Bases de données
Licence professionnelle
Logiciels Libres

Frédéric Lardeux

Université d'Angers

18 octobre 2011

Bases de données

Cours (15h) / TD (5h) / TP (20h)

Bases de données

Cours (15h) / TD (5h) / TP (20h)

- Modèle relationnel.
  - Données (attribut, clé, etc.).
  - Opérations (algèbre relationnelle).
- Utilisation de SGBD relationnels : **MySQL**, PostgreSQL et Oracle.

Chapitre I

Bases de données

Chapitre I

Bases de données

Introduction

La plupart des applications « réelles » manipulent de grandes quantités de données.
Les langages de programmation classiques n'offrent pas de facilités pour la gestion de données.

Introduction

La plupart des applications « réelles » manipulent de grandes quantités de données.
Les langages de programmation classiques n'offrent pas de facilités pour la gestion de données.

Défauts de la gestion des données directement dans une application :

- Non *indépendance des données*.
  Chaque programme utilise son format de stockage des données.

Introduction

La plupart des applications « réelles » manipulent de grandes quantités de données.
Les langages de programmation classiques n'offrent pas de facilités pour la gestion de données.

Défauts de la gestion des données directement dans une application :

- Non *indépendance des données*.
  Chaque programme utilise son format de stockage des données.
  Indépendance physique : insensibilité des applications à des changements tels que le changement du nombre d'octets pour la représentation d'un entier.
  Indépendance logique : insensibilité à l'ajout d'informations à certaines classes d'objets.

Gestion de données dans une application

- *Redondance de données*.
  Chaque logiciel utilise ses fichiers. Certaines informations doivent être utilisées par plusieurs logiciels : elles sont dupliquées $\rightarrow$ gestion plus complexe, mise à jour plus difficile.

Gestion de données dans une application

- *Redondance de données*.
  Chaque logiciel utilise ses fichiers. Certaines informations doivent être utilisées par plusieurs logiciels : elles sont dupliquées $\rightarrow$ gestion plus complexe, mise à jour plus difficile.
- Risque de problèmes d'*intégrité*.
  Les données peuvent être incohérentes.

Gestion de données dans une application

- *Redondance de données*.
  Chaque logiciel utilise ses fichiers. Certaines informations doivent être utilisées par plusieurs logiciels : elles sont dupliquées → gestion plus complexe, mise à jour plus difficile.
- Risque de problèmes d'*intégrité*.
  Les données peuvent être incohérentes.

⇒ Multiplication de routines de gestion de données peu réutilisables (coût, résultat peu satisfaisant).

Gestion de données dans une application

- *Redondance de données*.
  Chaque logiciel utilise ses fichiers. Certaines informations doivent être utilisées par plusieurs logiciels : elles sont dupliquées → gestion plus complexe, mise à jour plus difficile.
- Risque de problèmes d'*intégrité*.
  Les données peuvent être incohérentes.

⇒ Multiplication de routines de gestion de données peu réutilisables (coût, résultat peu satisfaisant).
⇒ Pas de mise en commun des données utilisées par plusieurs logiciels (saisies multiples, risques d'incohérences).

Gestion de données dans une application

- *Redondance de données*.
  Chaque logiciel utilise ses fichiers. Certaines informations doivent être utilisées par plusieurs logiciels : elles sont dupliquées → gestion plus complexe, mise à jour plus difficile.
- Risque de problèmes d'*intégrité*.
  Les données peuvent être incohérentes.

⇒ Multiplication de routines de gestion de données peu réutilisables (coût, résultat peu satisfaisant).
⇒ Pas de mise en commun des données utilisées par plusieurs logiciels (saisies multiples, risques d'incohérences).
⇒ Utilisations de représentations et d'opérations peu sûres et peu efficaces.

Système de gestion de bases de données (SGBD)

Un ensemble de routines permettant une gestion efficace et sûre de tous types de données.

Système de gestion de bases de données (SGBD)

Un ensemble de routines permettant une gestion efficace et sûre de tous types de données.
Regroupement de données pour former une *base* de données.
possibilité de formuler des requêtes faisant intervenir toutes les données de la base.

Système de gestion de bases de données (SGBD)

Un ensemble de routines permettant une gestion efficace et sûre de tous types de données.
Regroupement de données pour former une *base* de données.
possibilité de formuler des requêtes faisant intervenir toutes les données de la base.

Avantages

- Indépendance logique et physique. Pas de redondance, Meilleure intégrité.

Système de gestion de bases de données (SGBD)

Un ensemble de routines permettant une gestion efficace et sûre de tous types de données.
Regroupement de données pour former une *base* de données.
possibilité de formuler des requêtes faisant intervenir toutes les données de la base.

Avantages

- Indépendance logique et physique. Pas de redondance, Meilleure intégrité.
- Gestion de tous types de données adaptable à tout type d'application.

Système de gestion de bases de données (SGBD)

Un ensemble de routines permettant une gestion efficace et sûre de tous types de données.
Regroupement de données pour former une *base* de données.
possibilité de formuler des requêtes faisant intervenir toutes les données de la base.

Avantages

- Indépendance logique et physique. Pas de redondance, Meilleure intégrité.
- Gestion de tous types de données adaptable à tout type d'application.
- Interface avec des langages hôtes.

Chapitre I

Bases de données

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.
- Accéder efficacement à ces données.

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.
- Accéder efficacement à ces données.
- Supporter un modèle de données abstrait qui est la façon dont l'utilisateur perçoit les données.

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.
- Accéder efficacement à ces données.
- Supporter un modèle de données abstrait qui est la façon dont l'utilisateur perçoit les données.
- LDD : Langage de définition des données.
  LMD : Langage de manipulation de données (de requêtes).

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.
- Accéder efficacement à ces données.
- Supporter un modèle de données abstrait qui est la façon dont l'utilisateur perçoit les données.
- LDD : Langage de définition des données.
  LMD : Langage de manipulation de données (de requêtes).
- Permettre un accès concurrent aux données.

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.
- Accéder efficacement à ces données.
- Supporter un modèle de données abstrait qui est la façon dont l'utilisateur perçoit les données.
- LDD : Langage de définition des données.
  LMD : Langage de manipulation de données (de requêtes).
- Permettre un accès concurrent aux données.
- Vérifier les accès des utilisateurs (sécurité).

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.
- Accéder efficacement à ces données.
- Supporter un modèle de données abstrait qui est la façon dont l'utilisateur perçoit les données.
- LDD : Langage de définition des données.
  LMD : Langage de manipulation de données (de requêtes).
- Permettre un accès concurrent aux données.
- Vérifier les accès des utilisateurs (sécurité).
- Conserver la validité des données (intégrité).

Fonctions d'un SGBD

- Gérer de grandes quantités de données *persistantes*.
- Accéder efficacement à ces données.
- Supporter un modèle de données abstrait qui est la façon dont l'utilisateur perçoit les données.
- LDD : Langage de définition des données.
  LMD : Langage de manipulation de données (de requêtes).
- Permettre un accès concurrent aux données.
- Vérifier les accès des utilisateurs (sécurité).
- Conserver la validité des données (intégrité).
- Récupérer les données après panne.

Langage de définition des données

Description de la nature des données et de leurs *liens logiques*.
Permet la définition du schéma conceptuel.
Description des schémas externes.

Langage de définition des données

Description de la nature des données et de leurs *liens logiques*.
Permet la définition du schéma conceptuel.
Description des schémas externes.

- Permet d'exprimer des *contraintes d'intégrité* sur les données.
  Valeurs permises pour certaines données.
  ex : $prix\_vente > prix\_achat$.

Langage de définition des données

Description de la nature des données et de leurs *liens logiques*.
Permet la définition du schéma conceptuel.
Description des schémas externes.

- Permet d'exprimer des *contraintes d'intégrité* sur les données.
  Valeurs permises pour certaines données.
  ex : $prix\_vente > prix\_achat$.
- Définition de clés.
- Définition des droits d'accès aux données.
- ...

Langage de manipulation de données

Manipulation, modification, ajout, suppression des données.

- **SGBD avec langage hôte**
  Le LMD n'est pas un langage en soi, mais est une « sur-couche » d'un langage de programmation tel que C.

Langage de manipulation de données

Manipulation, modification, ajout, suppression des données.

- **SGBD avec langage hôte**
  Le LMD n'est pas un langage en soi, mais est une « sur-couche » d'un langage de programmation tel que C.
- **SGBD autonome**
  Interpréteur (et/ou compilateur) fourni avec le SGBD qui exécute des ordres rédigés dans un langage propre au SGBD (ex : *SQL*).

Langage de manipulation de données

Manipulation, modification, ajout, suppression des données.

- **SGBD avec langage hôte**
  Le LMD n'est pas un langage en soi, mais est une « sur-couche » d'un langage de programmation tel que C.
- **SGBD autonome**
  Interpréteur (et/ou compilateur) fourni avec le SGBD qui exécute des ordres rédigés dans un langage propre au SGBD (ex : *SQL*).

. . . ou utilisation d'un LMD autonome à partir d'un langage de programmation.
ex : *SQL dans Java*.

Module de contrôle d'accès aux données

- Vérification des droits (lecture, modification, ajout)

Module de contrôle d'accès aux données

- Vérification des droits (lecture, modification, ajout)
- Gestion des accès concurrents.
  Risque d'une perte d'intégrité.
  Techniques de verrouillage des données.

Module de contrôle d'accès aux données

- Vérification des droits (lecture, modification, ajout)
- Gestion des accès concurrents.
  Risque d'une perte d'intégrité.
  Techniques de verrouillage des données.

SGBD et SE

Habituellement un SGBD est construit au dessus d'un SE, et réutilise son système de gestion de fichiers (même si l'utilisateur du SGBD ne verra pas les fichiers), sa gestion de la mémoire, des exclusions mutuelles.

Modules d'administration

- Initialisation de la base (à vide, par importation).

Modules d'administration

- Initialisation de la base (à vide, par importation).
- Copie / Sauvegarde / Restauration.

Modules d'administration

- Initialisation de la base (à vide, par importation).
- Copie / Sauvegarde / Restauration.
- Gestion des journaux d'historique.
  - Identification des programmes, utilisateurs, postes.
  - Charge du serveur.
  - ...

Modules d'administration

- Initialisation de la base (à vide, par importation).
- Copie / Sauvegarde / Restauration.
- Gestion des journaux d'historique.
  - Identification des programmes, utilisateurs, postes.
  - Charge du serveur.
  - ...
- Réorganisation de la base (niveau physique).
- ...

Chapitre I

Bases de données

Modélisation et modèles

Définition (Modelisation)

Expression du monde réel observé utilisant les concepts d'un *modèle* de représentation.

Modélisation et modèles

Définition (Modelisation)

Expression du monde réel observé utilisant les concepts d'un *modèle* de représentation.

Représentation de la partie du monde « utile » à l'application.

Modélisation et modèles

Définition (Modelisation)

Expression du monde réel observé utilisant les concepts d'un *modèle* de représentation.

Représentation de la partie du monde « utile » à l'application.
Par exemple, la modélisation d'une « personne » sera différente dans une application gérant l'inscription d'étudiants à des diplômes, et dans la BD de la sécurité sociale.
Les données choisies pour la modélisation (d'une personne) différent.

Modélisation et modèles

Définition (Modelisation)

Expression du monde réel observé utilisant les concepts d'un *modèle* de représentation.

Représentation de la partie du monde « utile » à l'application.
Par exemple, la modélisation d'une « personne » sera différente dans une application gérant l'inscription d'étudiants à des diplômes, et dans la BD de la sécurité sociale.
Les données choisies pour la modélisation (d'une personne) différent.

Il existe plusieurs modèles de représentation.

- 1960 Modèle hiérarchique, modèle réseau.
- 1970 Modèle relationnel
- 1990 Modèle objet

Chapitre II

Modèle relationnel

Modèle relationnel

Fondé sur le concept mathématique de *relation* (sous-ensemble du *produit cartésien* de différents *domaines*).

Modèle relationnel

Fondé sur le concept mathématique de *relation* (sous-ensemble du *produit cartésien* de différents *domaines*).

Le modèle le plus utilisé dans les bases de données (SGBDR) :

- fondements mathématiques.

Modèle relationnel

Fondé sur le concept mathématique de *relation* (sous-ensemble du *produit cartésien* de différents *domaines*).

Le modèle le plus utilisé dans les bases de données (SGBDR) :

- fondements mathématiques.
- simple à comprendre.

Modèle relationnel

Fondé sur le concept mathématique de *relation* (sous-ensemble du *produit cartésien* de différents *domaines*).

Le modèle le plus utilisé dans les bases de données (SGBDR) :

- fondements mathématiques.
- simple à comprendre.
- muni de langages de manipulation.

Modèle relationnel

Fondé sur le concept mathématique de *relation* (sous-ensemble du *produit cartésien* de différents *domaines*).

Le modèle le plus utilisé dans les bases de données (SGBDR) :

- fondements mathématiques.
- simple à comprendre.
- muni de langages de manipulation.
- homogène : représentation de toutes les informations sous une même forme.

Chapitre II

Modèle relationnel

Données

Définition (Domaine)

D : Ensemble d'éléments de même type.

Données

Définition (Domaine)

D : Ensemble d'éléments de même type.

Exemple (Domaine)

$D_1 = \{RedHat\ Linux\ 9.0, Windows\ 2000, Windows\ xp\}$
$D_2 = \{RedHat, Microsoft\ corp\}$
$D_3 = \boldsymbol{N}$
$D_4 = Dates$

Données

Définition (Relation)

R : sous-ensemble du produit cartésien de n domaines D_i (non nécessairement distincts).
$R \subseteq D_1 \times D_2 \times \ldots D_n$.

Données

Définition (Relation)

R : sous-ensemble du produit cartésien de n domaines D_i (non nécessairement distincts).
$R \subseteq D_1 \times D_2 \times \ldots D_n$.

Exemple (Relation)

$Systèmes \subseteq D_1 \times D_2 \times D_3 \times D_4$

Données

Définition (Relation)

R : sous-ensemble du produit cartésien de n domaines D_i (non nécessairement distincts).
$R \subseteq D_1 \times D_2 \times \ldots D_n$.

Exemple (Relation)

$Systèmes \subseteq D_1 \times D_2 \times D_3 \times D_4$

Relation *n-aire* ou d'*arité n* ou de *degré n*.
Systèmes est d'arité 4.

Données

Définition (Attribut)

nom donné à une composante de la relation.

indispensable si un même domaine est utilisé plusieurs fois.

Données

Définition (Attribut)

nom donné à une composante de la relation.

indispensable si un même domaine est utilisé plusieurs fois.

Exemple (Schéma conceptuel)

Systèmes(version, éditeur, prix, date_install)

Données

Définition (Attribut)

nom donné à une composante de la relation.

indispensable si un même domaine est utilisé plusieurs fois.

Exemple (Schéma conceptuel)

Systèmes(version, éditeur, prix, date_install)

Définition (Tuple (ou n-uplet))

élément de la relation. $t \in R$

Données

Définition (Attribut)

nom donné à une composante de la relation.

indispensable si un même domaine est utilisé plusieurs fois.

Exemple (Schéma conceptuel)

Systèmes(version, éditeur, prix, date_install)

Définition (Tuple (ou n-uplet))

élément de la relation. $t \in R$

Exemple (Tuple)

$(d_1, d_2, \ldots d_n) \in R$.

$t_1 = (RedHat\ Linux\ 9.0, RedHat, 0, 23/10/2005)$

Données

Notation

Soit A un attribut d'une relation R et t un tuple de R on note $t.A$ la valeur de l'attribut A du tuple t.

$t_1.\mathit{éditeur} = \mathit{RedHat}$

Données

Notation

Soit A un attribut d'une relation R et t un tuple de R on note $t.A$ la valeur de l'attribut A du tuple t.

$t_1.\mathit{éditeur} = \mathit{RedHat}$

Soit R une relation, $a(R)$ est l'ensemble des attributs de R.

$a(\mathit{Systèmes}) = \{\mathit{version}, \mathit{éditeur}, \mathit{prix}, \mathit{date\_install}\}$

Données

Définition (Cardinalité d'une relation)

nombre de tuples de la relation.

Définition (Instance du schéma)

Un ensemble de tuples. (définition extensionnelle de la relation)

Données

Définition (Cardinalité d'une relation)

nombre de tuples de la relation.

Définition (Instance du schéma)

Un ensemble de tuples. (définition extensionnelle de la relation)

Exemple (Instance d'un schéma)

version	**éditeur**	**prix**	**date_install**
RedHat Linux 9.0	RedHat	0	23/10/2005
Windows 2000	Microsoft corp	350	10/05/2005

Clé

Il est souvent nécessaire de faire référence dans une relation à des tuples d'une autre relation.

Exemple

Stockage des logiciels disponibles à partir d'une installation d'un SE. *Logiciels(???, nom_logiciel, éditeur)*

Clé

Il est souvent nécessaire de faire référence dans une relation à des tuples d'une autre relation.

Exemple

Stockage des logiciels disponibles à partir d'une installation d'un SE. *Logiciels(???, nom_logiciel, éditeur)*

Définition (Clé)

attribut ou ensemble d'attributs dont les valeurs identifient de manière unique chaque tuple de la relation.

Clé

Il est souvent nécessaire de faire référence dans une relation à des tuples d'une autre relation.

Exemple

Stockage des logiciels disponibles à partir d'une installation d'un SE. *Logiciels(???, nom_logiciel, éditeur)*

Définition (Clé)

attribut ou ensemble d'attributs dont les valeurs identifient de manière unique chaque tuple de la relation.

Exemple

version est la clé primaire de Systèmes.
Il ne peut y avoir deux tuples d'une relation ayant mêmes valeurs des attributs de clé.

Clé

Il est souvent nécessaire de faire référence dans une relation à des tuples d'une autre relation.

Exemple

Stockage des logiciels disponibles à partir d'une installation d'un SE. *Logiciels(???, nom_logiciel, éditeur)*

Définition (Clé)

attribut ou ensemble d'attributs dont les valeurs identifient de manière unique chaque tuple de la relation.

Exemple

version est la clé primaire de Systèmes.
Il ne peut y avoir deux tuples d'une relation ayant mêmes valeurs des attributs de clé.
Logiciels(version_Systèmes, nom_logiciel, éditeur)

Exemple

Exemple

Représentation de commandes effectuées par des clients, de produits achetés chez des fournisseurs.

Fournisseurs(f_nom, adresse)
Produits(p_nom, f_nom, prix)
Commandes(cmd_numéro, date, cli_nom)
Clients(cli_nom, adresse)
Contenu_commandes(cmd_numéro, p_nom, quantité)

Exemple

Exemple

Représentation de commandes effectuées par des clients, de produits achetés chez des fournisseurs.

Fournisseurs(f_nom, adresse)
Produits(p_nom, f_nom, prix)
Commandes(cmd_numéro, date, cli_nom)
Clients(cli_nom, adresse)
Contenu_commandes(cmd_numéro, p_nom, quantité)

Exemple

Exemple

Représentation de commandes effectuées par des clients, de produits achetés chez des fournisseurs.

Fournisseurs(f_nom, adresse)
Produits(p_nom, f_nom, prix)
Commandes(cmd_numéro, date, cli_nom)
Clients(cli_nom, adresse)
Contenu_commandes(cmd_numéro, p_nom, quantité)

Fournisseurs $= \{$(*Canson, 1 rue de la Paix*), (*Pelikan, 3 boulevard Foch*)$\}$
Produits $= \{$(*500N, Canson, 12*), (*Cartouche HP500, Pelikan, 20*), (*1000C, Canson, 25*)$\}$
Commandes $= \{$(*23, 21/01/2002, Rudoux*)$\}$
Clients $= \{$(*Rudoux, Place de l'horloge*), (*Loucondre, Rue des directions*)$\}$
Contenu_commandes $= \{$(*23, 500N, 2*), (*23, Cartouche HP500, 1*)$\}$

Chapitre II

Modèle relationnel

Langages de manipulation

Le modèle relationnel ne définit pas *qu'*une méthode de représentation, mais aussi des langages permettant la manipulation des relations.

Langages de manipulation

Le modèle relationnel ne définit pas *qu'*une méthode de représentation, mais aussi des langages permettant la manipulation des relations.
Ces langages ne sont pas procéduraux, et sont plus simples d'utilisation que les langages de programmation. L'utilisateur n'a pas à exprimer **comment** les résultats doivent être obtenus (parcours des relations, etc.), il lui suffit d'indiquer le résultat qui l'intéresse (**quoi**).

Langages de manipulation

Le modèle relationnel ne définit pas *qu'*une méthode de représentation, mais aussi des langages permettant la manipulation des relations.
Ces langages ne sont pas procéduraux, et sont plus simples d'utilisation que les langages de programmation. L'utilisateur n'a pas à exprimer **comment** les résultats doivent être obtenus (parcours des relations, etc.), il lui suffit d'indiquer le résultat qui l'intéresse (**quoi**).

- **Langage algébrique** : combinaison de relations en utilisant des opérateurs. Les relations sont considérées comme des ensembles de tuples $\rightarrow$ opérations ensemblistes.

Langages de manipulation

Le modèle relationnel ne définit pas *qu'*une méthode de représentation, mais aussi des langages permettant la manipulation des relations.
Ces langages ne sont pas procéduraux, et sont plus simples d'utilisation que les langages de programmation. L'utilisateur n'a pas à exprimer **comment** les résultats doivent être obtenus (parcours des relations, etc.), il lui suffit d'indiquer le résultat qui l'intéresse (**quoi**).

- **Langage algébrique** : combinaison de relations en utilisant des opérateurs. Les relations sont considérées comme des ensembles de tuples $\rightarrow$ opérations ensemblistes.
- **Langage logique** : spécification de relations par des formules logiques.

Chapitre II

Modèle relationnel

Chapitre II

Modèle relationnel

Algèbre relationnelle
Opérations de base

5 opérateurs primitifs à partir desquels les autres peuvent être dérivés.

Algèbre relationnelle
Opérations de base

5 opérateurs primitifs à partir desquels les autres peuvent être dérivés.

Définition (Union et Différence)

R et S sont définies sur le même ensemble d'attributs $A_1, A_2, \ldots A_n$.

$U = R \cup S = \{t \in A_1 \times A_2 \times \ldots \times A_n / t \in R \text{ ou } t \in S\}$

$D = R - S = \{t \in A_1 \times A_2 \times \ldots \times A_n / t \in R \text{ et } t \notin S\}$

Algèbre relationnelle
Opérations de base

5 opérateurs primitifs à partir desquels les autres peuvent être dérivés.

Définition (Union et Différence)

R et S sont définies sur le même ensemble d'attributs $A_1, A_2, \ldots A_n$.

$U = R \cup S = \{t \in A_1 \times A_2 \times \ldots \times A_n / t \in R \text{ ou } t \in S\}$

$D = R - S = \{t \in A_1 \times A_2 \times \ldots \times A_n / t \in R \text{ et } t \notin S\}$

Définition (Produit cartésien)

R est définie sur l'ensemble d'attributs $A_1, A_2, \ldots A_n$. S est définie sur l'ensemble d'attributs $B_1, B_2, \ldots B_m$.

$P = R \times S = \{(a_1, \ldots, a_n, b_1, \ldots, b_m) \in$
$A_1 \times \ldots \times A_n \times B_1 \times \ldots \times B_m / (a_1, \ldots, a_n) \in R \text{ et } (b_1, \ldots, b_m) \in S\}$

Définition (Projection (sur un ensemble d'attributs))

R est définie sur l'ensemble d'attributs $A_1, A_2, \ldots A_n$.
$\Pi_{A_{i_1} \ldots A_{i_m}}(R) = \{t \in A_{i_1} \times \ldots A_{i_m} / \exists t' \in R$ avec $t'.A_{i_1} = t.A_{i_1}$ et $\ldots$ et $t'.A_{i_m} = t.A_{i_m}\}$

Définition (Projection (sur un ensemble d'attributs))

R est définie sur l'ensemble d'attributs $A_1, A_2, \ldots A_n$.
$\Pi_{A_{i_1} \ldots A_{i_m}}(R) = \{t \in A_{i_1} \times \ldots A_{i_m} / \exists t' \in R$ avec $t'.A_{i_1} = t.A_{i_1}$ et $\ldots$ et $t'.A_{i_m} = t.A_{i_m}\}$

« découpage vertical », « suppression de certaines colonnes ».

Définition (Projection (sur un ensemble d'attributs))

R est définie sur l'ensemble d'attributs $A_1, A_2, \ldots A_n$.
$\Pi_{A_{i_1} \ldots A_{i_m}}(R) = \{t \in A_{i_1} \times \ldots A_{i_m} / \exists t' \in R$ avec $t'.A_{i_1} = t.A_{i_1}$ et $\ldots$ et $t'.A_{i_m} = t.A_{i_m}\}$

« découpage vertical », « suppression de certaines colonnes ».

Exemple

EDT

Salle	***Matière***	***Jour***
A116	*BD*	*Lundi*
A115	*Objets*	*Lundi*
H007	*BD*	*Mardi*
A116	*Réseaux*	*Mardi*

$\Pi_{Salle}(EDT)$

Salle
A116
A115
H007

Définition (Sélection)

Soit C une formule logique utilisant des attributs de R.
$\sigma_C(R) = \{t \in R / C(t) = \text{vrai}\}$

« découpage horizontal », *Caractérisation d'un sous-ensemble de la relation.* Possibilité d'utiliser des opérateurs selon le type des attributs.

Définition (Sélection)

Soit C une formule logique utilisant des attributs de R.
$\sigma_C(R) = \{t \in R / C(t) = \text{vrai}\}$

« découpage horizontal », *Caractérisation d'un sous-ensemble de la relation.* Possibilité d'utiliser des opérateurs selon le type des attributs.

Exemple

$\sigma_{Matière\ =\ BD}(EDT)$

Salle	**Matière**	**Jour**
A116	*BD*	*Lundi*
H007	*BD*	*Mardi*

$\sigma_{(Salle\ =\ A116) \wedge (Jour\ =\ Lundi)}(EDT)$

Salle	**Matière**	**Jour**
A116	*BD*	*Lundi*

Chapitre II

Modèle relationnel

Autres opérations

Trois nouveaux opérateurs qui peuvent être définis à partir des 5 opérateurs de base.

Définition (Intersection)

R et S sont définies sur le même ensemble d'attributs $A_1, A_2, \ldots A_n$.

$I = R \cap S = \{t \in A_1 \times A_2 \times \ldots \times A_n / t \in R \text{ et } t \in S\}$

Autres opérations

Trois nouveaux opérateurs qui peuvent être définis à partir des 5 opérateurs de base.

Définition (Intersection)

R et S sont définies sur le même ensemble d'attributs $A_1, A_2, \ldots A_n$.
$I = R \cap S = \{t \in A_1 \times A_2 \times \ldots \times A_n / t \in R \text{ et } t \in S\}$

$R \cap S = R - (R - S)$

Jointure

Définition (Jointure)

Soit R et S deux relations définies sur des ensembles d'attributs à l'intersection non vide $a(R) \cap a(S) = \{A_1 \dots A_n\}$.

Jointure

Définition (Jointure)

Soit R et S deux relations définies sur des ensembles d'attributs à l'intersection non vide $a(R) \cap a(S) = \{A_1 \dots A_n\}$.
$R \bowtie S$ est une relation définie sur l'union des attributs de R et S contenant tous les tuples composés d'un tuple de R et un tuple de S qui coïncident sur $A_1 \dots A_n$.

Jointure

Définition (Jointure)

Soit R et S deux relations définies sur des ensembles d'attributs à l'intersection non vide $a(R) \cap a(S) = \{A_1 \dots A_n\}$.
$R \bowtie S$ est une relation définie sur l'union des attributs de R et S contenant tous les tuples composés d'un tuple de R et un tuple de S qui coïncident sur $A_1 \dots A_n$.

Expression de $\bowtie$ à partir de Π, σ et $\times$

Soit $a(R) \cap a(S) = \{A_1, \dots A_n\}$.
$a(R) = \{A_1, \dots A_n, B_1, \dots B_k\}$ et $a(S) = \{A_1, \dots A_n, C_1, \dots C_l\}$.

Jointure

Définition (Jointure)

Soit R et S deux relations définies sur des ensembles d'attributs à l'intersection non vide $a(R) \cap a(S) = \{A_1 \dots A_n\}$.
$R \bowtie S$ est une relation définie sur l'union des attributs de R et S contenant tous les tuples composés d'un tuple de R et un tuple de S qui coïncident sur $A_1 \dots A_n$.

Expression de $\bowtie$ à partir de Π, σ et $\times$

Soit $a(R) \cap a(S) = \{A_1, \dots A_n\}$.
$a(R) = \{A_1, \dots A_n, B_1, \dots B_k\}$ et $a(S) = \{A_1, \dots A_n, C_1, \dots C_l\}$.

$$R \bowtie S = \prod_{A_1,\dots,A_n,B_1,\dots,B_k,C_1,\dots,C_l} (\sigma_{\bigwedge_{i\in[1,n]}(R.A_i=S.A_i)}(R \times S))$$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

$Fournisseurs = \{(Canson,\ 1\ rue\ de\ la\ Paix), (Pelikan,\ 3\ boulevard\ Foch)\}$
$Produits = \{(500N,\ Canson,\ 12), (Cartouche\ HP500,\ Pelikan,\ 20), (1000C,\ Canson,\ 25)\}$
$Commandes = \{(23,\ 21/01/2002,\ Rudoux)\}$
$Clients = \{(Rudoux,\ Place\ de\ l'horloge), (Loucondre,\ Rue\ des\ directions)\}$

$Commandes \bowtie Clients$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

$Fournisseurs = \{(Canson, 1\ rue\ de\ la\ Paix), (Pelikan, 3\ boulevard\ Foch)\}$
$Produits = \{(500N, Canson, 12), (Cartouche\ HP500, Pelikan, 20), (1000C, Canson, 25)\}$
$Commandes = \{(23, 21/01/2002, Rudoux)\}$
$Clients = \{(Rudoux, Place\ de\ l'horloge), (Loucondre, Rue\ des\ directions)\}$

$Commandes \bowtie Clients\ (cmd\_numéro, date, cli\_nom, adresse)$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

$Fournisseurs = \{(Canson, 1\ rue\ de\ la\ Paix), (Pelikan, 3\ boulevard\ Foch)\}$
$Produits = \{(500N, Canson, 12), (Cartouche\ HP500, Pelikan, 20), (1000C, Canson, 25)\}$
$Commandes = \{(23, 21/01/2002, Rudoux)\}$
$Clients = \{(Rudoux, Place\ de\ l'horloge), (Loucondre, Rue\ des\ directions)\}$

$Commandes \bowtie Clients\ (cmd\_numéro, date, cli\_nom, adresse)$
$Commandes \bowtie Clients =$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

$Fournisseurs = \{(Canson, 1\ rue\ de\ la\ Paix), (Pelikan, 3\ boulevard\ Foch)\}$
$Produits = \{(500N, Canson, 12), (Cartouche\ HP500, Pelikan, 20), (1000C, Canson, 25)\}$
$Commandes = \{(23, 21/01/2002, Rudoux)\}$
$Clients = \{(Rudoux, Place\ de\ l'horloge), (Loucondre, Rue\ des\ directions)\}$

$Commandes \bowtie Clients\ (cmd\_numéro, date, cli\_nom, adresse)$
$Commandes \bowtie Clients = \{(23, 21/01/2002, Rudoux, Place\ de\ l'horloge)\}$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

$Fournisseurs = \{(Canson, 1\ rue\ de\ la\ Paix), (Pelikan, 3\ boulevard\ Foch)\}$
$Produits = \{(500N, Canson, 12), (Cartouche\ HP500, Pelikan, 20), (1000C, Canson, 25)\}$
$Commandes = \{(23, 21/01/2002, Rudoux)\}$
$Clients = \{(Rudoux, Place\ de\ l'horloge), (Loucondre, Rue\ des\ directions)\}$

$Commandes \bowtie Clients\ (cmd\_numéro, date, cli\_nom, adresse)$
$Commandes \bowtie Clients = \{(23, 21/01/2002, Rudoux, Place\ de\ l'horloge)\}$
$Produits \bowtie Fournisseurs$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

$Fournisseurs = \{(Canson, 1\ rue\ de\ la\ Paix), (Pelikan, 3\ boulevard\ Foch)\}$
$Produits = \{(500N, Canson, 12), (Cartouche\ HP500, Pelikan, 20), (1000C, Canson, 25)\}$
$Commandes = \{(23, 21/01/2002, Rudoux)\}$
$Clients = \{(Rudoux, Place\ de\ l'horloge), (Loucondre, Rue\ des\ directions)\}$

$Commandes \bowtie Clients\ (cmd\_numéro, date, cli\_nom, adresse)$
$Commandes \bowtie Clients = \{(23, 21/01/2002, Rudoux, Place\ de\ l'horloge)\}$
$Produits \bowtie Fournisseurs\ (p\_nom, f\_nom, prix, adresse)$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

Fournisseurs $= \{$(*Canson, 1 rue de la Paix*), (*Pelikan, 3 boulevard Foch*)$\}$
Produits $= \{$(*500N, Canson, 12*), (*Cartouche HP500, Pelikan, 20*), (*1000C, Canson, 25*)$\}$
Commandes $= \{$(*23, 21/01/2002, Rudoux*)$\}$
Clients $= \{$(*Rudoux, Place de l'horloge*), (*Loucondre, Rue des directions*)$\}$

Commandes $\bowtie$ *Clients (cmd_numéro, date, cli_nom, adresse)*
Commandes $\bowtie$ *Clients* $= \{$(*23, 21/01/2002, Rudoux, Place de l'horloge*)$\}$
Produits $\bowtie$ *Fournisseurs (p_nom, f_nom, prix, adresse)*

Produits $\bowtie$ *Fournisseurs* $=$

Exemple (Jointure)

Fournisseurs(f_nom, adresse) *Produits(p_nom, f_nom, prix)*
Commandes(cmd_numéro, date, cli_nom) *Clients(cli_nom, adresse)*

Fournisseurs $= \{$(*Canson, 1 rue de la Paix*), (*Pelikan, 3 boulevard Foch*)$\}$
Produits $= \{$(*500N, Canson, 12*), (*Cartouche HP500, Pelikan, 20*), (*1000C, Canson, 25*)$\}$
Commandes $= \{$(*23, 21/01/2002, Rudoux*)$\}$
Clients $= \{$(*Rudoux, Place de l'horloge*), (*Loucondre, Rue des directions*)$\}$

Commandes $\bowtie$ *Clients (cmd_numéro, date, cli_nom, adresse)*
Commandes $\bowtie$ *Clients* $= \{$(*23, 21/01/2002, Rudoux, Place de l'horloge*)$\}$
Produits $\bowtie$ *Fournisseurs (p_nom, f_nom, prix, adresse)*
Produits $\bowtie$ *Fournisseurs* $= \{$(*500N, Canson, 12, 1 rue de la Paix*), (*Cartouche HP500, Pelikan, 20, 3 boulevard Foch*), (*1000C, Canson, 25, 1 rue de la Paix*)$\}$

Quotient

Définition (Quotient)

Soit R et S deux relations telles que $a(S) \subseteq a(R)$.
$a(S) = \{A_1, \ldots, A_n\}$ et $a(R) = a(S) \cup \{B_1, \ldots, B_p\}$.

Quotient

Définition (Quotient)

Soit R et S deux relations telles que $a(S) \subseteq a(R)$.
$a(S) = \{A_1, \ldots, A_n\}$ et $a(R) = a(S) \cup \{B_1, \ldots, B_p\}$.

$Q = R \div S$ est la relation définie sur $\{B_1, \ldots, B_p\}$ et composée des tuples $(b_1, \ldots, b_p)$ tels que pour tout tuple $(a_1, \ldots, a_n) \in S$, $(a_1, \ldots, a_n, b_1, \ldots, b_p) \in R$.

Quotient

Définition (Quotient)

Soit R et S deux relations telles que $a(S) \subseteq a(R)$.
$a(S) = \{A_1, \ldots, A_n\}$ et $a(R) = a(S) \cup \{B_1, \ldots, B_p\}$.

$Q = R \div S$ est la relation définie sur $\{B_1, \ldots, B_p\}$ et composée des tuples $(b_1, \ldots, b_p)$ tels que pour tout tuple $(a_1, \ldots, a_n) \in S$, $(a_1, \ldots, a_n, b_1, \ldots, b_p) \in R$.

Plus formellement...

1. $Q \subseteq \prod_{a(R)-a(S)}(R)$
2. $Q \times S \subseteq R$
3. Q est maximale par (1) et (2).

Quotient

Définition (Quotient)

Soit R et S deux relations telles que $a(S) \subseteq a(R)$.
$a(S) = \{A_1, \ldots, A_n\}$ et $a(R) = a(S) \cup \{B_1, \ldots, B_p\}$.

$Q = R \div S$ est la relation définie sur $\{B_1, \ldots, B_p\}$ et composée des tuples $(b_1, \ldots, b_p)$ tels que pour tout tuple $(a_1, \ldots, a_n) \in S$, $(a_1, \ldots, a_n, b_1, \ldots, b_p) \in R$.

Plus formellement...

1. $Q \subseteq \prod_{a(R)-a(S)}(R)$
2. $Q \times S \subseteq R$
3. Q est maximale par (1) et (2).

Expression de $\div$ à partir de Π, $\times$ et $-$

$$R \div S = \Pi_{a(R)-a(S)}(R) - \Pi_{a(R)-a(S)}((\Pi_{a(R)-a(S)}(R) \times S) - R)$$

Exemple (Quotient)

R

Nom	***Sport***	***Niveau***
Jean	*Hauteur*	*2*
Pierre	*100 m*	*1*
Jean	*100 m*	*2*
Pierre	*Perche*	*1*
Pierre	*Longueur*	*1*
Jean	*Longueur*	*2*

S

Sport	***Niveau***
100 m	*1*
Perche	*1*
Longueur	*1*

Exemple (Quotient)

R

Nom	**Sport**	**Niveau**
Jean	*Hauteur*	*2*
Pierre	*100 m*	*1*
Jean	*100 m*	*2*
Pierre	*Perche*	*1*
Pierre	*Longueur*	*1*
Jean	*Longueur*	*2*

S

Sport	**Niveau**
100 m	*1*
Perche	*1*
Longueur	*1*

$R \div S$

Nom
Pierre

Semi-jointure

Définition (Semi-jointure)

$$R \ltimes S = \prod_{a(R)}(R \bowtie S)$$

Semi-jointure

Définition (Semi-jointure)

$R \ltimes S = \prod_{a(R)}(R \bowtie S)$

Attention : $R \bowtie S = S \bowtie R$ mais $R \ltimes S \neq S \ltimes R$
(sauf si $a(R) = a(S)$)

Semi-jointure

Définition (Semi-jointure)

$R \ltimes S = \prod_{a(R)}(R \bowtie S)$

Attention : $R \bowtie S = S \bowtie R$ mais $R \ltimes S \neq S \ltimes R$
(sauf si $a(R) = a(S)$)

Exemple

Clients $\ltimes$ *Commandes*

Semi-jointure

Définition (Semi-jointure)

$R \ltimes S = \prod_{a(R)}(R \bowtie S)$

Attention : $R \bowtie S = S \bowtie R$ mais $R \ltimes S \neq S \ltimes R$
(sauf si $a(R) = a(S)$)

Exemple

Clients $\ltimes$ *Commandes (cli_nom, adresse)*

Semi-jointure

Définition (Semi-jointure)

$R \ltimes S = \prod_{a(R)}(R \bowtie S)$

Attention : $R \bowtie S = S \bowtie R$ mais $R \ltimes S \neq S \ltimes R$
(sauf si $a(R) = a(S)$)

Exemple

Clients $\ltimes$ *Commandes* (*cli_nom, adresse*)
Clients $\ltimes$ *Commandes* =

Semi-jointure

Définition (Semi-jointure)

$R \ltimes S = \prod_{a(R)}(R \bowtie S)$

Attention : $R \bowtie S = S \bowtie R$ mais $R \ltimes S \neq S \ltimes R$
(sauf si $a(R) = a(S)$)

Exemple

Clients $\ltimes$ *Commandes (cli_nom, adresse)*
Clients $\ltimes$ *Commandes* $= \{(Rudoux, Place\ de\ l'horloge)\}$

θ-jointure

Définition (θ-jointure)

$$R \bowtie_{R.A\,\theta\,S.B} S = \sigma_{R.A\,\theta\,S.B}(R \times S)$$

avec $\theta \in \{=, \neq, <, >, \leq, \geq\}$.

Chapitre II

Modèle relationnel

L'algèbre relationnelle comme langage de requêtes

Les opérateurs de l'algèbre relationnelle permettent d'exprimer toute requête non récursive. On peut donc utiliser l'algèbre relationnelle pour exprimer une requête.

L'algèbre relationnelle comme langage de requêtes

Les opérateurs de l'algèbre relationnelle permettent d'exprimer toute requête non récursive. On peut donc utiliser l'algèbre relationnelle pour exprimer une requête.

Exemple

L'adresse du client de la facture 23.

L'algèbre relationnelle comme langage de requêtes

Les opérateurs de l'algèbre relationnelle permettent d'exprimer toute requête non récursive. On peut donc utiliser l'algèbre relationnelle pour exprimer une requête.

Exemple

L'adresse du client de la facture 23.

$\Pi_{adresse}(Clients \bowtie (\sigma_{cmd\_numéro=23} Commandes))$

L'algèbre relationnelle comme langage de requêtes

Les opérateurs de l'algèbre relationnelle permettent d'exprimer toute requête non récursive. On peut donc utiliser l'algèbre relationnelle pour exprimer une requête.

Exemple

L'adresse du client de la facture 23.

$\Pi_{adresse}(Clients \bowtie (\sigma_{cmd\_numéro=23} Commandes))$

$\{(Place\ de\ l'horloge)\}$

L'algèbre relationnelle comme langage de requêtes

Les opérateurs de l'algèbre relationnelle permettent d'exprimer toute requête non récursive. On peut donc utiliser l'algèbre relationnelle pour exprimer une requête.

Exemple

L'adresse du client de la facture 23.

$\Pi_{adresse}(Clients \bowtie (\sigma_{cmd\_numéro=23} Commandes))$

$\{(Place\ de\ l'horloge)\}$

cependant...

Langage pas très facile : l'utilisateur doit connaître le schéma de la base pour naviguer dans les relations.

L'algèbre relationnelle comme langage de requêtes

Les opérateurs de l'algèbre relationnelle permettent d'exprimer toute requête non récursive. On peut donc utiliser l'algèbre relationnelle pour exprimer une requête.

Exemple

L'adresse du client de la facture 23.

$\Pi_{adresse}(Clients \bowtie (\sigma_{cmd\_numéro=23} Commandes))$

$\{(Place\ de\ l'horloge)\}$

cependant...

Langage pas très facile : l'utilisateur doit connaître le schéma de la base pour naviguer dans les relations.

Ce n'est pas exacement un langage déclaratif (*quoi*), puisqu'une expression décrit *comment* la requête doit être résolue.

Chapitre III

SQL

Chapitre III

SQL

Présentation de SQL

SQL = Structured Query Language = langage d'interrogation structuré Langage de gestion de bases de données relationnelles pour

- interroger
- mettre à jour (LMD ; Langage de Manipulation des Données)
- définir les données (LDD ; Langage de Définition des Données gérées)
- contrôler l'accès aux données (LCD ; Langage de Contrôle de l'accès aux Données)

Définition (Norme SQL92 (ou SQL2))

Norme adoptée en 1992 par l 'ISO (International Organisation for Standardization) Presque complètement implémentée par les principaux SGBD : Oracle, DB2, Informix, MySQL, PostgreSQL, Access, SQL Server,. . .

Depuis, de nouvelle normes (SQL99 (ou SQL3), SQL:2003 et SQL:2008) ont été proposées. Elles ajoutent certaines fonctionnalités mais elles ne sont pas toutes implémentées dans tous les SGBD.

Chapitre III

SQL

- SGBDR (Relationnel) qui utilise le langage SQL
- Langage procédural suivant la norme SQL:2003

Nombreux programmes utilitaires

- mysql : SQL interactif en ligne de commandes
- mysql workbench : saisir/voir des données, créer/administrer des serveurs, ...
- phpmyadmin pour l'interface avec le Web
- ...

Identificateurs pour les objets manipulés

- 30 caractères au plus n lettres, chiffres, _, $ ou # (commence par une lettre)
- pas un mot clef

Quelques mots clefs : ASSERT, ASSIGN, AUDIT, COMMENT, DATE, DECIMAL, DEFINITION, FILE, FORMAT, INDEX, LIST, MODE, OPTION, PARTITION, PRIVILEGES, PUBLIC, SELECT, SESSION, SET,TABLE.

Relations stockées sous forme de tables composées de lignes et de colonnes
SQL92 : le nom d'une table est précédé du nom d'un schéma (pour réunir tous les objets liés à un même thème) Par défaut, le schéma est le nom de l'utilisateur connecté

Exemple (Exemple de table : DEPT)

Dept	NomD	Lieu
10	Finances	Paris
20	Recherche	Grenoble
30	Ventes	Lyon

Exemple (Exemple de table : EMP)

Matr	Nom	Sal	Com	Sup	Dept
1200	Dupond	2500	300	2200	10
2200	Durand	3000	500		10
1780	Boisier	2500		2200	20

- Toutes les données d'une colonne sont d'un même type
- Identificateur unique pour les colonnes d'une table, mais 2 colonnes dans 2 tables différentes peuvent avoir le même nom
- Le nom complet d'une colonne comprend le nom de la table à laquelle elle appartient (obligatoire en cas d'ambiguïté) : DEPT.dept, BERNARD.DEPT.dept

Chapitre III

SQL

- Types numériques
- Types pour les chaînes de caractères
- Types temporels (dates, heures, minutes,. . .)
- Types fichiers
- SQL ne permet pas à l'utilisateur de créer ses propres types

- Nombres entiers :
  - SMALLINT sur 2 octets
  - INTEGER sur 4 octets
- À virgule flottante :
  - REAL
  - DOUBLE PRECISION (ou FLOAT)
- Constantes : 253.8, -10, 1.3E-5

- Nombres décimaux à nombre fixe de décimales :
  - DECIMAL(nbChiffres, nbDécimales)
  - NUMERIC(nbChiffres, nbDécimales)
  - Les deux sont identiques en MySQL
- NUMERIC(8, 2) ou DECIMAL(8, 2) : 6 chiffres avant la virgule et 2 après
- Constantes : 253.8, -10

- CHAR(longueur) chaînes de caractères avec un nombre fixe de caractères
- VARCHAR(longueurMaximum) chaînes de caractères avec un nombre variable de caractères (mais un nombre maximum de caractères)
- CHAR(5) : chaîne de 5 caractères
- VARCHAR(20) : chaîne de 20 caractères au plus
- Constante : 'Comptabilité', 'Aujourd"hui'

- DATE pour les dates
- TIME pour les heures, minutes et secondes
- TIMESTAMP pour un moment précis : date et heures, minutes et secondes, avec une précision jusqu'à la microseconde (un millionième de seconde)

- BIT permet d'enregistrer un bit
- Exemples : BIT(1), BIT(4)

- Une valeur de type BLOB est un objet binaire de grande taille
- TINYBLOB, BLOB, MEDIUMBLOB, et LONGBLOB ne différent que par la taille maximale de données qu'ils peuvent stocker

- TINYTEXT, TEXT, MEDIUMTEXT, et LONGTEXT correspondent aux types BLOB
- différences aux niveau des tris et comparaisons : une valeur TEXT est une valeur BLOB insensible à la casse.

Chapitre III

SQL

Interrogations simples

Définition (Select simple)

SELECT expression1, expression2, ...
FROM table
WHERE prédicat

Exemple

select nomE, poste from emp;
select * from dept;
select nomE, sal + comm from emp;
select matr, nomE, sal * 1.15 from emp where sal + comm >= 12500;

Expressions

Ces expressions se trouvent à la suite du select ou du where ; elles peuvent comporter des

- noms de colonnes
- constantes
- opérateurs arithmétiques
- concaténations de chaînes de caractères (||)
- calculs sur les dates (+ et -)
- fonctions

Valeur NULL

- Valeur attribuée aux attributs qui n'ont pas reçu de valeur
- Les expressions qui contiennent la valeur NULL ont la valeur NULL :
  comm + sal est NULL si comm est NULL
- Sauf la fonction COALESCE

COALESCE

- COALESCE(expr1, expr2,. . .)
  retourne la valeur de la 1ère expression non NULL parmi les expressions expr1, expr2,. . .
- COALESCE(comm, 0)
- COALESCE(comm, salaire, 0)
- COALESCE(poste, 'simple soldat')

NVL

- NVL(expression, valeur)
  renvoie la valeur de l'expression si elle n'est pas NULL, et valeur sinon
- Exemple : NVL(comm, 0)

NULLIF

- Permet de récupérer des données d'une BD qui n'utilisait pas la valeur NULL
- Exemple : select nomE, NULLIF(comm, -1) from emp

Signification de NULL

- NULL signifie qu'une donnée est manquante et qu'on ne connaît donc pas sa valeur Cependant, dans la pratique il peut signifier (entre autres) :
  - **pas applicable** (seuls les commerciaux touchent une commission) ; à éviter, mais possible pour simplifier le schéma de la base
  - **valeur 0** : on peut considérer que la valeur NULL correspond à la valeur 0 ; à éviter !

Logique à 3 valeurs

- La valeur NULL implique une logique à 3 valeurs :
  - vrai
  - faux
  - on ne sait pas si c'est vrai ou faux
- Par exemple, la condition « salaire $>$ 1000 » pour un employé peut être
  - vrai, si le salaire est 1200
  - faux, si le salaire est 700
  - on ne sait pas, si le salaire est NULL

Implication de cette logique

- Soit une liste L qui contient NULL : (1, 8, NULL, 78, 7)
- Est-ce que 1 appartient à L ?
- Évidemment oui
- Est-ce que 10 appartient à L ?
- On ne sait pas !
- Est-ce que 10 n'appartient pas à L ?
- On ne sait pas !

Chapitre III

SQL

Création d'une table

Définition

Création d'une table CREATE TABLE table (
colonne1 type1,
colonne2 type2,
...
...)

Exemple

```
create table article (
ref char(5) not null,
nom varchar(20),
prix numeric(9,2),
dateMAJ date);
```

Option **not null** si la colonne doit obligatoirement être renseignée.

Valeur par défaut

On peut donner une valeur par défaut pour une colonne :

Exemple

```
create table dept (
numDept integer not null,
nomDept varchar(20),
ville varchar(30) default 'Nice');
```

On peut aussi donner une fonction comme valeur par défaut ; par exemple, **default sysdate**

DESCRIBE

Cette commande est à la base développée pour Oracle mais est maintenant intégrée dans MySQL. Elle affiche une description des colonnes d'une table :

Exemple

SQL > describe article;

Name	**Null ?**	**Type**
REF	Not null	CHAR(5)
NOM		VARCHAR(20)
PRIX		NUMBER(9,2)
DATEMAJ		DATE

Chapitre III

SQL

Contraintes d'intégrité

Définition

Contraintes d'intégrité

- Une contrainte d'intégrité est une contrainte que doivent vérifier les données d'une table
- Une commande est annulée par le SGBD si son exécution viole une des contraintes

Types de contraintes d'intégrité

- PRIMARY KEY : clé primaire
- FOREIGN KEY ... REFERENCES : clé étrangère
- UNIQUE : 2 lignes ne peuvent avoir la même valeur pour les colonnes spécifiées
- CHECK : contrainte de domaine, ou autre ; porte sur une seule ligne

Types de contraintes d'intégrité

2 types de contraintes :

- contrainte de colonne (concerne une seule colonne)
- contrainte de table

Définition des contraintes

- Les contraintes sont définies dans les commandes CREATE (ou ALTER) TABLE
  - à l'intérieur des définitions de colonnes pour les contraintes de colonne
  - au même niveau que les définitions de colonnes pour les contraintes de table
- Pour les contraintes de table : **CONSTRAINT** *nomContrainte définitionContrainte*

Clé primaire

- Si la clé primaire n'est formée que d'une seule colonne, le plus simple est d'ajouter une contrainte de colonne :

Exemple

```
create table emp (
matr integer primary key,
...
```

- Sinon, il faut ajouter une contrainte de table :

Exemple

```
create table participation (
matr integer,
codeP integer,
...,
constraint pkpar primary key(matr, codeP))
```

Une erreur à ne pas faire

- Si une table a une clé primaire formée de 2 colonnes, il ne faut pas déclarer 2 contraintes de colonne
- Il faut déclarer une seule contrainte de table portant sur les 2 colonnes :
  constraint pkpar primary key(matr, codeP)
- Aucune des colonnes de la clé primaire ne peut avoir la valeur **null**

Contrainte UNIQUE

- 2 lignes de la table ne pourront avoir la même valeur (sauf NULL)
- Correspond à un identificateur (clé candidate si minimal), si on ajoute une contrainte NOT NULL

Clé étrangère

Si une seule colonne forme la clé étrangère, le plus simple est d'utiliser une contrainte de colonne :

Exemple

```
create table emp (
...,
dept integer references dept(dept))
```

(dept) : Optionnel si la colonne référencée est clé primaire

Clé étrangère

Peut être une contrainte de table :

Définition

FOREIGN KEY (colonne1, colonne2,. . .)
REFERENCES table-ref [(col1, col2,. . .)]

Exemple

```
create table emp (
...,
dept integer,
constraint r_dept foreign key(dept) references dept(dept))
```

Clé étrangère

Les colonnes de l'autre table référencées (col1, col2,. . .) doivent avoir la contrainte PRIMARY KEY ou UNIQUE

constraint r_dept references dept(dept)

dept doit être clé primaire, ou unique

Option ON DELETE CASCADE (sans)

Exemple

```
create table emp (
...
dept integer references dept)
```

ou

Exemple

```
create table emp (
...
dept integer
constraint r_dept foreign key (dept) references dept)
```

Attention On ne peut supprimer un département s'il est référencé par une ligne de la table **emp**

Option ON DELETE CASCADE (avec)

Exemple

```
create table emp (
...
dept decimal(2,0),
constraint r_dept foreign key (dept) references dept on delete
cascade)
```

Attention La suppression d'un département entraîne automatiquement la suppression de toutes les lignes de la table **emp** qui référencent ce département.

Autres options pour les clés étrangères

- on delete set null
- on delete set default
- on update cascade
- on update set null
- on update set default

Exemples divers de contraintes

Exemple

create table emp (
matr integer **primary key**,
nomE varchar(30) **unique**,
dept smallint **references** dept **check** (dept in (10, 20, 30, 40)),
constraint nom_unique **check** (nomE = upper(nomE)));

Exemple

create table participation (
matr integer **references** emp,
codeP varchar(5) **references** projet,
. . . ,
constraint pkpart **primary key**(matr, codeP));

Modification des contraintes

Exemple

```
ALTER TABLE emp
DROP CONSTRAINT nom_unique
ADD (CONSTRAINT sal_min check(sal + coalesce(comm, 0) >
5000))
RENAME CONSTRAINT truc TO machin;
```

Attention On ne peut ajouter que des contraintes de table.

Vérification des contraintes

- En fonctionnement normal les contraintes sont vérifiées à chaque requête SQL.
- Cette vérification peut être gênante, en particulier lors de l'ajout de plusieurs lignes de données.
- Exemple : si on a cette contrainte sur la colonne SUP de la table EMP : constraint sup_ref_emp foreign key (SUP) **references EMP**
- La contrainte oblige à ajouter les supérieurs en premier.

Chapitre III

SQL

Dictionnaire des données

Définition

- Tables qui stockent les descriptions des objets de la base
- Tenues à jour automatiquement par le SGBD
- Peuvent être consultées au moyen du langage

INFORMATION_SCHEMA

- Ce dictionnaire est stocké dans la base **INFORMATION_SCHEMA**

Exemple

```
SELECT table_name, table_type, engine FROM
information_schema.tables;
```

- Il est aussi possible d'utiliser la commande **SHOW** (ex : show tables)

Tables de INFORMATION_SCHEMA :

- SCHEMATA : informations sur les bases de données
- TABLES : informations sur les tables
- COLUMNS : informations sur les colonnes dans les tables
- USER_PRIVILEGES : informations sur les droits globaux
- SCHEMA_PRIVILEGES : informations sur les droits des schémas
- TABLE_PRIVILEGES : informations sur les droits des tables
- COLUMN_PRIVILEGES : informations sur les droits reliés aux colonnes
- TABLE_CONSTRAINTS : informations sur les tables qui ont des contraintes
- KEY_COLUMN_USAGE : descriptions des contraintes sur les colonnes
- ROUTINES : informations sur les procédures stockées
- VIEWS : informations sur les vues

Chapitre III

SQL

Chapitre III

SQL

Langage de manipulation des données (LMD)

Commandes de manipulation des données :

- INSERT pour ajouter des lignes
- UPDATE pour modifier des lignes
- DELETE pour supprimer des lignes

Insertion

Définition

INSERT INTO table [(colonne1, colonne2,...)] VALUES (valeur1, valeur2,...)

ou

Définition

INSERT INTO table [(colonne1, colonne2,...)] select ...

La liste des colonnes est optionnelle ; par défaut, toutes les colonnes sont dans l'ordre donné lors de la création de la table.
Si la commande comporte une liste, les colonnes qui ne sont pas dans la liste auront la valeur NULL
Dans les programmes, il faut toujours donner la liste des colonnes dont on donne les valeurs pour faciliter la maintenance de l'application

Insertion

Exemple

```
insert into dept values (10, 'Finance', 'Paris');
insert into dept (lieu, nomD, dept) values ('Grenoble', 'Recherche',
20);
insert into emp (matr, nomE, dept, sal)
select matr + 100, nomE, 60, sal * 0.15
from emp
where dept = 10;
```

Modification

Définition

UPDATE table SET colonne1 = expr1, colonne2 = expr2, ... [WHERE prédicat]

Définition

UPDATE table [synonyme] SET (colonne1, colonne2, ...) = (select ...) [WHERE prédicat]

Le **select** ne doit renvoyer qu'une seule ligne

Exemples

Exemple

update emp set dept = 10 where nomE = 'Martin';
update emp set sal = sal * 1.1 where poste = 'Commercial';

Exemple

update emp set sal = (select avg(sal) * 1.1 from emp where poste = 'Secrétaire') where nomE = 'Clément';
update emp **E** set (sal, comm) = (select avg(sal), avg(comm) from emp where dept = **E**.dept) where poste = 'Secrétaire';

Suppressions

Définition

DELETE FROM table [WHERE prédicat]

Attention S'il n'y a pas de clause WHERE, toutes les lignes sont supprimées.

Exemple

delete from emp where dept = 10;

Chapitre III

SQL

Transaction

- Une requête forme un tout indivisible
- Si une erreur survient pendant l'exécution d'une requête SQL, toutes les modifications déjà effectuées par la requête sont annulées automatiquement.
- On peut généraliser ce comportement à un ensemble de requêtes SQL en utilisant les transactions
- Ensemble de modifications de la base qui forment un tout indivisible : à la fin de la transaction, toutes les modifications effectuées pendant la transaction sont sauvegardées ou annulées

Début d'une transaction

- Dans la norme SQL2 toute modification appartient à une transaction
- Par défaut, MySQL est lancé en mode autocommit (transaction effectuée après chaque opération)
- Une transaction démarre automatiquement après la fin d'une transaction (pas de commande pour démarrer une transaction)
- Une transaction démarre par START TRANSACTION
- La structure des transactions est « plate » : les transactions ne peuvent se chevaucher

Terminer une transaction

- Pour terminer une transaction on peut
  - valider la transaction (COMMIT) : toutes les modifications deviennent effectives
  - annuler la transaction (ROLLBACK) : toutes les modifications sont annulées
- Les ordres DDL (create table par exemple) provoquent un COMMIT automatique

Propriétés des transactions - ACID

- **A**tomicité : un tout indivisible
- **C**ohérence : une transaction doit laisser la base dans un état cohérent ; elle ne doit pas mettre les données dans un état « anormal »
- **I**solation : une transaction est isolée des autres transactions (dans une certaine mesure. . .)
- **D**urabilité : le SGBD doit garantir que les modifications d'une transaction validée seront conservées, même en cas de panne

Propriétés des transactions - ACID

- **AID** est du ressort du système transactionnel du SGBD
- **C** est du ressort de l'utilisateur mais il est aidé
  - par **I**, car il n'a pas à considérer les interactions avec les autres transactions
  - par la vérification automatique des contraintes d'intégrité par le SGBD
- **I** est assuré par le système de contrôle de la concurrence du SGBD et **AD** sont supportés par les procédures de reprise après panne du SGBD

Transaction

Exemple d'annulation d'une transaction

Exemple

```
start transaction;
insert into dept values (10, 'Finance', 'Paris');
delete from emp;
rollback;
```

Isolation des transactions

En fonctionnement standard les modifications effectuées par une transaction T ne sont connues par les autres transactions qu'après validation de T En fait, il existe plusieurs niveaux d'isolation (voir cours sur la concurrence)

Transactions longues

- Exemple : organisation par une agence de voyage d'un voyage Nice – Wuhan (Chine)
- Nécessite la réservation de plusieurs billets d'avion : Nice – Paris ; Paris – Beijing ; Beijing – Wuhan
- On commence par réserver les 2 premiers mais si on ne peut trouver de Beijing – Wuhan, il faut tout annuler
- On met donc toutes ces réservations dans une transaction ; ça peut être long si l'agence discute avec le client pendant la transaction

Problèmes avec les transactions longues

- Manque de souplesse : si on ne trouve pas de voyage Beijing – Wuhan, on annule tout
- On aurait pu garder le Nice – Paris et essayer de passer par Shanghai pour aller à Wuhan, en annulant seulement le Paris – Beijing
- Autre problème : le contrôle de la concurrence effectue des blocages sur les tables et les lignes qui ne sont relâchés qu'à la fin de la transaction
- Un problème de communication peut provoquer l'annulation des premières réservations alors qu'on pourrait simplement réessayer le lendemain

Transactions emboîtées

- Extension de la notion de transaction « plate »
- Évite les annulations complètes de transactions
- Apporte plus de souplesse dans les transactions longues et multi-sites
- Permet de limiter la durée des blocages des ressources système

Définition des transactions emboîtées

- Une transaction globale (mère) peut contenir des soustransactions filles qui, elles-mêmes, peuvent avoir des filles
- L'annulation d'une transaction n'annule pas nécessairement la transaction mère ; celle-ci peut
  - décider d'un traitement substitutif
  - reprendre la transaction annulée
  - s'annuler
  - ou même ignorer l'annulation (traitement pas indispensable)
- L'annulation d'une transaction provoque l'annulation automatique de toutes ses transactions filles

Points de reprise

- Sans passer au modèle des transactions emboîtées, on peut assouplir le modèle des transactions plates
- Désigner des points de reprise dans une transaction : **savepoint** *nomPoint*
- Possible d'annuler toutes les modifications effectuées depuis un point de reprise : **rollback to** *nomPoint*
- Évite d'annuler toute la transaction et permet d'essayer de pallier le problème si c'est possible

Points de reprise

Exemple

```
start transaction;
insert into ...;
savepoint p1;
delete from ...;
update ...;
savepoint p2;
insert into ...; – Problème !
rollback to p2;
insert into ...; – on essaie autre chose
commit;
```

Chapitre III

SQL

Syntaxe générale

Définition

SELECT ... FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...

- L'ordre des clauses est imposé
- SELECT et FROM sont obligatoires

Clause SELECT

Définition

select [distinct] expression1 [[AS] nom1], expression2 [[AS] nom2], . . .

nom1 :

- en-tête de la colonne (entre guillemets si mot-clef ou si contient plusieurs mots)
- alias pour désigner la colonne dans une autre partie du select

select * : Toutes les colonnes

Exemple

select distinct poste from emp;
select nomE, sal + coalesce(comm, 0) as "Salaire Total" from emp;

select dans une expression

Exemple

```
select nomE, sal / (select sum(sal) from emp) * 100 from emp
```

Le select de l'expression peut être synchronisé avec le select principal :

Exemple

```
select nomE, (select count(*) from emp where sal > e1.sal) + 1 as
rang from emp e1
```

e1 est un synonyme de emp pour lever l'ambiguïté dans le select interne

select dans une expression

On peut utiliser l'alias pour trier par ordre décroissant des salaires :

Exemple

select nomE, (select count(*) from emp where sal > e1.sal) + 1 as rang from emp e1 order by rang

Clause FROM

Définition

FROM table1 [synonyme1], table2 [synonyme2], ...

- Produit cartésien des tables s'il y en a plusieurs
- Possible de se restreindre à un sous-ensemble du produit cartésien (voir jointure)

Clause FROM

Exemple

```
select B.dept, A.nomD
from dept A, dept B
```

DEPT	NOMD
10	VENTES
20	RECHERCHE
30	FINANCE

DEPT	NOMD
30	VENTES
20	VENTES
10	VENTES
30	RECHERCHE
20	RECHERCHE
10	RECHERCHE
30	FINANCE
20	FINANCE
10	FINANCE

Clause FROM

Certains SGBDs (et la norme SQL-2) permettent l'utilisation d'un SELECT à la place du nom d'une table :

Exemple

```
select nomE, sal,
sal / total * 100 Pourcentage
from emp,
(select sum(sal) as total from emp);
```

Clause WHERE

La clause WHERE comporte de nombreuses possibilités :

- opérateurs de comparaison
- opérateurs logiques
- jointures
- sous-interrogations

Opérateurs de comparaison

- =, !=, <, >, <=, >=, BETWEEN, LIKE, NOT LIKE, IN, NOT IN, IS NULL, IS NOT NULL
- LIKE permet d'utiliser des jokers :
  - % pour une chaîne de caractères de longueur quelconque
  - _ pour un seul caractère
- Attention, expression = NULL n'est jamais vrai, il faut utiliser expression IS NULL

Opérateurs de comparaison

Exemple

```
select * from emp where poste = 'Secrétaire';
select * from emp where sal between 10000 and 15000;
select * from emp where dept in (10, 30);
select * from emp where comm is not null;
select * from emp where nomE like '%A%';
```

Opérateurs logiques

- AND, OR, NOT

Exemple

```
select nomE from emp
where dept = 30
and (sal > 10000 or comm is null);
```

Exemple

```
select * from emp
where not (poste = 'Directeur'
or poste = 'Secrétaire');
```

Jointures

Traduction de l'équi-jointure « emp ⋈ dept » :

Exemple

```
select nomE, nomD
from emp, dept
where emp.dept = dept.dept
```

Autre syntaxe :

Exemple

```
select nomE, nomD
from emp JOIN dept
ON emp.dept = dept.dept
```

Jointure de plus de 2 tables

Exemple

```
select nomE, nomP
from emp, participation, projet
where emp.matr = participation.matr
and participation.codeP = projet.codeP
```

Autre syntaxe :

Exemple

```
select nome, nomp
from emp
join participation
on emp.matr = participation.matr
join projet
on participation.codep = projet.codep
```

Jointure naturelle

- La jointure s'effectue sur toutes les colonnes qui ont le même nom dans les 2 tables ; ces colonnes ne sont pas répétées dans la jointure
- Les colonnes qui participent à la jointure ne doivent être préfixées par un nom de table

Exemple

```
select nomE, nomD, dept from emp NATURAL JOIN dept;
select nome, nomp from emp NATURAL JOIN participation
NATURAL JOIN projet;
```

Jointure d'une table avec elle-même

Alias indispensable pour le nom de la table afin de lever l'ambiguïté sur les colonnes :

Exemple

```
select emp.nomE "Employé",
supe.nomE "Supérieur"
from emp join emp supe
on emp.sup = supe.matr
```

Jointures « non équi »

Les jointures « non équi » peuvent être traduites comme les équi-jointures, en utilisant d'autres opérateurs de comparaison

Exemple

```
select emp1.nomE, emp2.nomE
from emp emp1
join emp emp2
on emp1.sal < emp2.sal
```

Jointure externe

- Dans une jointure n'apparaissent que les lignes qui ont une ligne correspondante dans l'autre table
- Dans l'exemple suivant, un département qui n'a pas d'employé n'apparaîtra pas :

Exemple

```
select nomE, nomD from emp join dept on emp.dept = dept.dept
```

- Si on veut qu'il apparaisse, on doit utiliser une jointure externe

Syntaxe de la jointure externe

Exemple

```
select nomE, nomD
from emp RIGHT OUTER JOIN dept
ON emp.dept = dept.dept
```

- RIGHT indique que l'on veut afficher toutes les lignes de la table de droite (dept)
- Ça revient à ajouter une « ligne fictive » dans l'autre table emp
- Cette ligne fictive aura toutes ses colonnes à null, sauf la colonne de jointure
- Il existe de même LEFT OUTER JOIN et FULL OUTER JOIN

Sous-interrogations

- Une clause WHERE peut comporter un ordre SELECT emboîté :

Exemple

```
select nomE from emp
where poste = (select poste from emp where nomE = 'Martin');
```

- Cette sous-interrogation doit ramener une seule ligne et une seule colonne
- Remplace le select interne par NULL s'il ne renvoie aucune ligne (ou erreur, suivant les SGBD)

Sous-interrogations

Des variantes de sous-interrogations ramènent plusieurs colonnes ou plusieurs lignes

Sous-interrogation ramenant 1 ligne, 1 colonne

Définition

WHERE expression op (SELECT ...)

où op est un des opérateurs de comparaison =,!=, <, >, <=, >=

Exemple

```
select nomE from emp
where sal >=
(select sal from emp
where nomE = 'Mercier')
```

Sous-interrogation ramenant plusieurs lignes

Définition

WHERE expression op ANY (SELECT ...)
WHERE expression op ALL (SELECT ...)
WHERE expression IN (SELECT ...)
WHERE expression NOT IN (SELECT ...)

où op est un des opérateurs de comparaison $=$,$!=$, $<$, $>$, $<=$, $>=$

- **ANY** : vrai si la comparaison est vraie pour **au moins** une des valeurs ramenées par le SELECT
- **ALL** : vrai si la comparaison est vraie pour **toutes** les valeurs ramenées par le SELECT

Remarque

- =ANY est équivalent à IN
- !=ALL est équivalent à NOT IN

Exemple

```
select nomE, sal from emp
where sal > all (select sal from emp where dept = 30);
select nomE, sal from emp
where sal > all (select sal
from emp
where dept = 38888)
```

Réflexion sur ALL

- Quand « x $>$ all (x1, x2, ..., xn) » est faux ?
- Quand $\exists$ un xi tel que xi $>=$ x
- Si $\nexists$ un xi tel que xi $>=$ x, l'expression est vraie
- Donc si la liste (x1, x2, ..., xn) est vide, l'expression est toujours vraie

Retour sur NULL

- La condition **expression not in (expr1, expr2, null)** n'est jamais vérifiée. Pourquoi ?
- Est-ce que la condition **expression in (expr1, expr2, null)** peut être vérifiée ?
- Rappel : la logique de SQL n'utilise pas seulement vrai et faux mais aussi « je ne sais pas », représenté par NULL
- Utile à savoir pour les sous interrogations qui renvoient NULL pour une des lignes

Sous-interrogations ; optimisation

- Soit un select qui comporte une sous-interrogation :

Exemple

```
select nom from emp
where dept in
(select dept from dept
where lieu = 'NICE')
```

- Pour chaque employé, le select peut lancer la sousinterrogation pour savoir si l'employé est dans un département qui se trouve à Nice
- En fait, le moteur de recherche du SGBD va optimiser en lançant d'abord la sous-interrogation et en conservant en mémoire les départements de Nice

Sous-interrogations synchronisées

- Cette optimisation n'est pas possible quand la sous-interrogation utilise une des valeurs ramenées par l'interrogation principale
- On dit que la sous-interrogation est synchronisée avec l'interrogation principale
- Notation pointée utilisée pour se référer dans la sous-interrogation à une colonne de l'interrogation principale : sous-interrogation

Exemple

```
select nomE from emp E
where dept != (select dept from emp
where matr = E.sup);
```

Sous-interrogation ramenant 1 ligne de plusieurs colonnes

Définition

WHERE (expr1, expr2,...) op (SELECT ...)

où op est = ou != (mais pas <, >, <=, >=) et où le select renvoie 1 seule ligne

Exemple

```
select nomE from emp
where (poste, sal) =
(select poste, sal from emp
where nomE = 'Mercier');
```

Relation d'ordre

- Il n'y a pas de relation d'ordre totale « naturelle » sur les tuples
- $5 > 3$, $12 > 7$
- On ne peut comparer (5, 7) et (3, 12)

Sous-interrogation ramenant plusieurs lignes de plusieurs colonnes

Définition

WHERE (expr1, expr2,...) op ANY (SELECT ...)
WHERE (expr1, expr2,...) op ALL (SELECT ...)
WHERE (expr1, expr2,...) IN (SELECT ...)
WHERE (expr1, expr2,...) NOT IN (SELECT ...)

où op est = ou != (mais pas <, >, <=, >=)

Exemple

```
select nomE from emp
where (poste, sal) in
(select poste, sal from emp
where dept = 10);
```

EXISTS

- La clause « EXISTS(select ...) » est vraie si le select renvoie au moins une ligne
- La sous-interrogation est le plus souvent synchronisée :

Exemple

```
select nomE from emp E
where exists
(select null from emp
where sup = E.matr);
```

EXERCICE : Division avec not exist

Chapitre III

SQL

Fonctions de groupe

Les fonctions de groupe peuvent apparaître dans une expression du select ou du having :

- AVG moyenne
- SUM somme
- MIN plus petite valeur
- MAX plus grande valeur
- VARIANCE variance
- STDDEV écart type
- COUNT(*) nombre de lignes
- COUNT(col) nombre de valeurs non NULL dans la colonne
- COUNT(DISTINCT col) nombre de valeurs distinctes

Fonctions de groupe

Exemple

```
select count(*) from emp;
select count(comm) from emp where dept = 10;
select sum(sal) from emp where dept = 10;
select max(sal) from emp where poste = 'INGENIEUR';
```

Niveaux de regroupement

- A un niveau de profondeur (relativement aux sous interrogations) d'un SELECT, les fonctions de groupe et les colonnes doivent être toutes du même niveau de regroupement
- Par exemple, si on veut le nom et le salaire des employés qui gagnent le plus dans l'entreprise, la requête suivante provoquera une erreur :

Exemple

```
select nome, sal from emp
where sal = max(sal)
```

- Solution :

Exemple

```
select nome, sal from emp
where sal = (select max(sal) from emp)
```

Clause GROUP BY

Permet de regrouper des lignes qui ont les mêmes valeurs pour des expressions :

Définition

GROUP BY expression1, expression2,...

Il n'est affiché qu'une seule ligne par regroupement de lignes

Exemple

```
select dept, count(*) from emp group by dept;
```

Clause GROUP BY

Exemple

```
select dept, poste, count(*) from emp group by dept, poste;
select dept, count(comm) from emp group by dept;
```

Schéma à retenir pour obtenir des optima sur des regroupements :

Exemple

```
select nome, dept, sal from emp
where (dept, sal) in (select dept, max(sal) from emp group by
dept);
```

Sans doute moins performant :

Exemple

```
select nome, dept, sal from emp E
where sal = (select max(sal) from emp
where dept = E.dept);
```

Clause GROUP BY

Exemple

```
select nomD, avg(sal) from emp natural join dept group by nomD;
```

Exemple

```
select dept, count(*) from emp where poste = 'SECRETAIRE'
group by dept;
```

Exemple

```
select count(*) "Nbre employés", count(comm) "Nbre
commissions", count(comm) / count(*) "Ratio" from emp;
```

Restrictions pour les expressions

Dans la liste des expressions du select ne peuvent figurer que des caractéristiques liées aux groupes :

- des fonctions de groupes
- des expressions figurant dans le GROUP BY

Exemple (Ceci est interdit)

```
select dept, nomE, sal
from emp
group by dept;
```

Restrictions pour les expressions

- Ceci ne fonctionne pas :

Exemple

```
select nomd, sum(sal)
from emp natural join dept
group by dept.dept
```

- Il aurait fallu écrire :

Exemple

```
select nomd, sum(sal)
from emp natural join dept
group by nomd
```

Clause HAVING

- Cette clause sert à sélectionner des groupes : HAVING prédicat
- Le prédicat ne peut porter que sur des caractéristiques de groupe

Clause HAVING

Exemple

```
select dept, count(*) from emp
where poste = 'Secrétaire'
group by dept
having count(*) > 1;
```

Exemple

```
select nomd "Département",
count(*) "Nombre de secrétaires"
from emp natural join dept
where poste = 'Secrétaire'
group by nomd
having count(*) = (select max(count(*)) from emp
where poste = 'Secrétaire'
group by dept);
```

Clause HAVING

Exemple

```
select dept, count(*) from emp group by dept having count(*) =
max(count(*));
```

Interdit ! car pas le même niveau de regroupement que le reste du select

Exemple

```
select dept, count(*) from emp
group by dept
having count(*) = (select max(count(*)) from emp
group by dept);
```

Clause ORDER BY

La clause ORDER BY précise l'ordre des lignes d'un SELECT :

Définition

ORDER BY expr1 [DESC], expr2 [DESC], ...

On peut aussi donner le numéro de la colonne qui servira de clé de tri

Exemple

```
select dept, nomD from dept order by nomD;
select dept, nomD from dept order by 2;
```

Clause ORDER BY

Exemple

select nome, poste from emp **order by dept, sal desc**;

select dept, sum(sal) "Total salaires" from emp group by dept **order by 2**;

select dept, sum(sal) "Total salaires" from emp group by dept **order by sum(sal)**;

Fonctions

- Fonctions arithmétiques : abs(n), mod(m, n), power(n, e), round(n, p), trunc(n, p), sign(n), sqrt(n), greatest(n1, n2,...), least(n1, n2,...)
- Conversions nombre - chaîne de caractères : to_char(n, format), number(chaîne)
- Fonctions date : round(date, précision), trunc(date, précision), sysdate
- Conversions date - chaîne de caractères : to_char(date, format), to_date(chaîne, format)

Fonctions

- Fonctions sur les chaînes de caractères : length(chaîne), substr(chaîne, position, longueur), locate(sous-chaîne, chaîne), upper(chaîne), lower(chaîne), lpad(chaîne, long, car), rpad(chaîne, long, car), ltrim(chaîne, car), rtrim(chaîne, car), translate(chaîne, lesCar1, lesCar2), replace(chaîne, ancienne, nouvelle)

Fonctions

Exemple

```
select nomE, to_char(datemb, 'DD/MM/YYYY')
from emp
where round(sysdate – datemb) > 3;
```

Conditionnelle

2 variantes :

Définition

```
CASE
WHEN condition1 THEN resultat1
WHEN condition2 THEN resultat2
ELSE resultat3
END
```

Définition

```
CASE expression
WHEN valeur1 THEN resultat1
WHEN valeur2 THEN resultat2
ELSE resultat3
END
```

Conditionnelle

Exemple

```
SELECT nome, poste FROM emp
order by
CASE poste
WHEN 'Président' THEN 1
WHEN 'Directeur' THEN 2
ELSE 3 END;
```

Exemple

```
SELECT nome, poste FROM emp
order by
CASE
WHEN poste = 'Président' THEN 1
WHEN poste = 'Directeur' THEN 2
ELSE 3
END;
```

Opérateurs ensemblistes

- On peut effectuer des opérations sur plusieurs select considérés comme des ensembles de lignes :

Définition

select ... **UNION** select ...
select ... **INTERSECT** select ...
select ... **MINUS** select ...

- Ensembles au sens mathématique : si 2 lignes des 2 select d'une union ont les mêmes valeurs, l'union n'aura qu'une seule ligne avec ces valeurs

Exemple d'opérateur ensembliste

Exemple (PostgreSQL)

```
select dept from dept
minus
select dept from emp;
```

Exemple (Mysql)

```
select dept from dept
not in (select dept from emp);
```

Exemple

```
select nomE, 'salaire' as TYPE, sal as MONTANT from emp union
select nomE, 'commission', comm from emp where comm is not
null;
```

nomE	**TYPE**	**MONTANT**
Toto	salaire	1200
Bibi	salaire	5000
Toto	commission	240

Opérateurs ensembliste

- UNION ALL : Pour conserver les doublons avec UNION
- Opérateurs ensembliste et order by : Seul le dernier select peut recevoir une clause order by

Limiter le nombre de lignes

Problème de portabilité si on ne veut afficher qu'une partie des lignes récupérées par un select

Exemple

```
SELECT matr, nomE
FROM emp
LIMIT 10
```

Chapitre IV

Langage de définition des données avancé

Création de table par copie

Définition

CREATE TABLE table (col1 type, ...) AS select ...

Exemple

```
create table dept2
(cle integer, nom varchar(20))
as select dept, nomd from dept;
create table dept10
as select * from emp
where dept = 10;
```

Modifier la définition d'une table

Définition

ALTER TABLE table ADD (col1 type1, col2 type2,...)
ALTER TABLE table MODIFY (col1 type1, col2 type2,...)
ALTER TABLE table DROP COLUMN colonne;

Modifier la définition d'une table

On ne peut modifier une colonne que si la colonne ne contient que des valeurs null ou si la nouvelle définition est compatible avec les valeurs déjà entrées dans cette colonne.

Exemple

```
alter table emp add (situ_famille char(1), nbEnfants smallint);
alter table emp modify (situ_famille char(2));
```

Supprimer une colonne

- Tous les SGBDs ne permettent pas de supprimer une colonne d'une table (Mysql l'accepte)
- Si c'est impossible, on peut mettre toutes les valeurs de la colonne à null pour gagner de la place
- On peut aussi
  - utiliser create table as pour transférer les autres données dans une nouvelle table qui n'a pas cette colonne
  - supprimer la 1ère table
  - renommer la nouvelle table

Supprimer une colonne

Définition

- ALTER TABLE emp DROP COLUMN comm
- Il n'est pas possible de supprimer une colonne
  - référencée par une clé étrangère
  - sur laquelle un index a été construit

Exemple

```
alter table emp drop column situ_famille;
```

Modifier la définition d'une table

- Supprimer une table : DROP TABLE table
- Renommer une colonne :ALTER TABLE table RENAME COLUMN ancienNom TO nouveauNom
- Renommer une table : RENAME ancienNom TO nouveauNom
- Commande équivalente pour renommer une table : ALTER TABLE ancienNom RENAME TO nouveauNom

Chapitre V

Vues, Index, Procédures stockées, ...

Chapitre V

Vues, Index, Procédures stockées, ...

Vues

- Une vue est une vision **virtuelle** partielle ou particulière des données d'une ou plusieurs tables
- La définition d'une vue est donnée par un select : les données de la vue sont celles retournées par le select
- Les utilisateurs peuvent consulter ou modifier la base à travers la vue comme si c'était une table réelle

Création et suppression d'une vue

Définition

CREATE VIEW vue [(col1, col2,...)]
AS select ... [WITH CHECK OPTION]

Le select peut contenir toutes les clauses d'un select sauf **order by**

Exemple

```
create view emp10 as
select * from emp
where dept = 10;
DROP VIEW vue
```

Exemple de création de vues

Exemple

```
create view
deptStat (nom, inf, moy, max, total)
as
select nomd,
min(sal), avg(sal), max(sal), sum(sal)
from emp natural join dept
group by dept.nomd;
```

Vue de vue

Il est possible de créer une vue de vue :

Exemple

```
create view stat2 as
select nom, max
from deptstat;
```

Utilisation des vues dans un select

Dans un select on peut utiliser une vue à la place d'une table

Exemple

```
select * from emp10;

select nom, total
from deptStat
where total > 100000;
```

Suppression avec une vue

On peut effectuer des delete à travers une vue, sous les conditions suivantes sur le select qui définit la vue :

- une seule table
- pas de group by
- pas de distinct
- pas de fonction de groupe

Modification avec une vue

On peut effectuer des update à travers une vue, sous les conditions suivantes :

- une seule table
- pas de group by
- pas de distinct
- pas de fonction de groupe
- les colonnes modifiées sont des colonnes réelles de la table (pas des expressions)

Exemple

```
update emp10 set sal = sal * 1.1;
```

Insertion avec une vue

On peut effectuer des insert à travers une vue sous les conditions suivantes :

- une seule table
- pas de group by
- pas de distinct
- pas de fonction de groupe
- les colonnes modifiées sont des colonnes réelles de la table (pas des expressions)
- toute colonne « not null » de la table représentée par la vue est présente dans la vue

Exemple

```
insert into emp10 (matr, nome, ...) values (1200, 'DUBOIS', ...);
```

Option CHECK

Si la vue a été créée avec **WITH CHECK OPTION**, toute modification au travers de la vue doit donner des données qui peuvent être affichées par la vue.

Exemple (Ceci est interdit)

```
update emp10 set dept = 20;
```

La mise à jour est interdite car la vue ne contient que les employés du departement 10.

Exemple (Rappel)

```
create view emp10 as
select * from emp
where dept = 10;
```

Utilité des vues

- Dissocier
  - la façon dont les utilisateurs voient les données
  - du découpage en tables
- On favorise ainsi l'indépendance entre les programmes et les données
- Si un programme utilise des vues, on peut, par exemple, remplacer une table par 2 tables sans modifier le programme de consultation des données ; il suffit de modifier la définition des vues

Utilité des vues (2)

- Simplifier la consultation de la base en enregistrant des select complexes
- Protéger des données :
  - on peut donner accès à une vue, sans donner accès à la table sous-jacente
  - une vue peut ne donner accès qu'à certaines colonnes ou lignes d'une table
  - les modifications des données peuvent être restreintes avec la clause WITH CHECK OPTION

Chapitre V

Vues, Index, Procédures stockées, ...

Index

- Un index utilise des techniques informatiques pour rendre très rapides les accès aux valeurs d'une colonne

Exemple

```
select * from emp
where nomE = 'Dupond'
```

- Cet exemple est très long si la table emp contient des millions de lignes
- Un index bien construit permet d'obtenir l'emplacement des informations sur Dupond en quelques accès disques (pas plus de 4, même s'il y a des millions de lignes dans la table EMP)

Création d'un index

Définition

CREATE [UNIQUE] INDEX nomIndex
ON table (col1, col2,...)

Exemple

```
create index nomE on emp(nomE);
```

Le nom choisi doit être unique parmi tous les index de **toutes** les tables

Utilisation d'un index

- Après sa création un index est géré automatiquement par le SGBD
- Il est transparent pour l'utilisateur : celui-ci interroge la base de la même façon que si l'index n'existait pas
- Le SGBD peut utiliser un index s'il pense que la requête sera accélérée
- Les index ralentissent les modifications des données

Suppression d'un index

Définition

DROP INDEX nomIndex

Chapitre V

Vues, Index, Procédures stockées, ...

Génération de clés

Utilité :

- Les identifiants de lignes non significatifs sont préférables
- Le plus simple est d'avoir des clés qui sont des nombres entiers
- Le problème : générer des entiers sans que 2 lignes puissent avoir le même identifiant, même en situation de concurrence entre plusieurs transactions

Une mauvaise solution

Prendre le plus grand nombre déjà utilisé dans la table comme identifiant et ajouter 1

Exemple (En pseudo code)

lock table;
val = select max(cle) from table;
insert into table values (val + 1,...);
commit;

Inconvénients de la solution 1

- Cette solution n'est pas performante :
  - nécessite un accès à la base
  - il faut trouver le plus grand
  - nécessite un blocage de la table pour éviter que 2 transactions n'obtiennent la même valeur
- Si on veut garder un historique, on peut se retrouver avec des lignes qui ont le même identificateur...
- ... (si la ligne de plus grand identificateur est supprimée)

Solution 2

Une table contient la prochaine clé à attribuer. La valeur est incrémentée à chaque nouvelle clé.
Variantes

- Variante 1 : Une seule table contient une seule valeur utilisée pour les identifiants de toutes les tables
- Variante 2 : Une table par clé
- Variante 3 : Une seule table qui contient une ligne par table qui a besoin d'une clé Quelle colonnes dans la table des clés ?

Pseudo-code de la variante 1

Exemple

```
lock table_cle;
update table_cle set cle = cle + 1;
val = select cle from table_cle;
commit;
insert into table values (val,...);
commit;
```

Inconvénients de l'utilisation d'une table

Cette solution n'est pas très performante :

- nécessite un accès à la base (mais cette petite table est conservée en mémoire centrale dans le cache du SGBD)
- il est nécessaire de bloquer l'accès à la valeur

Les séquences

- Les versions actuelles des SGBD offrent des solutions qui ne nécessitent pas d'accès aux données de la base
- Inconvénient : pas les mêmes solutions dans tous les SGBDs
- Les séquences sont disponibles avec Oracle, DB2 et PostgreSQL

Créer une séquence

Définition

CREATE SEQUENCE nom_séquence [INCREMENT BY entier1]
[START WITH entier2]

Exemple

```
create sequence seqdept
increment by 10
start with 10
```

Utilisation des séquences

Deux pseudo-colonnes permettent d'utiliser les séquences :

- CURRVAL retourne la valeur courante
- NEXTVAL incrémente la séquence et retourne la nouvelle valeur

Exemple

```
insert into dept(dept, nomd)
values (seqdept.nextval, 'Finances')
```

CURRVAL et NEXTVAL

- On ne peut utiliser CURRVAL qu'après avoir utilisé NEXTVAL au moins une fois dans la session de travail
- NEXTVAL modifie immédiatement la valeur future pour les autres transactions, même s'il est lancé dans une transaction non validée
- La valeur de CURRVAL ne dépend que des NEXTVAL lancés dans la même transaction

Modification des séquences

Définition

ALTER SEQUENCE nom_séquence
INCREMENT BY entier1
alter sequence seqdept
increment by 5

On ne peut modifier la valeur de départ

Récupérer plusieurs identificateurs

- Mettre un incrément supérieur à 1 permet d'obtenir plusieurs identificateurs en un seul appel pour obtenir de meilleures performances
- Par exemple, si on veut des identificateurs pour des lignes de factures, on en a souvent besoin de plusieurs en même temps

Informations sur les séquences pour Oracle

- Afficher la valeur d'une séquence : select seqdept.currval from dual
- Tables du dictionnaire des données : USER_SEQUENCES et ALL_SEQUENCES

Autres solutions

- DB2 et SQL Server ont une clause « IDENTITY » pour dire qu'une colonne est un identifiant, avec une valeur qui est générée automatiquement par le SGBD
- MySQL permet de déclarer une colonne **AUTO_INCREMENT**

Comparaison séquences et autres solutions

- Les séquences sont souvent plus souples si on veut récupérer la valeur des identifiants pendant l'enregistrement des données
- Exemple : dans l'enregistrement d'une facture avec ses lignes de factures, on doit disposer de l'identifiant de la facture pour le mettre en clé étrangère dans les lignes de facture

Exemple

facture(id, nom_client, date_facture,...)
ligne_facture(id_facture, nb_ligne, quantite,
id_article)

Chapitre V

Vues, Index, Procédures stockées, ...

Injection de code SQL

- Tous les langages de programmation permettent de lancer des requêtes SQL pour interagir avec une base de données
- Il est possible de construire une requête SQL dont le texte contient une partie entrée par l'utilisateur (par concaténation de chaînes de caractères)
- Attention alors à l'injection de code SQL

Exemple d'injection de code SQL

- Un programme demande son nom et son mot de passe à un utilisateur, et les range dans 2 variables nom et mdp
- Il lance cette requête et accepte l'utilisateur si elle renvoie bien une ligne "select * from utilisateur where nom = '" + nom + "' and mdp = '" + mdp + "'"
- Quel est le problème ?

Le problème

- Un pirate sait qu'un des utilisateurs autorisés s'appelle Dupond
- Il saisit « Dupond' – » pour le nom et « a » pour le mot de passe
- La requête devient :
  select * from utilisateur where nom = 'Dupond' –' and mdp = 'a'
- Mais « – » indique un commentaire avec le SGBD utilisé ; donc la requête exécutée sera :
  select * from utilisateur where nom = 'Dupond'

Les parades

- Toujours vérifier la saisie d'un utilisateur avant de s'en servir pour construire une requête SQL
- Pour l'exemple, il aurait suffit d'interdire le caractère « ' » ou de le doubler
- Les API fournies avec les langages pour accéder à une base de données offrent des facilités pour se protéger contre l'injection de code SQL
- Avec JDBC par exemple, il suffit d'utiliser des paramètres et PreparedStatement ; les caractères spéciaux comme « ' » sont alors traités comme des caractères ordinaires

Chapitre V

Vues, Index, Procédures stockées, ...

Procédures stockées

- Les SGBD modernes fonctionnent en client/serveur
- Chaque requête issue d'un client
  - transite sur le réseau
  - est « compilée » lorsqu'elle arrive au serveur : celuici recherche en particulier la meilleure façon de répondre à la requête
- La compilation peut être une étape complexe et longue à traiter et coûteuse en ressources
- Les procédures stockées sont des requêtes précompilées et stockées sur le serveur

3 étapes

1. Écriture du code de la procédure
2. Compilation et enregistrement sur le serveur ; on donne un nom à la procédure stockée
3. Exécution de la procédure en passant au serveur le nom de la procédure

Langages des procédures stockées

- Les procédures stockées peuvent inclure
  - des variables, des boucles et des tests
  - plusieurs requêtes SQL

  Le plus souvent, elles sont écrites dans un langage spécial
  - PL/SQL pour Oracle
  - PSM pour la norme SQL (PL/SQL lui ressemble mais n'est pas vraiment compatible)
  - Java pour les dernières versions d'Oracle

Avantages et inconvénients des procédures stockées

- Améliorent les performances et diminuent le trafic sur le réseau
- Encapsulent les processus métier (boîtes noires)
- Mais les langages d'écriture et les appels de ces procédures ne sont pas normalisés
- On perd donc de la portabilité si on utilise des procédures stockées

Syntaxe Officielle

Définition

CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...] routine_body

CREATE FUNCTION sp_name ([parameter[,...]]) [RETURNS type]
[characteristic ...] routine_body

- paramètre : [**IN** | OUT | INOUT] param_name type
- type : Any valid MySQL data type
- characteristic: LANGUAGE SQL | [NOT] DETERMINISTIC | **SQL SECURITY DEFINER** | INVOKER | COMMENT string
- routine_body : Commande(s) SQL valide(s)

Exemple (1/2)

Exemple

```
delimiter | – changement de délimiteur
create procedure augmentation
(in unDept integer, in pourcentage numeric, out cout numeric)
begin
select sum(sal) * pourcentage / 100 into cout
from EMP
where dept = unDept;
update EMP set sal = sal * (1 + pourcentage / 100)
where dept = unDept;
end|
```

Exemple (2/2)

Exemple

delimiter ; – *changement de délimiteur*
set @b=10; – *initialisation d'une variable*
call augmentation(@b,50,@a); – *appel de la procédure stockée*
select @a;

Langage de programmation (1/2)

- CALL : Appelle une procédure stockée précédemment définie, qui a été créée à l'aide de l'instruction CREATE PROCEDURE.
- BEGIN ... END : Contient un groupe de plusieurs instructions en vue de leur exécution.
- DECLARE : Sert à définir des variables locales, des conditions, des routines de gestion et des curseurs.
- SET : sert à modifier les valeurs des variables locales et des variables serveur globales.
- SELECT ... INTO : Sert à stocker les colonnes indiquées directement dans des variables.
- OPEN : Sert à ouvrir un curseur.
- FETCH : Extrait la ligne suivante à l'aide du curseur spécifié et avance le curseur d'une ligne.
- CLOSE : Sert à fermer un curseur ouvert.

Langage de programmation (2/2)

- IF : instruction conditionnelle IF-THEN-ELSE-END IF.
- CASE ... : construction conditionnelle d'une instruction CASE.
- LOOP : Structure de boucle simple; la sortie se fait à l'aide de l'instruction LEAVE.
- LEAVE : Sert à sortir des instructions IF, CASE, LOOP, REPEAT et WHILE.
- ITERATE : Utilisée dans les boucles pour recommencer au début de la boucle.
- REPEAT : Boucle avec le test conditionnel à la fin.
- WHILE : Boucle avec le test conditionnel au début.
- RETURNS : Renvoie une valeur d'une fonction stockée.

Utilisation d'une procédure stockée

Depuis un langage de 3ème génération (C, Java,. . .) on les appelle presque comme des procédures ou des fonctions du langage

Supprimer une procédure

Définition

DROP PROCEDURE nomProcédure

Compilation d'une procédure

- Une procédure stockée est compilée dès sa création
- Les erreurs éventuelles sont montrées par la commande **SHOW ERRORS**
- Si le schéma de la base ou la répartition des données a changé, il faut recompiler une procédure pour optimiser son exécution :

Définition

ALTER PROCEDURE nomProcédure COMPILE

Fonctions

- Comme les procédures stockées mais elles renvoient une valeur
- Elles peuvent être utilisées dans les requêtes SQL comme les fonctions prédéfinies

Exemple

```
create function euro_to_fr(somme numeric) returns number
begin
return somme * 6.55957;
end
```

Exemple

```
select nome, sal, euro_to_fr(sal) from emp
```

Chapitre V

Vues, Index, Procédures stockées, ...

Triggers (déclencheurs)

- Ils complètent les contraintes d'intégrité en permettant des contrôles et des traitements plus complexes
- Compilés et stockés sur le serveur
- Souvent écrits dans le même langage que les procédures stockées
- Normalisés en SQL3

Trigger (déclencheurs)

Définition

CREATE TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt

- La table associée doit être une table permanente (pas de vue)
- **trigger_time** est le moment d'action du trigger. Il peut être BEFORE (avant) ou AFTER (après)
- **trigger_event** indique le type de commande qui active le trigger. Il peut valoir INSERT, UPDATE ou DELETE. Il ne peut pas y avoir deux déclencheurs pour une même table avec les mêmes configurations de moment et de commande.
- **trigger_stmt** est la commande a exécuter lorsque le déclencheur s'active. Si vous voulez utiliser plusieurs commandes, il faut utiliser BEGIN ... END. Cela vous permet aussi d'utiliser les mêmes codes que ceux utilisés dans des procédures stockées.

Exemple de trigger

Exemple

```
CREATE TRIGGER totalSalaire
AFTER UPDATE ON EMP
FOR EACH ROW
begin
if (NEW.sal != OLD.sal)
then
update cumul set augmentation = augmentation + NEW.sal -
OLD.sal
where cumul.matricule = NEW.emp.matr;
end if;
end|
```

Utilisation de OLD et NEW

- **OLD**.col_name faire référence à une colonne d'une ligne existante avant sa modification ou son effacement.
- **NEW**.col_name faire référence à une colonne d'une ligne après insertion ou modification.

Clause WHEN

Exemple

```
CREATE TRIGGER totalSalaire
AFTER UPDATE ON EMP
FOR EACH ROW
when (NEW.sal != OLD.sal)
begin
update cumul set augmentation = augmentation + NEW.sal -
OLD.sal
where cumul.matricule = NEW.emp.matr;
end
```

Restriction

- Un trigger ne peut modifier par une requête SQL (update, insert, delete) les données de la table qui l'a déclenché
- Mais un trigger « for each row » peut modifier la ligne de la table concernée ; par exemple begin new.version = old.version + 1; end

Suppression de trigger

Définition

DROP TRIGGER tbl_name.trigger_name

Ordres interdits

- Les ordres COMMIT et ROLLBACK sont interdits dans un trigger

Chapitre VI

Sécurité

Chapitre VI

Sécurité

Contrôle de l'accès à la base

- Le contrôle de l'accès à la base est effectué en associant à chaque utilisateur
  - un nom de login
  - un mot de passe

  Chaque utilisateur a des privilèges d'accès à la base : droit ou non
  - de créer des tables ou des vues
  - de lire ou de modifier des tables ou des vues
  - ...

Propriétaire des données

- Une table (et les données qu'elle contient) appartient à celui qui l'a créé
- Le propriétaire d'une table peut donner à d'autres le droit de travailler avec sa table
- Les vues permettent d'affiner les droits que l'on donne sur ses propres données : on peut donner des droits sur des vues et pas sur les tables sous-jacentes

Accorder des droits sur des objets

Définition

GRANT privilège [(colonne,...)]ON table/vue
TO utilisateur
[WITH GRANT OPTION]

- L'utilisateur qui reçoit le privilège pourra le donner à d'autres utilisateurs
- Les privilèges :
  - SELECT
  - INSERT
  - UPDATE [(col1, col2,. . .)]
  - DELETE
  - ALTER
  - ALL PRIVILEGES

Accorder des droits (exemples)

Exemple

```
grant select on emp to clement;
grant select, update on emp to clement, chatel;
grant select on emp to public;
```

Changer son mot de passe :

Exemple

```
grant connect to bibi identified by motDePasse;
```

Reprendre les droits

Définition

REVOKE privilège ON table/vue FROM utilisateur

Chapitre VI

Sécurité

Gestion des accès concurrents

- Un SGBD est prévu pour travailler avec de nombreux utilisateurs/transactions
- ... qui peuvent consulter et modifier en même temps les mêmes données

Quelques questions

- A quel moment les modifications sont-elles **vues** par les autres transactions ?
- Que se passe-t-il lorsque plusieurs transactions veulent modifier les **mêmes** données ?
- Un ordre SQL (moyenne des salaires, par exemple) tient-il compte des modifications apportées par les autres transactions **validées** pendant son exécution ?

Problèmes liés aux accès concurrents

- Il peut se produire des pertes de données quand plusieurs processus veulent modifier les mêmes données en même temps
- Les principaux cas d'école sont :
  - mise à jour perdue
  - lecture inconsistante
  - lecture non reproductible
  - ligne fantôme

Mise à jour perdue

S = 500

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S

Mise à jour perdue

S = 1500

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	

Mise à jour perdue

$S = 2500$

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	
	s = s + 2000

Mise à jour perdue

$S = 1500$

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	
	s = s + 2000
Enregistrer s	

Mise à jour perdue

S = 2500

Temps

Transaction T1	Transaction T2
s = Lire S	
	s = Lire S
s = s + 1000	
	s = s + 2000
Enregistrer s	
	Enregistrer s

Éviter les mises à jour perdues

	Temps
Transaction T1	Transaction T2
Bloquer S	
s = Lire S	
	s = Lire S; attente ...
s = s + 1000	
Enregistrer s	
Débloquer S	... s = Lire S
	s = s + 2000
	Enregistrer s

Problèmes liés aux blocages : interblocage

Temps

Transaction T1	Transaction T2
Bloquer A	
	Bloquer B
Bloquer B; Attente ...	
	Bloquer A; Attente ...

Lecture inconsistante

	Temps
Transaction T1	Transaction T2
V = 100	
	v = Lire V (100)
ROLLBACK	
	Travaille avec v=100

Ce cas n'arrive pas si les modifications ne sont visibles par les autres qu'après un COMMIT.

Lecture inconsistante

Temps

Transaction T1	Transaction T2
Lire V	
	V = V + 100
	COMMIT
Lire V	

Pour éviter cela, T1 devrait bloquer les données lues (V)

Lignes fantômes

- Ce problème survient quand une transaction n'a pas perçu la création d'une ligne par une autre transaction
- Par exemple, une transaction lit d'abord le nombre d'employés et lance ensuite la lecture d'informations sur chacun de ces employés
- Si une autre transaction a ajouté des employés, le nombre lu auparavant ne correspond plus à la réalité

Lignes fantômes

	Temps
Transaction T1	Transaction T2
Récupère le nombre d'employés du dept 10 (35)	
	Ajoute un employé au dept 10
	COMMIT
Récupère les informations sur les employés du dept 10 (36 lignes)	

Pour éviter cela, T1 devrait bloquer la table des employés au début de la transaction (pas possible de bloquer des lignes qui n'existent pas !)

Transactions sérialisables

- On vient de voir que l'exécution de transactions **entrelacées** peut provoquer des pertes de données ou de cohérence
- Pour éviter ces problèmes, le SGBD doit s'arranger pour que l'exécution de plusieurs transactions entrelacées fournisse les mêmes résultats que l'exécution des mêmes transactions **les unes à la suite des autres** (dans un ordre quelconque)

Moyens de sérialiser

- Le moyen le plus courant s'appelle le **verrouillage à 2 phases** :
  - on doit bloquer un objet avant d'agir sur lui (lecture ou écriture)
  - on ne peut plus faire de blocage après avoir débloquer un objet
- On a donc 2 phases :
  1. acquisitions des verrous
  2. abandons des verrous (en pratique, au COMMIT ou ROLLBACK)

Inter-blocage avec le verrouillage à 2 phases

- Les situations d'entrelacement des transactions qui auraient provoqué des problèmes vont se traduire par des attentes ou des inter-blocages
- En cas d'inter-blocage, des transactions sont annulées et redémarrées

Estampillage

- L'estampillage est une autre stratégie que le verrouillage à 2 phases pour assurer la sérialisation des transactions
- L'ancienneté des transactions est repérée par une estampille donnée à la création de la transaction (un nombre incrémenté à chaque attribution)
- Chaque donnée accédée par une transaction reçoit l'estampille de cette transaction
- Les algorithmes qui utilisent l'estampillage assurent que l'exécution concurrente des transactions sera équivalente à l'exécution séquentielle de ces transactions dans l'ordre de leur estampille

Idée pour les algorithmes d'estampillage

- Une transaction T ne peut accéder à une donnée si cette donnée a déjà été accédée par une transaction plus jeune (donc d'estampille plus grande que celle de T)
- La transaction « trop vieille » T est annulée si elle s'est faite doubler par une transaction plus jeune
- Elle est relancée avec une estampille plus grande ; plus jeune, elle a plus de chance de pouvoir accéder à la donnée en passant « après »

Résultat de l'estampillage

- Cet algorithme est trop restrictif et ne va autoriser que peu d'exécutions parallèles des transactions
- La transaction redémarrée va peut-être entrer en conflit avec une autre transaction plus jeune
- D'autres algorithmes d'estampillage moins simplistes permettent d'améliorer le parallélisme, en particulier en distinguant les accès en lecture et les accès en écriture
- Malgré tout, ce type de traitement est souvent trop restrictif et nuit à la concurrence

Autres stratégies

- Pour améliorer les performances dans des situations de forte concurrence, les SGBD offrent la possibilité d'être plus permissif et de ne pas sérialiser les transactions

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ-UNCOMMITTED |
READ-COMMITTED | **REPEATABLE-READ**

SERIALIZABLE : les transactions s'exécutent totalement isolées des autres transactions et comme si elles s'exécutaient les unes après les autres

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ-UNCOMMITTED |
READ-COMMITTED | **REPEATABLE-READ**

READ COMMITED : les transactions ne voient les modifications des autres transactions qu'après les commit ; empêche les lectures inconsistantes mais pas les lectures non répétitives ni les lignes fantômes C'est l'isolation par défaut d'Oracle et de beaucoup de SGBD car c'est un bon compromis entre sécurité et performances

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ-UNCOMMITTED |
READ-COMMITTED | **REPEATABLE-READ**

READ UNCOMMITED : les transactions voient les modifications avant même le commit

Niveaux d'isolation des transactions (InnoDB)

Définition

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE | READ COMMITED | READ-UNCOMMITTED |
READ-COMMITTED | **REPEATABLE-READ**

REPEATABLE READ : empêche les lectures non répétitives mais pas les lignes fantômes

En résumé...

Niveaux par isolation décroissante :

- SERIALIZABLE (souhaitable, mais coûteux car provoque des blocages de tables)
- REPEATABLE READ
- READ COMMITED (souvent un bon compromis)
- READ UNCOMMITED (très rarement utilisé)

Traitement des accès concurrents par les SGBD

- Le but : favoriser au maximum les accès concurrents pour permettre l'exécution du plus grand nombre de transactions dans un temps donné (argument commercial important)
- Tous les SGBDs n'ont pas les mêmes solutions pour atteindre ce but

Durée des blocages

Un blocage n'a lieu que le temps d'une transaction

Accès concurrents

- Pas de verrou pour effectuer une lecture
- Les lectures ne sont pas bloquées par les écritures, et vice-versa
- Une transaction est bloquée si elle veut modifier des données qui sont en cours de modification par une autre transaction
- La granularité minimum de blocage est la ligne (dépand du moteur (InnoBD, MyISAM, ...)

Autre façon de traiter les accès concurrents

- Beaucoup de SGBD bloquent les données **lues**
  - ce blocage n'interdit pas leur lecture par d'autres transactions mais interdit leur modification

  Une écriture bloque les données **modifiées**
  - elles ne peuvent être modifiées par d'autres transactions (comme avec Oracle)
  - mais elles ne peuvent pas non plus être lues par d'autres transactions

Blocages explicites et implicites

- Des blocages sont effectués implicitement par certaines commandes (en particulier UPDATE, INSERT et DELETE)
- Si le comportement par défaut du SGBD ne convient pas pour un traitement, on peut effectuer des blocages explicites des lignes ou des tables
- Les inter-blocages sont détectés par le SGBD qui annule un des ordres qui a provoqué l'inter-blocage

Granularité des blocages

- 2 types de blocages :
  - au niveau d'une table
  - au niveau d'une ligne
- Dans certains SGBDs le blocage d'une ligne implique le blocage de toute la page/bloc (plusieurs lignes) qui contient la ligne
- Certains SGBD transforment de trop nombreux blocages de lignes d'une table en un blocage de toute la table

Réponses à quelques questions

- A quel moment les modifications sont-elles vues par les autres transactions ?
  après la validation de la transaction (COMMIT)
- Que se passe-t-il lorsque plusieurs transactions veulent modifier les mêmes données ?
  blocages implicites du SGBD $\Rightarrow$ attente

Réponses à quelques questions

- Quand un ordre SQL tient compte de plusieurs lignes (moyenne des salaires, par exemple), cet ordre tient-il compte des modifications apportées par les autres transactions pendant l'exécution de l'ordre ?
  Ca dépend du SGBD. Avec Mysql, non