SGBD - MYSQL

(SGBD – SQL - MYSQL)

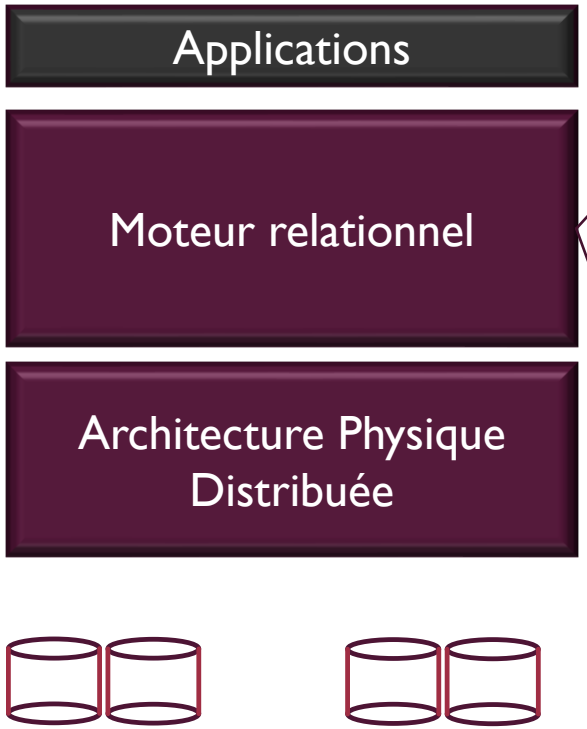
SGBDR

Quelques rappels…

INTRODUCTION

- Une base de données relationnelle :
  - Est une collection de données organisées formellement sous la forme de relations, le terme relation devant être entendu au sens mathématique de la théorie des relations elle-même construite à partir de la théorie des ensembles.
  - S'appuie sur un moteur relationnel, capable de manipuler des relations, c'est-à-dire des ensembles de données. En d'autres termes, dans le monde du relationnel on ne manipule pas des éléments mais des ensembles ! Le moteur relationnel traite les questions qu'on lui formalise sous la forme d'opérations d'algèbre relationnelle. Plus simplement, un moteur relationnel sait répondre à la question suivante : « Parmi l'ensemble des relations composant ma base de données, trouve moi l'ensemble (relation) qui a les caractéristiques suivantes ».
  - En corolaire, en relationnel on ne manipule pas des 'enregistrements' mais des ensembles entiers.
  - Modèle relationnel inventé par Edgar Frank « Ted » Codd en 1970 et construit et théorisé jusqu'au-delà des années 2000 (6NF en 2003)….

ELÉMENTS D'ARCHITECTURE



- Réalise l'indépendance entre le modèle logique (relationnel) et l'implantation physique (serveur, grappes, disques, fichier, organisation logique à l'intérieur du fichier, découpage physique du fichier, gestion index,…),
- Assure l'intégrité des données :
  - Gestion de clés étrangères,
  - Intégrité de domaine,
  - Gestion de transactions,
  - Unicité
  - …….
- Interface d'accès via un langage de requêtes intégrant l'algèbre relationnelle (ex SQL…)
- Optimisation des requêtes.

BASES DE DONNÉES RELATIONNELLES…..

- Alors oui, les SGBDr s'occupent du stockage des données, MAIS PAS QUE !…
- Quelques SGBDr :
  - Oracle RDBMS,
  - IBM DB2,
  - Microsoft SQL Server,
  - Oracle MySQL,
  - Maria DB,
  - PostgreSQL

COLLECTION DE DONNÉES ORGANISÉE EN TABLES

- L'organisation des tables (*) doit répondre à des règles de normalisation que l'on appelle les formes normales

  - Il existe 6 niveaux de Formes Normales 1NF, 2NF,…6NF plus une forme normale, dite de BOYCE-CODD qui, dans la pratique, rend inutile la vérification de la conformité 2NF et 3NF.
  - L'objectif du cours n'est pas de présenter formellement cette normalisation. La bible française sur le sujet est celle de François de Sainte Marie que l'on trouve ici :

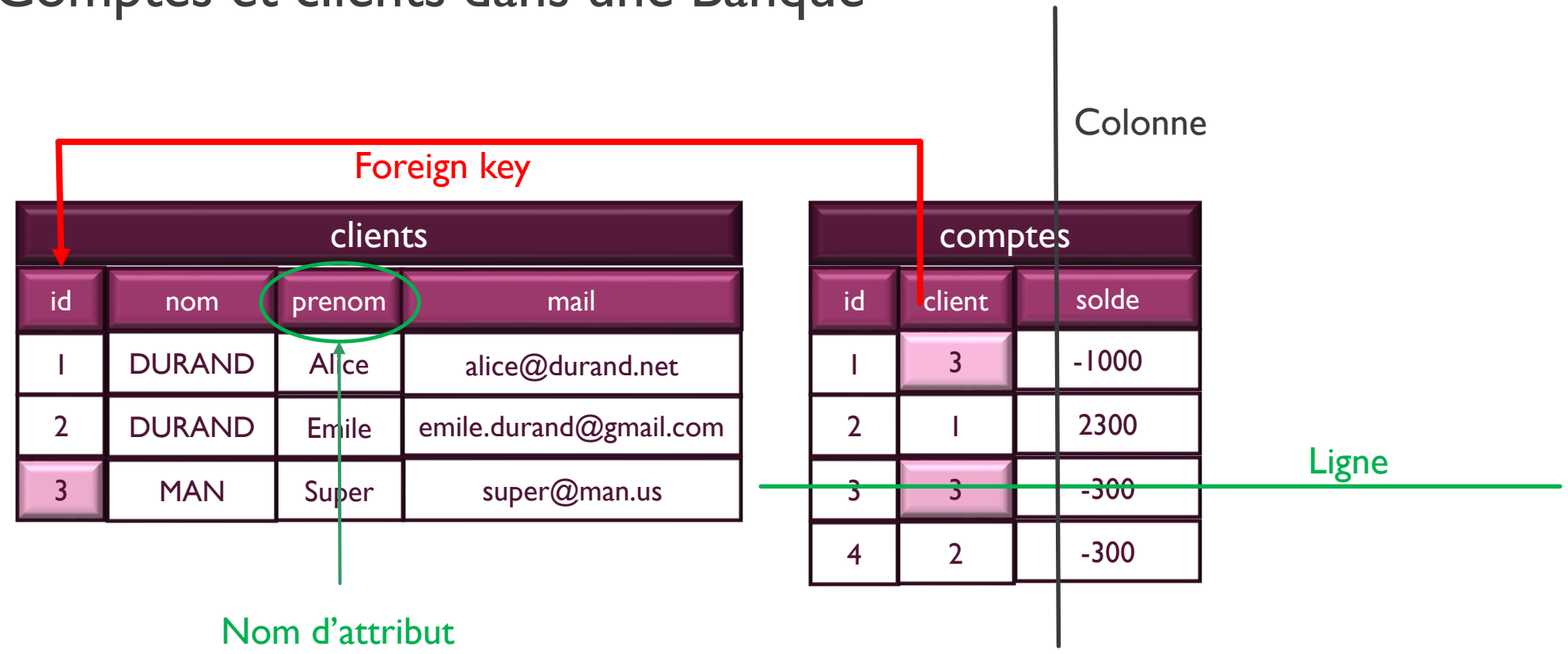
    https://fsmrel.developpez.com/basesrelationnelles/normalisation

    - Existe en version PDF (En bas à droite de la première page)
    - Relisez la bible une fois, deux fois, dix fois, il restera toujours quelque chose de nouveau à comprendre !!!

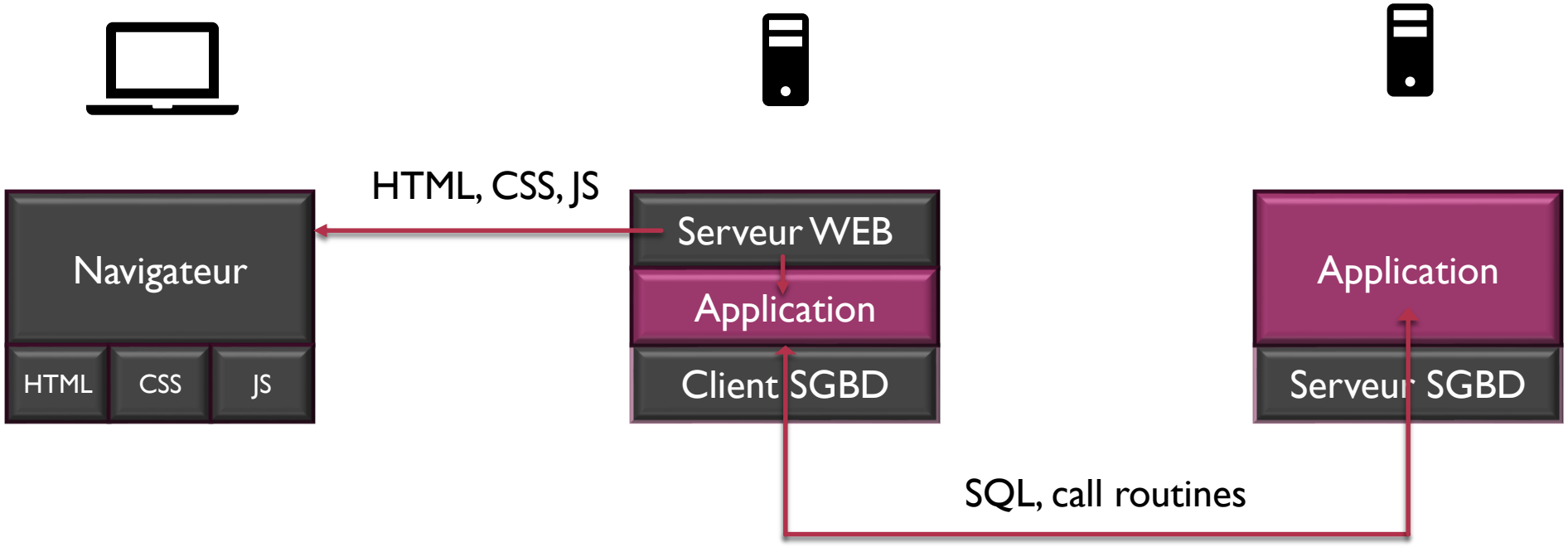
(*) *Dans le monde du relationnel on utilise plutôt le terme de relvar…*

EXEMPLES DE TABLES 'EN RELATIONS' DANS UN SGBDR

- Comptes et clients dans une Banque



SGBD ET ARCHITECTURES N TIERS



SQL

DDL, DML, Stored Programs…

STRUCTURED QUERY LANGAGE - STANDARD DES SGBDR

- SQL - AML (SQL – Account Management Language)
  - Gestion des utilisateurs
    - CREATE USER, ALTER USER, GRANT
- SQL - DDL (SQL - Data Definition Language)
  - Définition des tables
    - CREATE TABLE, DROP TABLE, ALTER TABLE
- SQL – DML (SQL – Data Manipulation Language)
  - Comme son nom l'indique : langage de manipulation de données
    - INSERT, UPDATE, DELETE, SELECT, START TRANSACTION/BEGIN, COMMIT, ROLLBACK, SAVEPOINT, ROLLBACK TO SAVEPOINT,…
- SQL – Stored Programs
  - Functions, Procedures, Triggers, Event Scheduler…

SQL - AML (SQL – ACCOUNT MANAGEMENT LANGUAGE)

- CREATE USER [IF NOT EXISTS]
  user [*auth_option*] [, *user* [*auth_option*]] ...
  [REQUIRE {NONE | *tls_option* [[AND] *tls_option*] ...}] [WITH
  resource_option [*resource_option*] ...] [*password_option* | *lock_option*]

On se reportera à la documentation MySQL :

https://dev.mysql.com/doc/refman/5.7/en/create-user.html

SQL - DDL (SQL - DATA DEFINITION LANGUAGE) (1/2)

- CREATE TABLE

  On se reportera à la documentation MySQL :
  https://dev.mysql.com/doc/refman/5.7/en/create-table.html

  A titre d'exemples :

```
delimiter $$
DROP        TABLE    IF    EXISTS    `it-akademy`.`clients`                    $$
CREATE      TABLE                    `it-akademy`.`clients`
(
      `id`        integer(10)     unsigned    NOT NULL     AUTO_INCREMENT    ,
      `nom`       varchar(128)                NOT NULL                       ,
      `prenom`    varchar(128)                NOT NULL                       ,
      `email`     varchar(255)                NOT NULL                       ,

  PRIMARY KEY (`id`)                                                         ,
  KEY `clients_nom_prenom_idx` (`nom`,`prenom`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8                                         $$
```

SQL - DDL (SQL - DATA DEFINITION LANGUAGE) (2/2)

- CREATE TABLE – Gestion des FK

```
delimiter $$
DROP      TABLE     IF    EXISTS    `it-akademy`.`comptes`                    $$
CREATE    TABLE                     `it-akademy` .`comptes`
(
          `id`      int(10) unsigned  NOT NULL      AUTO_INCREMENT  ,
          `clients` int(10) unsigned  NOT NULL                      ,
          `solde`   decimal(16,4)     NOT NULL DEFAULT '0.0000'     ,

 PRIMARY KEY (`id`)                                                 ,
 KEY `comptes_client_FK_idx` (`clients`)                            ,

 CONSTRAINT `comptes_client_FK`
      FOREIGN KEY (`clients`) REFERENCES `clients` (`id`)
           ON DELETE CASCADE
           ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8                                $$
```

MYSQL : TYPES DE DONNÉES ÉLÉMENTAIRES (1/3)

- ENTIERS
  - **tinyint (8 bits), smallint (16 bits) , mediumint (24 bits), int (32 bits), bigint (64 bits)**
  - **(unsigned)**
- BIT(M)
- BOOL (BOOLEAN)
  - Synonyme de tinyint(1), 0 = FALSE
- DECIMAL(M,D)
  - Nombre decimal avec précision fixe (M digits, D décimales)

MYSQL : TYPES DE DONNÉES ÉLÉMENTAIRES (2/3)

- VIRGULE FLOTTANTE
  - **float** (précision ≈ 7 décimales), **double** (précision 15 ≈ décimales)
  - **(unsigned)**
- Type DATE
  - DATE
  - DATETIME
  - TIMESTAMP
  - TIME
  - YEAR

MYSQL : TYPES DE DONNÉES ÉLÉMENTAIRES (3/3)

- Types String
  - CHAR, VARCHAR, BINARY, VARBINARY
  - TEXT (de tiny à long…)
  - BLOB (Binary String, de tiny à long…)
  - ENUM

SQL – MYSQL – TP1

- Configurer MySQL
- Télécharger MySQL Workbench sur le client
- Avec MySQL Workbench :
  - Créer un user ‘the_academy’
  - Créer la base ‘it-akademy’
  - Assigner des droits à ‘the_academy’ sur la base ‘it_akademy’
  - Créer les tables ‘clients’ et ‘comptes’

OPÉRATIONS RELATIONNELLES ET SQL (1/9)

- Opérateurs relationnels

Opération	Notation fonctionnelle	Notation algébrique
RESTRICTION	RESTRICT R_x (prédicat)	$\sigma_{\text{prédicat}}(R_x)$
PROJECTION	PROJECT(R_x / {liste attributs})	$\Pi_{\text{liste attributs}}(R_x)$
PRODUIT CARTESIEN	PRODUCT(R_x, R_y)	$R_x \times R_y$
JOINTURE	JOIN(R_x, R_y / prédicat)	$R_x \bowtie R_{y\ \text{Predicat}}$
UNION	UNION (R_x, R_y)	$R_x \cup R_y$
INTERSECTION	INTERSECT (R_x, R_y)	$R_x \cap R_y$
DIFFERENCE	MINUS (R_x, R_y)	$R_x - R_y$
DIVISION	DIVIDEBY (R_x, R_y)	$R_x \div R_y$

OPÉRATIONS RELATIONNELLES ET SQL (2/9)

- Principe de fermeture

  Opérer sur des tables (relationnelles) permet de produire de nouvelles tables qui, à leur tour pourront etre utilisées comme opérandes. Il y a cohérence totale entre les tables qui constituent ainsi un système fermé.

  Table = Operateur(Table1, Table2)

  ⇨ **SQL ne manipule que des tables et ne restitue que des tables.**

OPÉRATIONS RELATIONNELLES ET SQL (3/9)

- Thêta-restriction (Sélection)

  Cette opération permet, à partit d'une table releationnelle T, de produire une nouvelle table relationnelle T' dont les tuples sont les images de certains tuples de T.

  Si : - θ est un opérateur de comparaison scalaire (=, >, …),

  - A un attribut d'une table relationnelle T,

  - B un autre attribut de T ou une constante,

  On appelle thêta-restriction l'ensemble des tuples de T qui vérifie une condition donnée sous la forme A θ B : T' = RESTRICT T (A θ B)

  - Thêta-restriction en SQL : SELECT

    SELECT * FROM clients

    WHERE nom = 'DURAND' ;

id	nom	prenom	mail
1	DURAND	Alice	alice@durand.net
2	DURAND	Emile	emile.durand@gmail.com

OPÉRATIONS RELATIONNELLES ET SQL (4/9)

- Projection

La projection d'une table relationnelle $T(A_1, A_2, \ldots A_n)$ sur un sous ensemble $(A_a, A_b, \ldots A_m)$ d'attributs de T est une table relationnelle T' $(A_a, A_b, \ldots A_m)$ dont les tuples sont obtenus par élimination des valeurs des attributs de T qui ne figurent dans le schéma de T'. Les éventuels doublons dont éliminés.

T' = PROJECT (T / $\{A_a, A_b, \ldots A_m\}$)

- Projection en SQL :

SELECT nom, mail FROM clients ;

nom	mail
DURAND	alice@durand.net
DURAND	emile.durand@gmail.com
MAN	super@man.us

OPÉRATIONS RELATIONNELLES ET SQL (5/9)

- Union
  - Tables union-compatibles

    En pratique 2 tables relationnelles T1 et T2 sont union-compatibles si elles-ont exactement le même schéma.
  - Définition :

    L'union de 2 tables relationnelles T1 et T2 union-compatibles est une table T3, union-compatible, dont les tuples sont les images de tous les tuples de T1 et de tous les tuples de T2. Les éventuels doublons dont éliminés.

T3 = UNION (T1, T2)

- Union en SQL : **T1 UNION T2;**

OPÉRATIONS RELATIONNELLES ET SQL (6/9)

- Différence

  La différence entre 2 tables relationnelles T1 et T2 union-compatibles est une table T3, union-compatible, dont les tuples sont les images de tous les tuples appartenant à T1 et n'appartenant pas à T2.

  T3 = MINUS (T1, T2)

  - Différence en SQL : **T1 MINUS T2;**

    Opérateur non défini dans MySQL

    Simulation avec d'autres opérateurs disponibles

OPÉRATIONS RELATIONNELLES ET SQL(7/9)

- Intersection

  L'intersection entre 2 tables relationnelles T1 et T2 union-compatibles est une table T3, union-compatible, dont les tuples sont les images de tous les tuples communs à T1 et T2 à T2.

  T3 = INTERSECT (T1, T2)

  - Intersection en SQL : **T1 INTERSECT T2;**

    Opérateur non défini dans MySQL

    Simulation avec d'autres opérateurs disponibles

    Remarque : $T1 \cap T2 = T2 - (T2 - T1)$

OPÉRATIONS RELATIONNELLES ET SQL (8/9)

- Produit cartésien

  Le produit cartésien entre 2 tables relationnelles T1, de degré n_1 et de cardinalité c_1, et T2 , de degré n_2 et de cardinalité c_2, est une table T3 de degré $n_1 = n_1 + n_2$ et de cardinalité $c_3 = c_1 * c_2$, telle que chaque tuple de T3 est obtenu en concaténant chaque tuple de T1 avec chaque tuple de T2.

  T3 = PRODUCT (T1, T2)

  - Produit cartésien en SQL : Utilisation d'une jointure sans prédicat (CROSS JOIN)
  - Intérêt d'un produit cartésien 'pur' ?

OPÉRATIONS RELATIONNELLES ET SQL (9/9)

- Jointure interne

  La jointure (interne) entre 2 tables relationnelles T1 et T2 , est une table relationnelle T3 qui est un sous-ensemble du produit cartésien de T1 et T2 vérifiant un prédicat lui-même combinaison de prédicats élémentaires de la forme : **$T1.X_i$ thêta $T2.Y_j$** où :

  - Xi et Yj sont respectivement des attributs de T1 et T2 (dont les valeurs partage le même domaine !)
  - théta est un opérateur de comparaison scalaire (=, !=, >,..)

  On parle en fait de **thêta-jointure** et, si l'opérateur scalaire est celui de l'égalité, d'**équi-jointure**.

  Jointure en SQL :

```
SELECT      Xi,..Xn,Yj, …Yp
FROM        T1
INNER JOIN  T2
ON          Prédicat
```

SQL – MYSQL – TP2 (1/2)

- Afficher la liste des comptes clients sous la forme

nom	prenom	NoCompte	solde
DURAND	Alice	2	2300
DURAND	Emile	4	-300
MAN	Super	3	-300
MAN	Super	1	-1000

SQL – MYSQL – TP2 (2/2)

- Afficher la liste des comptes (même schéma) inférieurs à une certaine valeur
- Variable de session dans un script
  - Partagée par toute la session,
  - Syntaxe : @nomVariable
  - Affectation : SET @nomVariable = Expression ; / * on termine par un délimiteur par défaut ';' */

DML – LECTURE DE DONNÉES (1)

- SELECT – Syntaxe MySQL (Sans jointure)

SELECT [ALL | DISTINCT | DISTINCTROW] *select_expr* [, *select_expr* ...]

[FROM *table_references*

[WHERE *where_condition*]

[GROUP BY {*col_name* | *expr* | *position*} [ASC | DESC], ...

[HAVING *where_condition*]

[ORDER BY {*col_name* | *expr* | *position*} [ASC | DESC], ...]

[LIMIT {[*offset*,] *row_count* | *row_count* OFFSET *offset*}]

[INTO OUTFILE '*file_name*' [CHARACTER SET *charset_name*] *export_options* | INTO DUMPFILE '*file_name*' | INTO *var_name* [, *var_name*]]

[FOR UPDATE | LOCK IN SHARE MODE]] ;

DML LECTURE DE DONNÉES (2)

- SELECT – Exemple (Sans jointure)

```
SELECT      nom, prenom, mail
FROM        it-akademy.clients
WHERE       nom = 'DURAND'
ORDER BY    prenom DESC ;
```

nom	prenom	mail
DURAND	Emile	emile.durand@gmail.com
DURAND	Alice	alice@durand.net

DML – APARTÉ SUR LES NULL (1/2)

- Tous les SGBDR du marché autorisent qu'un attribut d'un tuple puisse être marqué à NULL. On signifie ainsi que la valeur de l'attribut est inconnue.
  - Combien de schéma de tables contiennent un attribut email autorisant le 'NULL' !
  - ⇨ Le problème est que 'NULL' n'est pas une valeur mais juste un marqueur ce qui peut occasionner bien des surprises par exemples, si la colonne 'nom' de la table clients supporte les NULL :

    SELECT * FROM clients

    WHERE nom != 'DURAND' ;

    Ne listera pas les tuples dont le nom est marqué à NULL !
  - ⇨ il faudra systématiquement traiter le cas du bonhomme NULL avec les prédicats IS NULL ou IS NOT NULL (et non avec = NULL ou != NULL !)

DML – APARTÉ SUR LES NULL (2/2)

- Dans la conception d'une base de données on veillera particulièrement à ne pas autoriser la valeur NULL !

DML – LECTURE DE DONNÉES (3)

- SELECT – Clause 'GROUP BY' (1/3)

  Cette clause permet de grouper les lignes de résultats selon un ou des attributs. Est généralement utilisé avec une fonction d'agrégation.

```
SELECT      client, SUM(solde) AS SoldeTotal
FROM        comptes
GROUP BY    client
ORDER BY    SoldeTotal DESC
```

client	SoldeTotal
1	2300
2	-300
3	-1300

DML – LECTURE DE DONNÉES (4)

- SELECT – Clause ‘GROUP BY’ (2/3)
  - Fonctions d’agrégations

    On trouvera ici la liste des fonction d’agrégation MySQL

DML – LECTURE DE DONNÉES (5)

- SELECT – Clause ‘GROUP BY’ (3/3)
  - Prédicat HAVING

    Permet de grouper les lignes de résultats selon un predicat portant sur les attributs.

```
SELECT      client, SUM(solde) AS SoldeTotal
FROM        comptes
GROUP BY    client
HAVING      SoldeTotal > 0
ORDER BY    SoldeTotal DESC
```

client	SoldeTotal
1	2300

DML – LECTURE DE DONNÉES (8)

- SELECT – Clause 'LIMIT' (MySQL)

  Cette clause permet de limiter le nombre de lignes retournées dans le SELECT

  Syntaxe (en fin de SELECT) [LIMIT {[offset,] row_count | row_count OFFSET offset}]

  - Avec 2 arguments, le 1er précise l'offset de la 1ère ligne à retourner (offset commence à 0), le second précise le nombre de lignes à retourner,
  - Avec un seul argument, on précise le nombre de ligne à retourner à partir de l'offset 0.

- SELECT – Clause 'DISTINCT'

  Elimine les tuples retournés doublons (Hum !)

- SELECT – Clause 'INTO var_name [, *var_name*]'

  Permet de récupérer le résultat (une seule ligne) d'un sélect dans des variables

DML – LECTURE DE DONNÉES (9)

- Jointure interne

  Vu avec les opérations relationnelles

- Jointure externe - Principe

  A la différence de la jointure interne, chaque tuple de chaque table peut avoir son image dans le résultat R. Certains attributs de R pourront être marqués à NULL

  - LEFT OUTER JOIN
    S'il n' y a pas de correspondance, relativement au prédicat, dans la table de droite un tuple avec toutes les attributs de la table de droite marqués à NULL est créé.
  - RIGHT OUTER JOIN

    On inverse les rôles de table de droite et de gauche par rapport à LEFT OUTER JOIN

DML – LECTURE DE DONNÉES (10)

- Jointure externe – Syntaxe SQL (MySQL)

```
SELECT              Xi,..Xn,Yj, …Yp
FROM                LeftTable
LEFT [OUTER] JOIN   RightTable
ON                  Prédicat
```

DML – LECTURE DE DONNÉES (11)

- Jointure externe – Exemple

```
SELECT      nom, prenom, CPT.id AS NoCompte, solde
FROM        clients     AS  CLI
LEFT JOIN   comptes     AS  CPT
ON          client  =   CLI.id  ;
```

clients

id	nom	prenom	mail
1	DURAND	Alice	alice@durand.net
2	DURAND	Emile	emile.durand@gmail.com
3	MAN	Super	super@man.us
4	DUPOND	Pierre	pierre.dupond@yahoo.fr

comptes

id	client	solde
1	3	-1000
2	1	2300
3	3	-300
4	2	-300

nom	prenom	NoCompte	solde
DURAND	Alice	2	2300
DURAND	Emile	4	-300
MAN	Super	1	-1000
MAN	Super	3	-300
DUPOND	Pierre	NULL	NULL

DML – LECTURE DE DONNÉES (12)

- Jointure externe – Remarque

Quel résultat donne la requête suivante ?

```
SELECT      nom, prenom, CPT.id AS NoCompte, solde
FROM        clients      AS  CLI
LEFT JOIN   comptes      AS  CPT
ON          client = CLI.id
WHERE       solde   <   0
```

DML – INSERTION DE DONNÉES (1)

- INSERT – Exemple Syntaxe 1 (In Extenso)

INSERT INTO *tbl_name*
[(*col_name* [, *col_name*] ...)]
{VALUES | VALUE} (*value_list*) [, (*value_list*)] ...

[ON DUPLICATE KEY UPDATE *assignment_list*] ;

Avec :

- value_list : value [, value]…
- value : {*expr* | DEFAULT}
- *assignment_list* : *assignment* [, *assignment*]
- *assignment* : *col_name* = *value*

DML – INSERTION DE DONNÉES (2)

- INSERT – Exemple Syntaxe 1 (1)

```
INSERT INTO it-akademy.clients
            (nom       , prenom       , email                          )
     VALUE  ('MOUSE' , 'Mickey'     , 'mickey.mouse@disney.com'      )     ;
```

clients			
id	nom	prenom	mail
1	DURAND	Alice	alice@durand.net
2	DURAND	Emile	emile.durand@gmail.com
3	MAN	Super	super@man.us
4	MOUSE	Mickey	mickey.mouse@disney.com

DML – INSERTION DE DONNÉES (3)

- INSERT – Exemple Syntaxe 1 (2)

```
SELECT   LAST_INSERT_ID()      INTO     @NewClient  ;      /* Récupération de la dernière id créée       */

INSERT INTO  it-akademy.comptes
                  (client          , solde           )
       VALUE    (@NewClient , 20000000.00 )      ;
```

comptes		
id	client	solde
1	3	-1000.00
2	1	2300.00
3	3	-300.00
4	2	-300.00
5	4	20000000.00

DML – INSERTION DE DONNÉES (4)

- INSERT – Exemple Syntaxe 2 (In Extenso)

INSERT INTO *tbl_name*
[(*col_name* [, *col_name*] ...)]
SET *assignment_list*

[ON DUPLICATE KEY UPDATE *assignment_list*] ;

DML – INSERTION DE DONNÉES (5)

- INSERT – Syntaxe 3 (simplifiée)

```
INSERT [INTO] tbl_name
    [(col_name [, col_name] ...)]
    SELECT ...
    [ON DUPLICATE KEY UPDATE assignment_list]
```

DML – MODIFICATION DE DONNÉES (1)

- UPDATE – Syntaxe mono-table

```
UPDATE      table_reference
    SET     assignment_list
[WHERE      where_condition]
[ORDER BY ...]
[LIMIT      row_count]    ;
```

DML – MODIFICATION DE DONNÉES (2)

- UPDATE – Syntaxe multi-tables

UPDATE *table_reference**s***
SET *assignment_list*
[WHERE *where_condition*] ;

Avec :

- *table_reference**s*** : table_reference [, table_reference]…

DML – SUPPRESSION DE DONNÉES (1)

- SUPPRESSION – Syntaxe simplifiée

DELETE FROM *tbl_name*
[WHERE *where_condition*]
[ORDER BY ...]
[LIMIT *row_count*] ;

DML – SUPPRESSION DE DONNÉES (2)

- DELETE – Exemple Syntaxe simplifiée

```
DELETE FROM     it-akademy.clients
WHERE           nom = 'DURAND' ;
```

Requête **ENSEMBLISTE** !

clients

id	nom	prenom	mail
3	MAN	Super	super@man.us
4	MOUSE	Mickey	mickey.mouse@disney.com

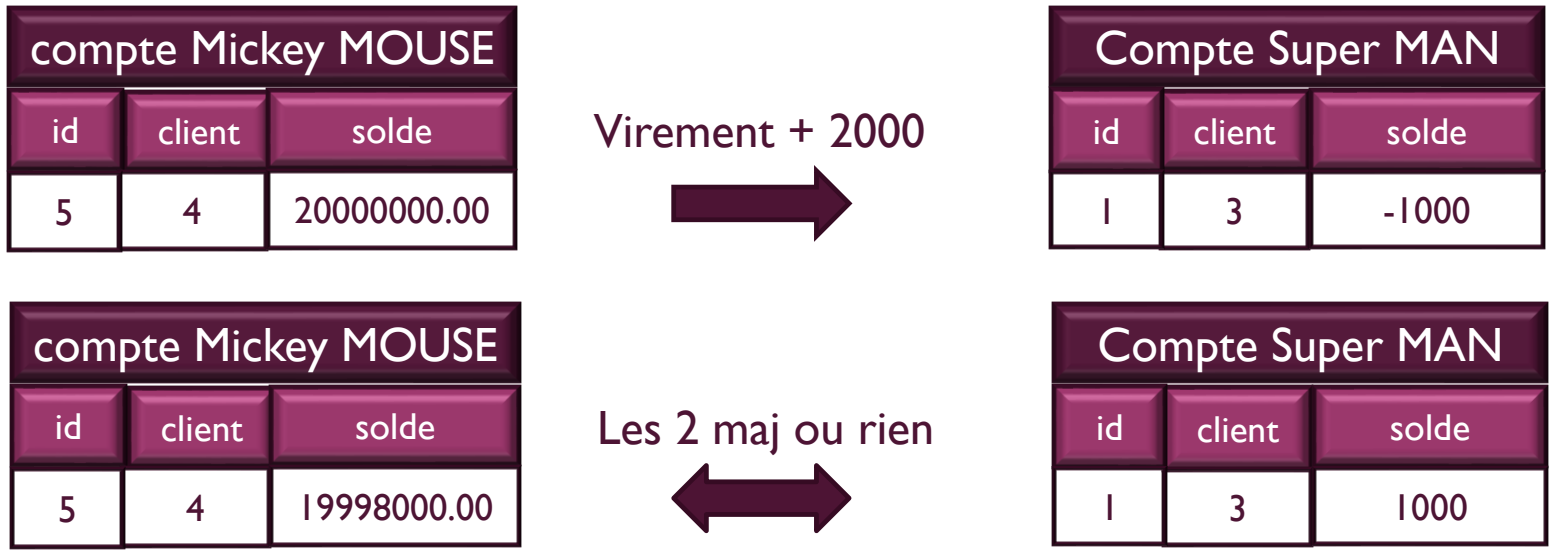
comptes

id	client	solde
1	3	-1000.00
3	3	-300.00
5	4	20000000.00

Suppression des comptes associés du fait de la foreign key

TRANSACTIONS AND LOCKING STATEMENTS (1)

- Positionnement du problème – Cas d'un virement bancaire



TRANSACTIONS AND LOCKING STATEMENTS (2)

- Propriétés ACID d'une transaction (1)
  - **ACID** comme **A**tomicité, **C**ohérence, **I**solation, **D**urabilité
    - **Atomicité**
      Assure qu'une transaction se fait au complet ou pas du tout : si une partie d'une transaction ne peut être faite, il faut effacer toute trace de la transaction et remettre les données dans l'état où elles étaient avant la transaction. L'atomicité doit être respectée dans toutes situations, comme une panne d'électricité, une défaillance de l'ordinateur.
    - **Cohérence**
      Assure que chaque transaction amènera le système d'un état valide à un autre état valide. Tout changement à la base de données doit être valide selon toutes les règles définies, incluant mais non limitées aux contraintes d'intégrité, aux rollbacks en cascade, aux déclencheurs de base de données, et à toutes combinaisons d'événements.

TRANSACTIONS AND LOCKING STATEMENTS (3)

- Propriétés ACID d'une transaction (2)
  - **Isolation**
    Toute transaction doit s'exécuter comme si elle était la seule sur le système. Aucune dépendance possible entre les transactions. La propriété d'isolation assure que l'exécution simultanée de transactions produit le même état que celui qui serait obtenu par l'exécution en série des transactions. Chaque transaction doit s'exécuter en isolation totale : si T1 et T2 s'exécutent simultanément, alors chacune doit demeurer indépendante de l'autre.
  - **Durabilité**
    Assure que lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'une panne d'électricité, d'une panne de l'ordinateur ou d'un autre problème. Par exemple, dans une base de données relationnelle, lorsqu'un groupe d'énoncés SQL a été exécuté, les résultats doivent être enregistrés de façon permanente, même dans le cas d'une panne immédiatement après l'exécution des énoncés.

TRANSACTIONS AND LOCKING STATEMENTS (4)

- Transaction – Syntaxe MySQL (cas général)

```
START TRANSACTION      ;
SELECT …..
FOR UPDATE…..          ;

UPDATE….
UPDATE…

INSERT…..

COMMIT                 ;   /*   En cas de succès     */
ROLLBACK               ;   /*   En cas d'anomalie    */
```

Déverrouille les données

Sans déclaration explicite de Transaction, le SGBD se place en autocommit. C'est une euphémisme qui signifie que toutes les mises à jour, insertions, suppressions **sont validées une par une.**
En d'autres termes, en cas d'erreur, **on ne peut plus revenir** sur les modifications de données préalablement réalisées.

SQL – MYSQL – TP3

- Ecrire un script de virement dans la base ‘it-akademy’
  - On crée un client et son compte qui seront l’objet du virement créditeur,
    - Utilisation de variables de sessions pour préciser les attributs à valoriser,
  - On valorise dans 2 variables supplémentaires nom d’un débiteur existant et le montant du virement
  - On réalise le virement du débiteur vers le créditeur

SQL – SOUS REQUÊTES (1)

- Sous requête externe

  Syntaxe type :

  SELECT …..

  WHERE {attributs_list} **operateur** (

  SELECT … /* sous-requête */

) ;

  - Operateurs
    - **[NOT] IN** : vérifie que {attributs_list} fait partie [ou non] de l'ensemble renvoyé par la sous-requête
    - **[NOT] ANY** : vérifie si la comparaison est vraie pour au moins [fausse pour toutes] une ligne renvoyé par la sous requêtes (un opérateur de comparaison scalaire (=, >,..) doit précéder [NOT] ANY
    - **SOME** : idem ANY
    - **ALL** : vérifie si la comparaison est vraie pour toutes les lignes renvoyés par la sous requêtes (un opérateur de comparaison scalaire (=, >,..) doit précéder ALL

SQL – SOUS REQUÊTES (2)

- Sous requête interne

Syntaxe type :

```
SELECT …..
        (
        SELECT …..
        )
        AS   table_name        ;
```

EXEMPLE :

```
SELECT      a1, a2,…an , b1, b2, … bk
FROM        table_name
INNER JOIN  (
            SELECT b1, b2,…bk
            FROM…
            INNER JOIN..
            ON …
            )
            AS  T1
ON          ai = bj        ;
```

SQL – FONCTIONS NATIVES MYSQL

- Il existe de nombreuses fonctions utilisables dans n'importe quelle requête MySQL en lieu et place d'un attribut, plus généralement dans n'importe quelle instruction SQL.
- Ces fonctions sont classées par catégories:
  - Mathématiques,
  - Traitement de chaines de caractères dont :
    - le traitement d'expressions régulières,
    - La recherche Full-Text
  - Gestion de dates et d'heures,
  - Transtypage (conversion)
  - Cryptage

  On se reportera ici à la documentation MySQL

MYSQL – ROUTINES STOCKÉES

- Une extension de SQL (SQL/**P**ersistent **S**tored **M**odules), conforme au standard SQL:2003 est implémentée dans MySQL afin de pouvoir directement stocker dans la base de données du code procédural.
  - Standard SQL:2003, donc normalement portabe,
  - Généralement du code métier
- Ces routines sont classées par catégories:
  - Fonctions, Procedures,
  - Triggers,
  - Event Scheduler

Pour les détails du langage, on se reportera à la documentation MySQL :
Documentation routines stockées MySQL

On trouvera de larges exemples ici : framework it-daas

MYSQL – ROUTINES STOCKÉES – PSM - COMMENTAIRES

- Commentaires
  - # commentaire jusqu'en fin de ligne
  - -- idem
  - /* Commentaire sur plusieurs lignes */

MYSQL – ROUTINES STOCKÉES – PSM - BLOCS

- Blocs d'instructions

  Syntaxe

  [*begin_label*:] BEGIN

  [*DECLARE private_var_name* [,…] type [DEFAULT value; [DECLARE..;]]

  [statement_list]

  END [*end_label*]

  - Les instructions doivent eêtre séparées par des ';'
  - Changer le delimiteur standard (';') avant la déclaration de routine.
    - Ex: DELIMITER $$ /* le délimiteur standard ';' devient'$$' */

MYSQL – ROUTINES STOCKÉES – PSM - VARIABLES

- Variables de session
  - Variables préfixées par '@', ex : @XmlOpeningTag
  - Ne se déclare pas, s'initialise par un SET @XmlOpeningTag = '<' ;
  - Visible par l'ensemble des processus/scripts de la session
  - Utilisé en particulier (et de manière conseillée uniquement !) pour définir des constantes (la notion de constante n'existe pas en PSM)

- Variables privées
  - Définies (clause DECLARE) et visibles à l'intérieur d'un bloc BEGIN…END

MYSQL – ROUTINES STOCKÉES – PSM – INST. CONTRÔLE(1)

- **IF..THEN…ELSE**

  Syntaxe :

  IF *search_condition*
  THEN *statement_list*
  [ELSEIF *search_condition*
  THEN *statement_list*] ...
  [ELSE *statement_list*]
  END IF

MYSQL – ROUTINES STOCKÉES – PSM – INST. CONTRÔLE(2)

- **CASE**

Syntaxe 1 :

```
CASE          case_value
    WHEN when_value
    THEN    statement_list
    [WHEN when_value
    THEN    statement_list]
     ...
    [ELSE    statement_list]
END CASE
```

Remarque : 'case_value' ne peut être le marqueur NULL car :
NULL = NULL est faux !

MYSQL – ROUTINES STOCKÉES – PSM – INST. CONTRÔLE(3)

- **CASE**

Syntaxe 2 :

```
CASE
    WHEN  search_condition
    THEN    statement_list
    [WHEN search_condition
    THEN    statement_list]
     ...
    [ELSE statement_list]
END CASE
```

MYSQL – ROUTINES STOCKÉES – PSM – BOUCLES (1)

- **WHILE**

  Syntaxe :

  [*begin_label*:] WHILE *search_condition*
  DO *statement_list*
  END WHILE [*end_label*]

MYSQL – ROUTINES STOCKÉES – PSM – BOUCLES (2)

- **REPEAT… UNTIL**

  Syntaxe :

  [*begin_label*:] REPEAT *statement_list*
  UNTIL *search_condition*
  END REPEAT [*end_label*]

MYSQL – ROUTINES STOCKÉES – PSM – BOUCLES (3)

- Autres instructions de gestions de Boucles
  - LOOP (+ LEAVE)
  - ITERATE

MYSQL – ROUTINES STOCKÉES – PSM – EXCEPTIONS (1)

Exceptions - Principe

- Lorsque MySQL détecte une erreur, il lève une exception
- Une exception peut être traité par l'application dans un gestionnaire d'exception (Handler)
- A défaut de gestionnaire d'exception, MySQL effectue un traitement par défaut de l'erreur
- L'application peut elle-même lever des exceptions 'utilisateur' (et éventuellement les traiter dans un Handler).

MYSQL – ROUTINES STOCKÉES – PSM – EXCEPTIONS (2)

Exceptions – Types d'erreurs

- Une erreur est identifiée par un numéro d'erreur MySQL, non standard, et un numéro « d'état SQL » ou SQLSTATE (Standard ANSI SQL) (Remarque ; il n'y a pas bijection !)
- Familles d'erreurs
  - WARNING,
  - SQL Exceptions,
  - NOT FOUND,
  - Others

  - Voir documentation MySQL ici

MYSQL – ROUTINES STOCKÉES – PSM – EXCEPTIONS (3)

- **Exceptions – Handler**

  Syntaxe :

  DECLARE *handler_action* HANDLER FOR *condition_value* [, *condition_value*] ...

  BEGIN

  Statements

  END

  handler_action : { CONTINUE | EXIT | UNDO }

  condition_value : { *mysql_error_code* | SQLSTATE [VALUE] *sqlstate_value* | *condition_name* | SQLWARNING | NOT FOUND | SQLEXCEPTION }

MYSQL – ROUTINES STOCKÉES – PSM – EXCEPTIONS (4)

- **Exceptions – Lever une exception**

  Se fait via l'instruction SIGNAL dont la syntaxe détaillée est fournie ici

  De manière pratique ;

```
SIGNAL    SQLSTATE              '45000'                    /*   $User Exception' */
          SET MESSAGE_TEXT      =    'libelle_message'
          MYSQL_ERRNO           =    1644
```

MYSQL – ROUTINES STOCKÉES – PSM – CURSEURS

Gestion de curseur

Cette fonctionnalité (volontairement non développée ici) permet de réaliser des traitements ligne à ligne (traitements non ensemblistes donc) à partir du résultat ensembliste d'un SELECT.

- Parfois nécessaire
- On se reportera, si besoin, à la documentation utilisateur : Gestion Curseurs MySQL

MYSQL – ROUTINES STOCKÉES – CONTRÔLE D'ACCÈS (1)

- **DEFINER**

  Lors de la création de la routine stockée est défini est propriétaire de celle-ci (DEFINER) qui est un des utilisateurs MySQL. Par défaut le 'DEFINER' est le créateur de la routine (qui doit avoir les droits de créer une routine !)

MYSQL – ROUTINES STOCKÉES – CONTRÔLE D'ACCÈS (2)

- **Paramètre SQL SECURITY**

  Ce paramètre est utilisé uniquement dans la définition d'une procédure ou d'une fonction. Il peut prendre 2 valeurs :

  - DEFINER

    Quelque soit l'utilisateur qui invoque la routine, celui-ci hérite des droits du DEFINER (ie du propriétaire de la routine) pendant toute l'exécution de la routine.

  - INVOKER

    Quelque soit les droits du DEFINER, la routine s'exécute avec les droits de celui qui invoque la routine

  Par défaut, ce paramètre vaut 'DEFINER'

MYSQL – ROUTINES STOCKÉES – TRIGGERS (1)

- Types de triggers

  - Un trigger est lié à une table
  - Un trigger peut être positionné sur une opération d'insertion, d'update ou de delete
  - Sur chaque type d'opération (insertion, update, delete), on peut positionner un trigger avant l'opération (trigger before) ou après (trigger after).

MYSQL – ROUTINES STOCKÉES – TRIGGERS (2)

- Syntaxe création Trigger

CREATE [DEFINER = { *user* | CURRENT_USER }] TRIGGER *trigger_name*

trigger_time trigger_event ON *tbl_name*

FOR EACH ROW [*trigger_order*]

trigger_body

trigger_time : { BEFORE | AFTER }

trigger_event : { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } *other_trigger_name*

MYSQL – ROUTINES STOCKÉES – TRIGGERS (3)

- Particularité : référence des valeurs d'une colonne pour la table cible d'un trigger

  - On utilise les mots clés NEW.nom_colonne ou OLD.nom_colonne pour faire référence à ces valeurs.
  - OLD permet de connaitre la valeur de la colonne avant opération et NEW est la valeur qui s'apprête à valoriser la colonne

SQL – MYSQL – TP4- TRIGGER

- Ecrire un trigger de la table comptes qui interdit qu'un compte devienne débiteur.

SQL – MYSQL – TP5- PROCEDURES STOCKÉES (1)

- Importer la base ad15
- Créer la procédure stockée suivante : pp_create
  - Synopsis arguments societes_create()
    - OUT $Status INTEGER(10) UNSIGNED ,
      OUT $Step INTEGER(10) UNSIGNED ,
      OUT $PpID INTEGER(10) UNSIGNED ,

      IN $Nom VARCHAR(128) ,
      IN $Prenom VARCHAR(128)
- Créer le script pour créer une personne physique pp

SQL – MYSQL – TP5 - PROCEDURES STOCKÉES (2)

- Créer la procédure stockée suivante : societes_create
  - Synopsis arguments societes_create()
    - OUT $Status INTEGER(10) UNSIGNED ,
      OUT $Step INTEGER(10) UNSIGNED ,
      OUT $SocieteID INTEGER(10) UNSIGNED ,

      IN $StatutJuridique VARCHAR(64) ,
      IN $CodeSociete VARCHAR(8) ,
      IN $RaisonSociale VARCHAR(128) ,
      IN $Immatriculation VARCHAR(64) ,
      IN $ImmatriculationSiege VARCHAR(64) ,
      IN $VatNumber VARCHAR(64) ,
      IN $Activite VARCHAR(255) ,
      IN $Capital DECIMAL(16,0)
- Créer le script pour créer une société

SQL – MYSQL – TP6 – REQUÊTES IMBRIQUÉES

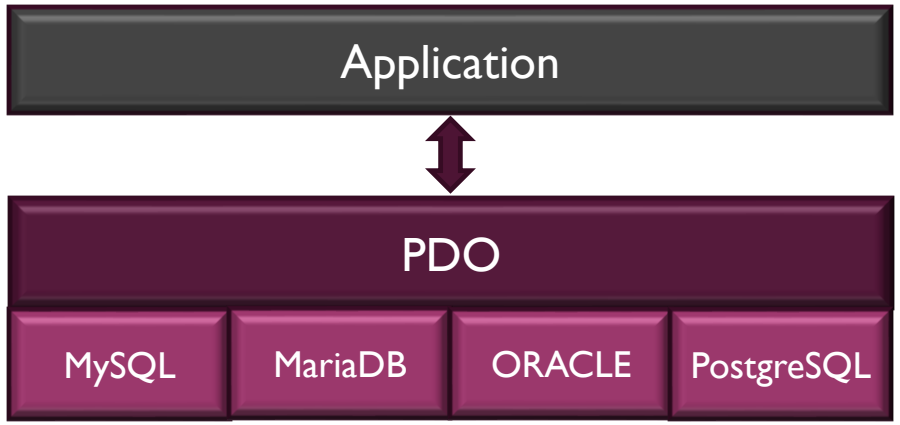
- Produire :
  - la liste des clients par sociétés,
  - le chiffre d'affaire par client pour chaque société
  - le montant des commissions versées à chaque commercial sur une période donnée.

PHP – INTERFACE AVEC UN SGBDR

PDO

PHP DATA OBJECTS (PDO)

- Interface OO d'abstraction au SGBDR (à partir de PHP 5.1)
  - Dans le cas de WAMP, à activer si nécessaire (menu PHP/Extensions)



PDO – CONNEXION AU SGBDR

- Syntaxe:

  $DataBase = **new** PDO(‘string **$DSN** [, string **$username** [, string **$password** [array $options]]]) ;

  - $**DSN** - **D**ata **S**ource **N**ame, se décompose en pratique de la manière suivante :
    - [Préfixe DSN][host:**namehost**][:port][;dbname=**database_name**][;charset=**charset_name**]
    - Pour MySQL, le préfixe est ‘**mysql:’**
    - Exemple : $DSN = ‘mysql:host=localhost;dbname=it-akademy;charset=utf8’

- **$username** et **$password**
  - Comme leur nom l’indique

PDO – DÉCONNEXION DU SGBDR

- Syntaxe:

```
$DataBase = null ;
```

PDO – FAIRE UNE REQUÊTE ÉLÉMENTAIRE - QUERY

- Méthode : query (PDO::query)

  Exécute une requête SQL en appelant une seule fonction, retourne le jeu de résultats (s'il y en a) de la requête en tant qu'objet PDOStatement.

  PDOStatement : Représente une requête préparée et, une fois exécutée, le jeu de résultats associé.

  - Syntaxe : $query = $DataBase->query('La Requête à exécuter') ;

  Ex : $client**s** = $DataBase->query('select nom, prenom, mail from clients where nom = 'DURAND' order by prenom DESC';) ;

  - On récupère ainsi un jeu de résultats (par exemple plusieurs lignes car un moteur relationnel retourne un ensemble) qu'il faut aller chercher un à un via la méthode PDOStatement::fetch

PDO – FAIRE UNE REQUÊTE ÉLÉMENTAIRE - FETCH

- Méthode : fetch(PDOStatement::fetch)

  Récupère la ligne suivante d'un jeu de résultats PDO

  - Syntaxe simple : $fetch = $query->fetch() ;
    - $fetch est un array qui contient l'ensemble des champs de la première ligne de réponse
    - Ex :

      ```
      $client  =  $clients->fetch()  ;
      $nom     =  $client['nom']     ;
      $prenom  =  $client['prenom']  ;
      $email   =   $client['email']  ;
      ```

    - En pratique, on récupère toutes les lignes au moyen d'une boucle while() controlée de la manière suivante :

      ```
      while($client = $clients->fetch())
              {….}
      ```

    - En fin de boucle, on ferme le curseur : $clients->closeCursor() ; /* PDOStatement::closeCursor */

PDO – EXERCICE QUERY

- A partir d'un formulaire où l'on saisit le nom, afficher tous les clients qui porte ce nom saisi (base it-akademy)
  - Pour gérer les erreurs d'accès à la base de données, on utilisera la gestion d'exceptions au moyens des blocs try{} et catch{}

```
try
  {
    /*
    **    Gestion accès via PDO à la base de données
    */
  }
catch(Exception $Exception)
  {
    die($Exception->getMessage())        ;    /* arrête le script en envoyant le message d'erreur détectée    */
  }
```

PDO –REQUÊTES PRÉPARÉES – PREPARE/EXECUTE (1)

- Méthodes : PDO::prepare() et PDOStatement::execute()
  - PDO::prepare()
    Prépare une requête SQL à être exécutée par la méthode PDOStatement::execute(). La requête SQL peut contenir zéro ou plusieurs noms (:nom) ou marqueurs (?) pour lesquels les valeurs réelles seront substituées lorsque la requête sera exécutée.
- PDOStatement::execute()
  - Exécute une requête préparée. Si la requête préparée inclut des marqueurs de positionnement, on utilisera pour passer les arguments en entrée :
    - Soit la méthode PDOStatement::bindParam() (or PDOStatement::bindValue()),
    - Soit un tableau de valeurs de paramètres
- PDOStatement::nextRowset()
  - Avance à la prochaine ligne de résultats (utilisé lorsque la requête contient plusieurs ordres SQL ou si une procédure stockée renvoie plusieurs lignes.

PDO –REQUÊTES PRÉPARÉES – PREPARE/EXECUTE (2)

- Exemple(1)

```
/*
**      (1)     Prepare
*/
$sql_request =      $database->prepare (      '       INSERT INTO comptes
                                                                    (client ,       solde)
                                                        VALUES      (:client        ,       :solde);
                                                '
                                            )                                                   ;
/*
**      (2)     Bind
*/
$sql_request->bindParam   (':client'      ,     $ClientID       ,     PDO:: PARAM_INT)      /* paramètre entier  */     ;
$sql_request->bindParam   (':solde'       ,     $Solde          ,     PDO:: PARAM_STR)      /* paramètre string  */     ;
/*
**      (3)     Execute
*/
$sql_request->execute()                                                                             ;
```

PDO –REQUÊTES PRÉPARÉES – PREPARE/EXECUTE (3)

- Exemple(2) – variante array pour passer les paramètres en entrée

```
/*
**    (2)   Pas de bindParam() et execute() direct
*/
$sql->execute (     array      (
                               'client'     =>   $ClientID       ,
                               'solde'      =>   $Solde
                               )
                )     ;
;
```

PDO – EXERCICE REQUÊTE PRÉPARÉE

- A partir d'un formulaire ou l'on saisit le nom, prénom et email, insérer le client ainsi saisi.

PDO –GESTION DE TRANSACTIONS – MISE EN ŒUVRE

- Méthodes
  - PDO::beginTransaction()
  - PDO::commit() ou PDO:: rollback()

PDO –GESTION DE TRANSACTIONS – TP7

- Ecrire un formulaire de virement qui débite le 1er compte d'un client et crédite le 1er compte d'un autre client
  - On saisira les noms et prénoms des clients ainsi que le montant du virement
- Restructurer ensuite le code en isolant dans un script spécifique tous les accès à la base de données
  - L'opération de virement pourra être codée sous la forme d'une procédure stockée ou directement en PHP