	Furniture
Peter	110	Randwick	Garden
Tom	120	Botany Bay	Toys

Mettre à jour une relation

- Le salaire de Mary a augmenté de 10%
- Phil est maintenant employé au magasin, son salaire est 140, il est affecté au rayon Furniture, son adresse est à Newtown.

John	120	Randwick	Toys
Mary	143	Wollongong	Furniture
Peter	110	Randwick	Garden
Tom	120	Botany Bay	Toys
Phil	140	Newtown	Furniture

Ajouter un nouveau type d'information

Chaque rayon est dirigé par un employé

Deux solutions..... ajouter une(des) colonne(s)
ajouter une(des) relation(s)

Ajouter une colonne (#1) :

- Ajouter une marque pour chaque employé

John	yes	120	Randwick	Toys
Mary	yes	143	Wollongong	Furniture
Peter	yes	110	Randwick	Garden
Tom	no	120	Botany Bay	Toys
Phil	no	140	Newtown	Furniture

Dur à lire et à décoder ...

Ajouter une colonne (#2) :

- Ajouter un nom de chef à chaque rayon

John	120	Randwick	Toys	John
Mary	143	Wollongong	Furniture	Mary
Peter	110	Randwick	Garden	Peter
Tom	120	Botany Bay	Toys	John
Phil	140	Newtown	Furniture	Mary

Redondance des données ...

"John est le chef du rayon Toys" est dit deux fois

Ajouter une relation

John	Toys
Mary	Furniture
Peter	Garden

Les requêtes sont un peu plus compliquées ...
"Donner le salaire du chef du rayon Toys"

Ensemble : définition et quelques opérations

- Un ensemble est une collection d'éléments (entre accolades) différents deux à deux et reliés à un domaine particulier.

$$A = \{ \text{'Furniture'}, \text{'Toys'}, \text{'Garden'} \}$$
$$F = \{ p \in \text{ThePersons} : \text{Sexe}(p) = \text{'female'} \}$$

Ensemble : définition et quelques opérations

- Un ensemble est une collection d'éléments (entre accolades) différents deux à deux et reliés à un domaine particulier.

$A = \{ \text{'Furniture'}, \text{'Toys'}, \text{'Garden'} \}$

$F = \{ p \in \text{ThePersons} : \text{Sexe}(p) = \text{'female'} \}$

- Appartenance : $3 \in \{1, 3, 5, 6\}, 8 \notin \{1, 3, 5, 6\}$

- Produit cartésien d'ensembles (noté \times) :

$$\{1, 6, 3, 5\} \times A =$$

$$\{ \langle 1, 'Furniture' \rangle, \langle 1, 'Toys' \rangle, \langle 1, 'Garden' \rangle, \\ \langle 3, 'Furniture' \rangle, \langle 3, 'Toys' \rangle, \langle 3, 'Garden' \rangle, \\ \langle 5, 'Furniture' \rangle, \langle 5, 'Toys' \rangle, \langle 5, 'Garden' \rangle, \\ \langle 6, 'Furniture' \rangle, \langle 6, 'Toys' \rangle, \langle 6, 'Garden' \rangle \}$$

- Produit cartésien d'ensembles (noté \times) :
 $\{1, 6, 3, 5\} \times A =$
 $\{ \langle 1, 'Furniture' \rangle, \langle 1, 'Toys' \rangle, \langle 1, 'Garden' \rangle,$
 $\langle 3, 'Furniture' \rangle, \langle 3, 'Toys' \rangle, \langle 3, 'Garden' \rangle,$
 $\langle 5, 'Furniture' \rangle, \langle 5, 'Toys' \rangle, \langle 5, 'Garden' \rangle,$
 $\langle 6, 'Furniture' \rangle, \langle 6, 'Toys' \rangle, \langle 6, 'Garden' \rangle \}$
- Intersection : $\{1, 3, 5, 6\} \cap \{10, 5, 3, 9\} = \{5, 3\}$

- Produit cartésien d'ensembles (noté \times) :
 $\{1, 6, 3, 5\} \times A =$
 $\{ \langle 1, 'Furniture' \rangle, \langle 1, 'Toys' \rangle, \langle 1, 'Garden' \rangle,$
 $\langle 3, 'Furniture' \rangle, \langle 3, 'Toys' \rangle, \langle 3, 'Garden' \rangle,$
 $\langle 5, 'Furniture' \rangle, \langle 5, 'Toys' \rangle, \langle 5, 'Garden' \rangle,$
 $\langle 6, 'Furniture' \rangle, \langle 6, 'Toys' \rangle, \langle 6, 'Garden' \rangle \}$
- Intersection : $\{1, 3, 5, 6\} \cap \{10, 5, 3, 9\} = \{5, 3\}$
- Union :
 $\{1, 3, 5, 6\} \cup \{10, 5, 3, 9\} = \{1, 5, 3, 6, 10, 9\}$

- Produit cartésien d'ensembles (noté \times) :
 $\{1, 6, 3, 5\} \times A =$
 $\{ \langle 1, 'Furniture' \rangle, \langle 1, 'Toys' \rangle, \langle 1, 'Garden' \rangle,$
 $\langle 3, 'Furniture' \rangle, \langle 3, 'Toys' \rangle, \langle 3, 'Garden' \rangle,$
 $\langle 5, 'Furniture' \rangle, \langle 5, 'Toys' \rangle, \langle 5, 'Garden' \rangle,$
 $\langle 6, 'Furniture' \rangle, \langle 6, 'Toys' \rangle, \langle 6, 'Garden' \rangle \}$
- Intersection : $\{1, 3, 5, 6\} \cap \{10, 5, 3, 9\} = \{5, 3\}$
- Union :
 $\{1, 3, 5, 6\} \cup \{10, 5, 3, 9\} = \{1, 5, 3, 6, 10, 9\}$
- Différence (asymétrique) :
 $\{1, 3, 5, 6\} - \{10, 5, 3, 9\} = \{1, 6\}$
 $\{10, 5, 3, 9\} - \{1, 3, 5, 6\} = \{10, 9\}$

- Produit cartésien d'ensembles (noté \times) :
 $\{1, 6, 3, 5\} \times A =$
 $\{ \langle 1, 'Furniture' \rangle, \langle 1, 'Toys' \rangle, \langle 1, 'Garden' \rangle,$
 $\langle 3, 'Furniture' \rangle, \langle 3, 'Toys' \rangle, \langle 3, 'Garden' \rangle,$
 $\langle 5, 'Furniture' \rangle, \langle 5, 'Toys' \rangle, \langle 5, 'Garden' \rangle,$
 $\langle 6, 'Furniture' \rangle, \langle 6, 'Toys' \rangle, \langle 6, 'Garden' \rangle \}$
- Intersection : $\{1, 3, 5, 6\} \cap \{10, 5, 3, 9\} = \{5, 3\}$
- Union :
 $\{1, 3, 5, 6\} \cup \{10, 5, 3, 9\} = \{1, 5, 3, 6, 10, 9\}$
- Différence (asymétrique) :
 $\{1, 3, 5, 6\} - \{10, 5, 3, 9\} = \{1, 6\}$
 $\{10, 5, 3, 9\} - \{1, 3, 5, 6\} = \{10, 9\}$
- Inclusion :
 $\{ \} \subseteq \{10, 5, 3, 9\}$ ($\{ \}$ est aussi noté \emptyset)
 $\{9, 10\} \subseteq \{10, 5, 3, 9\}$
 $\{9, 10\} \subset \{10, 5, 3, 9\}$
 $\{9, 10, 3, 5\} \not\subseteq \{10, 5, 3, 9\}$
 $\{1, 9, 10, 3, 5, 7\} \not\subseteq \{10, 5, 3, 9\}$

Domaine, Relation, Attribut, Schéma

- Un domaine est un ensemble de valeurs atomiques (chaînes, nombres,..). {'Furniture', 'Toys', 'Garden'}, entiers > 100

Domaine, Relation, Attribut, Schéma

- Un domaine est un ensemble de valeurs atomiques (chaînes, nombres,...). $\{'Furniture', 'Toys', 'Garden'\}$, entiers > 100
- Une relation est un sous-ensemble du produit cartésien d'un ensemble de domaines.

$$\{ \langle 'John', 120 \rangle, \langle 'Mary', 130 \rangle, \langle 'Peter', 110 \rangle, \langle 'Tom', 120 \rangle \} \subseteq$$

$$\{ 'John', 'Mary', 'Peter', 'Tom' \} \times \text{entiers} > 100$$

Domaine, Relation, Attribut, Schéma

- Un domaine est un ensemble de valeurs atomiques (chaînes, nombres,..). $\{\text{'Furniture'}, \text{'Toys'}, \text{'Garden'}\}$, entiers > 100
- Une relation est un sous-ensemble du produit cartésien d'un ensemble de domaines.

$$\{ \langle \text{'John'}, 120 \rangle, \langle \text{'Mary'}, 130 \rangle, \langle \text{'Peter'}, 110 \rangle, \langle \text{'Tom'}, 120 \rangle \} \subseteq$$

$$\{ \text{'John'}, \text{'Mary'}, \text{'Peter'}, \text{'Tom'} \} \times \text{entiers} > 100$$
- Un attribut indique le rôle joué par un domaine dans une relation.
domaine(Salary) = entiers > 100

Domaine, Relation, Attribut, Schéma

- Un domaine est un ensemble de valeurs atomiques (chaînes, nombres,..). $\{'Furniture', 'Toys', 'Garden'\}$, entiers > 100
- Une relation est un sous-ensemble du produit cartésien d'un ensemble de domaines.

$$\{ \langle 'John', 120 \rangle, \langle 'Mary', 130 \rangle, \langle 'Peter', 110 \rangle, \langle 'Tom', 120 \rangle \} \subseteq \{ 'John', 'Mary', 'Peter', 'Tom' \} \times \text{entiers} > 100$$
- Un attribut indique le rôle joué par un domaine dans une relation.
 $\text{domaine}(\text{Salary}) = \text{entiers} > 100$
- La structure de la relation est donnée par son nom et par un ensemble d'attributs. *Employees (firstname, salary, address, dept)*

Diagram illustrating the structure and data of a relation named **Employees**.

nom de la relation (name of the relation): **Employees**

attributs (attributes): **firstname**, **salary**, **address**, **dept**

structure de la relation (structure of the relation): The set of attributes.

nuplets (tuples): The rows of data.

valeur de la relation (value of the relation): The set of tuples.

firstname	salary	address	dept
John	120	Randwick	Toys
Mary	143	Wollongong	Furniture
Peter	110	Randwick	Garden
Tom	120	Botany Bay	Toys
Phil	140	Newtown	Furniture

Les attributs et les n-uplets n'ont pas d'ordre.

Interprétation

L'interprétation d'une relation est un prédicat :

Employees (firstname, salary, address, dept)

/ $\langle n, s, a, d \rangle \in \text{Employees} \iff$ l'employé identifié par son nom n gagne un salaire s . Il habite à l'adresse a et est affecté au rayon d . */*

Le prédicat est utile pour comprendre le schéma de la relation et le documenter.

Contraintes Relationnelles

- *Contraintes de domaine* :
 $domain(A)=T$ spécifie que les valeurs de A doivent être du type T .

Contraintes Relationnelles

- *Contraintes de domaine* :
 $\text{domain}(A)=T$ spécifie que les valeurs de A doivent être du type T .
- *Contraintes d'identification* :
 \underline{X} spécifie une contrainte d'unicité telle que les n -uplets de la relation sont distincts deux à deux pour X (X un ensemble d'attributs).

Contraintes Relationnelles

- *Contraintes de domaine :*
 $\text{domain}(A)=T$ spécifie que les valeurs de A doivent être du type T .
- *Contraintes d'identification :*
 \underline{X} spécifie une contrainte d'unicité telle que les n -uplets de la relation sont distincts deux à deux pour X (X un ensemble d'attributs).
- *Contraintes d'intégrité référentielle :*
 R projetée sur l'attribut X se réfère à S projetée sur l'attribut Y : tous les n -uplets de R , restreints à X doivent avoir un n -uplet correspondant dans S restreinte à Y . Ce que l'on note $R[X] \subseteq S[Y]$.

Employees (firstname, salary, address, dept) /* *firstname est l'identifiant* */
 /* $\langle n, s, a, d \rangle \in \text{Employees} \iff$ l'employé identifié par son nom n gagne un
 salaire s . Il habite à l'adresse a et est affecté au rayon d . */
 Leaderships (boss, dept) /* 2 identifiants : boss et dept */
 /* $\langle b, d \rangle \in \text{Leaderships} \iff$ l'employé b est affecté au rayon d . */

Les domaines :

domaine (firstname) = domaine (boss) =
 domaine (address) = domaine (dept) = chaînes
 domaine (salary) = entiers > 100

Contraintes d'intégrité référentielle :

Leaderships[boss, dept] \subseteq Employees[firstname, dept]

Préliminaires

SQL est un langage ANSI/ISO pour interroger et manipuler des données relationnelles.

Conçu pour être un langage lisible par les personnes.

- Opérations de définition des données
- Opérations de modification des données
- Opérations relationnelles
- Opérations d'agrégation

Préliminaires

SQL est un langage ANSI/ISO pour interroger et manipuler des données relationnelles.

Conçu pour être un langage lisible par les personnes.

- Opérations de définition des données
- Opérations de modification des données
- Opérations relationnelles
- Opérations d'agrégation

Facile d'exprimer des requêtes simples
TRÈS difficile d'exprimer des requêtes compliquées !

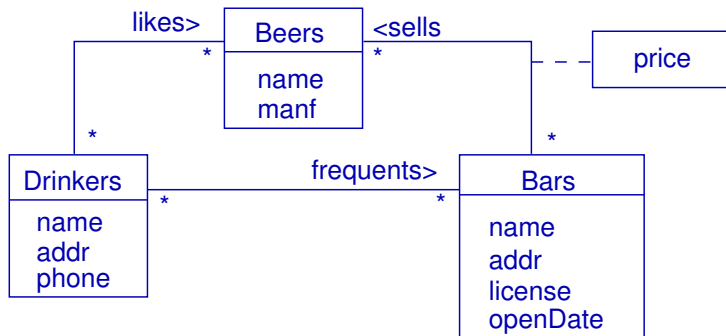
SQL : histoire⁴

Année	Appellation	Commentaires
1986	SQL-86 ou SQL-87	Édité par l'ANSI puis adopté par l'ISO en 1987.
1992	SQL-92 (en) alias SQL2	Révision majeure.
1999	SQL-99 (en) alias SQL3	Expressions rationnelles, requêtes récursives, déclencheurs, types non-scalaires et quelques fonctions orientées objet (les deux derniers points sont quelque peu controversés et pas encore largement implémentés).
2003	SQL:2003 (en)	Introduction de fonctions pour la manipulation XML, ordres standardisés et colonnes avec valeurs auto-produites (y compris colonnes d'identité).

SQL a survécu (et continu de survivre) à la montée des systèmes de gestion de bases de données à objets.

⁴Source : https://fr.wikipedia.org/wiki/Structured_Query_Language

Une BD jouet⁵



⁵Le schéma de cette base de données est tirée de : *Database Systems: The Complete Book (2e édition)*, écrit par Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom, 2008. Les valeurs sont d'inspiration personnelle.

Spécification des relations

Drinkers (name, addr, phone) /* *name est l'identifiant.* */

Beers (name, manf)

Bars (name, addr, license, openDate)

Likes (drinker, beer) /* *drinker, beer est l'identifiant.* */

Likes[drinker] \subseteq Drinkers[name]

Likes[beer] \subseteq Beers[name]

Sells (bar, beer, price)

price > 0

Sells[beer] \subseteq Beers[name]

Sells[bar] \subseteq Bars[name]

Frequents (drinker, bar)

Frequents[drinker] \subseteq Drinkers[name]

Frequents[bar] \subseteq Bars[name]

Il faudrait décrire aussi chacun des domaines.

Valeurs des relations

Bars

Name	Addr	License	openDate
Australia Hotel	The Rocks	123456	12/1/1940
Coogee Bay Hotel	Coogee	966500	31/8/1980
Lord Nelson	The Rocks	123888	11/11/1920
Marble Bar	Sydney	122123	1/4/2001
Regent Hotel	Kingsford	987654	29/2/2000
Rose Bay Hotel	Rose Bay	966410	31/8/2000
Royal Hotel	Randwick	938500	26/6/1986

Drinkers

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321
Marie	Rose Bay	9371-2126

Notion de requête

Une requête est un *programme déclaratif* pour retrouver des données d'une base de données. 2 modes :

- Interprété : le résultat est affiché sur la sortie courante.
- Encapsulé dans un autre programme : le résultat est traité par le programme lui-même.

Notion de requête

Une requête est un *programme déclaratif* pour retrouver des données d'une base de données. 2 modes :

- Interprété : le résultat est affiché sur la sortie courante.
- Encapsulé dans un autre programme : le résultat est traité par le programme lui-même.

Exemple :

```
select bar, price
from Sells
where beer = 'Victoria Bitter'
```

```
/* un ensemble de noms d'attributs */
/* un ensemble de noms de relations */
/* une expression booléenne */
```

Notion de requête

Une requête est un *programme déclaratif* pour retrouver des données d'une base de données. 2 modes :

- Interprété : le résultat est affiché sur la sortie courante.
- Encapsulé dans un autre programme : le résultat est traité par le programme lui-même.

Exemple :

```
select bar, price
from Sells
where beer = 'Victoria Bitter'
```

```
/* un ensemble de noms d'attributs */
/* un ensemble de noms de relations */
/* une expression booléenne */
```

*Donner dans quel bar et pour quel prix la bière
'Victoria Bitter' est vendue.*

Avec Oracle....

```
SQL> select bar, price  
2 from Sells  
3 where beer = 'Victoria Bitter';
```

BAR	PRICE
Coogee Bay Hotel	2.3
Marble Bar	2.8
Regent Hotel	2.2
Royal Hotel	2.3

4 rows selected

```
SQL>
```

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

En SQL :

```
select      /* projection */  
from        /* relations (une ou plusieurs) */  
where      /* sélection */
```

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

En SQL :

```
select  
from Bars  
where
```

```
/* projection */  
/* relations (une ou plusieurs) */  
/* sélection */
```

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

En SQL :

```
select  
from Bars  
where addr = 'Sydney'
```

```
/* projection */  
/* relations (une ou plusieurs) */  
/* sélection */
```


Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

En SQL :

```
select name, addr  
from Bars  
where addr = 'Sydney'
```

```
/* projection */  
/* relations (une ou plusieurs) */  
/* sélection */
```

Dans la requête *select ... from ... where P*, *P* est un prédicat construit avec, soit :

- Une simple condition de la forme :
<nom att.> <op. comp.> <nom att.> ou
<nom att.> <op. comp.> <val. cst.>

Dans la requête *select ... from ... where P*, *P* est un prédicat construit avec, soit :

- Une simple condition de la forme :
 <nom att.> <op. comp.> <nom att.> ou
 <nom att.> <op. comp.> <val. cst.>
- Ou une expression booléenne complexe de la forme :
 <cond.> <op. bool.> <cond.>
 où <cond.> est soit une condition basique soit une expression booléenne complexe.

Dans la requête *select ... from ... where P*, *P* est un prédicat construit avec, soit :

- Une simple condition de la forme :
 $\langle \text{nom att.} \rangle \langle \text{op. comp.} \rangle \langle \text{nom att.} \rangle$ ou
 $\langle \text{nom att.} \rangle \langle \text{op. comp.} \rangle \langle \text{val. cst.} \rangle$
- Ou une expression booléenne complexe de la forme :
 $\langle \text{cond.} \rangle \langle \text{op. bool.} \rangle \langle \text{cond.} \rangle$
 où $\langle \text{cond.} \rangle$ est soit une condition basique soit une expression booléenne complexe.

Exemple d'opérateurs de comparaison : $=$, \neq , $<$, $>$, etc.

Opérateurs booléens : not, and, or

P peut contenir des parenthèses

Évaluer les expressions suivantes :

- A and B or C
- A and (B or C)
- not A and A
- A and not A
- not A and C

lorsque A=true, B=false, C=true

Valeurs multiples (doublons)

Donner les bières qui coûtent moins de \$2.5

Le résultat attendu est :

Beer
New
Old
Victoria Bitter

Valeurs multiples (doublons)

Donner les bières qui coûtent moins de \$2.5

Le résultat attendu est :

Beer
New
Old
Victoria Bitter

En SQL :

```
select Beer from Sells
where price <= 2.5
```

Et le résultat est :

Beer
New
Old
Victoria Bitter
New
Victoria Bitter
New
Old
Victoria Bitter

Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
select distinct Beer from Sells where price <= 2.5
```


Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
select distinct Beer from Sells where price <= 2.5
```

A utiliser avec précaution :

Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
select distinct Beer from Sells where price <= 2.5
```

A utiliser avec précaution :

- Penser au coût de cette opération !

Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
select distinct Beer from Sells where price <= 2.5
```

A utiliser avec précaution :

- Penser au coût de cette opération !
- Souvent inutile : `select name from Drinkers`

Renommer les attributs dans un select

Le renommage d'attributs est implémenté via la clause AS dans la clause select

```
select name as Drinkers, addr, phone from Drinkers
```

Affichage :

drinkers	addr	phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321
Marie	Rose Bay	9371-2126
Adrian	Redfern	9371-1244

Ordonner les nuplets

La clause order by s'applique uniquement aux attributs contenus dans la clause select :

```
select bar, beer, price from Sells
order by price desc, bar asc
```

Résultat :

bar	beer	price
Lord Nelson	Three Sheets	3.75
Lord Nelson	Old Admiral	3.75
Australia Hotel	Burraborang Bock	3.5
Coogee Bay Hotel	Sparkling Ale	2.8
Marble Bar	New	2.8
Marble Bar	Victoria Bitter	2.8
Marble Bar	Old	2.8
Coogee Bay Hotel	Old	2.5
...		

Notation qualifiée

Le nom complet d'un attribut est qualifié par un nom de relation :

```
select Sells.bar, Sells.beer, Sells.price  
from Sells join Frequents on (Sells.bar = Frequents.bar)  
where Frequents.drinker = 'John'
```

- Chaque attribut est qualifié par la relation à laquelle il appartient (plus facile à lire)
- Nécessaire pour les attributs définis dans plus d'une relation de la clause from.

Notation qualifiée avec raccourcis

Une relation peut être renommée dans la clause from :

```
select S.Bar, S.Beer, S.Price  
from Sells S join Frequents F on (S.Bar = F.Bar)  
where F.Drinker = 'John'
```

Produit relationnel (interne)

```
select  
from R1 join R2 on .. join Rp on ...  
where
```

/ → produits de relations */*

Produit relationnel (interne)

```
select  
from R1 join R2 on .. join Rp on ...  
where
```

/ → produits de relations */*

Produit relationnel (interne)

```
select A1, A2, ..., An  
from R1 join R2 on .. join Rp on ...  
where C
```

/ → projection */*

/ → produits de relations */*

/ → sélection */*

Produit relationnel (interne)

```
select A1, A2, ..., An
from R1 join R2 on .. join Rp on ...
where C
```

/ → projection */*

/ → produits de relations */*

/ → sélection */*

Pour chaque bière vendue par l'Australia Hotel, donner son prix et son brasseur

Produit relationnel (interne)

```
select A1, A2, ..., An
from R1 join R2 on .. join Rp on ...
where C
```

/ → projection */*
/ → produits de relations */*
/ → sélection */*

Pour chaque bière vendue par l'Australia Hotel, donner son prix et son brasseur

En SQL :

```
select Beer, Price, Manf
from Beers join Sells on (Name = Beer)
where Bar = 'Australia Hotel'
```

/ projection */*
/ condition de produit */*
/ condition de sélection */*

/ La clause "on" peut contenir n'importe quel prédicat. */*

Produit relationnel (interne)

```
select A1, A2, ..., An
from R1 join R2 on .. join Rp on ...
where C
```

/ → projection */*

/ → produits de relations */*

/ → sélection */*

Pour chaque bière vendue par l'Australia Hotel, donner son prix et son brasseur

En SQL :

```
select Beer, Price, Manf
from Beers join Sells on (Name = Beer)
where Bar = 'Australia Hotel'
```

/ projection */*

/ condition de produit */*

/ condition de sélection */*

/ La clause "on" peut contenir n'importe quel prédicat. */*

Pour exprimer le produit entre deux (ou plus) relations

Produit naturel: forme 1

La condition de produit porte sur tous les attributs communs aux deux relations.

```
select bar, S.beer, S.price
```

```
/* L'attribut bar apparaît une seule fois dans le schéma résultat */
```

```
from Sells S natural join Frequents F
```

```
/* L'attribut bar est commun aux deux relations */
```

```
where F.drinker = 'John'
```

L'attribut bar (défini dans Sells et Frequents) ne peut être préfixé par aucun nom de relation.

Produit naturel : forme 2

La condition de produit porte sur un sous-ensemble des attributs en commun

```
select B.name, addr, D.name  
from Bars B join Drinkers D using (addr)  
/* La condition de produit s'applique sur l'attribut commun addr */
```

Une autre version :

```
select B.name, B.addr, D.name  
from Bars B join Drinkers D on (B.addr=D.addr)  
/* La condition de produit est explicite. */
```

Produit naturel : aucun attribut en commun

```
select ...  
from Bars B natural join Sells  
where B.name = 'Coogee Bay Hotel'
```

- Quel est le schéma de la relation resultat ?
- La valeur de la relation resultat est le produit cartésien de Bars et de Sells

Produit naturel : attention

```
select ... from Drinkers natural join Bars B where B.name = 'Coogee Bay Hotel'
```

- Quel est le schéma de la relation resultat ?
- La valeur de la relation est vide (il n'y a personne dont le nom est le nom d'un bar et dont l'adresse est celle d'un bar).

Produit naturel et produit relationnel dans la même clause from

Pour chaque bière, donner son nom, son brasseur, le nom des personnes qui l'apprécient et pour chacune de ces personnes, les bars qu'elle fréquente, et pour chaque bar qui la vend, le nom du bar et le prix de la bière.

```
select distinct B.name, B.manf, drinker, S.bar, S.price
from Beers B join Likes L on (B.name=L.beer)
      natural join Frequents F
      join Sells S on (B.name=S.beer F.bar=S.bar)
```

Changeons l'ordre des produits : l'expression ci-dessous est-elle correcte ?

```
select distinct B.name, B.manf, drinker, bar, price
from Beers B join Likes L on (B.name=L.beer)
      join Sells S on (B.name=S.beer)
      natural join Frequents F
```

Les produits ont tous la même priorité et ils sont évalués de gauche à droite. L'utilisation de parenthèses permet de forcer l'ordre d'évaluation.

Produit cartésien

```
select ....  
from Bars cross join Drinkers  
where Drinkers.name = 'John'
```

Produits : en résumé

Soient R et S définies comme : R (X, Y, Z) et S (Y, Z, T)

- Produit cartésien : `from R cross join S`
schéma : R.X, R.Y, R.Z, S.Y, S.Z, S.T
- Produit relationnel : `from R join S on (P)`
schéma : R.X, R.Y, R.Z, S.Y, S.Z, S.T
P est un prédicat valide sur R.X, R.Y, R.Z, S.Y, S.Z, S.T
La condition de produit est P
- Produit naturel (forme 1) : `from R natural join S`
schéma : R.X, Y, Z, S.T
La condition de produit est $R.Y = S.Y$ and $R.Z = S.Z$ (tous les attributs communs aux opérandes sont concernés)
- Produit naturel (forme 2) : `from R join S using (Y)`
schéma : R.X, Y, R.Z, S.Z, S.T
La condition de produit est $R.Y = S.Y$

Ambigüité de nom dans une requête

Plusieurs occurrences du même nom dans une requête

Pour chaque bar que John fréquente, donner son nom, les bières qu'il vend et leur prix.

```
select bar, beer, price
from Sells join Frequents on (bar = bar)
where drinker = 'John'
ERROR at line 2:
ORA-00918: column ambiguously defined
```

Citation multiple d'une même relation

Donner les paires de consommateurs différents qui aiment la même bière

Citation multiple d'une même relation

Donner les paires de consommateurs différents qui aiment la même bière

```
select L1.drinker, L2.drinker  
from Likes L1 join Likes L2  
on (L1.beer = L2.beer and L1.drinker <> L2.drinker)
```

Citation multiple d'une même relation

Donner les paires de consommateurs différents qui aiment la même bière

```
select L1.drinker, L2.drinker  
from Likes L1 join Likes L2  
on (L1.beer = L2.beer and L1.drinker <> L2.drinker)
```

Questions :

La clause *select distinct* est-elle nécessaire pour éliminer les doublons ?

Comment assurer l'antisymétrie ?

R est antisymétrique si :

$$\langle X, Y \rangle \in R \implies (\langle Y, X \rangle \notin R \text{ or } X = Y)$$

On sait déjà que pour tout X, $\langle X, X \rangle \notin R$

Comment assurer l'antisymétrie ?

R est antisymétrique si :

$$\langle X, Y \rangle \in R \implies (\langle Y, X \rangle \notin R \text{ or } X = Y)$$

On sait déjà que pour tout X, $\langle X, X \rangle \notin R$

```
select distinct L1.drinker, L2.Drinker
```

```
from Likes L1 join Likes L2
```

```
on (L1.Beer = L2.Beer and L1.Drinker < L2.Drinker)
```

Union, Intersection, Différence

Dans la suite les requêtes Q1 et Q2 sont de la forme `select... from....` dont les schémas sont compatibles :

- `select A, B from...` n'est pas compatible avec `select C from...`
- `select A, B from...` est compatible avec `select C, D from...` ssi A et C sont comparables, ainsi que B et D.

Opérateurs

- Union [all]
Pas d'élimination des doublons avec l'option all
- Intersect (intersection)
- Minus (différence)

Les doublons sont éliminés sauf lors de l'utilisation de union all.

Différence : exemple

Donner les bières vendues à moins de \$3 et que John n'aime pas

```
select Beer from Sells where Price < 3  
minus
```

```
select Beer from Likes where Drinker = 'John'
```

Les deux arguments doivent construire des relations de schémas compatibles.

Intersection : exemple

Donner les consommateurs et les bières tels que le consommateur aime la bière et fréquente un bar qui la vend.

```
select Drinker, Beer from Likes  
intersect  
select Drinker, Beer from Sells natural join Frequents
```

une autre version:

```
select distinct Drinker, Beer  
from Likes natural join Sells natural join Frequents  
/* Les produits sont évalués de gauche à droite. */
```

Union : exemple

Donner les consommateurs qui aiment la Sparkling Ale ou fréquentent le bar Lord Nelson.

```
select Drinker from Likes  
where Beer = 'Sparkling Ale'  
union  
select Drinker from Frequents  
where Bar = 'Lord Nelson'
```



Drinker
Gernot
Justin
John

Union : exemple

Donner les consommateurs qui aiment la Sparkling Ale ou fréquentent le bar Lord Nelson.

```
select Drinker from Likes
where Beer = 'Sparkling Ale'
union
select Drinker from Frequents
where Bar = 'Lord Nelson'
```



Drinker
Gernot
Justin
John

```
select Drinker from Likes
where Beer = 'Sparkling Ale'
union all
select Drinker from Frequents
where Bar = 'Lord Nelson'
```



Drinker
Gernot
Gernot
Justin
John

Requêtes dans la clause From

Principe : la clause from peut contenir des expressions SQL

Donner le nom et le brasseur des bières vendues à moins de \$3 et que John n'aime pas

On a déjà vu que la requête suivante retournait le nom des bières vendues à moins de \$3 et que John n'aime pas :

```
select Beer from Sells where Price < 3  
minus  
select Beer from Likes where Drinker = 'John'
```

Soit R la relation construite par cette expression.

La requête est donc (en utilisant le nom R) :

```
select Beer, Manf
from Beers join R on (Beer = Name)
```

En substituant R par son expression :

```
select Beer, Manf
from Beers join (select Beer from Sells where Price < 3
                minus
                select Beer from Likes where Drinker = 'John') R
/* < b > ∈ R ⇔ b est une bière vendue à moins de $3 et John
   n'aime pas b. */
on (Beer = Name)
```

La requête est donc (en utilisant le nom R) :

```
select Beer, Manf
from Beers join R on (Beer = Name)
```

En substituant R par son expression :

```
select Beer, Manf
from Beers join (select Beer from Sells where Price < 3
                minus
                select Beer from Likes where Drinker = 'John') R
/* < b > ∈ R ⇔ b est une bière vendue à moins de $3 et John
   n'aime pas b. */
on (Beer = Name)
```

Un nom doit être fixé pour la sous-requête (ici on choisit d'utiliser R).

Intérêts

- Décomposition de la requête en sous-requêtes plus simples.
- Expression et test des sous-requêtes indépendamment les unes des autres.

Règle : la relation imbriquée dans la clause from DOIT être spécifiée.

Intérêts

- Décomposition de la requête en sous-requêtes plus simples.
- Expression et test des sous-requêtes indépendamment les unes des autres.

Règle : la relation imbriquée dans la clause from DOIT être spécifiée.

Utilisation maladroite :

```
select beer, price
```

```
from (select distinct beer from Sells where price  $\geq$  10) X
```

/ < b > \in X \iff il existe un bar qui vend la bière b pour plus de \$10. */*

```
natural join Sells
```

```
where bar = 'Coogee Bay Hotel'
```

Intérêts

- Décomposition de la requête en sous-requêtes plus simples.
- Expression et test des sous-requêtes indépendamment les unes des autres.

Règle : la relation imbriquée dans la clause from DOIT être spécifiée.

Utilisation maladroite :

```
select beer, price
from (select distinct beer from Sells where price  $\geq$  10) X
/*  $\langle b \rangle \in X \iff$  il existe un bar qui vend la bière  $b$  pour plus de $10. */
natural join Sells
where bar = 'Coogee Bay Hotel'
```

Une expression plus concise, plus facile à lire :

```
select beer, price from Sells
where price  $\geq$  10 and bar = 'Coogee Bay Hotel'
```

Introduction de noms intermédiaires (clause with..as ())

La clause with.. as.. permet d'introduire des noms intermédiaires :

```
with R as (  
    select Beer from Sells where Price < 3  
    minus  
    select Beer from Likes where Drinker = 'John'  
)  
select .. from R ...
```

La durée de vie du nom introduit par la clause with est celle de la requête.

Clause with..as (select ...) : exemple avec Oracle

```
SQL > with R as (  
      2 select Beer from Sells where Price < 3  
      3 minus  
      4 select Beer from Likes where Drinker = 'John'  
5 )  
6 select * from R ;
```

La partie incluse entre with.. et le caractère ; constitue la requête courante.

Agrégation

Pour réduire une liste de valeurs à une valeur.

- $\text{Count} (*) \rightarrow$ nombre de n-uplets
- $\text{Count} (A) \rightarrow$ nombre de valeurs pour A
- $\text{Count} (\text{distinct } A) \rightarrow$ nombre de valeurs différentes pour A
- $\text{Avg} (A) \rightarrow$ valeur moyenne des valeurs de A
- $\text{Min} (A)$ (resp. Max) \rightarrow valeur minimum (resp. maximum) des valeurs de A
- $\text{Sum} (A) \rightarrow$ somme des valeurs de A

Agrégation : exemples

Quel est le prix moyen des bières vendues à l'Australia Hotel ?

Agrégation : exemples

Quel est le prix moyen des bières vendues à l'Australia Hotel ?

```
select      price  from Sells where Bar = 'Australia Hotel'
```

Agrégation : exemples

Quel est le prix moyen des bières vendues à l'Australia Hotel ?

```
select avg (price) from Sells where Bar = 'Australia Hotel'
```

Agrégation : exemples

Quel est le prix moyen des bières vendues à l'Australia Hotel ?

```
select avg (price) from Sells where Bar = 'Australia Hotel'
```

Combien de bars sont situés à The Rocks ?

Agrégation : exemples

Quel est le prix moyen des bières vendues à l'Australia Hotel ?

```
select avg (price) from Sells where Bar = 'Australia Hotel'
```

Combien de bars sont situés à The Rocks ?

```
select      *  from Bars where Addr = 'The Rocks'
```

Agrégation : exemples

Quel est le prix moyen des bières vendues à l'Australia Hotel ?

```
select avg (price) from Sells where Bar = 'Australia Hotel'
```

Combien de bars sont situés à The Rocks ?

```
select count (*) from Bars where Addr = 'The Rocks'
```

Agrégation : d'autres exemples

Combien de bars vendent des bières pour moins de \$2.5?

`select count (Bar) from Sells where price < 2.5`

Attention : la requête est incorrecte, pourquoi ?

Agrégation : d'autres exemples

Combien de bars vendent des bières pour moins de \$2.5?

`select count (Bar) from Sells where price < 2.5`

Attention : la requête est incorrecte, pourquoi ?

On compte un bar autant de fois qu'il vend une bière pour moins de \$2.5

Agrégation : d'autres exemples

Combien de bars vendent des bières pour moins de \$2.5?

```
select count (Bar) from Sells where price < 2.5
```

Attention : la requête est incorrecte, pourquoi ?

On compte un bar autant de fois qu'il vend une bière pour moins de \$2.5

Expression correcte de la requête :

```
select count (distinct Bar) from Sells where price < 2.5
```

Agrégation : d'autres exemples

Combien de bars vendent des bières pour moins de \$2.5?

```
select count (Bar) from Sells where price < 2.5
```

Attention : la requête est incorrecte, pourquoi ?

On compte un bar autant de fois qu'il vend une bière pour moins de \$2.5

Expression correcte de la requête :

```
select count (distinct Bar) from Sells where price < 2.5
```

Quels sont les bars qui vendent la bière New pour le prix le plus faible ?

```
select bar
```

```
from Sells join
```

```
(select min(price) as minP from Sells
```

```
where beer='New') Min
```

*/ * < m > ∈ Min ⇔ m est le prix le plus faible de la New parmi tous les bars. */*

```
on (price = minP)
```

```
where beer='New'
```

Partition

Partitionner une relation pour appliquer une agrégation sur chaque classe séparément

Combien de bières aime chaque consommateur ?

Drinker	Beer
Adam	3
Gernot	2
John	4
Justin	3

Résultat attendu :

Comment faire une partition

1. Construire une partition sur Likes

Adam	Crown Lager
Adam	Fosters
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Fosters
John	Three Sheets
Justin	Sparkling Ale
Justin	Fosters
Justin	Victoria Bitter

Comment faire une partition

1. Construire une partition sur Likes

Adam	Crown Lager
Adam	Fosters
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Fosters
John	Three Sheets
Justin	Sparkling Ale
Justin	Fosters
Justin	Victoria Bitter

En SQL : `select ...
from Likes
group by Drinker`

2. Réduire à un n -uplet chaque classe de la partition

liste de Drinker (critère de partition)

→ Drinker (une des valeurs)

liste de Beer

→ entier (nombre de valeurs)

2. Réduire à un n -uplet chaque classe de la partition

liste de Drinker (critère de partition)

→ Drinker (une des valeurs)

liste de Beer

→ entier (nombre de valeurs)

En SQL :

```
select Drinker, count (Beer)
from Likes
group by Drinker
```


Impact sur la sémantique de la clause select

Dans une requête contenant la clause group by, la clause select contient uniquement :

- Un ou plusieurs attributs parmi ceux du critère de la partition
- Une ou plusieurs agrégations appliquées aux autres attributs

Expression incorrecte :

```
select Drinker, Addr, count (Beer)
from Likes join Drinkers
on (Drinker=Name)
group by Drinker
```

Expression correcte :

```
select Drinker, Addr, count (Beer)
from Likes join Drinkers
on (Drinker=Name)
group by Drinker, Addr
```

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

from Sells S natural join Frequents F

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

```
from Sells S natural join Frequents F  
group by Bar
```

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

```
from Sells S natural join Frequents F  
group by Bar  
having count (distinct S.Beer) > 2
```

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

```
select Bar,  
       count (distinct S.Beer) as NbBeers,  
       count (distinct F.Drinker) as NbDrinkers  
from Sells S natural join Frequents F  
group by Bar  
having count (distinct S.Beer) > 2
```

Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Les bars qui vendent toutes les bières sont ceux, qui dans la relation Sells sont associés à un ensemble (soit A) de bières égal à l'ensemble des bières (vendues ou connues) soit B.

Or, on sait que : $|A| = |B| \wedge A \subseteq B \implies A = B$

En SQL, tester l'égalité de deux ensembles A et B lorsque $A \subseteq B$ revient donc à tester l'égalité des cardinalités des ensembles $|A|$ et $|B|$.

Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Les bars qui vendent toutes les bières sont ceux, qui dans la relation Sells sont associés à un ensemble (soit A) de bières égal à l'ensemble des bières (vendues ou connues) soit B.

Or, on sait que : $|A| = |B| \wedge A \subseteq B \implies A = B$

En SQL, tester l'égalité de deux ensembles A et B lorsque $A \subseteq B$ revient donc à tester l'égalité des cardinalités des ensembles $|A|$ et $|B|$.

Pour chaque bar, combien de bières ?

Soit X1(bar,nbBeers) la relation associée :

```
select bar, count(beer) as nbBeers from Sells group by bar
```


Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Les bars qui vendent toutes les bières sont ceux, qui dans la relation Sells sont associés à un ensemble (soit A) de bières égal à l'ensemble des bières (vendues ou connues) soit B.

Or, on sait que : $|A| = |B| \wedge A \subseteq B \implies A = B$

En SQL, tester l'égalité de deux ensembles A et B lorsque $A \subseteq B$ revient donc à tester l'égalité des cardinalités des ensembles $|A|$ et $|B|$.

Pour chaque bar, combien de bières ?

Soit X1(bar,nbBeers) la relation associée :

```
select bar, count(beer) as nbBeers from Sells group by bar
```

Combien de bières vendues au total ?

Soit X2(nbTot) la relation associée :

```
select count(distinct beer) as nbTot from Sells
```

Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Les bars qui vendent toutes les bières sont ceux, qui dans la relation Sells sont associés à un ensemble (soit A) de bières égal à l'ensemble des bières (vendues ou connues) soit B.

Or, on sait que : $|A| = |B| \wedge A \subseteq B \implies A = B$

En SQL, tester l'égalité de deux ensembles A et B lorsque $A \subseteq B$ revient donc à tester l'égalité des cardinalités des ensembles $|A|$ et $|B|$.

Pour chaque bar, combien de bières ?

Soit X1(bar,nbBeers) la relation associée :

```
select bar, count(beer) as nbBeers from Sells group by bar
```

Combien de bières vendues au total ?

Soit X2(nbTot) la relation associée :

```
select count(distinct beer) as nbTot from Sells
```

Quels sont les bars qui vendent le plus de bières ?

- Pour chaque bar combien de bières ? voir X1 ci-dessus.
- Dans X1, quel est le plus grand nombre de bières ? Soit Y(nbMax) la relation associée :

`select bar from X1 join (select max(nbBeers) as nbMax from X1) Y`

`- < m > ∈ Y ⇔ le bar qui vend le plus de bières, vend nbm`
`on (nbBeers = nbMax)`

La requête est finalement :

```
with X1 as (
  select bar, count(beer) as nbBeers from Sells group by bar
)
with Y as (
  select max(nbBeers) as nbMax
  from X1
)
select bar
from X1 join Y on (nbBeers = nbMax)
```

Pour chaque consommateur qui aime toutes les bières vendues à l'Australia Hotel, retrouver son nom et les bars qu'il fréquente.

/ Combien de bières sont vendues à l'Australia hotel ? */*

```
select count(Beer)
from Sells where Bar = 'Australia Hotel'
```

/ Combien chaque consommateur aime t-il de bières ? */*

```
select drinker, count(beer) as nbBeers from Likes
group by Drinker
```

La requête est finalement :

```
select drinker, bar
from (select count(Beer) as nbBeers
      from Sells where Bar = 'Australia Hotel') X1
natural join
(select drinker, count(beer) as nbBeers
 from Likes
 group by Drinker) X2
natural join
Frequents
```

Mais, cette expression est incorrecte, pourquoi ?

L'expression correcte est :



Sémantique opérationnelle des requêtes



Partition : erreur courante

Pour chaque bar, donner son nom, son adresse, le nombre de bières qu'il vend et les consommateurs qui le fréquentent.

```
select bar, count (distinct beer) as nbBeers, addr, drinker
*
from Bars join Sells on (name = bar)
        natural join Frequents
group by bar
```

ERROR at line 1:
ORA-00979: not a GROUP BY expression

Pourquoi ?

Résultat attendu

Bar	nbBieres	Addr	Drinker
Australia Hotel	1	The Rocks	{ John }
Coogee Bay Hotel	4	Coogee	{ Adam, John }
Lord Nelson	2	The Rocks	{ Gernot, John }
Marble Bar	3	Sydney	{ Justin }
Regent Hotel	2	Kingsford	{ Justin }

Le type de l'attribut Drinker est un ensemble.

Un ensemble n'est pas un type atomique !

Résultat possible

Bar	nbBieres	Addr	Drinker
Australia Hotel	1	The Rocks	John
Coogee Bay Hotel	4	Coogee	Adam
Coogee Bay Hotel	4	Coogee	John
Lord Nelson	2	The Rocks	Gernot
Lord Nelson	2	The Rocks	John
Marble Bar	3	Sydney	Justin
Regent Hotel	2	Kingsford	Justin

Pour résoudre le problème sur Addr

```
select bar, count (beer) as nbBeers, addr
/* distinct n'est pas nécessaire. */
from Bars join Sells on (name = bar)
group by bar, addr
```

Les deux expressions group by name et group by name, addr construisent la même partition car un bar a une seule adresse.

Pour résoudre le problème sur Drinker

```
select bar, nbBeers, addr, drinker
from (select bar, count (beer) as nbBeers, addr
      from Bars join Sells on (name = bar)
      group by bar, addr) X
/* <b, n, a> ∈ X ⇔ le bar b vend n bières et son adresse est a. */
natural join Frequents
```

2 types chaînes de caractères

char vs. varchar

char [n] : n caractères (longueur fixée), justifiée à gauche, espaces à la fin si nécessaire

varchar [n] : 0..n caractères (longueur variable), pas d'espace en fin

2 types chaînes de caractères

char vs. varchar

char [n] : n caractères (longueur fixée), justifiée à gauche, espaces à la fin si nécessaire

varchar [n] : 0..n caractères (longueur variable), pas d'espace en fin

A1 : char(6), A2 : char(10)

A1 ← 'coucou' (dans la mémoire A1='coucou')

A2 ← 'coucou' (dans la mémoire A2='coucou')

A1=A2 est faux.....

2 types chaînes de caractères

char vs. varchar

char [n] : n caractères (longueur fixée), justifiée à gauche, espaces à la fin si nécessaire

varchar [n] : 0..n caractères (longueur variable), pas d'espace en fin

A1 : char(6), A2 : char(10)

A1 ← 'coucou' (dans la mémoire A1='coucou')

A2 ← 'coucou' (dans la mémoire A2='coucou')

A1=A2 est faux.....

Le type char continue d'exister pour des raisons de compatibilité ascendante (Oracle est utilisé depuis plus de 30 ans !!).

Le type date Oracle (dépendant du système)

date : un instant (unité seconde)

Il faut un mécanisme pour :

- Forme externe d'une date → date (forme interne, illisible, sans sens pour les utilisateurs)
- Date → Forme externe d'une date (lisible par les utilisateurs)

qui s'adapte aux :

- Diverses cultures :
10/12/2014 → 10 décembre 2014
10/12/2014 → 12 octobre 2014
- Divers besoins des applications :
Lundi 8 décembre 2014 vs. Lu. 8-12-14
Lundi 8 décembre 2014 vs. Lundi, Sem. 50, 2014
- Divers systèmes d'unités ?
8 déc. 2014, 9H vs. 8 déc. 2014 9H00:00
8 déc. 2014, 9H vs. déc. 2014
8 déc. 2014, 9H vs. 21e siècle

Opérateurs sur les dates

- Forme externe d'une date → date
`to_date (string,string) → une date`
`/ to_date ('05 Dec 2001', 'DD Mon YYYY') */`*
- Date → Forme externe d'une date
`to_char (date, string) → une chaîne`
`/ to_char (d,f) renvoie une chaîne représentant d selon le format f. */`*
`/ to_char (to_date ('05 Dec 2001', 'DD Mon YYYY'), 'DD/MM/YY')`*
*`= '05/12/01' */`*

Fontions sur les dates

`add_months (date, integer) → une date`

/ add_months (d1, n) renvoie la date d1 plus n mois. */*

Extension aux dates, des opérateurs arithmétiques :

`d1, d2 : date, n : entier ≥ 0`

`d1 - d2` est le nombre de jours entre d1 et d2

`d1 + n` est la date n jours après d1

`d1 - n` est la date n jours avant d1

Autres fonctions : `sysdate` → une date, etc...

Exemples d'utilisation

Quels sont les bars qui ont ouvert un vendredi ?

```
select name from Bars  
where to_char(openSince,'day')='friday';
```

Qui retourne un résultat vide...

Exemples d'utilisation

Quels sont les bars qui ont ouvert un vendredi ?

```
select name from Bars  
where to_char(openSince,'day')='friday';
```

Qui retourne un résultat vide...

Attention :

Oracle gère les noms de jour comme des chaînes de caractères de longueur fixe (comme les valeurs de type `char(9)`, la longueur maximale des noms de jour soit 9 caractères, pour *wednesday*). Le même type de comportement est observé pour le nom des mois, et pour toutes les chaînes de caractères affichées.

Exemples d'utilisation

Quels sont les bars qui ont ouvert un vendredi ?

```
select name from Bars  
where to_char(openSince,'day')='friday';
```

Qui retourne un résultat vide...

Attention :

Oracle gère les noms de jour comme des chaînes de caractères de longueur fixe (comme les valeurs de type `char(9)`, la longueur maximale des noms de jour soit 9 caractères, pour *wednesday*). Le même type de comportement est observé pour le nom des mois, et pour toutes les chaînes de caractères affichées.

Pour s'affranchir de ce problème, on utilise le préfixe *fm* :

```
select name from Bars  
where to_char (openSince, 'fmday') = 'friday';
```

Pour chaque bar, donner son nom et sa date d'ouverture

```
select to_char (opendate, 'fmDDTh') || ' of ' ||
       to_char(opendate, 'fmMonth') || ', ' ||
       to_char (opendate,'YYYY') as OpenSince
from bars;
```

Qui retourne :

NAME	OPENSINCE
Australia Hotel	12TH of January, 1940
Rose Bay Hotel	31ST of August, 1920
Coogee Bay Hotel	31ST of August, 1980
Lord Nelson	11TH of November, 1920
Marble Bar	1ST of April, 2001
Regent Hotel	29TH of February, 2000
Royal Hotel	26TH of June, 1986

Alors que la requête qui n'utilise pas le préfixe 'fm' retourne :

NAME	OPENSINCE
Australia Hotel	12TH of January , 1940
Rose Bay Hotel	31ST of August , 1920
Coogee Bay Hotel	31ST of August , 1980
Lord Nelson	11TH of November , 1920
Marble Bar	01ST of April , 2001
Regent Hotel	29TH of February , 2000
Royal Hotel	26TH of June , 1986

Pour chaque bar, donner son nom et son adresse et calculer son âge

```
select name, addr,  
       to_number (to_char (sysdate, 'YYYY'))  
       - to_number (to_char (openDate, 'YYYY')) as age  
from Bars
```

NAME	ADDR	AGE
Australia Hotel	The Rocks	76
Rose Bay Hotel	Rose Bay	96
Coogee Bay Hotel	Coogee	36
Lord Nelson	The Rocks	96
Marble Bar	Sydney	15
Regent Hotel	Kingsford	16
Royal Hotel	Randwick	30

Des opérateurs et des fonctions variés sont disponibles

Pas d'itération

Considérons la relation LesPersonnes (nom, ami).

- Hypothèse : les amis de mes amis sont aussi mes amis
- Comment construire le résultat de la requête : *Pour chaque personne, donner tous ses amis ?*

Construire la fermeture de Personnes

Exemple :

Une valeur de la relation initiale :

nom	ami
Peter	Paul
Peter	John
Mary	Peter
Mary	David
David	Cath
Paul	Mary

1ère étape : pour chaque personne p, trouver les amis des amis de p (s'il y en a).

```
select distinct p1.nom, p2.ami
from LesPersonnes p1 join LesPersonnes p2 on (p1.ami = p2.nom)
where (p1.nom, p2.ami) not in
      (select nom, ami
       from Personnes)
```

L'union du résultat avec la relation initiale est :

nom	ami
Peter	Paul
Peter	John
Mary	Peter
Mary	David
David	Cath
Paul	Mary

(cont.)	
<i>Peter</i>	<i>Mary</i>
<i>Mary</i>	<i>Cath</i>
<i>Mary</i>	<i>John</i>
<i>Mary</i>	<i>Paul</i>
<i>Paul</i>	<i>David</i>
<i>Paul</i>	<i>Peter</i>

On recommence.

L'union du résultat avec la relation initiale est :

nom	ami
Peter	Paul
Peter	John
Peter	Mary
Mary	Peter
Mary	David
Mary	Cath

(cont.)	
Mary	John
Mary	Paul
Paul	Mary
Paul	David
Paul	Peter
David	Cath

(cont.)	
<i>Peter</i>	<i>Cath</i>
<i>Peter</i>	<i>David</i>
<i>Peter</i>	<i>Peter</i>
<i>Mary</i>	<i>Mary</i>
<i>Paul</i>	<i>Cath</i>
<i>Paul</i>	<i>John</i>
<i>Paul</i>	<i>Paul</i>

On recommence : le résultat est vide.

Plus aucun n-uplet n'est généré. La fermeture a été construite.

- Ce chapitre n'a couvert que la partie interrogation de SQL (la partie définition de relation, et modification de données est étudiée plus tard).
- Il y a souvent plusieurs expressions possibles pour une même requête.
 - Laquelle choisir ? Ce n'est pas une question simple...
 - Comme les requêtes sont optimisées par le SGBD, le programmeur n'a pas à se soucier du problème...
 - Cependant, dans le cas de certains SGBD, le temps de réponse peut être un problème.

Définition de données en SQL

Langage de Définition de Données de SQL :

- Créer et modifier un schéma de base de données : relations, vues, utilisateurs, ...
- Insérer et modifier des données
- les instructions LDD sont soumises comme des requêtes (via l'interpréteur SQL ou une application).

Une requête du LDD n'est exécutée par le SGBD uniquement lorsque le nouvel état de la base de données qu'elle produit est cohérent.

Intégrité des données

L'intégrité des données repose sur les *contraintes d'intégrité* qui :

- préviennent l'entrée de données invalides dans les tables de la base de données.
- précisent le sens des informations stockées dans la base de données.

Par exemple :

- Le numéro d'étudiant est un entier sur au plus 9 chiffres.
- Un adhérent de la bibliothèque ne peut pas avoir plus de 4 emprunts en cours.

Types de contraintes

Une relation, un état de la base de données

- Restriction de valeurs :

Le salaire doit être plus grand que 100

Chaque salaire doit être plus petit que le salaire moyen plus 20%

- Obligatoire :

Le nom ne peut pas être absent

- Unicité :

Tous les noms sont différents deux à deux

- Identification :

Chaque employé est identifié par son numéro.

Plusieurs relations, un état de la base de données

- Restriction de valeurs :

Chaque employé doit gagner un salaire plus petit que celui de son chef.

- Intégrité référentielle : une valeur (projection sur un attribut ou un ensemble d'attributs) dans une relation doit correspondre à au moins une valeur dans une relation reliée.

Chaque chef est un employé qui travaille dans le rayon.

Plusieurs relations, plusieurs états de la base de données

- Restriction de valeur :

Le salaire des employés ne peut pas diminuer.

- Autre exemple :

Pour être nommé chef d'un département, un employé doit avoir été employé au moins 2 ans.

Spécifier les contraintes en SQL : avantages

Déléguer au SGBD la responsabilité des contraintes

- **Facilité (requête SQL)**

Aucun programme supplémentaire n'est nécessaire. Le SGBD garde le contrôle.

- **Centralisation**

Associer les contraintes aux relations les rend plus faciles à manipuler que si elles étaient dispersées dans les programmes d'application.

- **Performance Supérieure**

La sémantique des contraintes d'intégrité est clairement définie, et des optimisations sont spécifiquement implémentées pour chaque cas.

- **Flexibilité**

Les contraintes d'intégrité peuvent être temporairement désactivées pour des besoins spécifiques.

Attention : quand une contrainte est désactivée, elle l'est pour toutes les applications !

Conclusion :

- La plupart des SGBD supportent seulement une partie de ces contraintes d'intégrité (une relation, un état de la base de données).
- Les contraintes d'intégrité qui ne sont pas supportées par le SGBD doivent être gérées au niveau de l'application (ou avec des déclencheurs (*triggers*) lorsque c'est possible).
- L'interface graphique peut quelques fois vérifier la cohérence des données :
 - Restrictions de type : saisie d'une date dans un format prédéfini.
 - Enumération et contrainte référentielle : choix dans un menu ou une liste

Il faut utiliser au maximum les possibilités du SGBD.

Créer des relations

Pour créer des relations (vides) :

```
create table T (  
    Liste_de_définitions_d'attributs  
    [,Liste_de_définitions_de_contraintes ]  
);
```

Définition des attributs

Liste de définitions d'attributs :

- *A1 number (3),*
- *A2 varchar (10),*
- *A3 number (3) default 1,*
- *A4 date,*
- ...

Pour le détail des types⁶ de données, voir Section 7 du Chapitre 3 du polycopié.

⁶L'ensemble des types proposés ainsi que les notations associées sont dépendants du système.

Spécifications de contraintes en SQL

constraint c1 check (P)

/ Plusieurs contraintes de restriction. P est un prédicat qui porte sur les attributs de la relation et/ou sur des constantes. */*

Spécifications de contraintes en SQL

constraint c1 check (P)

/ Plusieurs contraintes de restriction. P est un prédicat qui porte sur les attributs de la relation et/ou sur des constantes. */*

constraint c4 primary key (A1, A2)

/ A1,A2 forment une clef candidate. Une seule clef primaire par relation. */*

constraint c3 unique (A3,A1)

/ A3,A1 forment une autre clef (candidate) */*

Spécifications de contraintes en SQL

constraint c1 check (P)

/ Plusieurs contraintes de restriction. P est un prédicat qui porte sur les attributs de la relation et/ou sur des constantes. */*

constraint c4 primary key (A1, A2)

/ A1,A2 forment une clef candidate. Une seule clef primaire par relation. */*

constraint c3 unique (A3,A1)

/ A3,A1 forment une autre clef (candidate) */*

constraint c5 foreign key (A1,...,An) references Target(B1,...,Bn)

/ B1,...,Bn doivent être clef primaire dans la relation existante Target. B1,...,Bn doivent être compatibles avec A1,...,An. */*

Spécifications de contraintes en SQL

constraint c1 check (P)

/ Plusieurs contraintes de restriction. P est un prédicat qui porte sur les attributs de la relation et/ou sur des constantes. */*

constraint c4 primary key (A1, A2)

/ A1,A2 forment une clef candidate. Une seule clef primaire par relation. */*

constraint c3 unique (A3,A1)

/ A3,A1 forment une autre clef (candidate) */*

constraint c5 foreign key (A1,...,An) references Target(B1,...,Bn)

/ B1,...,Bn doivent être clef primaire dans la relation existante Target. B1,...,Bn doivent être compatibles avec A1,...,An. */*

A4 date constraint c2 not null

/ La contrainte de valuation obligatoire doit être placée avec la définition de l'attribut. */*

Un premier exemple

On considère la base de données initiale (page 48 du polycopié).

Bars (name, addr, openSince)

Drinker(name, addr, phone)

Frequents[drinker] \subseteq Drinker[name]

Frequents(drinker, bar)

Frequents[bar] \subseteq Bars[name]

Un premier exemple

On considère la base de données initiale (page 48 du polycopié).

Bars (name, addr, openSince)

Drinker(name, addr, phone)

Frequents(drinker, bar)

Frequents[drinker] \subseteq Drinker[name]

Frequents[bar] \subseteq Bars[name]

```
create table Bars (
    name varchar(30), addr varchar(30), openDate date,
    constraint b_c0 primary key (name));
```

Un premier exemple

On considère la base de données initiale (page 48 du polycopié).

Bars (name, addr, openSince)

Drinker(name, addr, phone)

Frequents(drinker, bar)

Frequents[drinker] \subseteq Drinker[name]

Frequents[bar] \subseteq Bars[name]

```
create table Bars (
    name varchar(30), addr varchar(30), openDate date,
    constraint b_c0 primary key (name));

create table Drinkers (
    name varchar(30), addr varchar(30), phone varchar(30),
    constraint d_C0 primary key (name)
);
```

Un premier exemple

On considère la base de données initiale (page 48 du polycopié).

Bars (name, addr, openSince)

Drinker(name, addr, phone)

Frequents(drinker, bar)

Frequents[drinker] \subseteq Drinker[name]

Frequents[bar] \subseteq Bars[name]

```
create table Bars (
    name varchar(30), addr varchar(30), openDate date,
    constraint b_c0 primary key (name));

create table Drinkers (
    name varchar(30), addr varchar(30), phone varchar(30),
    constraint d_C0 primary key (name)
);

create table Frequents (
    drinker varchar(30), bar varchar(30),
    constraint f_c0 primary key (drinker,bar),
    constraint f_c1 foreign key (bar) references bars(name),
    constraint f_c2 foreign key (drinker) references drinkers(name)
```


Observation de mises à jour

```
SQL> insert into Bars values ('Le bon coin', 'Grenoble', 'ertoifn',  
    to_date ('29/02/2011','DD/MM/YYYY'));
```

*

ERROR at line 1:

ORA-01839: date not valid for month specified

Observation de mises à jour

```
SQL> insert into Bars values ('Le bon coin', 'Grenoble', 'ertoifn',  
    to_date ('29/02/2011','DD/MM/YYYY'));
```

*

ERROR at line 1:

ORA-01839: date not valid for month specified

```
SQL> insert into drinkers values ('Pierre', 'Grenoble');
```

*

ERROR at line 1:

ORA-00947: not enough values

Observation de mises à jour

```
SQL> insert into Bars values ('Le bon coin', 'Grenoble', 'ertoifn',  
    to_date ('29/02/2011','DD/MM/YYYY'));
```

*

ERROR at line 1:

ORA-01839: date not valid for month specified

```
SQL> insert into drinkers values ('Pierre', 'Grenoble');
```

*

ERROR at line 1:

ORA-00947: not enough values

```
SQL> insert into drinkers values ('Pierre', 'Grenoble', 'rrrrr');
```

1 row created.

Observation de mises à jour

```
SQL> insert into Bars values ('Le bon coin', 'Grenoble', 'ertoifn',  
    to_date ('29/02/2011','DD/MM/YYYY'));
```

*

ERROR at line 1:

ORA-01839: date not valid for month specified

```
SQL> insert into drinkers values ('Pierre', 'Grenoble');
```

*

ERROR at line 1:

ORA-00947: not enough values

```
SQL> insert into drinkers values ('Pierre', 'Grenoble', 'rrrrr');  
1 row created.
```

```
SQL> insert into drinkers values ('Pierre', 'Voiron', 'aaaaa');
```

*

ERROR at line 1:

ORA-00001: unique constraint (BEERS.D_CO) violated

Observation du comportement de contraintes d'intégrité référentielle

```
SQL> insert into Frequents values ('Pierre', 'Bar les amis');  
insert into Frequents values ('Pierre', 'Bar les amis')  
*  
ERROR at line 1:  
ORA-02291: integrity constraint (BEERS.F_C1) violated  
- parent key not found
```

Observation du comportement de contraintes d'intégrité référentielle

```
SQL> insert into Frequents values ('Pierre', 'Bar les amis');  
insert into Frequents values ('Pierre', 'Bar les amis')  
*
```

ERROR at line 1:

ORA-02291: integrity constraint (BEERS.F_C1) violated
- parent key not found

```
SQL> insert into Frequents values ('Paul', 'Coogee Bay Hotel');  
insert into Frequents values ('Paul', 'Coogee Bay Hotel')  
*
```

ERROR at line 1:

ORA-02291: integrity constraint (BEERS.F_C2) violated
- parent key not found

Observation du comportement de contraintes d'intégrité référentielle (suite)

```
SQL> delete from Drinkers where name='John';  
delete from Drinkers where name='John'  
*  
ERROR at line 1:  
ORA-02292: integrity constraint (BEERS.F_C2) violated  
- child record found
```

Observation du comportement de contraintes d'intégrité référentielle (suite)

```
SQL> delete from Drinkers where name='John';  
delete from Drinkers where name='John'
```

*

```
ERROR at line 1:
```

```
ORA-02292: integrity constraint (BEERS.F_C2) violated  
- child record found
```

```
SQL> update Drinkers set name='Peter' where name='Adam';  
update Drinkers set name='Peter' where name='Adam'
```

*

```
ERROR at line 1:
```

```
ORA-02292: integrity constraint (BEERS.F_C2) violated  
- child record found
```


Règles associées aux contraintes d'intégrité référentielle

Permettent de spécifier le comportement du SGBD lors de la modification de données concernées par une contrainte d'intégrité.

Règles associées aux contraintes d'intégrité référentielle

Permettent de spécifier le comportement du SGBD lors de la modification de données concernées par une contrainte d'intégrité.

Exemple: $\text{Frequents}[\text{Drinker}] \subseteq \text{Drinkers}[\text{Name}]$

Drinkers

Name	Addr
Adam	Randwick
Gernot	Newtown
John	Clovelly
Justin	Mosman

Frequents

Drinker	Bar
Adam	Coogee Bay Hotel
Gernot	Lord Nelson
John	Coogee Bay Hotel
John	Lord Nelson
John	Australia Hotel
Justin	Regent Hotel
Justin	Marble Bar

Règles associées aux contraintes d'intégrité référentielle

Permettent de spécifier le comportement du SGBD lors de la modification de données concernées par une contrainte d'intégrité.

Exemple: $\text{Frequents}[\text{Drinker}] \subseteq \text{Drinkers}[\text{Name}]$

Drinkers

Name	Addr
Adam	Randwick
Gernot	Newtown
John	Clovelly
Justin	Mosman

Frequents

Drinker	Bar
Adam	Coogee Bay Hotel
Gernot	Lord Nelson
John	Coogee Bay Hotel
John	Lord Nelson
John	Australia Hotel
Justin	Regent Hotel
Justin	Marble Bar

- Supprimer <John, Clovelly> dans Drinkers ?
- Modifier "Adam" en "Peter" dans Drinkers ?
- Ajouter <Marie, Regent Hotel> dans Frequents ?

Plusieurs comportements possibles

- **Restrict**: interdit la modification ou la suppression des données référencées (défaut).

Plusieurs comportements possibles

- **Restrict**: interdit la modification ou la suppression des données référencées (défaut).
- **Set to null (resp. default)** : lorsque la donnée référencée est modifiée ou supprimée toutes les données qui en dépendaient sont mises à NULL (resp. à la valeur par défaut).
Ce qui pose problème lorsque ces dernières participent à une clef.

Plusieurs comportements possibles

- **Restrict**: interdit la modification ou la suppression des données référencées (défaut).
- **Set to null (resp. default)** : lorsque la donnée référencée est modifiée ou supprimée toutes les données qui en dépendaient sont mises à NULL (resp. à la valeur par défaut).
Ce qui pose problème lorsque ces dernières participent à une clef.
- **Cascade** : lorsque la donnée référencée est modifiée ou supprimée toutes les données associées sont mises à jour en conséquence (modifiées ou supprimées).

Sur l'exemple précédent :

create table Frequents (

....

constraint fk_drinker foreign key (drinker) references Drinkers(name)
on delete cascade on update cascade

...

- Absence de la clause on .. cascade \implies restrict
- on update cascade \implies "Adam" est remplacé par "Peter" dans Frequents
- on delete cascade \implies tous les n-uplets correspondant à "John" sont supprimés dans Frequents

Exemple : une (autre) bibliothèque

Spécification des relations :

LesLivres (titre, dateEdition)

LesExemplaires (numEx, dateAchat, titre)

titre not null

LesExemplaires[titre] = LesLivres[titre]

LesEmprunts (numEx, dateEmp, dateRetour, numAdh)

/ (numEx,dateRetour) est une autre clef */*

dateEmp < dateRetour

numAdh not null

LesEmprunts[numEx] \subseteq LesExemplaires[numEx]

domaine (dateEmp) = domaine (dateRetour) = date(Heure)

domaine (dateAchat) = domaine (dateEdition) = date(Jour)

Exemple : une (autre) bibliothèque

Spécification des relations :

LesLivres (titre, dateEdition)

LesExemplaires (numEx, dateAchat, titre)

titre not null

LesExemplaires[titre] = LesLivres[titre]

LesEmprunts (numEx, dateEmp, dateRetour, numAdh)

/ (numEx,dateRetour) est une autre clef */*

dateEmp < dateRetour

numAdh not null

LesEmprunts[numEx] \subseteq LesExemplaires[numEx]

domaine (dateEmp) = domaine (dateRetour) = date(Heure)

domaine (dateAchat) = domaine (dateEdition) = date(Jour)

- A chaque instant, chaque exemplaire est emprunté au plus une fois
- Chaque adhérent ne peut pas faire plus de 4 emprunts à la fois

En LDD SQL :

```
create table LesLivres ( titre varchar(30), dateEdition date,  
    constraint LesLivres_c1 primary key (titre) ) ;
```

En LDD SQL :

```
create table LesLivres ( titre varchar(30), dateEdition date,  
    constraint LesLivres_c1 primary key (titre) ) ;  
create table LesExemplaires ( numEx varchar(10), dateAchat date,  
    titre varchar(30) constraint numEx_c0 not null,  
    constraint numEx_c1 primary key (numEx),  
    constraint numEx_c2 foreign key (titre)  
        references LesLivres (titre) on delete cascade on update cascade );
```

En LDD SQL :

```
create table LesLivres ( titre varchar(30), dateEdition date,  
    constraint LesLivres_c1 primary key (titre) ) ;  
create table LesExemplaires ( numEx varchar(10), dateAchat date,  
    titre varchar(30) constraint numEx_c0 not null,  
    constraint numEx_c1 primary key (numEx),  
    constraint numEx_c2 foreign key (titre)  
        references LesLivres (titre) on delete cascade on update cascade );  
create table LesEmprunts ( numEx varchar(10), dateEmp date, dateRetour date,  
    numAdh varchar(10) constraint LesEmprunts_c1 not null,  
    constraint LesEmprunts_c1 primary key (numEx, dateEmp),  
    constraint LesEmprunts_c2 unique (numEx, dateRetour),  
    constraint LesEmprunts_c3 foreign key (numEx)  
        references LesExemplaires (numEx) on delete cascade on update cascade,  
    constraint LesEmprunts_c4 check (dateEmp < dateRetour) );
```

En LDD SQL :

```
create table LesLivres ( titre varchar(30), dateEdition date,
    constraint LesLivres_c1 primary key (titre) ) ;
create table LesExemplaires ( numEx varchar(10), dateAchat date,
    titre varchar(30) constraint numEx_c0 not null,
    constraint numEx_c1 primary key (numEx),
    constraint numEx_c2 foreign key (titre)
        references LesLivres (titre) on delete cascade on update cascade );
create table LesEmprunts ( numEx varchar(10), dateEmp date, dateRetour date,
    numAdh varchar(10) constraint LesEmprunts_c1 not null,
    constraint LesEmprunts_c1 primary key (numEx, dateEmp),
    constraint LesEmprunts_c2 unique (numEx, dateRetour),
    constraint LesEmprunts_c3 foreign key (numEx)
        references LesExemplaires (numEx) on delete cascade on update cascade,
    constraint LesEmprunts_c4 check (dateEmp < dateRetour) );
```

Quelles sont les contraintes à la charge de l'application ?

- A chaque instant, chaque exemplaire est emprunté au plus une fois
- Chaque adhérent ne peut pas faire plus de 4 emprunts à la fois
- Qu'en est-il de LesExemplaires[titre] = LesLivres[titre] ?

Spécifications des contraintes : bonnes pratiques

Les contraintes doivent être nommées :

- Maintenance du schéma : seules les contraintes nommées peuvent être désactivées/réactivées
- Gestion des exceptions dans les programmes : facilite l'analyse des exceptions retournées par le SGBD.

Attention aux règles de nommage !

En résumé

- Attention à l'ordre de définition des relations !
- Contraintes d'intégrité à la charge des applications :
 - Granularité du temps
(ex. domaine (DateRetour) = Date(Heure))
 - Restriction de valeur basée sur plus d'une relation
(ex. Chaque membre ne peut faire plus de 4 emprunts à la fois)
 - Contraintes d'intégrité référentielle définies sur des attributs non clés
 - Contraintes d'intégrité référentielle bi-directionnelles
 - Etc.
- Options "on delete cascade", "on update cascade" ?

Supprimer une relation

Afin de satisfaire les contraintes d'intégrité toute modification de schéma entraîne l'ajustement des données.

drop table T ; supprime la relation T et ses n -uplets

drop table LesEmprunts ;

Lorsque T est cible d'une contrainte d'intégrité référentielle, T ne peut pas être supprimée.

Modifier des relations

Il est possible de faire toute modification qui n'implique pas de suppression de données :

- Ajouter un attribut (chaque tuple existant dans la relation a la valeur absente pour le nouvel attribut).
- Ajouter une contrainte d'intégrité
- Redéfinir un attribut (type, taille, valeur par défaut)
- Activer, désactiver ou supprimer une contrainte d'intégrité

Quelques exemples :

```
alter table LesEtudiants add (anniv date, adresse varchar(20)) ;  
alter table LesEtudiants modify (adresse varchar (30)) ;  
alter table LesEtudiants disable constraint LesEtudiants_c1 ;
```

Insérer des données dans des relations

Insérer des données :

- *insert into nom_relation values ...*

Deux options :

```
/* un n-uplet à la fois */  
insert into LesBatiments values (210, 'John') ;  
insert into LesExemplaires values (12, sysdate, 'Dune') ;  
/* Pour une relation existante */  
insert into LesBatiments  
select A, B from R where .....
```

Rappel : une requête est exécutée uniquement lorsqu'elle construit un état de la base de données qui est cohérent.

Modifier des données

Modifier des données :

- *update nom_relation*
 set nom_attribut₁ =
 nouvelle_val₁, ...
 nom_attribut_N =
 nouvelle_val_N
 where condition ; — la clause where est optionnelle

Exemple :

```
update LesEmployés
  set salaire = salaire * 1.2
  where salaire < 10000 ;
```

Supprimer des données

Supprimer des données : Soient : LesBatiments (noB, nom) et LesEtages (noEt, noB, acces)

$\text{LesEtages}[\text{noB}] \subseteq \text{LesBatiments}[\text{noB}]$

- *delete from nom_relation where condition ;*

La clause where est optionnelle

Exemple :

`delete from LesBatiments ;`

Supprime tous les n-uplets des deux relations LesBatiments et LesEtages (à cause de l'option on delete cascade)

Les modifications sont faites si et seulement si elles satisfont toutes les contraintes de la base de données définies en SQL.

Vue : définition

Une vue est une relation dont les données sont *dynamiquement* calculées par une requête :

```
create view LesPersonnes (NumPers, prénom, nom, age) as
  select NumPers, prénom, nom, to_number (to_char (sysdate, 'YYYY'))
     - to_number (to_char (dateNais, 'YYYY'))
  from LesEmployés
 union
  select NumEt, prénom, nom, to_number (to_char (sysdate, 'YYYY'))
     - to_number (to_char (dateNais, 'YYYY'))
  from LesEtudiants
```

Une vue peut être pensée comme

- une représentation personnalisée de données stockées dans une ou plusieurs relations
- une *requête sauvegardée* qui définit une relation dérivée.

Avantages

Les vues sont souvent utilisées pour :

- Cacher la complexité des données,
- Simplifier des requêtes,
- Présenter des données sous différents angles,
- Sauvegarder des requêtes complexes,
- Offrir un niveau supplémentaire de confidentialité

Comme dans une relation, les données associées à une vue peuvent être :

- interrogées,
- modifiées avec certaines restrictions
(les modifications sont propagées aux relations de base de la vue)

Exemple

```
/* <a, nb> ∈ CombienDePrêts ⇔ l'adhérent a a actuellement nb prêts. */
create view CombienDePrêts (NumAdh, nbEmp) as
  select NumAdh, count (NumEx) from LesEmprunts
  where DateRetour is null
  group by NumAdh
```

Donner les adhérents qui ont actuellement 4 emprunts. Pour chaque emprunt, donner le numéro de l'exemplaire et le titre

```
select NumAdh, NumEx, titre
from CombienDePrêts
  natural join LesEmprunts
  natural join LesExemplaires
where nbEmp = 4
```

Utilisation maladroite des vues

```
create view EmpruntsARendre (NumEx, DateRendu)
  select NumEx, to_char (DateEmp + 15, 'DD/MM/YYYY')
  from LesEmprunts
  where NumAdh = &NumAdh and
         DateRetour is null and sysdate - DateEmp >= 15
```

Le paramètre &NumAdh est lié à la valeur donnée à l'exécution de la requête (c.a.d. le moment de création de la vue). Donc, toutes les futures requêtes sur la vue seront évaluées avec la même valeur de &NumAdh.
Mieux :

```
create view EmpruntsARendre (NumAdh, NumEx, DateRendu)
  select NumAdh, NumEx, to_char (DateEmp + 15, 'DD/MM/YYYY')
  from LesEmprunts
  where DateRetour is null and sysdate - DateEmp >= 15
```


Supprimer une vue

`drop view V;` supprime la vue `V` (mais pas les `n`-uplets des relations de base!)

`drop view CombienDePrêts ;`

Le langage de définition des données de SQL permet de :

- Créer, ajouter, modifier des relations et des vues
 - Spécifier certaines contraintes
- D'autres restent sous la responsabilité des programmes applicatifs :
- Restrictions de domaines spécifiques (le salaire d'un employé ne peut être plus haut que celui de son chef)
 - Contraintes dynamiques (un salaire ne peut diminuer)
- ...

Possibilités d'Oracle non étudiées ici : index, user, grant, tablespace... .

World Wide Web

Un peu d'histoire :

- 1980 : hypertexte, inventé par T. Berners-Lee et R. Caillau (format pour des échanges de documents au CERN, Genève)
- 1990 : protocole HTTP + HTML \longrightarrow WWW
- 1992 : premier serveur installé hors Europe (Stanford)
- 1994: création de W3C (WWW Consortium)

HyperText Markup Language

Description de documents hypertextes basés sur des balises :

- Les balises en paire : `<tag-name tag-attributes> texte </tag-name>`
- Les balises orphelines : `<tag-name tag-attributes/ >`

où tag-attributes est une séquence de 0, une ou plusieurs expressions de la forme attribute-name="value".

Les balises servent à décrire :

- Contenu
- Présentation
- Liens vers d'autres documents

Exemple

```

<html>
  <head>
    <meta name="author" content="M.-C. Fauvet"
    </meta>
    <title> un exemple </title>
  </head>
  <body>
    <h1> bienvenue à Grenoble! </h1>
    <br><b>une vue de Grenoble :</b>
    <br>
    <br><a href="http://www.ujf-grenoble.fr">
      cliquer ici pour visiter l'UJF</a>
    <br><i>Amusez-vous bien</i>
    <!-- un commentaire qui peut aider -->
  </body>
</html>

```

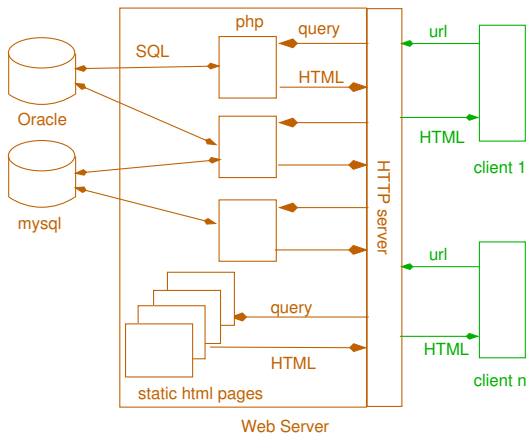
Un peu de culture...

- Créé en 1994 par Rasmus Lerdorf
- PHP signifie :
 - Personal Home Page (au début)
 - PHP : Hypertext Preprocessor (officiellement)
 - People Hate Perl (moins officiellement)
- La version courante est la 7.1.0 (depuis début décembre 2016)
- Compilé depuis la version 4

Site officiel : <http://www.php.net/>

La documentation en français : <http://www.php.net/manual/fr/index.php>

Architecture (vue grossière)



Principe

- Le code PHP code est embarqué dans les pages HTML (inclus entre les marques `<?php` et `?>`)
- Le serveur s'occupe d'interpréter le code
- Le code PHP ne peut pas être vu par les clients

Rôles (client et serveur)

- Du côté serveur :

```
<html>
```

```
<head> <title> Heure </title> </head>
```

```
<body> <p>
```

```
L'heure courante est <?php print date("h:i:s") ; ?>
```

```
</body> </html>
```

- du côté client (un browser) :

```
<html>
```

```
<head> <title> Heure </title> </head>
```

```
<body> <p>
```

```
L'heure courante est 16:39:27
```

```
</body> </html>
```

Exemples

Les noms de variables commencent par \$

```
$reply = 42; // entier
$pi = 3.14159; // réel
$action = "naviguer"; // chaîne
$sentence = "J'aime $action"; // chaîne interprétée
$sentence = 'Cela coûte $100'; // chaîne non interprétée
$foo = $action; // variable
$E = $m*$c*$c; // expression
```

Règles de typage

- Le type d'une variable est dérivé de sa valeur

```
<html> <body>
```

```
<?php
```

```
    $a = 3 ; // $a est un entier
```

```
    $a = 3.4 ; // $a est maintenant un réel
```

```
    $b = (int) $a ; // $b a la valeur de $a castée en entier (3)
```

```
?>
```

```
</body></html>
```

Type d'une variable

<code>boolean is_null(\$var)</code>	renvoie vrai si \$var est affectée (a une valeur)
<code>string gettype(\$var)</code>	renvoie la chaine décrivant le type de \$var
<code>boolean is_double(\$var)</code>	renvoie vrai si \$var est de type double
<code>boolean is_integer(\$var)</code>	renvoie vrai si \$var est de type entier
<code>boolean is_string(\$var)</code>	renvoie vrai si \$var est une chaîne
<code>boolean is_type(\$var, \$t)</code>	renvoie vrai si \$var est du type \$t
<code>...</code>	<code>...</code>

Exemple

```
<html> <body>
<?php
    $a=3 ;
    echo "type = "; echo gettype($a) ; echo "<br>" ;
    $a = 3.4 ;
    echo "type = "; echo gettype($a) ; echo "<br>" ;
    $b = (boolean) $a ;
    echo "type = "; echo gettype($b) ; echo "<br>" ;
    echo $b ;
?>
</body> </html>
```

Opérateurs (exemples)

- Sur les nombres : +, -, *, /, % (modulo), et les opérateurs arithmétiques habituels : cos, sin, floor, ceil, etc..

- Sur les chaînes de caractères :

```
<?php
```

```
$a = "Hello ";
```

```
$b = $a . "World!";
```

```
// la valeur de $b est "Hello World!"
```

```
$a = "Hello ";
```

```
$a .= "World!";
```

```
// la valeur de $a est "Hello World!"
```

```
$rest = substr("abcdef", 0, 1);
```

```
// la valeur de $rest est "a"
```

```
$rest = substr("abcdef", 4);
```

```
// retourne "e"
```

```
?>
```

- De comparaison : ==, <>, !=, >=, >, <=, <
- Logiques : NOT, !, AND, &&, OR, ||, XOR

Tableaux

- Initialisation :
`$tab = array("h", "1", "test") ;`
- Ajout d'éléments :
`$tab[3] = "suivant" ;`
`$i = 10; $tab[$i] = "autre" ;`
`$tab[] = "dernier" ; // pour insérer à la fin`
- Est-ce que `$var` est un tableau ?
`boolean is_array($var) ;`

Structures de contrôle

- if, else, elseif, while, do, for, foreach,...

Très proche de C ...

...

```
if (is_null ($login) or is_null ($motdepasse)) {
    $codeerreur = "problemevariables";
}
```

```
else {
    $codeerreur = "sanserreur";
}
```

while (\$nbRes > 0) {	do {
....;;
\$nbRes = \$nbRes - 1 ;	\$nbRes = \$nbRes - 1;
....;;
} ;	} while (\$nbRes > 0);

Inclusion de programme

- `include("foo.php")`
 - Inclut un autre programme
 - Alerte lorsque le programme n'existe pas
- `require("foo.php")`
 - Inclut un autre programme
 - Erreur lorsque le programme n'existe pas

Fonctions et actions

```
<?php
function test_fonction ($x) {
    return $x+1 ;
} ?>

<?php
function test_action ($x, &$y) {
    /* x est un paramètre donné, y un paramètre
    résultat. */
    $y = $x+1 ;
}
?>
```

Interagir avec un utilisateur via un formulaire (en HTML)

Exemple de formulaire :

login : password :

 Exemple de formulaire :

```
<form method=post action="1-action.php" >
```

```
  login : <input type="text" name="login" >
```

```
  /* Déclaration de la variable login */
```

```
  password : <input type="password" name="pwd" >
```

```
  /* Déclaration de la variable pwd */
```

```
<br>
```

```
<input type="submit" value="connexion" >
```

```
<input type="reset" value="remise à zéro" >
```

```
</form>
```

- Le programme 1-action.php s'occupe de traiter les valeurs entrées.
- login et pwd sont 2 variables dont les valeurs sont affectées après que l'utilisateur ait cliqué sur le bouton connexion, qui déclenche la soumission du formulaire.

Passages de valeurs de 1.php vers 1-action.php

1.php

Exemple de formulaire

```
<form method=post action="1-action.php">
login :   <input type="text" name="login">
passwd : <input type="password" name="pwd">
<input type="submit" value="connexion">
<input type="reset" value="remise &agrave; z&eacute;ro">
</form>
```

1-action.php

```
<php
    $login=$_POST["login"];
    $pwd=$_POST["pwd"];
    if (isset($login) and isset($pwd)) {
        .....
        echo "Connexion de ".$login." OK.";
        .....
    } else {
        echo "Connexion impossible";
    }
?>
```

\$login, \$pwd



1.php & 1-action.php : rendu à l'exécution

Exemple de formulaire :

login : **password :**

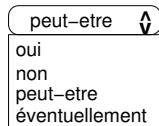
connexion

remise à zéro

Connexion fauvet OK

Sélection parmi un choix

Choix dans une liste :



...

```
<form method=post multiple=no size=1 action=2-action.php>
  <select name=reponse>
    <option value=1> oui </option>
    <option value=0> non </option>
    <option value=2 selected> peut-être </option>
    <option value=3> éventuellement </option>
  </select>
```

....

```
</form>
```

reponse est une variable dont la valeur est soit 0, 1, 2, ou 3 après que le formulaire ait été soumis.

Variables utilisées par plusieurs pages (variables de sessions)

Le programme 3.php :

```
<?php session_start();                                // Initialisation d'une session
...
$_SESSION['login_s'] = $_POST['login'];
$_SESSION['pwd_s'] = $_POST['pwd'];
?>
<b> Saisie d'un numéro de cage : </b>
<form method=post action="3-action.php" >
    Numéro : <input type="text" name="noCage" > <br>
    <input type="submit" value="valider" >
    <input type="reset" value="remise &agrave; z&eacute;ro" >
</form>
```

A la suite du déclenchement de la soumission du formulaire dont 3.php est l'action associée, les variables de session \$login_s et \$pwd_s prennent respectivement les valeurs de \$login et \$pwd initialisées après que l'utilisateur ait validé le formulaire.

Le programme 3-action.php

```
<?php session_start();
if (if_null($_SESSION['login_s']) and is_null($_SESSION['pwd_s'])) {
    // Vérification des paramètres de connexion (login_s et pwd_s)
    $lien = .... // tentative de connexion
    if ($lien) {
        echo "Connexion OK" ;
        $noCage_l = $_POST['noCage'] ;
        ... }
    else {
        echo = "Pas de connexion" ; }
?>
...
```

La variable locale \$noCage_l prend la valeur de la variable initialisée à la suite du formulaire présent dans le programme 3.php.

Introduction

PHP permet l'accès à plusieurs SGBD : Oracle, SQL server, Postgres, MySQL, etc.

- Les interfaces sont similaires mais leurs implémentations sont spécifiques.
- Principales interactions entre PHP et le SGBD :
 - Connexion à la base de données, déconnexion.
 - Exécution d'une requête (select, update, ...).
 - Traitement des résultats (une liste de nuplets ou une erreur).

Connexion

```
$lien = oci_connect ("john","john_passwd","$host:$port/$service_name");  
if ($lien) {  
    echo "Connexion OK";  
    ....  
    oci_close ($lien);  
}  
else {  
    echo "Connexion impossible";  
}
```

Exécution d'une requête d'interrogation

```
// La connexion est établie : variable $lien affectée
// définir la requête qui comprend le paramètre n
$query = "select noCage, nomA from zoo.LesAnimaux
        natural join zoo.LesCages natural join zoo.LesGardiens
        where lower (nomE) = lower(:n)
        order by noAllee" ;
$curseur = oci_parse ($lien,$query) ; // ouvrir un curseur et analyser la requête
oci_bind_by_name ($curseur, ":n", $nom) ;
// $nom est une variable affectée plus haut.
$res = oci_execute ($curseur) ; // évaluation de la requête
```

```
// Exploitation des données retournées par le SGBD.
$res = oci_fetch ($cursueur); if (!$res ) {
    // aucune sélection
    echo "Inconnu" ; }
else {
    // Traiter les données
    do {
        $scageld = oci_result ($cursueur, 0) ;
        $salleeld = oci_result ($cursueur, 1) ;

        ....
    } while (oci_fetch ($cursueur));
    oci_close ($cursueur) ;
}
```

Exécution de modifications (transaction)

```
// Connexion établie : variable $lien est affectée
if ($lien) {
    $requete = "insert into zoo.lesMaladies values (:animal, :maladie) ";
    // construction de la requete
    $curseur = oci_parse ($lien, $requete) ;
    // analyse de la requete et association au curseur
    $nomAn = "Milou"; $nomMal = "rage de dents";
    oci_bind_by_name ($curseur, ":animal", $nomAn); // affectation des paramètres
    oci_bind_by_name ($curseur, ":maladie", $nomMal);
    $res = oci_execute ($curseur, OCI_NO_AUTO_COMMIT) ; // exécution de la req
```

```

if ($res) {
    echo "Enregistrement effectué" ;
    oci_commit ($lien) ; }          // terminaison de la transaction : validation
else {
    echo "Enregistrement rejeté;" ;
    switch (oci_error ($casseur)) {
        // Analyse du code erreur retourné par Oracle
        case 1:
            $message = "Maladie ".$nomMal." déjà enregistrée pour ".$nomAn ;
            break ;
        case 2291:
            $message = $nomAn." inconnu. " ; break;
        default:
            $message = "Autre message : ".oci_error ($casseur) ; break ; }
    oci_rollback ($lien) ; }        // terminaison de la transaction : annulation
oci_close ($casseur) ; }

```

Notion de transactions

- Une ou plusieurs requêtes sur la base de données
- Toutes sont validées, ou aucune
- Une transaction typique :
 - 1 Connexion à la base de données
 - 2 Exécutions sans erreur de requêtes (dont certaines sont des mises à jour)
 - 3 Une erreur ? annulation (rollback)
 - 4 Aucune erreur ? validation (commit)
 - 5 Déconnexion de la base de données

On reviendra sur cette notion centrale dans le cours sur SQL Langage de définition des données.

- Transactions : quelques bases
- Application Php / Oracle : quelques bonnes pratiques

Transaction

Définition :

Une transaction est une séquence d'opérations constituant une unité logique de traitement. Cette séquence inclut des accès (au moins un) à une base de données.

Exemple :

- Réservation d'un billet de train
- Augmentation des salaires de tous les employés du magasin
- Enregistrement d'une maladie pour un animal du Zoo
- etc.

Un exemple de transaction : une opération bancaire

Soit la relation LesComptes :

```
create table LesComptes (  
    noC number, solde number,  
    constraint comptes_c1 primary key (noC),  
    constraint comptes_c2 check (solde >=0) );
```

On étudie le transfert de 100 du compte de numéro 1 vers celui de numéro 2 :

```
update LesComptes set solde = solde - 100 where noC = 2;  
update LesComptes set solde = solde + 100 where noC = 1;
```

Question 1 : que se passe t-il si la première modification est exécutée mais pas la seconde ?

Atomicité des transactions : définition

L'exécution d'une transaction est atomique : tous ses effets persistent⁷, ou aucun.

- Tous les employés ont été augmentés ou aucun
- Le compte 1 a été débité et le compte 2 a été crédité, ou les deux comptes n'ont pas été modifiés.

Cette propriété est assurée par le SGBD (à condition que la transaction ait été correctement programmée).

⁷persistent = sont enregistrés durablement

Portée des transactions (SQL)

Mode non "autocommit"

Une transaction se termine par :

- L'enregistrement de tous ses effets (requête commit)
- L'annulation de tous ses effets (requête rollback)
- Fin de la connexion

Une transaction commence par :

- L'établissement d'une connexion à la base de données
- La fin de la transaction précédente

Mode "autocommit"

Une transaction = une requête SQL

Enregistrement des effets de la transaction (avec l'interprète Oracle)

```
SQL> set autocommit off
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	100
2	0
3	200
4	5000

```
SQL> update LesComptes
      set solde = solde - 100
      where noC = 1;
1 row updated.
```

```
SQL> update LesComptes
      set solde = solde + 100
      where noC = 2;
1 row updated.
```

```
SQL> commit;
Commit complete.
```

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	100
3	200
4	5000

Annulation des effets de la transaction (avec l'interprète Oracle)

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	100
3	200
4	5000

```
SQL> update LesComptes
      set solde = solde - 199
      where noC = 3;
1 row updated.
```

```
SQL> update LesComptes
      set solde = solde + 199
      where noC = 2;
1 row updated.
```

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	299
3	1
4	5000

```
SQL> rollback ;
Rollback complete.
```

```
SQL> select * from LesComptes;
      NOC      SOLDE
-----
1         0
2        100
3        200
4       5000
```

Interruption de la transaction (événement externe)

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	100
3	200
4	5000

```
SQL> update LesComptes
      set solde = solde - 199
      where noC = 3;
1 row updated.
```

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	100
3	1
4	5000

```
SQL> Terminated
```

```
> sqlplus fauvet
```

```
SQL*Plus: Release 11.2.0.1.0 Production
on Mon Apr 16 19:26:31 2012
```

```
Copyright (c) 1982, 2009, Oracle. All
rights reserved.
```

```
Enter password:
```

```
Connected to:
```

```
Oracle Database 11g Enterprise Edition ...
```

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	100
3	200
4	5000

Interruption de la transaction (violation de contrainte)

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	100
3	200
4	5000

```
SQL> update LesComptes
      set solde = solde - 199
      where noC = 1;
```

```
update LesComptes
```

```
*
```

```
ERROR at line 1:
```

```
ORA-02290: check constraint
(FAUVET.COMPTES_C2) violated
```

```
SQL> select * from LesComptes;
```

NOC	SOLDE
-----	-----
1	0
2	100
3	200
4	5000

En Php

```
// $lien est une connexion valide
// $montant est le montant à transférer du compte $source vers le compte $cible
$req1 = "update LesComptes set solde = solde - :montant where noC = :source";
$curs1 = oci_parse ($lien, $req1);
oci_bind_by_name ($curs1, ":source", $source);
oci_bind_by_name ($curs1, ":montant", $montant);
$res1 = oci_execute ($curs1,OCI_NO_AUTO_COMMIT);
if ($res1) {
    $req2 = "update LesComptes set solde = solde + :montant where noC = :cible";
    $curs2 = oci_parse ($lien, $req2);
    oci_bind_by_name ($curs2, ":cible", $cible);
    oci_bind_by_name ($curs2, ":montant", $montant);
    $res2 = oci_execute ($curs2,OCI_NO_AUTO_COMMIT);
    if ($res2) {
        oci_commit ($lien); echo "<br> Transfert effectué.";
    } else {
        $e = oci_error ($curs2); oci_rollback($lien);
        echo "<br> Transfert refusé : ".$e['message']." (code : ".$e['code'].")";
    }
} else {
    $e = oci_error ($curs1); oci_rollback ($lien);
    echo "<br> Transfert refusé : ".$e['message']." (code : ".$e['code'].")";
}
```

Programme incorrect : voir démonstration

<http://goedel.e.ujf-grenoble.fr/~fauvetm/ComptesBancaires/>

Cas de la requête update

```
SQL> select * from LesComptes;
```

NOC	SOLDE
1	100
2	0
3	200
4	5000

```
SQL> update LesComptes set solde = solde - 1000 where noC = 4 ;  
1 row updated.
```

```
SQL> update LesComptes set solde = solde + 1000 where noC = 5 ;  
0 rows updated.
```

```
SQL> select * from LesComptes;
```

NOC	SOLDE
1	100
2	0
3	200
4	4000

La mise à jour de 0 n-uplet ne renvoie pas d'erreur (voir oci_num_rows).

Plusieurs transactions en concurrence

Transfert de 100 du compte de numéro 1 vers celui de numéro 2 :

```
update LesComptes set solde = solde - 100 where noC = 2;  
... interruption
```

Question 2 : que se passe t-il si une autre transaction lit le solde du compte 2 après la première instruction, avant que les effets de la transaction ne soient annulés par le SGBD ?

Voir démo..

Conclusion

Les propriétés des transactions :

- Atomicité : toutes les opérations sont exécutées, ou aucune
- Cohérence : à l'issue de la transaction les données sont cohérentes
- Isolation : les effets en cours d'une transaction ne sont pas visibles par les autres
- Durabilité : les effets de la transactions persistent

Ces propriétés doivent être garanties par le SGBD.

- Transactions : quelques bases
- Application Php / Oracle : quelques bonnes pratiques

Contraintes d'intégrité : depuis l'IHM jusqu'à la BD

Créditer un compte bancaire pour un montant donné

Principe de la solution :

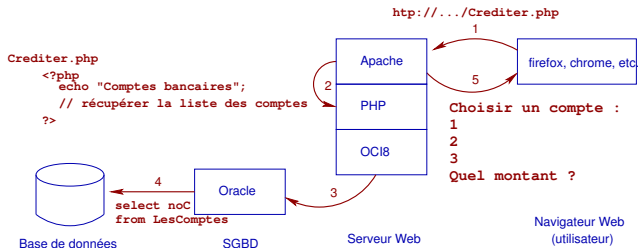
- 1 Interactions avec l'utilisateur : choix du compte dans la liste des comptes existants, saisie du montant à créditer
- 2 Modification du compte dans la base de données.

Crediter-action.php

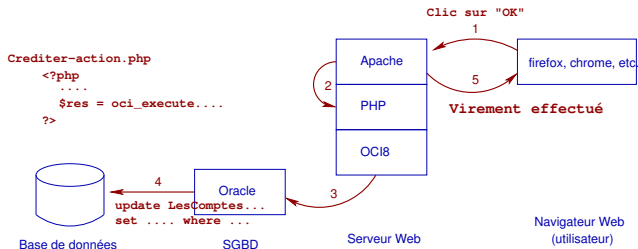
```
// $lien est une connexion valide
// $montant est le montant à créditer sur le compte $noc
$requete = "update lescomptes set solde = solde + to_number(:m)
           where noc = to_number(:n) ";
$curseur = oci_parse ($lien, $requete) ;
oci_bind_by_name ($curseur, ":n", $noc) ;
oci_bind_by_name ($curseur, ":m", $montant);
$res = oci_execute ($curseur,OCI_NO_AUTO_COMMIT) ;
if (!$res) {
    $error = oci_error ($curseur);
    $code = $error['code'];
    $message = $error['message'];
    echo "<p><b> Virement impossible : </b>
        (code erreur : $code message : $message ) </p>" ;
    oci_rollback($lien);
} else {
    echo "<p><b> Virement effectu&eacute; </b></p>";
    oci_commit($lien);
}
```

Voir démo....

Créditer un compte : détail des interactions (1)



Créditer un compte : détail des interactions (2)



Conclusion

Quels enseignements doit-on tirer des observations effectuées ?