BASE DE DONNEES RELATIONNELLE
Algèbre relationnelle et SQL

1 : Select mono-table

Manuels de référence du SQL :
Voir après le sommaire

Bertrand Liaudet

SOMMAIRE

Edition : décembre 2019 (contient l'ORACLE spécifique + MySQL 8.0 + Group By 8.0 + Variables et Vue).

Remarque d'utilisation du cours : les paragraphe précédés d'une (*) ne sont pas obligatoires pour un cours MySQL précédé d'un cours de modélisation. (**) veut dire (*) jusqu'à la fin du chapitre de niveau 2 (environ 15 pages, reste environ 80 pages).

Manuels de référence du SQL

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

Mémo SQL : http://www.volubis.fr/bonus/SQL_memo.htm

Adresses des manuels en ligne (édition novembre 2018) :

http://www.sqldata.fr/documentations-en-ligne/manuels-des-sgbd/

MySQL

Site officiel MySQL : https://www.mysql.com

Documentations MySQL : http://dev.mysql.com/doc/index.html

Documentation MySQL 8.0 en français :

https://dev.mysql.com/doc/refman/8.0/en/

Documentation MySQL 5.0 en français :

http://bliaudet.free.fr/IMG/pdf/MySQL-refman-5.0-fr.pdf

C'est la bible ! On fait les recherches dans le « Search manuel ».

Quand on passe plusieurs mots (logique de « et »), il faut mettre des « + » devant chaque mot.

Pour chercher une phrase, il faut la mettre entre guillemets.

MariaDB

MariaDB est la suite opensource de MySQL

Site officiel MariaDB : https://mariadb.org/

ORACLE

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

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

https://docs.microsoft.com/fr-fr/sql/t-sql/queries/queries?view=sql-server-2017

(*) 1 - MODELISATION : BASES DU 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
	TURNER	SALESMAN	8-SEP-81	3000	0	10
	JAMES	CLERK	3-DEC-81	1800	NULL	30
	WARD	SALESMAN	22-FEB-81	2500	500	20
4 tuples :	TURNER	ANALYST	3-DEC-81	5000	NULL	10

Vocabulaire

Relation = tableau = table = classe = ensemble = collection

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

Attribut = colonne du tableau = caractéristique = propriété = champ

BD = toutes les lignes de toutes les tables

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.

RELATION 7 attributs :

Employés	**NE**	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept
	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 tuples :	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10

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

Une clé primaire est toujours renseignée.

Exemples 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).
- Une adresse mail peut servir de clé primaire
- 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'Employe). Ca peut-être un simple entier.

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 social. 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.
- 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. Elle est fonction du contexte. Dans un autre contexte, la clé significative pourrait être le prénom.

NULL

- NULL signifie : « **non renseigné** ».
- C'est la seule information codée qu'on rentre dans une table.
- La valeur « 0 », par contre, ne signifie pas du tout « non renseignée », mais bien « valeur = 0 », comme on dirait « valeur = 500 ».
- Quand un attribut **peut valoir NULL**, on dit qu'il n'est « **pas obligatoire** ».
- Quand un attribut **ne peut pas valoir NULL**, on dit qu'il est « **obligatoire** » ou « **NOT NULL** ».
- A noter qu'en général, les attributs sont obligatoires.
- En règle générale, il vaut mieux limiter les valeurs NULL. A noter que MySQL, pour gagner facilement en performance, préconise d'éviter les valeurs NULL et de mettre '0' à la place.

Formalisme pour les tables

- **La clé primaire** est notée en premier et est soulignée.
- **Les clés secondaires** sont notées après la clé primaire et mises entre parenthèses.
- La table des employés représente une réalité physique. On l'appelle « **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.

BD – Schéma de la BD

Définition d'une BD

- Une BD c'est un ensemble de tables avec leurs tuples.
- Un SGBD gère une ou plusieurs BD.

Schéma d'une tables et schéma de la BD

- Le schéma d'une table consiste à écrire la table avec les noms de code des attributs, sans les tuples. La clé primaire est en premier et souligné. On précise parfois les clés secondaires et les attributs pouvant être NULL.

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

Schéma de la BD

- L'ensemble des schémas des tables constitue et les relations entre les tables constituent le schéma de la BD.

2 – INTRO. AU SQL ET A LA « CALCULETTE » SQL

PRINCIPALES NOTIONS	
Algèbre relationnelle	SQL
Manuel de référence	Calculette SQL

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** (create, drop, alter)
- Créer, modifier, détruire des tuples : **DML** (insert, update, delete)
- Consulter les tuples : **DSL** (select)

La consultation des tuples d'une table consiste à :

- **Filtrer les attributs** (choisir une ou plusieurs colonnes)
- **Créer des attributs calculés** (créer une ou plusieurs colonnes)
- **Filtrer les tuples** (choisir une ou plusieurs lignes)
- **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).

SQL ou algèbre relationnelle ?

- Le **SQL** (Structured Query Langage) 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 **« calculette SQL »** qui, au lieu de travailler sur des nombres, travaille sur des tableaux de données.

Intérêt de l'algèbre relationnelle 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 relationnelle

- Le SQL est un langage normalisé (ANSI - ISO) et implanté dans tous les SGBD-R
- Le SQL permet de créer les BD et de tester les calculs.
- 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.

2. Classification des commandes du SQL et manuel de référence

4 types de commandes

Les commandes du SQL se divisent en 4 sous-ensembles :

- Le **DDL** : data definition language
- Le **DML** : data manipulation language
- Le **DSL** : data select language (formulation non standard)
- Le **DCL** : data control language

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

Ce sont les commandes qui vont permettre de créer, modifier, détruire les tables : **CREATE TABLE**, **ALTER TABLE**, **DROP TABLE**.

Ces commandes permettront aussi de créer, modifier et supprimer d'autres objets de la BD : les BD, les séquences (auto-incréments ORACLE), les index, les vues, etc. **CREATE DATABASE**, **CREATE VIEW**, **CREATE INDEX**, etc

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

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

Le DSL : SELECT : l'algèbre relationnelle

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

Le SELECT me en œuvre l'algèbre relationnelle.

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 validée 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: cliquez sur le titre !

3 - SQL : CONSULTATION DE LA BASE DE DONNEES

PRINCIPALES NOTIONS	
Projection - Restriction	**SELECT / FROM / Where**
Distinct	**year / month / day**
Tri	**length / substr / concat**
Attribut calculé	**in, between, like**
Projection primaire	**Order by / asc / desc**

1. 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 réalisables 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 5 grandes catégories d'opérations s'appliquant à une seule table :

- Les filtres sur les colonnes = **Projection.** On choisit une ou plusieurs colonnes.
- La création d'attributs calculés = **Projection.** On crée une colonne par du calcul.
- Les filtres sur les lignes = **Restriction**. On choisit une ou plusieurs lignes.
- Les tris
- Les opérations statistiques

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 qui travaillent plutôt sur des tuples de même nature (donc sur la même table).
- Le **produit cartésien** et la jointure associée qui permet de travailler sur deux colonnes de 2 tables différentes.
- Les imbrications d'opérations.

Exemples de questions traitées par le SELECT :

- Quel est le nom de tous les employés qui sont vendeurs (filtre des lignes et des colonnes) ?
  - SELECT ne, nom FROM employes WHERE fonction = « vendeur » ;

- 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).
  - SELECT * FROM employes ORDER BY nom ;

- 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) ?
  - SELECT AVG(sal) FROM employes WHERE fonction = « vendeur » ;

- Quels sont les employés qui gagnent plus que la moyenne des salaires de l'entreprise ? (calcul statistique avec regroupements).

- Afficher la liste des employés avec leur nom, leur fonction, leur salaire et leur salaire net approximatif. Pour calculer le salaire net, il faut retirer 23% de cotisations à peu près. C'est un attribut calculé.
  - SELECT ne, nom, fonction, salaire, salaire*(1-0,23) FROM employes;

- Combien y a-t-il d'employés par tranche de 1000 de salaire (attribut calculé) ?

Manuels de référence du Select

ORACLE

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

MySQL

https://dev.mysql.com/doc/refman/8.0/en/select.html

PostgreSQL

http://www.postgresql.org/docs

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

SQL Server

https://docs.microsoft.com/fr-fr/sql/t-sql/queries/select-transact-sql?view=sql-server-2017

2. Projection = filtre des colonnes

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

Exemples

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

```
SELECT *
FROM emp;
```

C'est assez rare qu'on veuille tous les attributs. Il faut en tout cas se limiter à ce qui est nécessaire.

- ***Donner les noms de tous les employés :***

```
SELECT NE, nom
FROM emp;
```

- ***ATTENTION :***
  - **Il est nécessaire de projeter NE** même s'il n'est pas explicitement demandé.
  - En effet, la table produite ne doit pas contenir de tuples en double.
  - Pour distinguer les homonymes, on projette la clé primaire de la relation.

Schéma

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 jaune : les colonnes sélectionnées.

Syntaxe SQL

```
SELECT liste d'attributs [1]
FROM table ;
```

Syntaxe AR

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

1 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 : on va parler de « **projection primaire** ».
- Eliminer les doublons : on va parler de « **projection avec distinct** ».

La projection primaire

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**.

NE	NOM	FONCTION
7369	SMITH	CLERK
7499	ALLEN	SALESMAN
7521	WARD	SALESMAN
7566	JONES	MANAGER
7654	MARTIN	SALESMAN
7698	BLAKE	MANAGER
7782	CLARK	MANAGER
7788	SCOTT	ANALYST
7839	KING	PRESIDENT
7844	TURNER	SALESMAN
7900	JAMES	CLERK
7902	FORD	ANALYST
7903	ADAMS	CLERK
7904	MILLER	CLERK

Syntaxe

```
SELECT clé primaire, liste d'attributs
FROM table ;
```

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

Projection avec élimination des doublons : la clause distinct

Exemple

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

- ***Tous les métiers de la société :***

```
SELECT distinct fonction
FROM emp ;
```

FONCTION
CLERK
SALESMAN
MANAGER
ANALYST
PRESIDENT

La clé primaire de cette table est : FONCTION.

- ***Tous les métiers de la société par numéro de département :***

```
SELECT distinct ND, fonction
FROM emp ;
```

La clé primaire de la table des résultats est : ND, FONCTION.

Syntaxe SQL

```
SELECT distinct 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.

Remarques sur la clé primaire

- ***Remarque 1 :***

  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 ;
```

  équivaut à :

```
SELECT clé primaire, liste d'attributs FROM table ;
```

  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 produits des métiers (« fonction ») en partant d'employés.

- ***Remarque 2***

  Quand on met un distinct, il faut chercher la clé primaire parmi les attributs projetés.

  S'il n'y a qu'un attribut projeté, c'est forcément la c

  S'il y de plusieurs attributs, que ce soit sur une requête mono-table ou sur une requête multi-table, il vaut mieux trouver la clé primaire et la mettre en premier avec la clé significative pour savoir de quoi on parle.

Création d'attributs calculés

Création d'un attribut

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 ;
```

- ***tous les salaires, commissions et salaires totaux des salariés :***

```
SELECT NE, nom, sal, comm, sal + comm
FROM emp;
```

NE	NOM	SAL	COMM	SAL + COMM
7369	SMITH	800.00	*NULL*	*NULL*
7499	ALLEN	1600.00	300.00	1900.00
7521	WARD	1250.00	500.00	1750.00
7566	JONES	2975.00	*NULL*	*NULL*
7654	MARTIN	1250.00	1400.00	2650.00
7698	BLAKE	2850.00	*NULL*	*NULL*
7782	CLARK	2450.00	*NULL*	*NULL*
7788	SCOTT	3000.00	*NULL*	*NULL*
7839	KING	5000.00	*NULL*	*NULL*
7844	TURNER	1500.00	0.00	1500.00
7900	JAMES	950.00	*NULL*	*NULL*
7902	FORD	3000.00	*NULL*	*NULL*
7903	ADAMS	1100.00	*NULL*	*NULL*
7904	MILLER	1300.00	*NULL*	*NULL*

Autres exemples

- ***nombre de lettres du nom de chaque employé :***

```
SELECT NE, nom, length(nom)
FROM emp;
```

- ***année d'embauche de chaque employé***

```
SELECT NE, nom, datemb, year(datemb)
FROM emp;
```

- ***Salaire par tranche : petit (<1000), moyen (<2000), gros : autres.***

```
SELECT NE, nom,
CASE
  WHEN sal < 1000 THEN "petit"
  WHEN sal < 2000 THEN "moyen"
  ELSE "gros"
END
FROM emp;
```

Renommer un attribut

- ***Exemple de renommage***

```
SELECT NE, nom, sal, comm, sal + comm AS salaireTotal
FROM emp;
```

NE	NOM	SAL	COMM	SalaireTotal
7369	SMITH	800.00	*NULL*	*NULL*
7499	ALLEN	1600.00	300.00	1900.00
7521	WARD	1250.00	500.00	1750.00
7566	JONES	2975.00	*NULL*	*NULL*
7654	MARTIN	1250.00	1400.00	2650.00
7698	BLAKE	2850.00	*NULL*	*NULL*
7782	CLARK	2450.00	*NULL*	*NULL*
7788	SCOTT	3000.00	*NULL*	*NULL*
7839	KING	5000.00	*NULL*	*NULL*
7844	TURNER	1500.00	0.00	1500.00
7900	JAMES	950.00	*NULL*	*NULL*
7902	FORD	3000.00	*NULL*	*NULL*
7903	ADAMS	1100.00	*NULL*	*NULL*
7904	MILLER	1300.00	*NULL*	*NULL*

- ***Principes du renommage***

  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.

  On met **« AS »** entre l'ancien nom et le nouveau nom.

```
SELECT ancien_nom  AS nouveau_nom FROM table;
```

- ***Remarques***

  Si le nouveau nom contient des espaces, on écrira :

```
SELECT ancien_nom AS "nouveau nom" FROM table;
```

  On peut se passer du AS :

```
SELECT ancien_nom nouveau_nom FROM table;
```

Les opérations possibles pour calculer un attribut

Opérateurs et fonctions classiques

On peut utiliser **tous les opérateurs et toutes les fonctions** comme on l'a vu dans les premiers exemples.

Les chapitres suivants montreront d'autres opérateurs et d'autres fonctions.

IF (pas standard)

Le IF permet de faire des tests dans le SELECT avec 2 alternatives, 2 embranchements. Le passage par un embranchement permet de renvoyer une valeur et une seule : celle de l'attribut calculé.

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 1 : IF simple***

On crée un attribut, « categorie » qui aura deux valeurs : « bas » et « gros » en fonction du salaire. On met la limite à 2000.

Syntaxe : if(condition, résultat si vrai, résultat si faux)

```
SELECT NE, nom, sal,
  IF(sal<2000,"1:bas","3:gros") AS categorie
FROM emp
ORDER BY categorie, nom;
```

- ***Exemple 2 : IF imbriqué***

Pour avoir plus de deux alternatives, on peut imbriquer les IF

```
SELECT NE, nom, sal,
  IF(sal<1200,"1:bas",IF(sal<2500,"2:moyen","3:gros"))
  AS categorie
FROM emp
ORDER BY categorie, nom;
```

CASE WHEN : standard

- ***Présentation***

  Le “case when” est **une façon plus lisible d’écrire des IF imbriqués**.

  En général, il a **deux usages et deux syntaxes** :

  - usage **« else-if » classique** : succession de tests indépendants.
  - usage **« switch » classique** : comparaison d’égalité entre une expression unique de départ.

  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.

➢ ***Première syntaxe : équivalent de IF imbriqués (else-if)***

La première 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.

Syntaxe :

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

Exemple :

```
SELECT NE, nom, sal,
CASE
  WHEN sal<1200 THEN "1:petit"
  WHEN sal<2500 THEN "2:moyen"
  ELSE "3:gros"
END AS categorie
FROM emp;
```

NE	nom	sal	categorie
7369	SMITH	800.00	1:petit
7499	ALLEN	1600.00	2:moyen
7521	WARD	1250.00	2:moyen
7566	JONES	2975.00	3:gros
7654	MARTIN	1250.00	2:moyen
7698	BLAKE	2850.00	3:gros
7782	CLARK	2450.00	2:moyen
7788	SCOTT	3000.00	3:gros
7839	KING	5000.00	3:gros
7844	TURNER	1500.00	2:moyen
7900	JAMES	950.00	1:petit
7902	FORD	3000.00	3:gros
7903	ADAMS	1100.00	1:petit
7904	MILLER	1300.00	2:moyen

- ***Deuxième syntaxe : switch classique***

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

  Syntaxe :

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

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

  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.

Exemple :

```
SELECT NE, nom, sal,
CASE sal div 1000
  WHEN 0 THEN "< mille"
  WHEN 1 THEN "mille et plus"
  WHEN 2 THEN "2 mille et plus"
  WHEN 3 THEN "3 mille et plus"
  ELSE "plus de 4 mille"
END AS categorie
FROM emp;
```

NE	nom	sal	categorie
7369	SMITH	800.00	< mille
7499	ALLEN	1600.00	mille et plus
7521	WARD	1250.00	mille et plus
7566	JONES	2975.00	2 mille et plus
7654	MARTIN	1250.00	mille et plus
7698	BLAKE	2850.00	2 mille et plus
7782	CLARK	2450.00	2 mille et plus
7788	SCOTT	3000.00	3 mille et plus
7839	KING	5000.00	plus de 4 mille
7844	TURNER	1500.00	mille et plus
7900	JAMES	950.00	< mille
7902	FORD	3000.00	3 mille et plus
7903	ADAMS	1100.00	mille et plus
7904	MILLER	1300.00	mille et plus

Select imbriqué dans la projection

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

Exemple :

```
SELECT NE, nom, sal,
  (SELECT max(sal)FROM emp) AS maxSal
FROM emp;
```

Usage :

MIEUX VAUT EVITER LES SELECT IMBRIQUES !

On réabordera cette syntaxe dans le chapitres sur les SELECT multi-tables.

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.

3. Restriction = filtre des lignes

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

Exemples

Tous les employés du département 30 avec un salaire > 1000 avec tous leurs attributs :

```
SELECT *
FROM emp
WHERE ND = 30 and Sal >1000;
```

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

Il existe aussi trois opérateurs spécifiques au SQL :

in, like, between

Cf. paragraphe 3.5 sur les opérations sur les opérateurs de comparaison.

Syntaxe AR

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

Schéma

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 jaune : les lignes sélectionnées.

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

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

Exemple

tous les employés avec leur date d'embauche dont le salaire est compris entre 1200 et 1400

```
SELECT NE, nom, datemb, sal
FROM emp
where sal >= 1200 and sal <= 1400 ;
```

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

schéma

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
tuple 1					
tuple 2					
…					
tuple n					

En jaune: les informations sélectionnées.

Syntaxe AR

T1 = Rest (*table ; formule logique de sélection des tuples)*

Tres = Proj (T1 ; *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 = Proj (Rest (*table ; formule logique de sélection des tuples)* ; *liste d'attributs)*

Cependant, on évitera cette écriture qui est peu lisible.

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**

Restriction-projection primaire

Exemple

tous les employés avec leur date d'embauche dont le salaire est compris entre 1200 et 1400

```
SELECT NE, nom, datemb, sal
FROM emp
where sal >= 1200 and sal <= 1400 ;
```

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

NE	nom	datemb	sal
7521	WARD	1981-02-22	1250.00
7654	MARTIN	1981-09-28	1250.00
7904	MILLER	1982-01-23	1300.00

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

On peut aussi écrire la requête avec un between :

```
SELECT NE, nom, datemb, sal
FROM emp
where sal between 1200 and 1400 ;
```

Syntaxe générale SQL

La syntaxe restriction-projection primaire est :

```
SELECT clé primaire, clé significative,
  attributs demandés, attributs de restriction
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.

Attributs projetés

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

CP – (CS) – demandés – (restriction)

0) Les attributs entre parenthèses sont facultatifs mais fortement recommandés.

1) CP : on projette la clé primaire en premier pour savoir de quoi-qui 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 notre exemple, 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 restriction pour vérifier que les conditions sont bien réalisées. Département, salaire et fonction dans nos exemples.

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

Exemple 1

Pour obtenir les différents métiers de la société avec un sal > 2000 , on écrira :

```
SELECT distinct fonction
FROM emp
WHERE sal >2000;
```

fonction
MANAGER
ANALYST
PRESIDENT

La clé primaire de la table résultat, c'est la fonction. On produit une table de fonctions.

Exemple 2

Pour obtenir les différents métiers de la société avec un sal > 2000, par numéro de département, on écrira :

```
SELECT distinct ND, fonction
FROM emp
WHERE sal >2000
ORDER BY ND, fonction ;
```

On a intérêt à trier pour rendre le résultat lisible :

ND ▲ 1	fonction ▲ 2
10	MANAGER
10	PRESIDENT
20	ANALYST
20	MANAGER
30	MANAGER

La clé primaire de la table résultat, c'est le couple (ND, fonction). On produit une table de couples (ND-fonction).

Syntaxe SQL

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

```
SELECT distinct liste d'attributs  FROM table  ;
```

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 finale, clé significative finale,
    attributs demandés, attributs de restriction
FROM table
where formule logique de sélection des tuples ;
```

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

Toutefois, on écrit « clé primaire finale » pour préciser que la clé primaire devient clé primaire après élimination des doublons. Idem pour la clé significative.

5. 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.

http://www.lirmm.fr/~jq/Cours/MASS/Licence/BaseDeDonnees/doc/calculEnMySQL.html

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***

  https://dev.mysql.com/doc/refman/8.0/en/functions.html

- ***PostgreSQL***

  http://www.postgresql.org/docs

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

Les opérations arithmétiques : la calculette arithmétique

Présentation

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

+, -, *, /, div, mod, (,)

div est l'opérateur de division entière. mod c'est le reste de la division entière (modulo). Div est équivalent à floor( ) en MySQL.

Mais aussi toutes les fonctions mathématiques standards :

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

Quelques fonctions mathématiques en vrac

https://dev.mysql.com/doc/refman/8.0/en/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.

Manuel de référence

- ***MySQL***

  https://dev.mysql.com/doc/refman/8.0/en/functions.html

  https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html

  https://dev.mysql.com/doc/refman/8.0/en/arithmetic-functions.html

La calculette arithmétique

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

- ***Calculette mysql :***

On peut écrire :

```
SELECT 4*3 ;
```

Ou bien

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

Ou bien

```
SELECT 24*sin(3.1416/6) ;
```

On obtiendra le résultat du calcul : 12.

```
SELECT concat('Hello', ' ', 'world !') ;
```

Ca donne « Hello world ! »

- ***Calculette oracle : table DUAL***

On peut écrire :

```
SELECT 4*3 FROM DUAL ;
```

Ou bien

```
SELECT 4*log(2,8) FROM DUAL ;
```

Ou bien

```
SELECT 24*sin(3.1416/6) FROM DUAL ;
```

On obtiendra le résultat du calcul : 12.

```
SELECT 'FROM DUAL ;
```

On obtiendra le résultat du calcul : 12.

```
SELECT 'Hello' || ' ' || 'world !' FROM DUAL ;
```

Ca donne « Hello world ! ». On peut aussi utiliser la fonction concat( ) sous ORACLE.

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**'

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

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

not, *, +, <, and, or

Opérateurs spéciaux :

- ***Between … and…***

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

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

```
SELECT NE, nom, sal FROM emp
where sal between 1200 and 1400 ;
```

- ***In et not in***

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

  Tous les employés travaillant dans les départements 10 ou 30 :

```
SELECT NE, nom, ND FROM emp
where ND in (10, 30) ;
```

- ***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.

- ***(*) like ORACLE avec option ESCAPE***

  Si on veut les employés qui ont un « _ » dans leur nom :

```
SELECT NE, nom FROM emp
where nom like '%\_%' ESCAPE '\';
```

  Si on veut chercher un mot avec un '_' ou un '%', on place le caractère cherché dans le format précédé d'un caractère au choix, ici « \ », et on ajoute ESCAPE du caractère choisi.

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.

- ***Exemple :***

On veut remplacer les NULL par un 0

```
SELECT NE, nom, sal, ifnull(comm, 0)  FROM  emp;
```

(*) Equivalence dans les autres SGBD

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

Les fonctions de traitement de chaîne

Manuel de référence

- ***MySQL***

  https://dev.mysql.com/doc/refman/8.0/en/string-functions.html

  https://dev.mysql.com/doc/refman/8.0/en/string-comparison-functions.html

Les constantes

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

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 ;
```

concat est une fonction de concaténation

```
SELECT concat (chaine1, chaine2, chaine3) …
```

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).

Sous ORACLE, on peut utiliser l'opérateur || à la place de la fonction concat

```
SELECT chaine1 || chaine2  || chaine3) …
```

Autres fonctions de traitement de chaînes en vrac

https://dev.mysql.com/doc/refman/8.0/en/string-functions.html

https://dev.mysql.com/doc/refman/8.0/en/string-comparison-functions.html

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

length(attribut) ;

upper(texte) ; **lower**(texte) ; passe en majuscules ou minuscules.

substring(attribut from déb for lgr) : sous-chaîne commençant en déb de longueur lgr ;

substring ⇔ mid() ⇔ substr()

substring(att from 11) : la sous-chaîne à partir du 11ème 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 ;

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.

Reverse(att**)** : renverse l'ordre de la chaîne.

(*) Regexp : expression régulière

- ***Présentation***

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

  MySQL propose le REGEXP, qui 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.

- ***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.

- ***Sensibilité aux accents et à la casse***

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

  Elles sont toujours sensibles aux accents.

- ***Manuel de référence***

  https://dev.mysql.com/doc/refman/8.0/en/regexp.html

Les opérateurs de traitement de date

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

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

Il y a deux aspects spécifiques à prendre en compte :

- Les fonctions de manipulation des dates
- La question des formats

Mieux vaut régler le problème au niveau du SQL plutôt qu'au niveau du langage seveur.

La présentation ci-dessous présente quelques éléments dans différents SGBD

Les dates dans MySQL

Manuel de référence

https://dev.mysql.com/doc/refman/8.0/en/date-calculations.html

https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

- ***Exemples***

Aujourd'hui

```
SELECT curdate();   //  2018-10-31
```

Le mois du jour

```
SELECT month(curdate());     // 10
```

Aujourd'hui, l'année dernière :

```
SELECT from_days(to_days(curdate())-365);
```

Nombre de jours entre deux dates

```
SELECT datediff('2018-10-31','2018-10-21');     // 10
```

Principales fonctions de traitement de dates

year(date), **month**(date), **day**(date), **hour**(date), **minute**(date), **second**(date) : renvoient l'année, le mois et le jour, etc.

curdate(), **current_date, utc_date()** : date du jour ; **curtime**() , **current_time, utc_time()** : heure actuelle; **now**(), **current_timestamp, utc_timestamp** : date et heure actuelle ; **sysdate()** : equivalent de now mais calculé en temps réel; **utc** : date et heure de Greenwich.

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

from_days(nb jours) : renvoie la durée correspondant à un nombre de jours. Cette durée a le format date. from_days démarre à 366 : 1 janvier de l'année 1.

datediff(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.) ;

date **+ 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 ; (***type*** : year, month, day, week, hour, minute, second, microsecond, etc.) ;

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

Quelques conversions

- ***Date vers jours et réciproquement***

  to_days (date) : Nombre de jours depuis le 1er janvier 0

  from_days (nb jours) : durée en année, mois, jour à partir d'un nombre de jour

  Remarque: from_days démarre à 366 : 1 janvier de l'année 1 (from_days(0 à 365) vaut 0.

- ***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 = 20161010 ; (2016 10 10)

  date(20161010) = 20 octobre 2016

  On peut mettre 20161010 ou 20161010' ou "20161010"

  date(20161010)+10 = 20 octobre 2016

- ***Conversion d'une heure en nombre et réciproquement***

  curtime( ) + 0 = 161055 ;

  SELECT time(3666) ; = 16 heures 10 minutes et 55 secondes

- ***Date vers secondes et réciproquement***

  Nombre de secondes depuis le 1er janvier 1970 : **unix_timestamp** (date)

  Inverse : **from_unixtime**(nb secondes)

  Remarque : unix_timestamp ( ) ⇔ unix_timestamp (now())

Gestion des formats d'affichage : date_format ( )

https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html - function_date-format

- ***Exemple***

```
SELECT date_format(
    now( ),
    '%W %d %M %H heures %i minutes %s sec.'
);
```

Monday 31 October 10 heures 23 minutes 24 secSyntaxe

- ***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.

- ***(**) Affichage au format français***

  https://dev.mysql.com/doc/refman/8.0/en/locale-support.html

  Pour passer en français côté client :

  SET lc_time_names = 'fr_FR';

  Pour passer en américain côté client :

  SET lc_time_names = 'en_US';

  Pour afficher la langue côté client :

  SELECT @@lc_time_names;

  Pour afficher la langue côté serveur :

  SELECT @@global.lc_time_names;

  Pour modifier la langue côté serveur :

  SET global lc_time_names = 'fr_FR';

  Pour afficher la date en français :

```
SET lc_time_names = 'fr_FR';
SELECT date_format(
   now( ),
   '%W %d %M %H heures %i minutes %s sec.'
);
```

  lundi 31 octobre 11 heures 45 minutes 04 sec.

```
SELECT date_format(now(),'%d/%m/%Y %Hh%imin%ss') ;
```

  31/10/2016 11h48min04s

(*) Les dates dans Oracle

Dates et heures système

```
Select current_date from dual ; // date client
Select localetimestamp ; // 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 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

https://dev.mysql.com/doc/refman/8.0/en/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 !!!
```

6. Présentation des résultats : tris, limites, mode page

Trier les résultats - Exemple

Présentation : clause ORDER BY

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**.

Exemple 1 : tri simple par attribut continu

Un attribut continu est un attribut qui peut prendre un très grand nombre de valeurs continues (par opposition à un attribut catégoriel). Par exemple le salaire qui est un réel.

Tous les employés avec leur fonction et leur salaire, classés par salaire croissant :

```
SELECT NE, nom, fonction, sal
FROM emp
ORDER BY sal
```

NE	nom	fonction	sal
7369	SMITH	CLERK	800.00
7900	JAMES	CLERK	950.00
7903	ADAMS	CLERK	1100.00
7521	WARD	SALESMAN	1250.00
7654	MARTIN	SALESMAN	1250.00
7904	MILLER	CLERK	1300.00
7844	TURNER	SALESMAN	1500.00
7499	ALLEN	SALESMAN	1600.00
7782	CLARK	MANAGER	2450.00
7698	BLAKE	MANAGER	2850.00
7566	JONES	MANAGER	2975.00
7902	FORD	ANALYST	3000.00
7788	SCOTT	ANALYST	3000.00
7839	KING	PRESIDENT	5000.00

Pour rendre la table de résultats lisible, on a projeté l'attribut de tri en dernier.

Exemple 2 : tri simple par attribut catégoriel

Un attribut catégoriel est un attribut qui a une liste de valeurs limitée (par opposition à un attribut continu).

Tous les employés avec leurs départements et leurs salaires triés par fonction :

```
SELECT fonction NE, nom, ND, sal
FROM emp
ORDER BY fonction
```

Le tri met au jour des regroupements : les employés ANALYST, puis les CLECK, les MANAGER, le PRESIDENT, les SALESMAN.

fonction ▲ 1	NE	nom	ND	sal
ANALYST	7788	SCOTT	20	3000.00
ANALYST	7902	FORD	20	3000.00
CLERK	7903	ADAMS	20	1100.00
CLERK	7900	JAMES	30	950.00
CLERK	7369	SMITH	20	800.00
CLERK	7904	MILLER	10	1300.00
MANAGER	7698	BLAKE	30	2850.00
MANAGER	7782	CLARK	10	2450.00
MANAGER	7566	JONES	20	2975.00
PRESIDENT	7839	KING	10	5000.00
SALESMAN	7654	MARTIN	30	1250.00
SALESMAN	7844	TURNER	30	1500.00
SALESMAN	7521	WARD	30	1250.00
SALESMAN	7499	ALLEN	30	1600.00

Pour rendre la table de résultats lisible, on a projeté l'attribut de tri en premier.

Exemple 3 : tri sur plusieurs attributs catégoriels

Tous les employés triés par département et fonction avec leur salaire :

```
SELECT ND, fonction, NE, nom, sal
FROM emp
ORDER BY ND, fonction
```

Le tri met au jour des regroupements : les employés du département 10, ceux du 20, ceux du 30. Dans chaque groupe, il met au jour les fonctions. Par exemple : dans le département 30, on a 1 MANAGER et 4 SALESMAN.

ND	fonction	NE	nom	sal
10	CLERK	7904	MILLER	1300.00
10	MANAGER	7782	CLARK	2450.00
10	PRESIDENT	7839	KING	5000.00
20	ANALYST	7902	FORD	3000.00
20	ANALYST	7788	SCOTT	3000.00
20	CLERK	7903	ADAMS	1100.00
20	CLERK	7369	SMITH	800.00
20	MANAGER	7566	JONES	2975.00
30	CLERK	7900	JAMES	950.00
30	MANAGER	7698	BLAKE	2850.00
30	SALESMAN	7654	MARTIN	1250.00
30	SALESMAN	7521	WARD	1250.00
30	SALESMAN	7844	TURNER	1500.00
30	SALESMAN	7499	ALLEN	1600.00

Pour rendre la table de résultats lisible, on a projeté d'abord les attributs de tri.

Exemple 4 : tri sur un attribut catégoriel et un attribut continu

Tous les employés avec leurs départements et leurs salaires triés par fonction et par salaire :

```
SELECT fonction NE, nom, ND, sal
FROM emp
ORDER BY fonction, sal
```

Le tri met au jour des regroupements : les employés ANALYST, puis les CLECK, les MANAGER, le PRESIDENT, les SALESMAN.

Le tri met au jour des regroupements : les employés ANALYST, puis les CLECK, les MANAGER, le PRESIDENT, les SALESMAN. Dans chaque regroupement, les employés sont triés par salaire. Les 4 CLERK vont de 800 à 1300.

NE	nom	fonction ▲ 1	sal ▲ 2
7788	SCOTT	ANALYST	3000.00
7902	FORD	ANALYST	3000.00
7369	SMITH	CLERK	800.00
7900	JAMES	CLERK	950.00
7903	ADAMS	CLERK	1100.00
7904	MILLER	CLERK	1300.00
7782	CLARK	MANAGER	2450.00
7698	BLAKE	MANAGER	2850.00
7566	JONES	MANAGER	2975.00
7839	KING	PRESIDENT	5000.00
7521	WARD	SALESMAN	1250.00
7654	MARTIN	SALESMAN	1250.00
7844	TURNER	SALESMAN	1500.00
7499	ALLEN	SALESMAN	1600.00

Pour rendre la table de résultats lisible, on a projeté l'attribut de tri en dernier.

ASC, DESC, alias

- ***Exemple***

  Tous les employés triés par ordre alphabétique de fonction et par salaire total décroissant

```
SELECT fonction, NE, nom, sal+ifnull(comm) as salaireTotal,
FROM emp
ORDER BY fonction ASC, salaireTotal DESC, nom;
```

- ***Ordre du tri : ASC par défaut, DESC***

  On tri par ordre croissant par défaut : on peut préciser ASC.

  Si on veut un ordre décroissant, on précise DESC

- ***Usage d'alias dans le tri***

  Un alias défini dans la projection peut être réutilisé dans le tri. Ici « salaireTotal ».

  A noter que l'alias ne peut pas être utilisé dans un « group by » ou dans un « having »... ce qui n'est pas pratique !

Attributs projetés

Il faut projeter les catégories de tri dans l'ordre, en premier ou en dernier.

Quand on a un attribut de tri continu, mieux vaut le projeter en dernier.

Il faut maintenir ensemble le couple CP-CS.

Syntaxe AR

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

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 au premier élément

- ***Solution MySQL et PostgreSQL :***

```
SELECT NE, nom, sal FROM emp
limit 1;
```

- ***Solution Oracle :***

```
SELECT NE, nom, sal FROM emp
Where rownum = 1;
```

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;
```

Commencer la projection après les 5 premiers éléments et la limiter

- ***Solution MySQL :***

```
SELECT NE, nom, sal FROM emp
limit 1 offset 5 ;
```

L'offset permet de passer les 5 premiers éléments.

On peut aussi écrire :

```
SELECT NE, nom, sal FROM emp
limit 5,1;
```

- ***Solution Oracle :***

Si on fait :

```
SELECT rownum, NE, nom, sal
FROM emp
WHERE rownum = 5;
```

On ne récupère aucun tuple !

Le rownum ne peut s'utiliser qu'avec le signe « < » ! (ou seulement « =1 »).

Pour obtenir le 5ème élément, il faudra utiliser une imbrication dans le from :

```
SELECT NE, nom, sal
FROM (SELECT rownum r1, NE, nom, sal from emp) t1
WHERE r1=5;
```

Il faut renommer le rownum du select imbriqué pour qu'il ne soit pas confondu avec celui du select imbriquant. Ensuite on récupère le numéro de ligne souhaité.

Projection de 5 en 5

On prend les tuples de 5 en 5. On veut récupérer les tuples de 11 à 15

- ***Solution MySQL :***

Les 5 premiers :

```
SELECT NE, nom, sal FROM emp
limit 5 offset 0;
```

Les 5 suivants :

```
SELECT NE, nom, sal FROM emp
limit 5 offset 5;
```

Les 5 suivants :

```
SELECT NE, nom, sal FROM emp
limit 5 offset 10;
```

Autre solution :

```
SELECT NE, nom, sal FROM emp
limit 0,5;
```

Les 5 suivants :

```
SELECT NE, nom, sal FROM emp
limit 5,5;
```

Les 5 suivants :

```
SELECT NE, nom, sal FROM emp
limit 10,5;
```

- ***Solution Oracle :***

```
SELECT NE, nom, sal
FROM (SELECT rownum r1, NE, nom, sal from emp) t1
WHERE r1>=11 AND r1<=15;
```

Il faut renommer le rownum du select imbriqué pour qu'il ne soit pas confondu avec celui du select imbriquant. Ensuite on récupère le numéro de ligne souhaité.

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 la valeur plus haut salaire :

```
SELECT sal
FROM emp
ORDER BY sal desc
limit 1 ;
```

Attention, on ne peut pas projeter l'employé avec le plus haut salaire car il peut y avoir plusieurs employés avec le même plus haut salaire.

- ***Solution Oracle :***

Attention : le order by intervient après le rownum.

Si on fait :

```
SELECT sal
FROM emp
WHERE rownum = 1
ORDER BY sal desc
```

Ca ne marche pas ! Le order by intervient après le rownum.

Le rownum ne peut s'utiliser qu'avec le signe « < » ! (ou seulement « =1 »).

Pour obtenir le 5ème élément, il faudra utiliser une imbrication dans le from :

```
SELECT sal
FROM (SELECT rownum r1, sal from emp) t1
WHERE r1=1
ORDER BY sal desc ;
```

Attention, on ne peut pas projeter l'employé avec le plus haut salaire car il peut y avoir plusieurs employés avec le même plus haut salaire.

Tri aléatoire : order by rand()

Principe

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

- ***Solution MySQL : order by rand()***

Tous les employés avec tous leurs attributs classés aléatoirement :

```
SELECT * FROM emp
ORDER BY rand() ;
```

- ***Solution ORACLE : order by dbms_random.value***

Tous les employés avec tous leurs attributs classés aléatoirement :

```
SELECT * FROM emp
ORDER BY dbms_random.value ;
```

Sélection d'un élément au hasard :

Exemple : un employé au hasard avec tous ses attributs :

- ***Solution MySQL : order by rand() et limit***

```
SELECT * FROM emp
ORDER BY rand()
LIMIT 1;
```

- ***Solution ORACLE : order by dbms_random.value et rownum***

Tous les employés avec tous leurs attributs classés aléatoirement :

```
SELECT *
FROM (
    SELECT *
    FROM emp
    ORDER BY dbms_random.value
) t1
WHERE rownum = 1 ;
```

Afficher les résultats en mode « page » : uniquement mysql

Présentation

Un Select renvoie normalement une liste de résultat sous la forme d'un tableau avec un individu par ligne et plusieurs attributs pour chaque ligne.

Présentation « page » de mysql : \G

Mysql permet d'afficher les résultats en mode « page », avec un attribut par ligne.

Il faut placer un \G à la fin de la requête plutôt qu'un ;

- ***Solution MySQL :***

```
SELECT * FROM emp \G
```

4 - CALCULS STATISTIQUES EN SQL : LES FONCTIONS DE GROUPE ET LES AGREGATS

PRINCIPALES NOTIONS	
Fonction de groupe	**Sum()**
Count()	**Max(), Min(), Avg()**
Agrégat	**Group by**
Having	

1. Calculs statistiques : les fonctions de groupe

Calcul statistique élémentaire : les fonctions de groupe

Présentation

- Les fonctions de groupe sont des fonctions qui permettent de faire des calculs statistiques sur un ensemble de tuples.
- Les fonctions de groupe interviennent dans la projection : elles s'intéressent à la colonne projetée.

Attention !!!

- 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.

Exemples

Soit la table des Employés suivantes :

NE	Nom	Salaire
1	Toto	2000
2	Titi	1000
3	Tutu	4000
4	Tata	3000

- ***Combien y a-t-il d'employés ?***

Pour obtenir cette information, on écrit :

```
SELECT count(*) AS nbEmployes
FROM emp ;
```

Le résultat est le suivant :

nbEmployes
4

- ***Quelle est la moyenne des salaires ?***

Pour obtenir cette information, on écrit :

```
SELECT avg(sal) AS moySal
FROM emp ;
```

Le résultat est le suivant :

moySal
2500

Remarque importante

- Une fonction de calcul statistique (ici count() et avg() produit donc **un tuple et un seul comme résultat**.
- Elle permet donc de créer un nouveau tuple à partir d'un attribut en entrée.
- On ne peut pas projeter d'autres attributs avec une fonction de groupe (sauf s'il y a un Group By : cf. suite du cours).

- La table ci-dessous ne veut rien dire. Un nom à côté d'une moyenne ne veut rien dire.

moySal	nom
2500	Toto

- On ne doit donc pas écrire :

```
SELECT avg(sal) as moySal, nom
FROM emp ;
```

Plusieurs statistiques en même temps

On veut le nombre d'employés et la moyenne des salaire.

```
SELECT count(*) as nbEmployes, avg(sal) as moySal
FROM emp ;
```

nbEmployes	moySal
4	2500

Les 5 fonctions de groupe standards

Il existe 5 fonctions de groupe (notées ***fdg*** 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 suivante :

```
SELECT fdg ( attribut ) FROM table ;
```

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(*).

  A priori le count(*) est équivalent au count(id) car dès qu'il y a un attribut not null, le système peut savoir qu'il faut tout compter sans s'intéresser aux autres attributs. Cependant, comme pour toutes les questions d'optimisation, il faut regarder l'usage.

- ***Remarque 2***

  On peut renommer le calcul :

  ```
  SELECT count(NE) as nbEmployes FROM emp ;
  ```

Usages spéciaux de la fonction count()

- ***Compter uniquement les valeurs non NULL***

  Le count() ne compte que les valeurs non NULL.

  Combien y a-t-il de personnes qui sont susceptibles d'avoir une commission ?

```
SELECT count(comm) FROM emp ;
```

- ***Compter les valeurs distinctes***

  On peut faire un disctinct en paramètre du count() et donc supprimer les doublons avant de compter.

  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 ;
```

Autres fonctions de groupe

https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html

Selon les SGBD, on peut trouver d'autres fonctions de groupe proposant des calculs statistiques plus élaborés : variance -> variance( ), écart-type -> stddev(), etc.

Clé primaire de la table résultat

Quand on a une fonction de groupe, la table résultat n'a pas la même clé primaire que la table de départ.

Quelle est la clé primaire de la table résultat ?

L'information récupérée, par exemple le nombre d'employés, est un attribut de quoi ?

La ligne projetée est un attribut d'un objet correspondant à l'ensemble de la table de départ, ici « les employés » considéré comme un objet d'un autre ensemble à déterminer. Ici, il s'agit par exemple de l'ensemble des entreprises. On projette les caractéristiques d'une entreprise.

On pourrait imaginer avoir plusieurs entreprises, chacune ayant un nom, et le nombre d'employés caractériserait le nom de l'entreprise qui serait la clé primaire.

Deux entreprises pourraient avoir le même nombre d'employés. Ca ne peut donc pas être la clé primaire.

Bilan

Avec une fonction de groupe :

CP = NULL**. Table =** Entreprise (si on veut préciser le nom de la table).

2. Calculs statistiques : agrégats ou GROUP BY

Les agrégats ou regroupement : le GROUP BY

Présentation de la clause GROUP BY

- ***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 1***

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

```
SELECT fonction, count(*) as nbEmployesParFonction,
                 avg(sal) as salaireMoyenParFonction
FROM emp
GROUP BY fonction;
```

fonction	nbEmployesParFonction	salaireMoyenParFonction
ANALYST	2	3000.000000
CLERK	4	1037.500000
MANAGER	3	2758.333333
PRESIDENT	1	5000.000000
SALESMAN	4	1400.000000

CP : fonction. La clé primaire de la table des résultats est « fonction ». C'est une table de fonctions.

➢ ***Exemple 2***

ou encore quels sont les salaires maximums de chaque département :

```
SELECT ND, max(sal) as salaireMaxParDepartement
FROM emp
GROUP BY ND ;
```

ND	salaireMaxParDepartement
10	5000.00
20	3000.00
30	2850.00

CP : ND. La clé primaire de la table des résultats est « ND». C'est une table de départements.

➢ ***Remarque***

Avec un GROUP BY 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

La syntaxe de la clause group by est la suivante :

```
4 : SELECT attributs du GB, liste de fdg(attribut)
1 : FROM table
2 : [WHERE ...]
3 : GROUP BY attributs du GB
5 : [ORDER BY fdg]
;
```

WHERE et ORDER BY sont facultatifs.

La numérotation correspond à l'ordre d'exécution du calcul.

Etapes du déroulement d'un GROUP BY

1) On prend tous les tuples de la table
2) On en élimine s'il y a un WHERE
3) Le GROUP BY consiste d'abord en un ORDER BY : les tuples restants sont triés selon les valeurs croissantes de la liste des attributs du group by ; ca génère des sous-ensembles.
4) On projette les attributs du GROUP BY : un DISTINCT est appliqué. On projette aussi des fonctions de groupe : elles s'appliquent alors à chaque sous-ensembles définis par le ORDER-BY initial.
5) On peut enfin faire un ORDER BY qui s'applique aux attributs du GROUP BY ou aux fonctions de groupes (on trie par moyenne de salaire, par exemple)

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.**

Eviter les GROUP BY sans fonction de groupe ⇔ DISTINCT

On veut répondre à la question suivante :

Quels sont les fonctions par département trié par département et par fonction.

- ***SOLUTION propre : DISTINCT***

```
SELECT distinct ND, fonction
FROM emp
ORDER BY ND, fonction ;
```

C'est la bonne façon de faire : elle permet de choisir l'ordre du tri.

ND	fonction
10	CLERK
10	MANAGER
10	PRESIDENT
20	ANALYST
20	CLERK
20	MANAGER
30	CLERK
30	MANAGER
30	SALESMAN

- ***SOLUTION « sale » : GROUP BY sans fonction de groupe***

```
SELECT ND, fonction
FROM emp
GROUP BY ND, fonction ;
```

Il faut éviter les group by sans fonctions de groupe.

GROUP BY trié

- ***Versions 5.5, 5.6 et 5.7***

  En version 5.5, le GROUP BY fait un tri selon les attributs du GROUP BY :

```
GROUP BY fonction ;
```

  équivaut à :

```
GROUP BY fonction
ORDER BY fonction ;
```

  Idem avec plusieurs attributs :

```
GROUP BY ND, fonction ;
```

  équivaut à :

```
GROUP BY ND, fonction
ORDER BY ND, fonction ;
```

- ***Version 8.0***

  Il n'y a plus d'ORDER BY automatique avec un GROUP BY.

Restrictions supplémentaires : la clause HAVING

Principes

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

Exemple : quelles sont les fonctions pour lesquelles travaillent plus de trois personnes :

```
SELECT fonction, count(*) nbEmployes
FROM emp
GROUP BY fonction
HAVING count(*) >=3;
```

ou encore:

```
SELECT fonction
FROM emp
GROUP BY fonction
HAVING count(*) >=3;
```

fonction	nbEmployes
CLERK	4
MANAGER	3
SALESMAN	4

Syntaxe SQL

La syntaxe de la clause group by est la suivante :

```
5 : SELECT attributs du GB, liste de  fdg(attribut)
1 : FROM table
2 : [WHERE ...]
3 : GROUP BY attributs du GB
4 : HAVING formule logique de sélection des fdg(attribut )
6 : [ORDER BY attributs ] ;
```

La numérotation correspond à l'ordre d'exécution du calcul.

Tri après un group by

Principe

On peut aussi trier les résultats.

La clause de tri est placée en dernier.

Exemple : quelles sont les fonctions pour lesquelles travaillent plus de trois personnes, triées par nombre de personnes décroissant et ordre alphabétique de function

```
SELECT fonction, count(*) nbEmployes
FROM emp
GROUP BY fonction
HAVING count(*) >=3
ORDER BY count(*) desc, fonction ;
```

fonction ▲ 2	nbEmployes
CLERK	4
SALESMAN	4
MANAGER	3

Syntaxe SQL

La syntaxe de la clause group by est la suivante :

```
5 : SELECT attributs du GB, liste de  fdg(attribut)
1 : FROM table
2 : [WHERE ...]
3 : GROUP BY attributs du GB
4 : HAVING formule logique de sélection des fdg(attribut )
6 : ORDER BY attributs ;
```

Usage d'alias

Usage standard

On peut utiliser un alias dans la clause ORDER BY :

```
SELECT fonction, count(*) AS nb
FROM emp
GROUP BY fonction
HAVING count(*) >=3
ORDER BY nb desc;
```

Usage non standard – spécificité MySQL

On peut utiliser un alias dans la clause HAVING :

```
SELECT fonction, count(*) AS nb FROM emp
GROUP BY fonction
HAVING nb >=3;
```

Attention, c'est une spécificité MySQL qui ne fonctionne que si la variable système SQL_MODE ne contient pas ONLY_FULL_GROUP_BY.

Cet usage n'étant pas standard, il vaut mieux l'éviter.

Le fonctionnement du SQL_MODE est du ONLY_FULL_GROUP_BY est détaillé dans le cours sur les jointures.

Ordre de projection des attributs

Rappel : cas des requêtes sans fonction de groupe :

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

Ou

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

Cas des requêtes avec fonction de groupe

On prend plutôt la deuxième version (TRI à la fin)

En général : attributs du group by = CP + CS

D'où l'ordre de projection suivant :

GB - Demandés – (Restriction=Having) – Tri

3. 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) et Group By --> tous les SGBD

Count (distinct attribut) dans un group by

Exemple

Combien d'employés et de départements par fonction

```
SELECT fonction, count(ne) nbEmp, count(distinct nd) nbDept
FROM emp
GROUP BY fonction;
```

fonction	nbEmp	nbDept
ANALYST	2	1
CLERK	4	3
MANAGER	3	3
PRESIDENT	1	1
SALESMAN	4	1

2 analyst répartis dans 2 départements

3 clerk (employé, commis) répartis dans 2 départements, etc.

Group concat --> MySQL

https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html - function_group-concat

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 fonction, group_concat(ne ORDER BY ne)  AS LesEmployes
FROM emp
GROUP BY fonction;
```

fonction	LesEmployes
ANALYST	7788,7902
CLERK	7369,7876,7900,7934
MANAGER	7566,7698,7782
PRESIDENT	7839
SALESMAN	7499,7521,7654,7844

With Rollup --> MySQL

https://dev.mysql.com/doc/refman/8.0/en/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 fonction, nd, count(*) nb
FROM emp
GROUP BY fonction, nd
WITH ROLLUP ;
```

fonction	nd	nb
ANALYST	20	2
ANALYST	*NULL*	2
CLERK	10	1
CLERK	20	2
CLERK	30	1
CLERK	*NULL*	4
MANAGER	10	1
MANAGER	20	1
MANAGER	30	1
MANAGER	*NULL*	3
PRESIDENT	10	1
PRESIDENT	*NULL*	1
SALESMAN	30	4
SALESMAN	*NULL*	4
NULL	*NULL*	14

Les résultats présentent le nombre d'employés par fonction et n°de département et présentent, en gris, le total des employés par fonction.

La dernière ligne donne le total des employés concernés par la requêtes.

- ***Equivalents ORACLE :***

  GROUP BY ROLLUP, GROUP BY GROUPING SETS, ROUP 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 part d'une table des ventes avec un montant, une année et un pays :

idVentes	Montant	Pays	Année
1	1000	France	2015
2	150	France	2015
3	300	Allemagne	2015
4	250	Angleterre	2017
Etc.			

On veut un tableau de total des ventes par année et par pays
C'est un tableau croisé : on croise les années, en ligne, avec les pays, en colonne.

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

ANNEE         FRANCE   ALLEMAGNE   ANGLETERRE
-------------------------------------------
2015          29825     26825        27825
2016          25825     22825        29825
2017          32825     29000        28825
```

(*) 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
```

(*) 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 leur 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;
```

5 - VARIABLES DANS LES REQUETES

Présentation

Principe général

- Les requêtes SQL vont contenir des variables.
- Par exemple, on cherche les employés qui ont telle fonction. Le nom de la fonction se retrouve dans une variable.
- Ce sera utilisé par les langages serveur, type PHP ou Java.
- On peut aussi passer des variables au niveau de la calculette, avec des techniques variables selon le SGBD

Exemple PHP

```
$reqSQL='Select NE, nom, job where job= :v_job;

$requete=$bdd->prepare($reqSQL)

$requete->execute(array(
          'v_job'=> $_GET['get_job']
));
while($ligne=$requete->fetch()){
          echo '<p>' ;
          echo $ligne[NE] .' '. $ligne[nom] .' '. $ligne[nom] ;
          echo '</p>' ;
}
```

- La syntaxe est un peu particulière : v_job est un « alias » qui prendra finalement la valeur de $_GET('get_job'), c'est-à-dire une information transmise par exemple par un formulaire HTML.
- Le while et le fetch permettent de parcourir les lignes du résultat du select.

Exemples MySQL : set @variable

Exemple 1 :

```
Set @fonction = 'manager' ;

SELECT NE, nom, fonction
FROM employes
WHERE fonction = @fonction;
```

@fonction est une variable visible par le client uniquement.

« Set » permet de donner une valeur à une variable.

- ***Résultats :***

```
+------+-------+----------+
| NE   | nom   | fonction |
+------+-------+----------+
| 7698 | BLAKE | MANAGER  |
| 7782 | CLARK | MANAGER  |
| 7566 | JONES | MANAGER  |
+------+-------+----------+
```

Exemple 2 :

```
Set @nom = 'A%' ;

SELECT NE, nom, fonction
FROM employes
WHERE fonction like @nom;
```

- ***Résultats :***

```
+------+-------+----------+
| NE   | nom   | fonction |
+------+-------+----------+
| 7902 | FORD  | ANALYST  |
| 7788 | SCOTT | ANALYST  |
+------+-------+----------+
```

6 – LA NOTION DE VUE

Présentation

- Une vue est un **nouvel objet enregistré dans la BD**, au même titre qu'une table ou qu'un utilisateur.
- Une vue est une **vue sur une table** (ou plusieurs). Elle est **définie par un SELECT**.
- Un SELECT renvoie une table. **Une vue correspond à la table des données renvoyées par son SELECT de définition**.
- **On peut faire des SELECT sur une vue comme sur une table**.

Création d'une vue : CREATE or REPLACE VIEW

- On veut créer la vue des employés CLERK, sans les informations de la commission.
- On utilise un CREATE or REPLACE : si la vue existe déjà, elle est remplacée.
- Le select pour ces employés est :

```
SELECT NE, nom, fonction, datemb, ND
FROM emp
WHERE fonction = 'CLERK';
```

Pour créer la vue on écrit :

```
CREATE OR REPLACE VIEW empClerk AS
SELECT NE, nom, fonction, datemb, ND
FROM employes e
WHERE fonction = 'CLERK';
```

Utiliser une vue

Principes

- La vue est un SELECT.
- On va pouvoir relancer ce SELECT avec un simple SELECT * FROM maVue ;
- On pourra utilise la vue comme n'importe quelle table.
- On peut même ajouter, modifier ou supprimer des tuples à partir d'une vue (DML).

Tous les tuples de la vue

- Si on veut tous les employés de la vue « empClerk » on écrit :

```
SELECT * FROM empClerk;
```

Certains tuples de la vue

- Si on veut les employés de la vue « empClerk » dont le salaire est supérieur à 1000 on écrit :

```
SELECT * FROM empClerk
WHERE sal > 1000 ;
```

Lister les vues crées : SHOW TABLES

- Les vues sont des tables.
- Un « show tables » liste les tables et les vues.

Suppression d'une vue : DROP VIEW

```
DROP VIEW empClerk ;
```

Modification d'une vue

- Il n'y a pas de modification d'une vue. Pour modifier une vue, on la supprime et on la recrée.

Afficher le code d'une vue : SHOW CREATE VIEW

```
SHOW CREATE VIEW empClerk \G
```

- Le résultat demande un effort de lecture, mais on retrouve bien le code de création de la vue.

```
mysql> SHOW CREATE VIEW empClerk \G
*************************** 1. row ***************************
                View: empclerk
         Create View: CREATE ALGORITHM=UNDEFINED
DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `empclerk` AS select
`e`.`NE` AS `NE`,`e`.`NOM` AS `nom`,`e`.`FONCTION` AS
`fonction`,`e`.`DATEMB` AS `datemb`,`e`.`ND` AS `ND` from `employes` `e`
where (`e`.`FONCTION` = 'CLERK')
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
1 row in set (0,00 sec)1 row in set (0,00 sec)
```

- Mis en forme on obtient :

```
select
   `e`.`NE` AS `NE`,
   `e`.`NOM` AS `NOM`,
   `e`.`FONCTION` AS `FONCTION`,
   `e`.`DATEMB` AS `DATEMB`,
   `e`.`SAL` AS `SAL`,
   `e`.`ND` AS `ND`,
from `employes` `e`
where (`e`.`FONCTION` = 'CLERK')
```

7 - BILAN SYNTAXIQUE

Ordre des clauses du Select mono-table

```
4 : SELECT [DISTINCT] liste d'attributs
1 : FROM table
2 : [ WHERE formule de restriction ]
3 : [ GROUP BY liste d'attributs
5 : [ HAVING formule de restriction ] ]
6 : [ 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 attributs

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.