



Institut Supérieur d'Informatique Appliquée
27 Rue de Fontarabie
75 020 Paris
Tél : 01 56 98 21 30
www.insia.org

INSIA
BASES DE DONNEES RELATIONNELLE
Algèbre relationnelle et SQL
1 : Select mono-table

B. LIAUDET

SOMMAIRE

SOMMAIRE	<u>2</u>
INTRODUCTION	<u>3</u>
1. Objectifs généraux	3
2. Réquisits	3
3. Généralités sur les bases de données	4
4. Les 3 objectifs majeurs d'une BD	6
5. Généralités sur les SGBD	8
8. SGBD-R et SQL	11
MODÉLISATION : MODELE RELATIONNEL	<u>14</u>
1. La modélisation	14
2. Le modèle relationnel	15
SQL : INTERROGATION DE LA BD – MONO-TABLE	<u>23</u>
1. Algèbre relationnelle et SQL	23
2. La commande de recherche : le select	26
3. Projection = filtre des colonnes	28
4. Restriction = filtre des lignes	32
5. Restriction et projection = filtre des lignes et des colonnes	33
6. Opérateurs et fonctions	36
7. Tris et limites	43
8. Calculs statistiques élémentaires : les fonctions de groupe	46
9. Calculs statistiques : agrégats ou group by	49
10. Calculs statistiques : statistiques et agrégations avancées	53
11. Bilan syntaxique	56
12. Détails syntaxiques MySQL	57

Edition : août 2011

Evolutions prévues : intégrer DDL, Select : MySQL, ORACLE, PG. Faire un doc d'installation-environnement de dev pour MySQL-ORACLE-PG-SQL-Server.

INTRODUCTION

1. Objectifs généraux

L'objectif de ce cours est de faire comprendre ce qu'est une base de données (ce qui n'a rien d'évident), et particulièrement une base de données relationnelle, de comprendre comment on la fabrique et comment on s'en sert, c'est-à-dire comment on stocke les données et comment on consulte les données de la base de données.

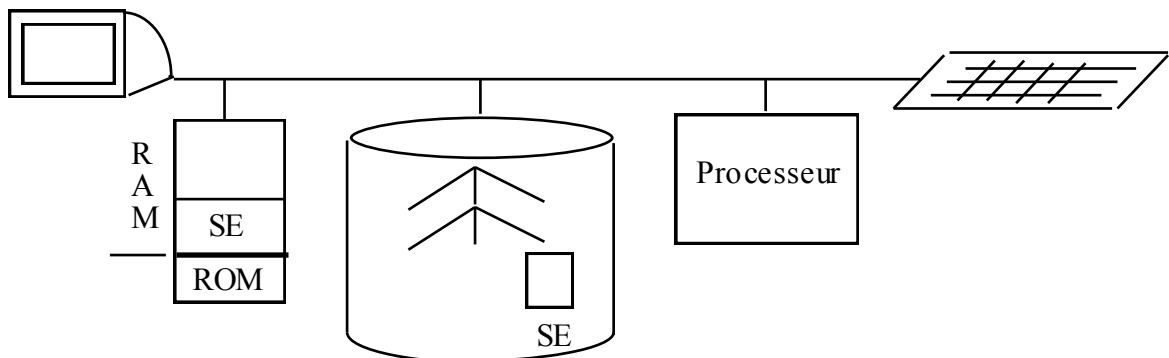
Pour cela nous serons amenés à utiliser le langage SQL et un SBB. Le langage SQL étant standard, on pourra utiliser à peu près indifféremment MySQL, PostgreSQL, ORACLE, SQL Server. Les versions (ORACLE 10, ORACLE 11, SQL Server 2005, SQL Server 2008, MySQL 4, MySQL 5, etc.) auront peut d'impact sur la base de l'apprentissage du langage SQL.

2. Réquisits

Il n'est pas nécessaire de connaître les langages de troisième génération (Pascal, C, C++, php, etc...) ni l'algorithmique.

Il vaut mieux (mais ce n'est pas obligé) connaître l'architecture et le fonctionnement général d'un ordinateur : unité centrale, processeur, périphériques d'entrée et de sortie, bus, mémoire, ROM (Read Only Memory, mémoire morte), RAM (Read Access Memory, mémoire vive), disque dur, répertoires, fichiers, logiciels, système d'exploitation.

➤ Architecture des ordinateurs :



Rappelons que quand l'ordinateur démarre, le petit programme codé dans la ROM va chercher le logiciel « système d'exploitation » (XP, LINUX, UNIX, MacOS, etc... SE sur le schéma) sur le disque dur et le recopie sur la RAM. L'utilisateur communique alors avec le système d'exploitation. Le système d'exploitation offre trois grands types de fonctionnalité : la gestion des fichiers (dir, cd, grep, etc.), la gestion des périphériques et le lancement des logiciels.

3. Généralités sur les bases de données

Définition générale d'une base de données

Présentation

La notion de base de données n'est pas facile à définir précisément. On peut toutefois en donner une définition intuitive très large :

Une base de données est constituée de deux éléments :

- un regroupement de données (d'informations) en grand nombre. Ces données décrivent (représentent) des objets du monde réel.
- un ensemble d'**outils de gestion** permettant de **consulter** ces données et les objets qu'elles représentent, mais aussi d'en **ajouter**, d'en **retirer** et d'en **modifier** : c'est le **SGBD** (système de gestion de la base de données)

➤ *Remarques*

Une base de données n'est pas constituée uniquement par des objets du monde réel mais aussi par leurs représentations.

Une base de données n'est pas nécessairement informatisée.

➤ *Exemples*

Une bibliothèque gère est une base de données. Les livres et les adhérents sont les objets du monde réel. Il existe un fichier avec les informations représentant tous les livres (numérotation, classement, etc.), tous les adhérents (nom, adresse, etc.), tous les emprunts, et aussi des fichiers pour les recherches bibliographiques. Ces outils sont les outils de gestion de la BD.

Un magasin avec ses articles en rayon gère une base de données. Les articles sont les objets du monde réel.

Un disque dur avec ses fichiers est une base de données. Les fichiers sont les objets du monde réel. Le système d'exploitation permet de manipuler

Un cerveau (plus ou moins rempli !) est une base de données. Les connaissances sont des « objets » du monde réel. L'accès aux connaissances est plus ou moins efficace !

Définition d'une base de données informatique

Présentation

Une base de données informatique est une base de données dont

les données et les outils de gestion sont informatisés.

Une base de données informatique est constituée de deux éléments :

- un regroupement de données informatiques en grand nombre. Ces données décrivent des objets du monde réel.
- un ensemble d'**outils de gestion informatisés** permettant de **consulter** ces données, mais aussi d'en **ajouter**, d'en **retirer** et d'en **modifier** : c'est le **SGBD** (système de gestion de la base de données).

➤ *Exemples*

Le catalogue informatisé d'une bibliothèque : il permet d'accéder à la liste de tous les ouvrages de la bibliothèque, de savoir s'ils sont disponibles ou pas, etc.

Le système de réservation de la SNCF (Système Socrate) : il permet d'accéder à tous les trains et à toutes les places dans les trains, de savoir si elles sont disponibles, etc.

Les sites internet de journaux (Le Monde, le Figaro, etc.) mais aussi les petits sites sous SPIP (système de publication sur internet) sont des bases de données d'articles : ils permettent d'accéder aux articles pour les lecteurs, de les mettre en ligne pour les auteurs, etc.

Problème de vocabulaire

Quand on parle de base de données (BD), on parle :

- Soit de l'ensemble constitué par les données et le SGBD (par exemple : « Cours de base de données »)
- Soit des données uniquement (par exemple : « on utilise la base de données des employés »)
- Soit du SGBD uniquement (par exemple « j'utilise une BD postgresSQL »)

Dans ce cours, on fera en général la distinction entre les données (la base de données ou BD) et les outils de gestion (le SGBD).

4. Les 3 objectifs majeurs d'une BD

L'intégrité des données : altération et incohérence

Garantir l'intégrité des données, c'est éviter l'altération et l'incohérence des données.

L'altération des données

Il y a plusieurs sources d'altération possibles : l'usure, les pannes, les erreurs, les malveillances. Une BD (et un SGBD) aura comme objectif d'en limiter la possibilité.

L'incohérence des données

Une donnée est incohérente si elle est contradictoire avec une autre donnée.

Il y a deux grands types d'incohérence :

- **La duplication des données avec des valeurs différentes.** Exemple : deux adresses différentes pour une même personne.
- **Les valeurs aberrantes.** Exemples : un âge négatif ou supérieur à 150 ; une donnée faisant référence à une autre donnée qui n'existe pas.

La BD a pour objectif d'être un **réservoir d'informations canonique** (unique et commun), **garantie sans incohérences** (donc sans duplication de données).

La distinction entre données et traitements

Les données existent indépendamment des traitements qu'on leur applique. Ainsi, on a d'un côté les données et leur modèle et de l'autre les traitements rationnalisés.

Modèle des données indépendant des traitements

➤ *Signifiante directe des données*

L'information correspondant à une donnée doit être directement intelligible : elle ne correspond pas à un calcul ou à un codage (pas de code spéciaux dans la BD).

➤ *Vision unifiée des données manipulées*

La BD permet de **produire un modèle des données** qui permet d'apporter une **vision unifiée des données manipulées** (dans une entreprise ou n'importe quel système d'informations, scientifique par exemple), indépendamment des traitements qui leur sont appliqués.

Cette vision unifiée permet une meilleure compréhension de la réalité représentée par les données.

Traitements rationnalisés

Une fois les données définies, les traitements se ramènent essentiellement à ajouter, retirer, modifier et consulter les données.

A cela, s'ajouteront ensuite les algorithmes de transformations (calculs) plus ou moins compliqués.

Performance et optimisation

Une BD doit fournir des performances acceptables par l'utilisateur. C'est la problématique de l'optimisation.

5. Généralités sur les SGBD

Définition

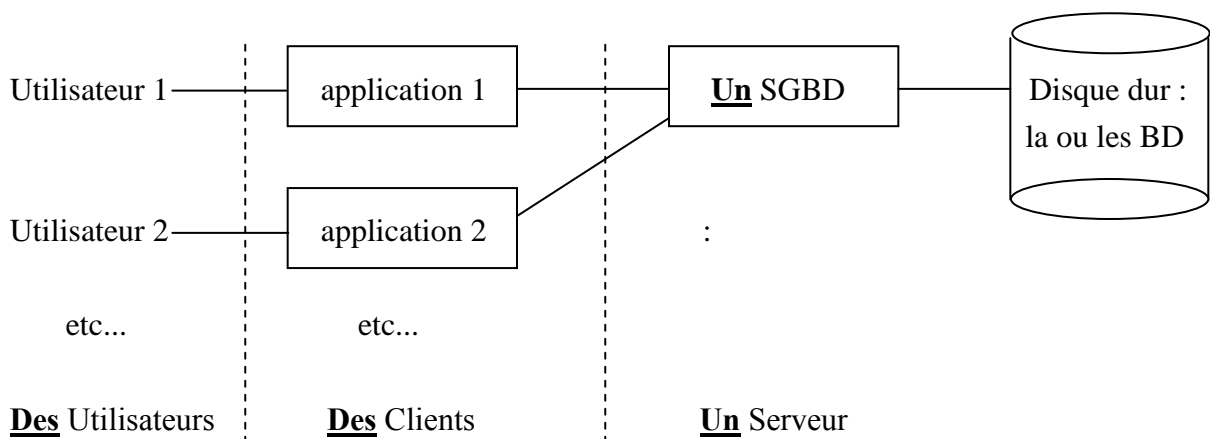
Le SGBD, Système de Gestion de Base de Données, est le logiciel qui réalise les outils de gestion de la base de données. Il permet essentiellement de :

- Définir les données qu'on veut manipuler,
- Ajouter, modifier, supprimer des données
- Consulter des données

Le SGBD répond aux objectifs de la BD.

Architecture

L'architecture la plus courante des SGBD est une architecture Client — Serveur : le SGBD est le serveur avec lequel des clients peuvent communiquer.



- Une BD est stockée sur un ou plusieurs **disques durs**.
- Pour une BD, il y a **un SGBD et un seul**. Le SGBD et la BD sont en général sur la même machine (pris en charge par le même SE). Le SGBD peut gérer plusieurs BD.
- Pour un SGBD donné, il peut y avoir **plusieurs applications** qui communiquent avec lui. Le SGBD est le serveur. Chaque application est un client. Une application peut être sur une machine différente de celle du serveur, ou sur la machine du serveur.
- Une **application** est un programme (un logiciel) qui envoie des commandes au SGBD pour manipuler d'une façon ou d'une autre les données de la BD sur le disque dur. Par exemple : un **programme PHP ou java** qui permet de communiquer avec une SGBD MySQL, PostgreSQL, ORACLE.

Les 3 types d'utilisateur d'un SGBD

Il y a trois types d'utilisateur d'un SGBD :

L'administrateur

Il gère les accès et les droits des utilisateurs : problème de sécurité.

Il gère aussi le bon fonctionnement des bases de données : problème d'efficacité.

Le développeur

Il programme une application qui sera cliente du SGBD. Il a accès à la BD à travers l'application cliente dite : « calculatrice SQL ».

L'utilisateur final

Il utilise une application cliente du SGBD. En général, il n'a accès à la BD qu'à travers une application cliente spécifique.

Les différents types d'objets gérés par un SGBD

Un SGBD gère un certains nombre d'objets :

Des tables : pour mettre les données

Des bases de données : pour distinguer des applications différentes

Des index : pour optimiser les requêtes

Des contraintes d'intégrité : pour garantir la cohérence des données

Des utilisateurs : pour se connecter à la BD et leur donner des droits particuliers

Des droits : pour limiter l'accès des utilisateurs à la BD

Etc.

Pour chacun de ces objets, le SGBD permet la création, modification, suppression.

Deux grands types d'applications autour d'un SGBD

Il y a deux grands types d'application autour d'un SGBD :

L'informatique « transactionnelle » : OLTP

Les applications d'informatique transactionnelle regroupent à la fois les applications de gestion (système d'information, site marchand, etc.) et des applications industrielles qui utilisent une BD. Ce sont toujours des applications de grosse taille.

On parle aussi d'OLTP : On-Line Transaction Processing

L'informatique « décisionnelle » : OLAP

Les applications d'informatique décisionnelle servent à exploiter les données le plus souvent issues d'applications d'informatique transactionnelle.

Ces applications mettent en œuvre des notions de statistiques, d'analyse de données et de datamining. Elles peuvent s'appuyer sur des fonctionnalités SQL proposées par le SGBD (Group By, Pivot, etc.).

Ces applications sont en général de petite taille. Leur finalité est l'aide à la décision.

Les notions de «Business Intelligence» : BI, data-mining et data-warehouse sont des notions connexes à ce domaine.

On parle aussi d'OLAP : On-Line Analytical Processing.

Objectif majeur d'un SGBD : l'intégrité des données

L'objectif majeur d'un SGBD est le même que celui de la BD : il s'agit de garantir l'intégrité des données.

Cet objectif se déploie sur plusieurs registres :

La conformité au modèle

- La gestion de base de l'intégrité consiste à vérifier que les données saisies sont cohérentes par rapport à la définition qu'on en a donnée.

Les accès concurrents

- La gestion des accès concurrents permet d'éviter des incohérences dues à des modifications faites en même temps sur une même donnée.

La sécurité sur panne

- La gestion de la sécurité sur panne permet de garantir le maintien de la cohérence des données, même quand une panne intervient au cours d'une période de modification des données.

Les autorisations d'accès

- La gestion des droits des utilisateurs permet de limiter les possibilités des utilisateurs sur les données.

8. SGBD-R et SQL

Petit historique des modèles de données

Il existe différents types de BD en fonction du modèle d'organisation des données qu'elles utilisent.

A ce modèle est associé un langage de manipulation des données.

- **Années 60** : les anciens modèles. Modèle hiérarchique, modèle réseau, langages d'accès navigationnels permettant de circuler dans des structures de types graphes.
- **1970** : Création du modèle relationnel. Article de CODD.
- **Années 80** : Premiers SGBD-Relationnels sur le marché (ORACLE, Ingres, Informix, sybase, Dbase IV, etc.)
- **Années 90** : Premiers SGBD-Objets.
- **Aujourd'hui** : L'Objet n'a pas remplacé le Relationnel. Le marché est dominé par les SGBD-R (ORACLE, SQL-Server (microsoft), DB2 (IBM), MySQL, PostgreSQL, ACCESS...). Mais les technologies sont souvent mixtes (Relationnel et objet).

SGBD-R

Aujourd'hui, le modèle relationnel domine le marché et la théorie des bases de données.

Les SGBD correspondant sont des SGBD-R.

Le SQL

Le SQL est le langage qui permet d'envoyer toutes les requêtes possibles à un SGDB-R : création, modification, suppression, consultation, etc.

SQL signifie Structured Query Language, c'est-à-dire langage structuré de requêtes.

Il est aussi parfois appelé : « SEQUEL » (prononcer : sicouel), pour Structured English Query Language.

Le SQL n'est pas un langage de programmation qui permet d'écrire des programmes composés d'une suite d'opérations élémentaires (comme le langage PHP, par exemple, et comme tous les langage dit « impératif » : C, C++, Pascal, VB, PHP, etc.).

Le SQL est un langage qui permet d'écrire des opérations élémentaires qui sont indépendantes les unes des autres.

SQL ou algèbre relationnelle ?

Le **SQL** un langage de programmation **fondé sur l'algèbre relationnelle** qui est la théorie des opérations qu'on peut appliquer à un objet mathématique particulier : les tableaux de données (appelées « relations » en algèbre relationnelle).

Toutefois, le SQL peut être considéré comme un **ensemble d'opérateurs** plutôt que comme un langage de programmation.

En ce sens, l'application cliente du SGBD qui permet d'utiliser directement le SQL peut être considérée comme une « **calculette SQL** » qui, au lieu de travailler sur des nombres, travaille sur des tableaux de données.

La calculette SQL

Il existe une application particulière qui permet de créer, modifier, détruire et consulter une BD : le « Command Line Client » et ses dérivés. Cette application concerne les programmeurs et pas les utilisateurs finaux.

Cette application particulière, on l'appellera : **la calculette SQL**. Par exemple : **psql** (postgresql), **mysql**, **SQL*PLUS** (ORACLE), etc.

Il existe des **versions graphiques des calculettes**. Par exemple : **phpMyAdmin** (MySQL), MySQL GUI tools, phpPgAdmin (postgresql), PgAdminIII (postgresql), **ORACLE developer**, **SQL Server Management Studio Express** (SSMSE), etc.

On va apprendre à se servir de cette calculette SQL

PL-SQL

Dans la plupart des SGBD-R, il existe un langage de programmation à proprement parler qui permet d'écrire des successions d'opérations relationnelles (SQL) sous la forme d'un programme en utilisant les mécanismes usuels de la programmation impérative type Pascal, VB ou C, à savoir : les variables, les affectations, les tests, les boucles et les fonctions. Ce langage s'appelle : le PL-SQL, c'est-à-dire : Programmation Language SQL.

Le PL-SQL ne permet pas de développer une interface utilisateur très élaborée. Il ne remplace donc pas le développement d'une application cliente.

Le PL-SQL permet de :

- éviter aux applications clientes de répéter les mêmes séries d'opérations SQL : principe de modularité
- mieux gérer l'intégrité des données (triggers et règles).
- séparer les modules de calcul des modules d'affichage

Dans ce cours, on ne traitera pas du tout le PL-SQL.

Historique du SQL

Années 70 : premiers prototypes de SQL à la suite de l'article de CODD.

1979 : première version de SQL, proposé par ORACLE.

1986 : SQL ANSI (American National Standard Institute)

1989 : SQL-1, ISO et ANSI (International Standard Organisation)

1992 : SQL-2, ISO et ANSI (jointures externes et « join »)

1999 : SQL-3, ISO et ANSI (SQL orienté objet)

Présentation

On voit que le SQL existe depuis 1980 en pratique et 1970 en théorie.

C'est un langage pérenne : il a vocation à durer !

Pourquoi :

Qui dit SI dit BD : la BD est le cœur du SI.

L'algèbre relationnelle est une théorie mathématique simple et solide (et non pas une technologie).

L'implantation massive du SQL rend son élimination coûteuse.

Les solutions « NO SQL » restent localisées.

Pourquoi changer ?

Pour détrôner le SQL, il faudrait :

Une meilleure théorie : l'algèbre relationnelle étant fondée sur la logique de base et celle-ci étant très stable, il est peu probable qu'une meilleure théorie se développe.

Des outils de calcul pour remplacer la théorie : l'intérêt du SQL, c'est qu'il permet de ranger correctement et ainsi d'éviter les incohérences et d'accélérer les traitements. Si des algorithmes permettaient d'obtenir les mêmes résultats sans avoir à ranger, le SQL perdrait de son intérêt. Toutefois, si cette solution se réalisait (ce qui veut dire des ordinateurs beaucoup plus puissants), elle se positionnerait probablement en interface au dessus d'une BD SQL.

Un pari sans grand risque

Aujourd'hui (années 2010), on peut parier sans grand risque que le SQL sera toujours présent d'ici à la fin de la carrière des étudiants !

MODÉLISATION : MODELE RELATIONNEL

1. La modélisation

La modélisation est l'activité qui consiste à produire un modèle.

Un modèle est ce qui sert ou doit servir d'objet d'imitation pour faire ou reproduire quelque chose.

On s'intéresse ici à la modélisation des données.

Un modèle des données est une représentation de l'ensemble des données. Cette représentation prend en compte un outil de représentation (un langage) et un niveau de précision (des contraintes méthodologiques).

Il existe plusieurs modèles de représentation des données : hiérarchique, relationnel, entité-association, objet, ensembliste, etc.

Les deux modèles dominants actuellement sont : le modèle relationnel : MR (qui correspond aux SGBD-R) et le modèle entité-association : MEA (qui est indépendant du type de SGBD utilisé). Ces deux modèles correspondent à 2 langages différents.

Les schémas entité-relation et les diagrammes de classes UML peuvent être utilisés comme autres langages à peu près équivalents au MEA. On peut aussi les utiliser pour le MR.

La méthode MERISE, utilisée quasi-exclusivement en France, distingue entre 3 types de modèles selon des critères méthodologiques : le modèle conceptuel des données : MCD, le modèle logique des données : MLD et le modèle physique des données : MPD. L'usage tend à rendre équivalents MCD et MEA, MLD et MR, MPD et SQL.

Le MCD est du niveau de l'analyse fonctionnelle : il est adapté à la maîtrise d'ouvrage (MOA) et à la partie fonctionnelle de la maîtrise d'œuvre (MOE).

Le MLD est du niveau de l'analyse organique et est adapté à la maîtrise d'œuvre (MOE).

La « jungle » des modèles !			
Méthode	MCD	MLD	MPD
Langage	MEA, schema E-R, UML	MR	SQL
Type de langage	Algo client	Algo informaticien	Code
Niveau	Analyse fonctionnelle (externe). MOA ou MOE	Analyse organique (interne, technique) MOE	Réalisation MOE

2. Le modèle relationnel

Présentation

Le modèle relationnel a été inventé par Codd à IBM-San Jose en 1970.

C'est un modèle mathématique rigoureux basé sur un concept simple : celui de relation (ou table, ou tableau).

Ce modèle, c'est celui qui est implanté dans les SGBR-R.

Il permet à la fois de fabriquer la BD et de l'interroger.

Table, tuple, attribut, clé primaire

Exemple traité

Un service de ressource humaine dans une entreprise veut gérer le personnel. Dans un premier temps, on veut pouvoir connaître le nom, la fonction, la date d'entrée, le salaire, la commission (part de salaire variable) de chaque employé et le numéro du département dans lequel travaille chaque employé.

Chaque employé a donc les caractéristiques suivantes :

Nom, fonction, date d'entrée, salaire, commission, numéro de département

Table, tuples et attributs

Pour ranger ces données, on peut faire un tableau à 6 colonnes :

RELATION

6 attributs :

Employés	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept
4 tuples :	TURNER	SALESMAN	8-SEP-81	3000	0	10
	JAMES	CLERK	3-DEC-81	1800	NULL	30
	WARD	SALESMAN	22-FEB-81	2500	500	20
	TURNER	ANALYST	3-DEC-81	5000	NULL	10

Vocabulaire

Relation = tableau = table = classe = ensemble = collection

Tuple = ligne du tableau = élément = enregistrement = individu = objet = donnée

Attribut = colonne du tableau = caractéristique = propriété = champ

BD = toutes les lignes de toutes les tables

NULL

NULL est la seule information codée qu'on rentre dans une table : elle signifie « non renseignée ». La valeur « 0 », par contre, ne signifie pas du tout « non renseignée », mais bien « valeur = 0 », comme on dirait « valeur = 500 ».

Clé primaire

On souhaite pouvoir distinguer facilement chaque ligne d'une autre ligne. Or, certains employés ont le même nom.

Pour distinguer chaque ligne, on introduit la notion de clé primaire.

La clé primaire est un attribut qui identifie un tuple et un seul. Cela veut dire que quand on connaît la clé primaire, on sait de quel tuple on parle, et on peut donc accéder sans ambiguïté à toutes les autres informations du tuple.

Une clé primaire est toujours renseignée.

Exemple type de clé primaire : le numéro de sécurité sociale dans un tableau de personne. Quand on connaît le numéro de sécurité sociale, on sait de qui on parle, donc tous les attributs sont déterminés (même si on ne connaît pas leur valeur à un instant donné).

Dans le tableau des employés, la clé primaire pourrait être un numéro de référence choisi par l'entreprise. On le nomme NE (pour Numéro d'Employé).

RELATION

7 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept
4 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30
	3	WARD	SALESMAN	22-FEB-81	2500	500	20
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10

Clé secondaire

Une clé secondaire est un attribut qui pourrait être clé primaire, mais qui ne l'est pas.

Par exemple, dans le tableau des employés, on pourrait avoir le numéro de sécurité sociale. Cet attribut détermine tous les autres. Si on garde le numéro d'employé comme clé primaire, le numéro de sécurité sociale est alors clé secondaire.

En l'occurrence, on a tout intérêt à ne pas faire du numéro de sécurité sociale la clé primaire car on peut imaginer que l'employé existe sans que cette information soit renseignée.

Une clé secondaire peut ne pas être renseignée.

Clé significative

La clé significative, c'est l'attribut qui sert de clé dans le langage ordinaire. Dans le cas des employés, c'est leur nom. Toutefois, il peut y avoir des homonymes : la clé significative est utile dans le langage ordinaire pour savoir de quoi on parle, mais elle est insuffisante dans le langage mathématique pour garantir l'identification de l'individu.

Schéma des tables et schéma de la BD

Le schéma d'une table consiste à écrire la table sur une ligne avec les noms de code des attributs:

EMP(NE, nom, fonction, dateEmb, sal, comm)

L'ensemble des schémas des tables constitue le schéma de la BD.

Formalisme usuel

La clé primaire est notée en premier et est soulignée.

La table des employés représente une réalité physique. On peut l'appeler « **table-nom** ».

Le nom donné à une table-nom est un nom commun, au pluriel puisqu'une table contient plusieurs tuples. Toutefois, on peut aussi les mettre au singulier si ces tables sont destinées à produire des classes dans un mapping relationnel objet car le nom d'une classe, en tant que type, est par contre au singulier.

La clé primaire d'une table-nom est N (pour numéro) suivi des premières lettres du nom de la table (une seule éventuellement). Cette technique n'est d'usage que pédagogique ! On peut aussi les nommer « idNomDeLaTable » ou encore « id », quelle que soit la table.

Expression ultra-schématique

De façon ultra-schématique, la table EMP peut s'écrire ainsi :

E (E, Ex)

E, c'est le nom de la table (initial de Emp).

E : c'est la clé primaire de la table E

Ex : ce sont les attributs de la table E

L'expression ultra schématique consiste à ne distinguer qu'entre les attributs clés et les attributs non clés. On abordera l'intérêt de cette représentation dans le cas de modélisation complexe.

Clé étrangère

Présentation

Au problème précédent de gestion des ressources humaines, on ajoute les spécifications suivantes :

Le service du personnel souhaite aussi connaître le nom du département dans lequel l'employé travaille. L'entreprise est répartie dans plusieurs villes. Les départements sont donc caractérisés par leur nom et par leur ville. Un employé travaille dans un département et un seul. Il peut y avoir plusieurs départements qui ont le même nom.

On va donc ajouter une table : la table des départements :

RELATION		3 attributs :	
Départements	ND	Nom	Ville
4 tuples :	10	ACCOUNTING	NEW YORK
	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

ND est le numéro de département : il était déjà dans la table des employés.

ND est clé primaire dans la table des départements.

Désormais, le numéro de département ND de la table des employés fait référence au numéro de département de la table des départements.

ND est ainsi clé étrangère dans la table des employés.

Définition

Une clé étrangère est un attribut qui fait référence à une clé primaire.

Schéma de la BD

Le schéma de la BD consiste à écrire chaque table sur une ligne avec les noms de code des attributs :

EMP(NE, nom, fonction, dateEmb, sal, comm., #ND)

DEPT (ND, nom, ville)

Formalisme usuel pour la clé étrangère

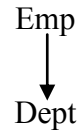
Le nom d'une clé étrangère est en général le **nom de la clé primaire** qu'elle référence.

Les clés étrangères sont mises **en dernier dans la liste** des attributs.

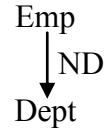
Le nom des clés étrangères est **précédé d'un « # »** sur le papier, mais pas dans la programmation !

Graphe des tables

Le graphe des tables montre les tables et le lien entre les tables :



On peut aussi présenter le nom de l'attribut commun :



Expression ultra-schématique

De façon ultra-schématique, les tables EMP et DEPT peuvent s'écrire ainsi :

E (E, Ex, #D)

D (D, Dx)

E, c'est le nom de la table Emp (initial de Emp).

E : c'est la clé primaire de la table E

Ex : ce sont les attributs de la table E

#D : c'est une clé étrangère qui ramène sur la clé primaire D

D, c'est le nom de la table Dept (initial de Dept)

D : c'est la clé primaire de la table D

Dx : ce sont les attributs de la table D

L'expression ultra-schématique distingue uniquement les attributs clés primaires, les attributs clés étrangères et les attributs non clés.

Cette représentation permet de se focaliser sur la question des liens entre les tables. C'est utile dans les modèles complexes.

Pourquoi avoir créé 2 tables ?

Une autre solution au problème posé aurait pu être d'ajouter 2 attributs dans la table des employés :

RELATION

7 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept	Nom	Ville
4 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10	ACCOUNTING	NEW YORK
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30	SALES	CHICAGO
	3	WARD	SALESMAN	22-FEB-81	2500	500	20	RESEARCH	DALLAS
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10	ACCOUNTING	NEW YORK

Cette solution pose deux problèmes :

- 1) Elle ne permet pas de définir le département n°40
- 2) Elle risque de conduire à des incohérences dans la base :

RELATION

7 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept	Nom	Ville
4 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10	ACCOUNTING	DETROIT
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30	SALES	CHICAGO
	3	WARD	SALESMAN	22-FEB-81	2500	500	20	RESEARCH	DALLAS
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10	ACCOUNTING	NEW YORK

INCOHERENCE !!!

Si dans le tuple numéro 1, on modifie la ville, remplaçant NEW YORK par DETROIT, on aura une BD incohérente puisque dans le tuple 1, le département 10 est à DETROIT et dans le tuple 4, ce même département est à NEW YORK

Clé étrangère réflexive

Présentation

Au problème précédent de gestion des ressources humaines, on ajoute les spécifications suivantes :

Chaque membre du personnel a un supérieur hiérarchique et un seul lui-même membre du personnel, sauf le président qui n'a pas de supérieur hiérarchique.

La table résultat ressemblera à celle-ci

RELATION

8 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	# ND	*NEchef
5 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10	5
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30	1
	3	WARD	SALESMAN	22-FEB-81	2500	500	20	4
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10	5
	5	BOSS	PRESIDENT	10-DEC-80	7000	NULL	5	NULL

Définition

Une clé étrangère réflexive est un attribut qui fait référence la clé primaire de sa table.

Schéma de la BD

Le schéma de la BD consiste à écrire chaque table sur une ligne avec les noms de code des attributs :

EMPLOYES(NE, nom, job, datemb, sal, comm., #ND, *NEchef)

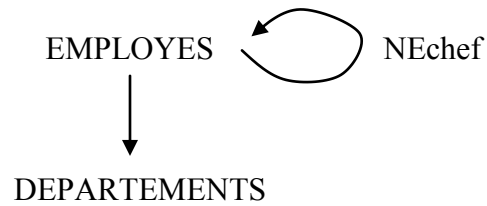
DEPARTEMENTS (ND, nom, ville)

Formalisme complet

1. Les clés **primaires** sont soulignées et placées en premier dans la liste des attributs.
2. Le **nom de la clé primaire** est constitué de : « N »+1^{ère} lettre de la table (NE).
3. Les clés **étrangères** sont précédés d'un #.
4. Les clés **étrangères** sont mises en dernier dans la liste des attributs.
5. Les clés **étrangères réflexives** sont précédés d'un *.
6. Les clés **étrangères réflexives** sont mises après les clés étrangères non-réflexives.
7. Le **nom des clés étrangères réflexives** est constitué de : clé primaire + code de l'attribut (NEchef)

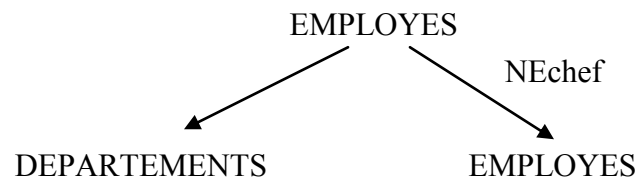
Graphe des tables

Le graphe des tables montre les tables et le lien entre les tables :



On précise le nom de l'attribut clé étrangère réflexive.

On peut aussi dupliquer la table des EMPLOYES :



Définition d'une BD

Une BD c'est un ensemble de tables avec leurs tuples.

Un SGBD gère une ou plusieurs BD.

SQL : INTERROGATION DE LA BD – MONO-TABLE

1. Algèbre relationnelle et SQL

Présentation de l'algèbre relationnelle

L'algèbre relationnelle c'est l'algèbre des relations, c'est-à-dire l'algèbre des tables.

De même que l'algèbre des nombres (c'est-à-dire l'algèbre commune) est la théorie des opérations portant sur les nombres, l'algèbre relationnelle est la théorie des opérations portant sur les tables.

L'intérêt du modèle relationnel réside dans ses capacités de représentation des données, mais aussi dans le fait qu'il permet de développer une algèbre constituée d'un petit nombre d'opérations qui permettent tous les traitements possibles sur les relations.

De même que les opérations de l'algèbre des nombres portent sur des nombres et produisent des nombres comme résultats, **les opérations de l'algèbre relationnelle portent sur des tables et produisent des tables comme résultats**.

Les opérations de l'algèbre relationnelle

L'algèbre relationnelle permet de :

- Créer, modifier, détruire des tables (et donc des attributs) : DDL
- Créer, modifier, détruire des tuples : DML
- Consulter les tuples : DSL.

La consultation des tuples d'une table consiste à :

- Filtrer les attributs
- Filtrer les tuples
- Trier les données
- Faire des opérations statistiques

Ces opérations peuvent s'appliquer à une ou plusieurs tables.

Exemple d'opérations :

- Créer la table des employés (création d'une table et de ses attributs : DDL).
- Créer les employés dans la table des employés (création de tuples dans une table : DML).
- Augmenter le salaire de tous les employés (modification de tuples dans une table : DML).
- Quel est le nom de tous les employés qui sont vendeurs ? (filtre des lignes et des colonnes : DSL).
- Quel est le nom de tous les employés travaillant dans tel département ? (filtre des lignes et des colonnes à partir de deux tables : DSL).

- Donner la liste des employés par ordre alphabétique (tri : DSL).
- Quel est le salaire moyen de tel métier de l'entreprise ? (filtre et calcul statistique : le résultat est une table avec un tuple et un attribut : DSL).

Le SQL ou algèbre relationnelle ?

Le **SQL** (Structured Query Language) est un langage de programmation **fondé sur l'algèbre relationnelle** qui est la théorie des opérations qu'on peut appliquer à un objet mathématique particulier : les tableaux de données (appelées « relations » en algèbre relationnelle).

Toutefois, le SQL peut être considéré comme un **ensemble d'opérateurs** plutôt que comme un langage de programmation.

En ce sens, l'application cliente du SGBD qui permet d'utiliser directement le SQL peut être considérée comme une « **calculatrice SQL** » qui, au lieu de travailler sur des nombres, travaille sur des tableaux de données.

Intérêt de l'algèbre relationnel par rapport au SQL :

- Indépendance par rapport à tout SGBD
- Formalisme mathématique plus concis
- Un opérateur en plus : celui de division

Intérêt du SQL par rapport à l'algèbre relationnel

Le SQL permet de créer les BD et de tester les calculs.

Le SQL est un langage normalisé (ANSI - ISO) et implanté dans tous les SGBD-R

Le formalisme du SQL est très proche de celui de l'algèbre relationnel.

L'opérateur de division est un opérateur complexe et rarement utilisé et équivalent à la combinaison d'autres opérateurs du SQL.

Les commandes du SQL

Les commandes du SQL se divisent en 4 sous ensembles :

Le DSL : data select language (formulation non standard)

Le DML : data manipulation language

Le DDL : data definition language

Le DCL : data control language

Le DSL : SELECT : l'algèbre relationnelle

Le SELECT est la commande qui permet de consulter les données de la BD.

C'est le SELECT qui va mettre en oeuvre l'algèbre relationnelle.

Le DML : commandes des tuples : INSERT, UPDATE, DELETE

Ce sont les commandes qui vont permettre de créer, modifier, détruire les tuples.

Le DDL : commandes des tables : CREATE, ALTER, DROP

Ce sont les opérateurs qui vont permettre de créer, modifier, détruire les tables.

Ces commandes permettront aussi de créer, modifier et supprimer d'autres objets de la BD : les séquences, les indexs, les vues, etc.

Le DCL : commandes de contrôle

La commande CREATE USER permet de créer des utilisateurs qui pourront se connecter à la base de données. DROP USER et ALTER USER permettront la suppression et la modification.

La commande GRANT permet de donner des droits aux utilisateurs : droits de création, modification, suppression de tables, et droits de création, modification, suppression et consultation de tuples. Cette commande permet aussi de créer des utilisateurs.

La commande REVOKE permet de supprimer les droits créés par la commande GRANT.

La commande COMMIT permet de valider les modifications dans la BD (les commandes du DML). Selon les environnements, les modifications peuvent être validées automatiquement par défaut ou pas (variable d'environnement AUTOCOMMIT à vrai ou à faux).

La commande ROLLBACK permet au contraire de revenir en arrière, s'il n'y a pas eu de validation manuelle ou automatique.

Manuels de référence du SQL

Chaque SGBD propose son manuel de référence en ligne sur internet.

ORACLE

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/toc.htm

MySQL

<http://dev.mysql.com/doc/refman/5.0/fr/index.html>

PostgreSQL

<http://www.postgresql.org/docs/9.0/static/index.html>

SQL Server

<http://technet.microsoft.com/fr-fr/library/ms173372%28SQL.90%29.aspx>

2. La commande de recherche : le select

La commande SELECT permet de faire des opérations de recherche et de calcul à partir des tables de la base de données.

On peut diviser toutes les opérations réalisable par un select en deux grandes catégories :

Les opérations s'appliquant à une seule table

Les opérations s'appliquant à plusieurs tables

Les opérations s'appliquant à une seule table

Il y a 4 grandes catégories d'opérations s'appliquant à une seule table :

- Les filtres sur les colonnes : projection
- Les filtres sur les lignes : restriction
- Les tris
- Les opérations statistiques

Exemples de questions traitées par le SELECT :

- Quel est le nom de tous les employés qui sont vendeurs ? (filtre des lignes et des colonnes).
- Quel est le nom de tous les employés travaillant dans tel département ? (filtre des lignes et des colonnes à partir de deux tables).
- Donner la liste des employés par ordre alphabétique (tri).
- Quel est le salaire moyen de tel métier de l'entreprise ? (filtre et calcul statistique : le résultat est une table avec un tuple et un attribut).
- Quels sont les employés qui gagnent plus que la moyenne des salaires de l'entreprise ? (calcul statistique avec regroupements).
- Combien y a-t-il d'employés par tranche de 1000 de salaire ?

Les opérations s'appliquant à plusieurs tables

Il y a 3 grandes catégories d'opérations s'appliquant à plusieurs tables :

- Les opérations ensemblistes classiques : union, intersection, différence.
- Le produit cartésien et la jointure associée
- Les imbrications d'opérations.

ORACLE

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/queries001.htm#i2053893

MySQL

<http://dev.mysql.com/doc/refman/5.0/fr/sql-syntax.html>

PostgreSQL

<http://www.postgresql.org/docs>

Dans un moteur de recherche : chercher postgresql docs et le ou les mots clés

SQL Server

<http://technet.microsoft.com/fr-fr/library/ms174113%28SQL.90%29.aspx>

3. Projection = filtre des colonnes

Projection = filtre des colonnes : tous les tuples, certains attributs

Présentation

La projection d'une table est une nouvelle table constituée de tous les tuples et de certains attributs de la table de départ.

TABLE	Attribut 1	attribut 2	attribut 3	...	attribut n
tuple 1					
tuple 2					
tuple n					

En gris : les colonnes sélectionnées.

Syntaxe AR

$Tres = Proj(table ; liste\ d'attributs)$

Syntaxe SQL

La syntaxe d'une projection est la suivante¹ :

Select liste d'attributs ² **from** table ;

Exemples

➤ *Donner les noms de tous les employés :*

Select NE, nom **from** emp;

Remarque 1 :

En toute rigueur, il est nécessaire de projeter NE même s'il n'est pas explicitement demandé dans la question, car la table produite ne doit pas contenir de tuples en double. Pour distinguer les homonymes, on projette la clé primaire de la relation.

➤ *Donner la liste de tous les employés avec tous leurs attributs :*

Select * **from** emp;

1 Remarques sur le métalangage utilisé : il ne prétend pas être parfaitement formel! Son objectif est d'associer pédagogie et rigueur formelles. Les mots clés du langage SQL sont en gras (**Select**). Les expressions générales sont en italiques (*liste d'attributs*). Les explications concernant ces expressions générales sont données soit en note, soit dans le texte, soit à travers des exemples. Les cas particuliers des exemples sont au format standard (NE, nom).

2 Les attributs de la liste d'attributs sont séparés par une virgule.

Deux types de projection

Quand on crée une nouvelle table, en toute rigueur, il faut éviter qu'il y ait des tuples en double.

Il y a deux manières d'éviter les doublons :

- Projeter la clé primaire (projection primaire)
- Eliminer les doublons (projection avec distinct)

a) Projection primaire

La syntaxe est la suivante :

```
Select clé primaire, liste d'attributs from table ;
```

La clé primaire de la table résultat est la clé primaire de la table de départ.

Exemple : tous les employés avec leurs fonctions.

```
Select NE, nom, fonction  
from emp;
```

Le résultat est une table d'employés avec moins d'attributs.

Remarque :

Puisqu'on veut les employés, il faut aussi projeter l'attribut donnant le nom (nom), c'est-à-dire la **clé significative**.

b) Projection avec élimination des doublons : la clause distinct

La clause **distinct** permet, à partir d'une projection, d'éliminer les tuples en doubles

```
Select distinct liste d'attributs from table ;
```

Exemples : pour obtenir les différents métiers de la société, on écrira :

```
Select distinct fonction from emp ;
```

Pour obtenir les différents métiers de la société par numéro de département, on écrira :

```
Select distinct ND, fonction from emp ;
```

Remarque 1 :

La clé primaire de la table résultat est constituée par la liste des attributs projetés. Les tuples produits sont donc conceptuellement différents des tuples présents dans la table d'origine. Dans l'exemple, on produit des métiers (« fonction ») en partant d'employés.

Remarque 2 :

On ne doit jamais projeter la clé primaire auquel cas on retomberais sur la table d'origine.

Inversement, si la liste d'attributs projetés contient la clé primaire, alors le distinct ne sert à rien :

```
Select distinct clé primaire, liste d'attributs  
from table ;
```

équivalent à :

```
Select clé primaire, liste d'attributs from table ;
```

Remarque 3:

La table est classée dans un ordre croissant (cf. clause **order by** à venir). Ceci vient du fait que l'ordre initial n'a plus aucun sens dans le résultat d'un distinct.

Création d'attributs calculés

Présentation

Il est possible de créer des attributs qui soient le résultat d'une opération arithmétique ou autre faite à partir d'autres attributs.

```
Select liste d'attributs avec des opérations sur attributs  
from table ;
```

Les opérations possibles

➤ *Opérateurs et fonctions classiques*

Opérateurs arithmétiques, opérateurs de comparaison, opérateurs de traitements de chaînes, de traitements de date, etc.

Cf. le paragraphe sur les opérateurs et les fonctions dans ce document.

➤ *Select*

On peut imbriquer un select dans les attributs projetés à condition que ce select ne renvoie qu'une seule valeur.

➤ *CASE WHEN*

Le "case when" est un cas particulier d'attribut calculé. Il est assez standard.

Le principe est de faire un test avec plusieurs embranchements possibles. Le passage par un embranchement permet de renvoyer une valeur et une seule : celle de l'attribut calculé.

En général, il a deux usages et deux syntaxes :

usage « switch » classique (comparaison d'égalité entre une expression unique de départ)

usage « else-if » classique (succession de tests indépendants)

Dans tous les cas, le CASE WHEN renvoie une seule valeur : c'est celle qui sera prise en compte dans la projection.

En général le ELSE final ou un DEFAULT permet de garantir que tous les cas possibles seront traités.

➤ *IF*

Le IF est un cas particulier du CASE qui réduit les valeurs possibles à 2 :

IF(expr1,expr2,expr3) : retourne expr2 si expr1 est différent de 0 et de null, expr3 sinon.

➤ *Autres possibilités*

Selon les SGBD-R, on peut trouver d'autres possibilités.

ORACLE propose la clause DECODE qui est à peu près équivalente à un CASE WHEN, et aussi la clause WIDTH_BUCKET qui permet de créer un attribut catégoriel à partir d'un attribut continu.

MySQL propose la fonction « if » qui permet de coder l'équivalent d'un CASE.

Exemples

tous les salaires, commissions et salaires totaux des salariés :

```
Select NE, nom, sal, comm, sal + comm  
from emp;
```

nombre de lettres du nom de chaque employé :

```
Select NE, nom, length(nom)  
from emp;
```

année d'embauche de chaque employé

```
Select NE, nom, year(datemb)  
from emp;
```

Salaires par tranche : petit (<1000), moyen (<2000), gros : autres.

```
Select NE, nom, case when sal < 1000 then "petit"  
When sal < 2000 then "moyen"  
Else "gros"  
from emp;
```

Changer les noms des attributs

Présentation

On peut renommer les attributs projetés : le nouveau nom s'affichera à la place du nom de l'attribut. Ce nouveau nom n'est valable que le temps de la requête.

```
Select ancien_nom nouveau_nom from table;
```

Si le nouveau nom contient des espaces, on écrira :

```
Select ancien_nom "nouveau nom" from table;
```

Remarques

On peut mettre des apostrophes à la place des guillemets

On peut mettre « as » entre l'ancien nom et le nouveau nom

```
Select ancien_nom as nouveau_nom from table;
```

4. Restriction = filtre des lignes

Restriction = filtre des lignes : certains tuples, tous les attributs

Présentation

La restriction d'une table est une nouvelle table constituée de certains tuples et de tous les attributs de la table de départ.

TABLE	attribut 1	attribut 2	Attribut 3	...	attribut n
tuple 1					
tuple 2					
tuple n					

En gris : les lignes sélectionnées.

Syntaxe AR

Tres = Rest(*table* ; *formule logique de sélection des tuples*)

Syntaxe SQL

La syntaxe d'une restriction est la suivante :

```
Select * from table
where formule logique de sélection des tuples ;
```

La formule de sélection des tuples fait intervenir les opérateurs habituels de l'algèbre booléenne:

=, !=, >, >=, <, <=, or, and, not

Cf. paragraphe 3.6 sur les opérations sur les opérateurs de comparaison

Exemples

Tous les employés du département 30 avec tous leurs attributs :

```
Select * from emp
where ND = 30;
```

Il existe aussi trois opérateurs spécifiques au SQL :

in, between, like.

in est vrai si le terme de gauche se trouve dans la liste proposée par le terme de droite.

between est vrai si le terme de gauche se trouve entre deux valeurs proposées.

like est vrai si la chaîne de caractères du terme de gauche est au format proposé dans le terme de droite.

Nous verrons la syntaxe de ces opérateurs dans les exemples du paragraphe suivant.

5. Restriction et projection = filtre des lignes et des colonnes

Restriction- projection = filtre des lignes et des colonnes : certains tuples, certains attributs

Présentation

La restriction-projection d'une table est une nouvelle table constituée de certains tuples et de certains attributs de la table de départ.

On fait d'abord la restriction, puis on projette.

C'est l'opération la plus courante.

TABLE	attribut 1	attribut 2	Attribut 3	...	attribut n
tupl 1					
tupl 2					
tupl n					

En gris clair : les lignes et les colonnes sélectionnées.

En gris foncé : le croisement des lignes et des colonnes sélectionnés.

2 types de restriction-projection

Quand on crée une nouvelle table, il faut éviter qu'il y ait des tuples en double. Il y a deux manières d'éviter les doubles qui correspondent à trois types de restriction-projection :

- la restriction-projection primaire (avec projection de la clé primaire)
- la restriction-projection avec élimination des tuples en doubles

Syntaxe AR

$T1 = \text{Rest}(\text{table} ; \text{formule logique de sélection des tuples})$

$Tres = \text{Proj} (T1 ; \text{liste d'attributs})$

Les deux opérations sont faites l'une après l'autre : d'abord la restriction, puis la projection.

On peut aussi écrire :

$Tres = \text{Proj} (\text{Rest}(\text{table} ; \text{formule logique de sélection des tuples}) ; \text{liste d'attributs})$

Cependant, on évitera cette écriture qui est peu lisible.

Restriction-projection primaire

Présentation

La syntaxe restriction-projection primaire est :

```
Select clé primaire, liste d'attributs from table  
where formule logique de sélection des tuples ;
```

La restriction-projection avec clé projette la clé de la table de départ. Conceptuellement, elle produit donc des tuples qui sont des objets restreints (ayant moins d'attributs) et appartenant à la table de départ.

Exemple

tous les employés dont le salaire est compris entre 1200 et 1400

```
Select NE, nom, sal from emp  
where sal between 1200 and 1400 ;
```

Le résultat est une table d'employés avec moins d'attributs.

Attributs projetés

La liste des attributs projetés est la suivante, dans l'ordre

CP – (CS) – demandés – (restriction)

Les attributs entre parenthèses sont facultatifs mais fortement recommandés.

- 1) CP : on projette la clé primaire en premier pour savoir de quoi-qu'on parle et pour éviter les doublons.
- 2) CS : on projette la « clé significative », CS en second. La CS est l'attribut qui fait office de clé primaire dans le langage courant. Projeter la CS n'est pas obligatoire d'un point de vue mathématique, par contre c'est nécessaire pour rendre les résultats lisibles. Dans nos exemples, la CS, c'est le nom.
- 3) Attributs demandés : on projette obligatoirement les attributs demandés.
- 4) Attributs de restriction : on projette les attributs qui participent aux restrictions pour vérifier que les conditions sont bien réalisées. Département, salaire et job dans nos exemples.

Syntaxe générale SQL d'une restriction-projection primaire

La syntaxe générale d'une restriction-projection primaire est donc :

```
Select clé primaire, clé significative, attributs  
demandés, attributs de restriction  
from table  
where formule logique de sélection des tuples ;
```

Restriction-projection avec élimination des tuples en double : la clause DISTINCT

Syntaxe SQL

La clause **distinct** permet, à partir d'une projection, d'éliminer les tuples en doubles

```
Select distinct liste d'attributs from table ;
```

Exemple

Pour obtenir les différents métiers de la société, on écrira :

```
Select distinct job from emp ;
```

Pour obtenir les différents métiers de la société par numéro de département, on écrira :

```
Select distinct ND, job from emp ;
```

Remarques

1) A la différence d'une restriction-projection primaire, dans le cas de l'utilisation d'un distinct, on ne veut pas - et on ne doit pas - projeter la clé primaire. En effet, si on projette la clé primaire, les tuples produits seront des objets restreints de la table de départ, tandis qu'avec la clause distinct on produit des objets qui, conceptuellement, n'appartiennent pas à la table de départ. Les métiers ne sont pas des employés, les couples (numéro de département, métier) non plus.

2) La table est classée dans l'ordre d'apparition de la première occurrence des valeurs projetées.

3) Dans ce cas, à la différence d'une restriction-projection primaire, les tuples obtenus ne correspondent pas à un et un seul tuple de la table de départ.

4) Si la liste d'attributs projetés contient la clé primaire, alors le distinct ne sert à rien :

```
Select distinct clé primaire, liste d'attributs  
from table ;
```

équivalent à :

```
Select clé primaire, liste d'attributs from table ;
```

Syntaxe AR

Le distinct n'existe pas en algèbre relationnelle car il est considéré comme étant effectué systématiquement : une table résultat ne contient jamais de doublons, de toute façon.

Syntaxe générale SQL d'une restriction-projection avec distinct

La syntaxe générale d'une restriction-projection avec distinct est la suivante :

```
Select distinct clé primaire, clé significative,  
attributs demandés, attributs de restriction  
from table  
where formule logique de sélection des tuples ;
```

C'est la même que pour une restriction-projection primaire.

6. Opérateurs et fonctions

Présentation

Principes

Il existe de nombreuses fonctions et de nombreux opérateurs fournis par le SQL standard mais aussi des fonctions et des opérateurs spécifiques à chaque SGBD. Ces outils permettent de faire des calculs puissants de façon simple : il faut s'y intéresser en fonction de du SGBD qu'on utilise.

La présentation ci-dessous fixe les principales catégories de fonctions et d'opérateurs. Toutefois, pour les découvrir de façon plus exhaustive, il faut se référer au manuel de référence du SGBD accessible sur internet.

Manuels de référence

➤ **Oracle**

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/operators.htm#SQLRF003

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions.htm#SQLRF006

➤ **MySQL**

<http://dev.mysql.com/doc/refman/5.0/fr/functions.html>

➤ **SQL Server**

➤ **PostgreSQL**

<http://www.postgresql.org/docs>

Dans un moteur de recherche : chercher postgresql docs et le ou les mots clés

Les opérations arithmétiques : la calculatrice arithmétique

Présentation

On peut faire des opérations arithmétiques en utilisant les opérateurs arithmétiques habituels :

$+$, $-$, $*$, $/$, $\%$, $($, $)$

$\text{round}()$, $\text{floor}()$, $\text{ceil}()$

Mais aussi toutes les fonctions mathématiques standards :

\sin , \cos , \log , \exp , power , sqrt , etc.

La calculette arithmétique

A noter que la calculette SQL est aussi une calculette arithmétique.

On peut écrire :

```
select 4*log(2,8) ;
```

On obtiendra le résultat du calcul : 12.

Les opérateurs de comparaison

Présentation

On peut faire des comparaisons en utilisant les opérateurs de comparaisons habituels :

'=', '<>', '!=', '<', '<=', '>', '>='

On peut aussi utiliser les opérateurs booléens

'AND', 'OR', 'NOT'

On peut aussi utiliser les opérateurs spéciaux :

in, not in

between ... and...

like

in est vrai si le terme de gauche se trouve dans la liste proposée par le terme de droite.

between est vrai si le terme de gauche se trouve entre deux valeurs proposées.

like est vrai si la chaîne de caractères du terme de gauche est au format proposé dans le terme de droite.

Exemples

tous les employés dont le salaire est compris entre 1200 et 1400

```
select NE, nom, sal from emp
where sal between 1200 and 1400 ;
```

Tous les employés travaillant dans les départements 10 ou 30 :

```
select NE, nom, ND from emp
where ND in (10, 30) ;
```

Tous les employés qui ont un E comme troisième lettre de leurs noms :

```
select NE, nom from emp
where nom like ' _ E %' ;
```

le "_" signifie : n'importe quel caractère. Le "%" signifie n'importe quelle chaîne de caractères, dont la chaîne vide.

Attention à l'ordre de priorité des opérateurs !!!

L'ordre de priorité des opérateurs est l'ordre habituel de l'arithmétique

not, *, +, <, and, or

Les opérations sur les valeurs NULL

Présentation

Quand une valeur NULL intervient dans une opération, quelle qu'elle soit, le résultat de cette opération vaut NULL

Opérateurs

Pour savoir si une valeur vaut NULL, on utilise les opérateurs :

is
is not

Remplacer la valeur NULL par 0

On peut remplacer la valeur NULL par une valeur au choix (0 le plus souvent) avec la fonction suivante :

ifnull (attribut, 0)

L'intérêt de cette transformation est de forcer les opérations malgré la présence d'une valeur NULL.

➤ *Equivalence dans les autres SGBD*

av1 (oracle), **isnul** (sqlserveur), **ifnull** (mysql), **coalesce** (postgres)

Les opérateurs de traitement des chaînes de caractères

like

like est vrai si la chaîne de caractères du terme de gauche est au format proposé dans le terme de droite.

Tous les employés qui ont un E comme troisième lettre de leurs noms :

```
Select NE, nom from emp
where nom like '___E %' ;
```

le "_" signifie : n'importe quel caractère. Le "%" signifie n'importe quelle chaîne de caractères, dont la chaîne vide.

Les trois fonctions de base du traitement de chaînes de caractères

length fournit la longueur d'un attribut chaîne de caractères :

```
Select length (attribut ) from table ;
```

substr fournit un morceau d'un attribut chaîne de caractères :

```
Select substr (attribut, début, longueur ) from table ;
```

Exemple : substr('bonjour', 2, 4) vaut 'onjo'

concat permet de concaténer deux chaînes de caractères (constante ou attribut) :

```
Select concat (chaine1 , chaine2)
```

Ou

```
Select chaine1 || chaine2 ;
```

Avec ces trois fonctions, on peut faire tous les traitements possibles sur les chaînes (sauf les traitements liés à la distinction entre minuscule et majuscule).

Autres fonctions de traitement de chaînes

upper() : passe la chaîne en majuscule.

lower() : passe la chaîne en minuscule.

initcap() : met la première lettre en majuscule, le reste en minuscule, de tous les mots de la chaîne.

Expression régulière : REGEXP LIKE

De nombreux SGBD propose un opérateur d'analyse d'expressions régulières.

ORACLE propose le REGEXP_LIKE, qui, comme son nom l'indique, est un « LIKE » dont les caractères spéciaux ne se limiteront pas aux « % » et « _ » mais à ceux qu'on trouve dans les expressions régulières standards.

Des fonctions supplémentaires de manipulation des expressions (comptage, remplacement, etc.) existent et sont spécifiques à chaque SGBD.

Les opérateurs de traitement de date

Présentation

La gestion des dates est la partie la moins standard entre les différents SGBD.

Il y a deux aspects à prendre en compte :

Les fonctions de manipulation des dates

La question des formats

La présentation ci-dessous présente quelques éléments dans différents SGBD

Les dates dans Oracle

➤ ***Dates et heures système***

```
Select current_date from dual ; // date client  
Select localtimestamp ; // date et heure client  
Select systimestamp ; // date et heure serveur
```

➤ ***Extraction d'une partie de la date ou de l'heure***

Extract (*format from expression*)

Formats de base : year, month, day, hour, minute, second.

Exemple :

```
Select extract (year from current date) from dual ;
```

➤ ***Les opérateurs de traitement de chaîne***

Les dates sont aussi considérées comme des chaînes de caractères. On peut donc leur appliquer tous les opérateurs de traitement de chaînes.

➤ ***Conversion d'une chaîne en date : to_date***

To_date (*chaîne, format*)

La fonction permet de transformer une chaîne de caractères en date selon le format proposé.

Le format utilise les parties : YYYY, MM, DD

Il existe plusieurs formats de date :

mm/dd/yyyy

dd.mm.yyyy

yyyy-mm-dd

Ces formats permettent de gérer les conversions de chaînes de caractères en date.

Plusieurs formats d'écriture des dates sont acceptés :

15 mars 2005

15 mar 2005

15/3/2005

15-3-2005

15-MAR-2005

Exemple :

```
Select to_date(current_date, 'DD-MM-YY') from dual ;
```

Current_date est ici considéré comme une chaîne

➤ **Conversion d'une date en texte : to_char**

To_char (*date, format*)

La fonction permet de transformer un attribut date en une chaîne de caractère selon le format précisé. On pourra à cette occasion n'extraire, par exemple, que l'année.

Exemple :

```
Select to_char(current_date, 'DD-YYYY-MM') from dual ;
```

Current_date est ici considéré comme une date

➤ **Manuel de référence**

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/expressions007.htm#i1047500

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#i88893

Les dates dans MySQL

➤ **Principales fonctions de traitement de dates**

year(), month(), day() : renvoient l'année, le mois et le jour dans le mois.

current_date : renvoie la date du jour

to_days(date) : renvoie le nombre de jours depuis le 1^{er} janvier 0. to_days(« 0-1-1 ») vaut 1.

From_days(nb jours) : renvoie la durée correspondant à un nombre de jours. Cette durée a le format date.

➤ **Les opérateurs de traitement de chaîne**

Les dates sont aussi considérées comme des chaînes de caractères. On peut donc leur appliquer tous les opérateurs de traitement de chaînes.

➤ ***Manuel de référence***

<http://dev.mysql.com/doc/refman/5.0/fr/date-calculations.html>

<http://dev.mysql.com/doc/refman/5.0/fr/date-and-time-functions.html>

Les dates dans PostgreSQL

➤ ***Principales fonctions de traitement de dates***

now() : renvoie la date et heure du moment

current_date : date du jour

current_time : heure du moment

extract (type from valeur) : le type peut être : century, year, month, etc.

exemple : select extract(century from now()) ;

➤ ***Les opérateurs de traitement de chaîne***

Les dates sont aussi considérées comme des chaînes de caractères. On peut donc leur appliquer tous les opérateurs de traitement de chaînes.

➤ ***Manuel de référence***

<http://www.postgresql.org/docs>

Dans un moteur de recherche, chercher : postgresql docs extract

Le transtypage : conversion de type

Présentation

Les résultats produits par les différentes opérations sont typés, selon le typage classique : entier, réel, chaîne de caractères, etc.

Les types seront abordés plus précisément dans le chapitre sur le DDL.

Toutefois, on a la possibilité de changer le type d'un résultat ou d'une variable : c'est le transtypage.

Bien sur le transtypage n'est possible que s'il est logique : on ne peut pas transformer 'bonjour' en entier !

Les fonctions ou opérateurs de transtypage sont variables selon les SGBD.

Exemple MySQL

<http://dev.mysql.com/doc/refman/5.0/fr/cast-functions.html>


➤ ***cast***

Select cast('19.4' as signed); // 19

Select cast(4/3 as decimal(2,1)); // 1.3

➤ ***convert***

```
Select convert ( 4/3, decimal(3,2) ); // 1.33  
Select convert ( 4/3, decimal(2,2) ); // 0.99 !!! ATTENTION !!!  
Select convert ( 4/3, decimal(1,2) ); // ERREUR !!!
```



7. Tris et limites

Présentation

Présentation

Les tuples d'une table sont présentés dans n'importe quel ordre. On peut choisir de les trier selon les valeurs de certains attributs avec la clause **order by**.

Syntaxe AR

Tres = Tri(*table ; liste d'attributs*)

Syntaxe SQL

La syntaxe du tri est la suivante :

```
Select liste_1 d'attributs from table
order by liste_2 d'attributs ;
```

Dans la liste 2 d'attributs triés, on peut préciser l'ordre du tri, ascendant ou descendant, par les mots-clés **asc** et **desc** qu'on fait suivre chaque attribut de tri. Par défaut, le tri est ascendant (il n'est donc pas nécessaire de préciser **asc**).

Attributs projetés

Pour que le résultat affiché soit compréhensible, on utilisera la forme syntaxique suivante :

```
table      Select liste_1 d'attributs , liste_2 d'attributs from
            order by liste_1 d'attributs ;
```

La liste des attributs projetés est donc la suivante, dans l'ordre

Tri - CP – (CS) – Demandés – (Restriction)

Ou bien :

CP – (CS) – Demandés – (Restriction) – Tri

Tri : attributs de tri : on les projette en premier pour rendre lisible le résultat. Ce choix est discutable : on peut aussi les mettre à la fin, ou juste après CP et CS.

Exemple

tous les employés classés par jobs, salaires décroissants et ordre alphabétique de noms :

```
Select job, sal, nom, NE from emp
order by job asc, sal desc, nom;
```

Remarque

On voit sur cet exemple que cet ordre d'affichage n'est pas forcément le meilleur : ici on préférerait projeter soit : job, sal, NE, nom; soit : NE, nom, job, sal.

Tri aléatoire : order by rand()

Principe

On peut souhaiter sortir les résultats dans n'importe quel ordre

Exemple

Tous les employés avec tous leurs attributs classés aléatoirement :

➤ *Solution MySQL :*

```
Select * from emp  
order by rand();
```

LIMIT et ROWNUM: limiter le nombre de tuples du résultat

Présentation

Les SGBD offrent différentes solutions permettant de limiter le nombre de tuples projetés dans un select.

Limiter la projection aux 5 premiers éléments :

➤ *Solution MySQL et PostgreSQL :*

```
Select NE, nom, sal from emp  
limit 5;
```

➤ *Solution Oracle :*

```
Select NE, nom, sal from emp  
Where rownum < 5;
```

Récupération du minimum ou du maximum

La technique consiste à récupérer le minimum ou le maximum en triant et en ne gardant que le premier.

➤ *Solution MySQL et PostgreSQL :*

Exemple : quel est l'employé qui a le plus haut salaire :

```
Select NE, nom, sal from emp  
order by sal desc  
limit 1 ;
```

A noter que cette technique n'est pas très pertinente puisqu'elle ne permet pas d'afficher les ex aequo.

On verra une solution avec les fonctions de groupe et les requêtes imbriquées. La solution Oracle est plus complexe à mettre en œuvre avec le rownum.

Récupération des 10 suivants

➤ *Solution MySQL :*

```
Select job, sal, nom, NE from emp
order by job asc, sal desc, nom
limit 10, 5;
```

Ici, on commence au 11^{ème} et on va jusqu'au 15^{ème}.

Autre solution :

```
Select job, sal, nom, NE from emp
order by job asc, sal desc, nom
limit 5 offset 10;
```

Sélection d'un élément au hasard : order by rand() et limit

Exemple : un employé au hasard avec tous ses attributs :

➤ *Solution MySQL :*

```
Select * from emp
order by rand() limit 1;
```

8. Calculs statistiques élémentaires : les fonctions de groupe

Présentation

Principes

Les fonctions de groupe permettent de faire des calculs statistiques sur l'ensemble des tuples projetés.

Une fonction de groupe est une fonction qui s'applique non pas à la valeur d'un attribut pour un tuple (comme la fonction sinus ou la fonction logarithme), mais à toutes les valeurs d'un attribut pour tous les tuples de la table traitée, donc à toute la colonne pour un attribut donné.

Par exemple, `max()` est une fonction de groupe qui détermine le maximum des valeurs d'un attribut pour tous les tuples de la table traitée.

Une fonction de groupe produit donc un tuple et un seul comme résultat.

Elle permet donc de créer un nouvel attribut à partir d'un attribut en entrée et de toutes les valeurs de cet attribut pour les tuples projetés.

On peut projeter plusieurs fonctions de groupe (calculer le max, le min, la moyenne).

Par contre, la projection d'une fonction de groupe empêche la simple projection d'un autre attribut de la table.

Les 5 fonctions de groupe standards

Il existe 5 fonctions de groupe (notées *fdg*³ dans cet exposé) et leur syntaxe est la suivante :

`count()`, `max()`, `min()`, `sum()`, `avg()`

Syntaxe SQL

La syntaxe d'une projection de fonction de groupe est la suivante⁴ :

```
select fdg( attribut ) from table ;
```

Interdiction syntaxique (et sémantique)

Quand on projette une fonction de groupe, on ne peut plus projeter d'attribut simple.

On ne peut pas écrire :

```
select fdg(att1), att2 from table ;
```

Ici, la projection de l'attribut « att2 » n'a pas de sens.

3 *fdg* (fonction de groupe) n'est pas un mot clé du SQL mais une expression de notre métalangage.

4 Remarques sur le métalangage utilisé : il ne prétend pas être parfaitement formel! Son objectif est d'associer pédagogie et rigueur formelle. Les mots clés du langage SQL sont en gras (**Select**). Les expressions générales sont en italiques (*liste d'attributs*). Les explications concernant ces expressions générales sont données soit en note, soit dans le texte, soit à travers des exemples. Les cas particuliers des exemples sont au format standard (NE, nom).

La fonction count ()

Présentation

count permet de compter le nombre de tuples renseignés (non NULL) d'une table pour le ou les attributs spécifiés.

Exemple

pour savoir combien il y a de salariés dans la société, on écrira :

```
Select count (*) from emp ;
```

➤ *Remarque 1*

On peut aussi écrire :

```
Select count (NE) from emp ;
```

NE étant la clé primaire, NE ne peut pas être NULL et le résultat est le même qu'avec count(*).
Quand on veut compter toutes les lignes de la table, mieux vaut utiliser count(*).

➤ *Remarque 2 : interdiction syntaxique (et sémantique)*

Il est impossible d'écrire

```
Select count (*), NE from emp ;
```

Une fois le nombre d'employés projeté, on ne peut plus projeter un numéro d'employé.

Usages spéciaux

➤ *Compter les valeurs non null*

Combien il y a personnes qui sont susceptibles d'avoir une commission ?

```
Select count (comm) from emp ;
```

➤ *Compter les valeurs distinctes*

Combien y a-t-il de fonctions différentes dans la société ?

```
Select count (distinct fonction) from emp ;
```

Les fonctions avg, sum, max et min

Présentation

avg, sum, max et min fournissent respectivement la moyenne, la somme, le maximum et le minimum d'un attribut donné :

Exemples

Quel est le salaire moyen de la société :

```
Select avg(sal) from emp ;
```

Quelle est la somme des salaires de la société :

```
Select sum (sal) from emp ;
```

Quels sont les salaires minimums et maximums de la société :

```
Select min (sal), max(sal) from emp ;
```

Interdiction syntaxique (et sémantique)

Il est impossible d'écrire

```
Select nd, avg(sal) from emp ;
```

Une fois la moyenne des salaires projetée, on ne peut plus projeter un numéro de département. On pourra faire ça avec un « group by ».

Autres fonctions de groupe

Selon les SGBD, on peut trouver d'autres fonctions de groupe proposant des calculs statistiques plus élaborés : variance, médiane, etc.

9. Calculs statistiques : agrégats ou group by

Présentation

Principe

La clause group by permet de faire des regroupements par valeur possible d'un attribut et de faire des statistiques sur les regroupements ainsi définis.

Exemple

On souhaite connaître, pour chaque fonction (fonction), le nombre d'employés et le salaire moyen :

```
Select fonction, count(*), avg(sal)
from emp
group by fonction;
```

ou encore quels sont les salaires maximums de chaque département :

```
Select ND, max(sal) from emp
group by ND ;
```

On obtient plusieurs tuples avec une fonction de groupe. En effet, la clause group by permet d'appliquer des fonctions de groupes à des sous ensembles de la table de départ.

Syntaxe SQL

En pratique, la syntaxe usuelle de la clause group by est la suivante :

```
Select liste d'attributs , liste de fdg(attribut)
from table
group by liste d'attributs ;
```

En général, la liste de tri et de restriction et la même que la liste de projection : la liste 1 est identique à la liste 2.

En théorie, on peut avoir :

```
Select liste 1 d'attributs , liste de fdg(attribut)
from table
group by liste 2 d'attributs ;
```

avec liste 1 incluse dans liste 2.

Détail du déroulement d'un group by

Le group by se divise en 4 étapes :

- 1) d'abord un "order by" : les tuples de la table de départ sont triés selon les valeurs croissantes de la liste des attributs du group by ;
- 2) Ensuite les calculs statistiques des fonctions de groupe : ces calculs sont appliqués aux sous-ensemble ayant la même valeur pour la liste d'attribut du group by ;
- 3) Ensuite on projette les attributs du group by avec un distinct.
- 4) Pour chaque ligne de la nouvelle table, on peut mettre le résultat des opérations statistiques.

Clé primaire d'un group by

La clé primaire d'un groupe by, c'est la concaténation des attributs du group by.
Elle peut aussi être constituée d'une partie seulement des attributs du group by

Group by sans fonctions de groupe : à éviter !

Un group by sans fonctions de groupe c'est soit une restriction-projection primaire, soit une restriction-projection avec distinct à l'ordre by près.

Soit le group by suivant :

```
Select liste d'attributs from table
group by liste d'attributs
order by liste d'attributs;
```

Si la liste d'attributs ne contient pas la clé primaire, la clause group by est équivalente à la clause distinct associée à la clause order by :

```
Select distinct liste d'attributs from table ;
```

Si la liste contient la clé primaire, alors il n'y a pas de restriction (pas de distinct) et la clause group by est équivalente à la clause order by :

```
Select liste d'attributs from table
order by liste d'attributs;
```

Dans tous les cas, il faut éviter les group by sans fonctions de groupe.

Restrictions supplémentaires : la clause having

La clause **having** permet d'ajouter une condition sur les résultats des fonctions de groupe associées à une clause group by:

```
Select liste d'attributs , liste de fdg(attribut )
from table
group by liste d'attributs
having formule logique de sélection des fdg(attribut );
```

Quelles sont les fonctions (fonction) pour lesquelles travaillent plus de trois personnes:

```
Select fonction, count(*) from emp
group by fonction
having count(*) >=3;
```

ou encore:

```
Select fonction from emp
group by fonction
having count(*) >=3;
```

On préférera la première écriture car on préfère projeter les attributs de restriction.

Tri après une fonction de groupe

On peut aussi trier les résultats.

A noter que certains SGBD (MySQL par exemple) tri par défaut selon les attributs du group by.

La clause de tri est placée en dernier.

Exemple

```
select fonction, count(*) nb from emp
group by fonction
having nb >=3
order by nb desc;
```

Ordre de projection des attributs

Rappel : cas des requêtes sans fonction de groupe :

Tri - CP – (CS) – Demandés – (Restriction)

Ou

CP – (CS) – Demandés – (Restriction) - Tri

Le principe est le même :

En général : CP = CS = Attributs de tri = tous les attributs du group by.

Parfois : CP = une partie des attributs du group by. CS = une partie des attributs du group by et il peut rester des attributs du group by.

D'où l'ordre de projection suivant :

Tri - CP – (CS) – GB - Demandés – (Restriction=Having)

ONLY FULL GROUP BY de MySQL

Présentation

La présentation précédente décrit le fonctionnement standard du group by : on ne peut projeter que les attributs de regroupement et ceux des fonctions de groupe.

MySQL, dans sa version de base, permet de projeter n'importe quel autre attribut.

C'est pratique dans certain cas (requêtes avec jointure). Toutefois, ce n'est pas standard.

On évitera donc systématiquement cette pratique.

« sql_mode » et « ONLY FULL GROUP BY »

La variable « sql_mode » permet de paramétrer un « group by » standard.

Pour cela, elle doit prendre la valeur : ONLY_FULL_GROUP_BY

Par défaut, à l'installation, la variable « sql_mode » contient :

STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION

Soit 3 caractéristiques différentes.

Pour ajouter une quatrième caractéristique, on passe la commande suivante :

```
set
@@local.sql_mode=concat(@@sql_mode,',ONLY FULL GROUP BY');
```

Pour que cette caractéristique soit prise en compte par tout nouveau client, on passe la commande suivante :

```
set
@@global.sql_mode=concat(@@sql_mode,',ONLY FULL GROUP BY');
```

On peut aussi modifier la valeur du sql_mode dans le fichier de configuration : my.ini :

[mysqld]

sql_mode=

STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,
ONLY_FULL_GROUP_BY'

10. Calculs statistiques : statistiques et agrégations avancées

Présentation

Les SGBD-R offrent tous des fonctions statistiques supplémentaires et aussi des clause qui permettent de faire des regroupements avancés.

Count (distinct attribut) : universel

Count (distinct attribut) dans un group by

➤ *Exemple*

Combien d'employés et de départements par fonction

Select job, count(ne) nbEmp, count(distinct nd) nbDept from emp group by job;

job	nbEmp	nbDept
Ingé	5	3
Com	2	2
Etc.		

5 ingénieurs répartis dans 3 départements

2 commerciaux répartis dans 2 départements, etc.

MAX (SUM (SALAIRE)) : ORACLE

On peut appliquer une fonction de groupe sur les attributs de la table résultant d'un group by.

Valeur maximum des moyennes des salaires par département

```
SELECT MAX( AVG( sal)) FROM EMP GROUP BY deptno;
```

```
MAX (AVG (SAL) )
-----
      2387,5
```

Group concat : MySQL

http://dev.mysql.com/doc/refman/5.0/fr/group-by-functions.html#function_group-concat

Group concat : pour violer la forme normale 1 ! Pour une valeur du group by, on regroupe toutes les valeurs d'un attribut :

```
Select attributGB, group_concat(liste d'attributs)
from table
group by attributGB;
```

On peut trier et choisir son séparateur :

```
group_concat(attribut order by attribut separator ' et ')
```

➤ **Exemple**

Liste des employés par fonction :

```
Select job, group_concat(ne order by ne)
from emp
group by job;
```

Ingé	1, 4, 5, 10
Com	2, 7
Boss	3
Secrétaire	6, 8, 9

Fonction OVER() : ORACLE, SQL Server

La fonction over() permet d'ajouter le résultat d'une fonction de groupe à une table.

Tous les employés avec leurs salaire et la moyenne des salaires :

```
select empno, ename, sal, avg(sal) over()
from emp;
```

EMPNO	ENAME	SAL	AVG(SAL) OVER()
7839	KING	5000	1988,33333
7782	CLARK	2450	1988,33333
7934	MILLER	1300	1988,33333
7935	DUPONT	800	1988,33333
etc...			

C'est équivalent à un select dans le select :

```
select empno, ename, sal, deptno, (select avg(sal) from emp)
from emp
where deptno=10;
```

Ou à un select dans le from :

```
select empno, ename, sal, deptno, avgsal
from emp, (select avg(sal) avgsal from emp) t1
where deptno=10;
```

With Rollup : MySQL

<http://dev.mysql.com/doc/refman/5.0/fr/group-by-modifiers.html>

Le with rollup permet de faire les totaux par catégorie après un group by.

```
Select ...
Group by (au moins deux attributs) WITH ROLLUP ;
```

➤ **Exemple**

Les employés travaillent dans un département et ont une fonction. On veut connaître le nombre d'employés par fonction et par département avec le total par fonction.

```
Select job, nd, count(*) nb
from emp
```

```
group by job, nd with rollup;
```

JOB	ND	NB
Ingé	dept10	4
Ingé	dept20	2
Ingé	NULL	6
Com	dept10	1
Com	NULL	1

➤ **Equivalents ORACLE :**

GROUP BY ROLLUP, GROUP BY GROUPING SETS, GROUP BY CUBE

Ces clauses fonctionnent avec plus ou moins d'attributs et font plus ou moins de totaux intermédiaires.

PIVOT : ORACLE 11 G, SQL Server

Le PIVOT permet de produire des tableaux croisés avec deux critères de regroupement.

(Pour comprendre les tableaux croisés, utiliser la fonction tableaux croisés dynamiques d'Excel).

➤ **Exemple de résultat produit ORACLE :**

On veut un tableau de total des ventes par année et par pays

```
Select *  
(Select ANNEE, PAYS, MONTANT  
Pivot (SUM(MONTANT) for PAYS)  
Order by ANNEE ;
```

ANNEE	FRANCE	ALLEMAGNE	ANGLETERRE
2008	29825	26825	27825
2009	25825	22825	29825
2010	32825	29000	28825

11. Bilan syntaxique

Ordre des clauses du Select mono-table

```
SELECT [DISTINCT] liste d'attributs  
FROM table  
[ WHERE formule de restriction ]  
[ GROUP BY liste d'attributs  
[ HAVING formule de restriction ] ]  
[ ORDER BY liste d'attributs ]
```

A noter que:

Il n'y a qu'un seul WHERE : mais l'expression logique peut être complexe en incluant des AND, des OR, etc.

C'est la même chose pour le HAVING. Toutefois, le HAVING ne peut être présent que s'il y a un GROUP BY

Ordre raisonnable de projection des tuples

TRI – CP – [CS] – Demandés – [Restriction]

Ou

CP – [CS] – Demandés – [Restriction] - TRI

La clé significative (CS) est facultative dans l'absolu. Toutefois, la seule présence d'une clé primaire (un numéro le plus souvent) est rarement suffisante pour rendre le résultat exploitable.

Les attributs de restriction sont facultatifs. On ne les projette que pour vérifier si les restrictions ont bien été prises en compte.

Les attributs de tri peuvent être mis devant ou derrière indifféremment, ou juste après CP et CS. L'objectif est de rendre lisible le résultat.

Ces règles de projection ne sont pas absolues : toutefois, elles permettent dans la grande majorité des cas d'obtenir une bonne lisibilité de la table résultant du Select.

A noter que parfois, on ne souhaite pas projeter la CP : si l'ensemble des attributs projetés est suffisant pour identifier correctement chaque ligne, la CP n'est pas nécessaire.

12. Détails syntaxiques MySQL

CASE - WHEN

Première syntaxe : case classique

La première syntaxe permet de calculer le nouvel attribut à partir d'une **succession de comparaison d'égalité** entre une expression unique de départ et différentes valeur : c'est un « case » ou un « switch » algorithmique classique.

```
CASE expression
WHEN valeur THEN résultat1
[* WHEN ... THEN ...]
[ELSE résultat]
END
```

Valeur et résultat sont des expressions (constantes littérales, attributs, expressions arithmétiques, etc.)

➤ *Remarque*

Le ELSE permet de garantir que tous les tuples seront pris en compte. Si il n'y a pas de ELSE et que tous les cas ne sont pas pris en compte dans les alternatives du ELSE, le tuple ne sera pas éliminé (seul le WHERE peut faire ça), mais la valeur de l'attribut calculé vaudra NULL.

Deuxième syntaxe : else-if

La seconde syntaxe permet de calculer le nouvel attribut à partir d'une succession de tests distincts les uns des autres. C'est un « else if » algorithmique classique.

```
CASE WHEN expression opérateur valeur
THEN résultat
[* WHEN ... THEN ...]
[ELSE résultat]
END
```

Exemple

On veut classer les employés en 3 catégories de salaire : 1 :bas, 2 :moyen, 3 :fort. Un salaire moyen est compris entre 1200 et 2500.

```
Select NE, nom, case when sal <1200 then '1:bas' when sal <2500
then '2:moyen' else '3:gros' end as categorie
from emp order by categorie, nom;
```

IF

Le IF est un cas particulier du CASE qui réduit les valeurs possibles à 2 :

IF(expr1,expr2,expr3) : retourne expr2 si expr1 est différent de 0 et de null, expr3 sinon.

Exemple

On reprend l'exemple du CASE :

```
Select NE, nom, if(sal <1200, '1:bas', if(sal <2500, '2:moyen',  
'3:gros')) as categorie  
from emp order by categorie, nom;
```

Fonctions mathématiques

<http://dev.mysql.com/doc/refman/5.0/fr/mathematical-functions.html>

abs (nb), **ceil** (nb) : arrondi vers le haut ; **floor** (nb) : arrondi vers le bas ; **round** (nb) arrondi au plus proche ; **power** (nb, puissance) : élévation à la puissance ; **sqrt** (nb) : racine ; **truncate** (nb, nbDécimales) : arrondi en vers le bas en précisant le nombre de décimales conservées ; **pi** () : renvoie PI.

Fonctions de chaînes

<http://dev.mysql.com/doc/refman/5.0/fr/string-functions.html>

<http://dev.mysql.com/doc/refman/5.0/fr/string-comparison-functions.html>

Les constantes chaînes de caractères sont entre guillemets, apostrophes ou accents grave (altGr+7, espace).

Like, **length**(attribut) ; **substring**(attribut from déb for lgr) : sous-chaîne commençant en déb de longueur lgr ; **substring**(att from 11) : la sous-chaîne à partir du 11^{ème} caractère ; **strcmp**(texte1, texte2) : renvoie 0 si les deux textes sont identiques, -1 si le premier est premier en ordre alphabétique, 1 sinon ; **trim**(texte) : élimine les espaces au début et à la fin ; **upper**(texte) ; **lower**(texte) ; **replace**(texte, ancien, nouveau) ; **locate**(mot, texte) : position d'un mot dans un texte ; **locate**(mot, texte, début) : position d'un mot dans un texte à partir de début ; **concat**(text1, text2, etc.) ; **concat_ws**(séparateur, text1, text2, etc.) : concaténation avec un séparateur ; **binary** : opérateur unaire qui rend un texte sensible à la casse : 'paris' = binary 'PARIS' est faux.

Fonctions de date

<http://dev.mysql.com/doc/refman/5.0/fr/date-and-time-functions.html>

to_days(date), **from_days**(entier) : conversion d'une date en jours. **from_days** démarre à 366 : 1 janvier de l'année 1. **To_days** démarre au 1 janvier 0.

date_diff(date1, date2) : nb jours entre 2 dates ;

date_add(date, **interval** nb **type**) : ajout d'un nombre de mois, jours, heures, etc. (**type** : year, month, day, week, hour, minute, second, microsecond, etc.) ;

+ **interval** nb **type** : augmente ou diminue directement une date, ou date et heure. Ne marche pas pour les heures seules ; exemple : select now + interval -1 month ;

extract (**type from** date) : extraction d'une année, mois, jour, heure, etc. (**type** : year, month, etc. + year_month, hour_minute, etc.)

year(date) ; **month**(date) ; **day**(date) ; **hour**(date) ; **minute**(date) ; **second**(date) ;
curdate() : date ; **curtime**() : heure ; **now**() : date et heure ;
current_date : date ; **current_time** : heure ; **current_timestamp** : date et heure ;
utc_date : date ; **utc_time** : heure ; **utc_timestamp** : date et heure ; date et heure de greenwich
(« universel »)

formats d'affichage des dates

Exemple

`date_format(now(), '%W %d %M %H heures %i minutes %s sec.')`
Friday 27 February 16 heures 12 minutes 23 sec.

Formats

%k : heure sur 1 chiffre si possible ; %w : jour de 0 à 6, 0 pour dimanche ; %a : jour abrégé ;
%e : numéro du jour du mois sur 1 chiffre si possible ; %m : mois sur deux chiffres ; %c : mois
sur 1 chiffre si possible ; %Y : année sur 4 chiffres ; %y : année sur 2 chiffres.

Conversions de date

Date vers secondes et réciproquement

Nombre de secondes depuis le 1^{er} janvier 1970 : `unix_timestamp (date)`
Inverse : `from_unixtime(nb secondes)`
Remarque : `unix_timestamp ()` ⇔ `unix_timestamp (now())`

Heure vers seconde et réciproquement

Nombre de seconde depuis minuit : `time_to_sec (heure)`
Inverse : `sec_to_time (nb secondes)`

Conversion d'une date en nombre et réciproquement

`Curdate() + 0 = 20090220 ;`
`Date(20090220) = 20 février 2009`

Conversion d'une heure en nombre et réciproquement

`Curtime() + 0 = 161055 ;`
`Time(161055) = 16 heures 10 minutes et 55 secondes`

Regexp : expression régulière

<http://dev.mysql.com/doc/refman/5.0/fr/string-comparison-functions.html>

« Regexp » est un opérateur du même genre que « Like », mais en plus élaboré.

Syntaxe du select

WHERE attribut REGEXP 'expression régulière'

Syntaxe de l'expression régulière

Classe de caractères : [1236], [1-4], [^1-4]

Nombre de caractères : {5}, {5,6}, {0,1}, ?, {1,}, +, {0,}, *

Début : ^

Fin : \$

N'importe quel caractère : « . »

Caractère « ^ » : \^

Caractère « . » : \.

Alternative (ou) : |

Classes prédéfinies : [:alpha :], [:lower :], [:upper :], [:alnum :], [:space :], [:punct :],

Début de mot : [[:< :]]

Fin de mot : [[:> :]]

Exemples

'^[BL] | [0-9]-?[0-9]{4}\$' veut dire : un B ou un L ou un chiffre, suivi d'un tiret ou pas, suivi de 4 chiffres.

'^[SA][A-Z]*\.[^0-9]{3}[[:alnum:]]*\$' veut dire : un S ou un A, suivi de plusieurs majuscules ou aucune, suivi de un point, suivi de pas de chiffre sur 3 caractères, suivi des chiffres ou des lettres ou rien.

Les expressions régulières sont sensibles ou pas à la casse selon la collation utilisée.

Elles sont toujours sensibles aux accents.