BASE DE DONNÉES

JOURNÉE 2 – APRES MIDI

1

Karine BRIFAULT

BASE DE DONNÉES
SQL AVANCÉ

- La majeur partie des bases de données incluent des fonctions complémentaires dans le SQL.
- ATTENTION : la plus part ne sont pas normalisées ⇔change d'une base à l'autre
- Le risque ⇔refaire toutes ses requêtes si changement de base de données.

BASE DE DONNÉES
SQL AVANCÉ

- Une liste est disponible sur http://dev.mysql.com/doc/refman/5.7/en/func-op-summary-ref.html
- Nous allons voir les plus utiles.
- En cas d'erreur, la plus part des fonctions retournent une valeur NULL.

BASE DE DONNÉES
SQL AVANCÉ

- Mathématiques :
  - **COS**, **SIN**, **TAN**, **ACOS**, **ASIN**, **ACOS**, **ATAN**, **ATAN2**, **COT** : fonctions algébriques standards
  - **LOG**, **LOG2**, **LN** : fonctions logarithmiques
  - **ABS**(*X*) : récupère la valeur absolue de X
  - **CEIL**(X) ou **CEILING**(X) : retourne l'entier le plus proche en valeur strictement supérieure (1,2 => 2)
  - **FLOOR**(X) : retourne l'entier le plus proche en valeur strictement inférieure (1,2 => 1)
  - **ROUND**(X) : l'entier le plus proche de X (1,2 => 1, 1,6 => 2)
  - **POW** ou **POWER**(X, Y) : donne X puissance Y
  - **SIGN**(X) : donne le signe de X
  - **RAND** : donne un chiffre flottant aléatoire : 0<=x<1.0

BASE DE DONNÉES
SQL AVANCÉ

- Dates :
  - **ADDATE**, **DATE_ADD**, **DATE_SUB** : permet d'ajouter/retirer du temps à une date :
    - **DATE_ADD**('2008-01-02', **INTERVAL** 31 **DAY**)
  - **ADDTIME** ajoute du temps à un temps :
    - **ADDTIME**('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
  - **CURDATE**(), **CURRENT_DATE**(), **CURRENT_DATE** : donne la date courante du **serveur** ('2008-06-13').
  - **CURRENT_TIME**(), **CURRENT_TIME**, **CURTIME**() : donne le temps courant du **serveur** ('23:50:26').

BASE DE DONNÉES
SQL AVANCÉ

- Dates :
  - **DATE_FORMAT**(date,format) : permet de formater une date selon son choix
    - **DATE_FORMAT**('2009-10-04 22:23:00', '%W %M %Y'); = 'Sunday October 2009'
  - **DATEDIFF** : soustrait deux dates pour donner le nombre de jours entre les deux
  - **DAY**, **DAYOFMONTH**, **DAYOFYEAR**, **DAYOFWEEK**, **MONTH**, **YEAR**, **HOUR**, **MINUTE**, **SECOND**, **WEEK**, **WEEKDAY**, **WEEKOFYEAR** : permet de récupérer une information ciblée sur la date (le jour, l'année, …)
    - Raccourcis à l'utilisation de EXTRACT(unit FROM date)
      - **EXTRACT(YEAR FROM '2009-07-02'); ⇔ YEAR('2009-07-02')**

BASE DE DONNÉES
SQL AVANCÉ

- Dates :
  - **LAST_DAY** : donne le dernier jour du mois
  - **NOW**, **LOCALTIME**, **LOCALTIMESTAMP** : donne la date et le temps du **serveur**.
  - **STR_DO_DATE**(chaine, format) : transforme une chaine de caractères en date (contraire de DATE_FORMAT)
    - **STR_TO_DATE('01,5,2013','%d,%m,%Y');** ⇔ '2013-05-01'

BASE DE DONNÉES
SQL AVANCÉ

- Conversions :
  - **BINARY** : pour transformer une chaine en binaire
    - select BINARY 'abcd'; ⇔ BLOB(61,62,63,64)
  - **CAST**(expr AS type) ou CONVERT (expr ,type) transforme une expression en autre chose
    - select convert(1.5, CHAR(3)); ⇔ '1.5‘
    - Type = BINARY(N) / CHAR(N) / DATE / DATETIME / DECIMAL(M[,D]) / SIGNED / TIME / UNSIGNED
  - **CONVERT**(expr USING transcod) : permet de passer d’un encodage à un autre
    - SELECT CONVERT('abc' USING utf8);

BASE DE DONNÉES
SQL AVANCÉ

- Cryptage :
  - **AES_DECRYPT**, **AES_ENCRYPT** : utilisation de AES
  - **DES_DECRYPT**, **DES_ENCRYPT** : utilisation de DES
  - **COMPRESS**, **UNCOMPRESS** : Zip / dezip une chaine de caractères
  - **MD5**, **SH1**, **SHA1**, **SHA2** : calcule un MD5, SHAx
  - **PASSWORD** : fabrique un password
  - **FROM_BASE64** : décode une chaine en base 64
  - **TO_BASE64** : encode une chaine en base 64

BASE DE DONNÉES
SQL AVANCÉ

- Chaines de caractères :
  - **CHAR_LENGTH**(str), **CHARACTER_LENGTH**(str) : donne la taille de la chaine
  - **CONCAT** : concatène n éléments
    - **SELECT CONCAT('My', 'S', 'QL');** ⇔ 'MySQL'
    - **SELECT CONCAT(14.3);** ⇔ **'14.3'**
  - **CONCAT_WS** : concatène deux chaines avec un séparateur
  - **FORMAT**, **HEX** : transforme un chiffre en chaine formatée
  - **INSERT**(str, pos, length, str2) : insert *str2* dans *str* à la position *pos* sur une longueur *length*
    - **SELECT INSERT('Quadratic', 3, 4, 'What');** ⇔ 'QuWhattic'
  - INSTR(str, what), LOCATE, POSITION : donne l'index de la première occurrence de *what* dans *str*
    - **SELECT INSTR('foobarbar', 'bar');** ⇔ 4

BASE DE DONNÉES
SQL AVANCÉ

- Chaines de caractères :
  - **LCASE**(str), **LOWER** : met en minuscule
  - **UCASE**(str), **UPPER** : met en majuscule
  - **LTRIM**(str), **RTRIM**(str) : enlève les caractères espaces à gauche (LeftTRIM) ou à droite (RightTRIM)
  - **TRIM**(str) : enlève les caractères espaces à gauche et à droite
  - **REPEAT** : fabrique une chaine en répétant la valeur donnée
  - **REPLACE** : remplace une occurrence dans une chaine
  - **SPACE** : fabrique une chaine avec un nombre donné d'espace
  - **SUBSTR**, **SUBSTRING** : récupère une sous chaine
    - **SELECT SUBSTRING('Quadratically',5,6);** ⇔ 'ratica'
  - **STRCMP** : compare deux chaines, retourne 0 si elles sont égales, -1 si la première est < à la seconde, +1 si la première est supérieure à la seconde

BASE DE DONNÉES
SQL AVANCÉ

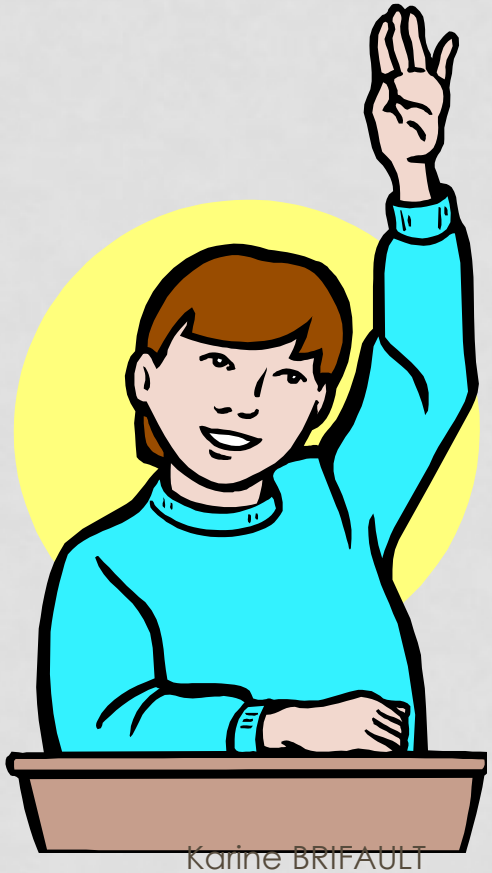
- Utilitaires :
  - **CURRENT_USER**() : utilisateur@nom de machine
  - **USER**() : utilisateur
  - **FOUND_ROWS** : nombre d'élément renvoyé par un select
    - SELECT SQL_CALC_FOUND_ROWS * FROM *tbl_name* WHERE id > 100 LIMIT 10; SELECT FOUND_ROWS(); ⇔ 10
  - **LAST_INSERT_ID** = dernier id inséré sur une colonne autoincrement
  - **ROW_COUNT** : nombre de ligne mises à jour

BASE DE DONNÉES
NOTION D'INDEX

- Se base sur le même principe qu'un index dans un livre.
- Est un élément de la base de données (comme les tables, les schémas, …), il peut être explicitement nommé.
- Permet d'optimiser / accélérer les recherches.

BASE DE DONNÉES
NOTION D'INDEX

- Pourquoi ne pas coller des indexes partout alors ?



BASE DE DONNÉES
NOTION D'INDEX

- Fabriquer un index :
  - Choisir la/les colonne(s) d'une table concerné(s)
  - Réfléchir au typage de l'index
    - UNIQUE : un élément ne peut y être qu'une seule fois
    - ASC/DESC : les éléments sont rangés par ordre croissant ou décroissant
  - Pour certaine base, il est possible d'ajouter d'autres options liées à l'espace physique occupé par les indexes

- En SQL :
  - **CREATE** *UNIQUE* **INDEX** NomIndex **ON** NomTable (NomColonne1 *ASC/DESC*, NomColonne2 *ASC/DESC*, …)

BASE DE DONNÉES
NOTION D'INDEX

- Une clef primaire ⇔une forme d'index UNIQUE

- Où poser des indexes
  - Regarder ses requêtes SQL
  - Localiser les colonnes ciblées
  - Ajouter un index si la requête est très souvent appelée

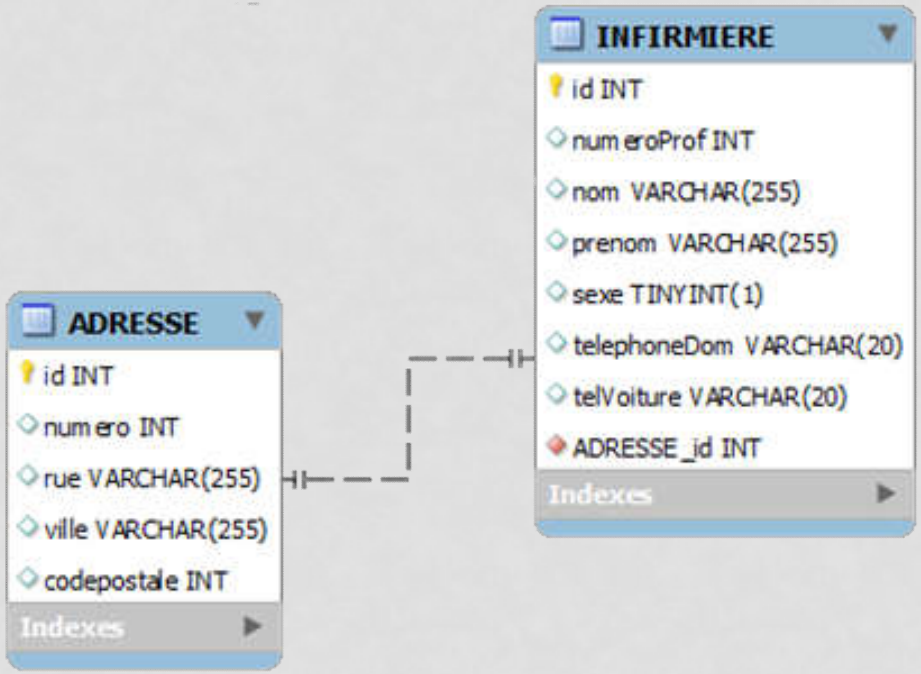
- Le gain en performance peut être énorme x2 ou x3, lors d'un select, donc ne pas les négliger.

BASE DE DONNÉES
NOTION DE VUE

- Une vue est une forme particulière de table virtuelle.
- On parle de vue car il s'agit simplement d'une représentation des données dans le but d'une exploitation visuelle.
- Elle est le résultat d'un select. C'est donc un agrégat de une ou plusieurs tables.

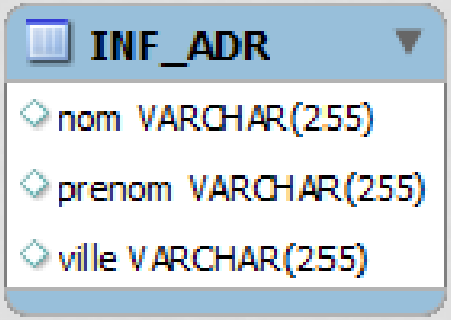
BASE DE DONNÉES
NOTION DE VUE

- Si je fais une vue entre les deux tables suivantes sur les colonnes Inf(nom, prenom) et adr(ville):



BASE DE DONNÉES
NOTION DE VUE

- C'est comme si j'avais fait une table :

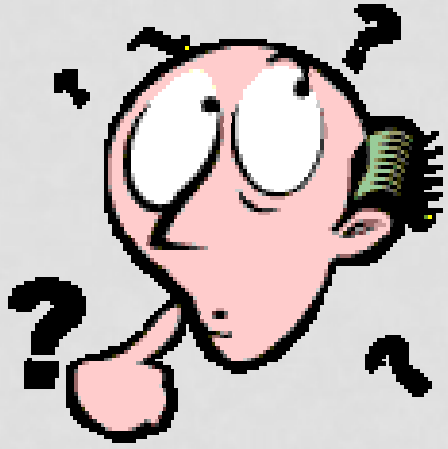


- Notez qu'une vue n'a pas de clef primaire, en fait, elle n'a même aucune contrainte (pas de clef étrangère, pas de colonne nulle ou non, …)

BASE DE DONNÉES
NOTION DE VUE

- Une vue = résultat d'un select

- Mais qui lance le select pour garantir la cohérence des données dans la vue ?



BASE DE DONNÉES
NOTION DE VUE

- Création d'une vue :

- **CREATE VIEW** NomDeLaVue (NomCol1, NomCol2, …) **AS SELECT …**

BASE DE DONNÉES
NOTION DE VUE

- Quand faire des vues
  - Pour une problématique de sécurité
  - Pour une problématique d'optimisation d'accès aux données
  - Pour une problématique de restructuration des données

- Attention :
  - le cout d'une vue se fait lors de l'insert/update/delete

BASE DE DONNÉES
LES TRIGGERS

- Un déclencheur (*trigger*) permet d'associer automatiquement des traitements chaque fois qu'un certain type d'opération est exécuté dans la base.

- Ce traitement peut s'exécuter avant ou après une commande INSERT, UPDATE ou DELETE.

BASE DE DONNÉES
LES TRIGGERS

- Syntaxe du Trigger en MySQL

```
CREATE TRIGGER NomTrigger
BEFORE INSERT ON NomTable
FOR EACH ROW
BEGIN
        InstructionSQL 1;
        InstructionSQL 2;
        InstructionSQL 2;
END
```

BASE DE DONNÉES
LES TRIGGERS

- **BEFORE** ou **AFTER** pour indiquer quand exécuter le trigger
- **INSERT**, **UPDATE** ou **DELETE** pour indiquer l'évènement sur lequel le trigger est branché.
- IMPORTANT : Un seul trigger possible pour les trois informations (table, quand, évènement)

BASE DE DONNÉES
LES TRIGGERS

- Dans la trigger vous pouvez faire usage de l'élément ajouté/mis à jour/supprimé avec les deux mots clefs
  - **NEW** : représente le nouvel état de la ligne modifiée
  - **OLD** : représente l'ancien état de la ligne modifiée

- Vous pouvez ainsi, dans une trigger, modifier l'état de l'élément avec le mot clef SET :
  - SET NEW.nomColonne = valeur;

BASE DE DONNÉES
LES TRIGGERS

- Exemple le plus courant d'utilisation de triggers :
  - Mettre un champ à la date du jour automatiquement
  - Utiliser un mécanisme d'historisation des états
  - Supprimer des éléments en cascade

- Attention : la syntaxe d'une trigger est souvent fortement liée à la base de données.