MySQL, en bref

- MySQL est le SGBDR libre et gratuit et qui appartient à Oracle.
- Le SQL de MySQL répond au standard de SQL 2008. (idem pour Oracle).h
- Mis à part la couche SQL (standard), ORACLE et MYSQL sont deux SGBD très différents.
- Pour notre utilisation:
  - Il y a une différence majeure au niveau de l'architecture, MYSQL possède une couche supplémentaire: Couche base de données. Avant de créer les tables il faut un CREATE DATABASE
  - les différences essentielles avec Oracle se situent au niveau des types de données et de certaines fonctions SQL.
  - Le définition des contraintes d'intégrité est la même pour les deux.
  - L'écriture des jointure est la même :au FROM et non au WHERE.
  - Les deux SGBDs supporte les procédures stockées et les triggers. Cependant MYSQL n'a pas la notion de Packages. Le détails de l'écriture des procédures et triggers viendra plus loin.

MySQL, en bref

- Les types de données :
- Les entiers

Type	Storage	Minimum Value	Maximum Value
	(Bytes)	(Signed/Unsigned)	(Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

MySQL, en bref

- DECIMAL et NUMERIC: le type NUMERIC en MySQL est implémenté comme un DECIMAL, et acceptent deux paramètres : la précision et l'échelle.
  - La précision définit le nombre de chiffres significatifs stockés, donc les 0 à gauche ne comptent pas. En effet 0024 est équivalent à 24. Il n'y a donc que deux chiffres significatifs dans 0024.
  - L'échelle définit le nombre de chiffres après la virgule.
  - Le nombre maximum de chiffres pour un DECIMAL est de 65.
  - Exemple: Salaire DECIMAL(5,2).
- Les Types: FLOAT, DOUBLE

MySQL, en bref

- Les types DATE au format ‘AAAA-MM-JJ, et DATETIME format ‘AAAA-MM-JJ HH:MM:SS’.
- Les chaines de caractères:
  - Pour stocker un texte relativement court (moins de 255 caractères), vous pouvez utiliser les types CHAR et VARCHAR. Ces deux types s'utilisent avec un paramètre qui précise la taille que peut prendre votre texte (entre 1 et 255).
  - La différence entre CHAR et VARCHAR est la manière dont ils sont stockés en mémoire. Un CHAR(x) stockera toujours *x* caractères, en remplissant si nécessaire le texte avec des espaces vides pour le compléter, tandis qu'un VARCHAR(x) stockera jusqu'à *x* caractères (entre 0 et *x*), et stockera en plus en mémoire la taille du texte stocké.
    Si vous entrez un texte plus long que la taille maximale définie pour le champ, celui-ci sera tronqué.
  - Pour stocker un text plus que 255 caractères, utilisez le type TEXT
  - Le type BLOB est également utilisé pour les gros fichiers

MySQL, en bref

- Le type ENUM, utilisé pour énumérer un ensemble de valeur (fonction DECODE de Oracle).
- Exemple:

```
create table etudiants
(
numad int unsigned auto_increment,
nom varchar (30) not null,
prenom varchar(30) not null,
groupe enum('gr1','gr2','gr3'),
constraint pketudiant primary key (numad)
);
```

MySQL, en bref

- Les requêtes de définition de données:
  - CREATE TABLE, la définition des contraintes reste la même qu'avec Oracle.
    - La clé primaire se définie au niveau colonne ou Table. La définition au niveau table est conseillée -> nom de contrainte
    - La définition de la contrainte de la Foreign Key reste la même: Au niveau TABLE.
    - Les autres contraintes (CHECK , not null) se définissent au niveau table ou colonne.
  - ALTER TABLE même syntaxe,
  - DROP table même syntaxe.
- Les requêtes de manipulation des données: même syntaxe.
- SELECT, même syntaxe. La jointure se fait avec un INNER JOIN et non dans le WHERE. Pour restreindre le nombre de lignes on utilise LIMIT a la place du ROWNUM.
  - Exemple: select * from employes order by salaire limit 5;

MySQL, en bref

- Les procédures stockées:(fonctions)
  - Le corps est le même qu'avec ORACLE (SQL)
  - Il faut un DELIMITER pour chaque procédure et chaque fonction.
  - Le mot réservé DECLARE est obligatoire (ce qui n'est pas le cas avec ORACLE)
  - MySQL n'a pas la notion de package.
  - Dans la définition d'une fonction (pas dans le code SQL) c'est RETURNS;
  - Le IN|OUT des paramètres est transmis en premier;
  - Pas de mot réservé IS ou AS dans la déclaration des procédures
  - Pour exécuter une fonction : SELECT nomfonction(paramètres). Pas de DUAL
  - Pour exécuter une procédure: CALL procedure(parametres); (ce n'est pas execute)
  - ET pas de curseur pour retourner plusieurs enregistrements !
- Exemples:

MySQL, en bref

```
delimiter $$
create procedure insertEtudiants
(IN pnom varchar(30),
IN pprenom varchar(30),
IN pgroupe char(3))
Begin
insert into etudiants (nom, prenom,groupe) values (pnom, pprenom,pgroupe);
commit;
end $$
```

Appel de la procedure:

```
call insertEtudiants('Remi','Ruby','gr1');
```

MySQL, en bref

```
delimiter |
create procedure afficher(IN pgroupe char(3))
Begin
select * from etudiants where groupe =pgroupe;
end |
```

Appel de la procedure.

```
call afficher('gr1');
```

MySQL, en bref

```
delimiter |
create function lister(pgroupe char(3)) returns int
Begin
declare resultat int;
select count(*) into resultat from etudiants  where groupe =pgroupe;
return resultat;
end |
Appel de la fonction
select lister('gr1');
```

MySQL, en bref (suite)

Pour initialiser une variable on lui affecte une valeur par défaut avec DEFAULT.
Pour affecter une valeur à une variable, on utilise le mot réservé SET.
Exemple:

```
delimiter |
create procedure updatenote(in pnum integer)
Begin
declare notep integer default 0;
set notep=20;
update etudiants set note = notep where pnum = numad;
commit;
end |
```

MySQL, en bref(suite)

Les structures de contrôles:

Pour les conditions:

```
IF condition THEN instructions
    [ELSEIF condition THEN instructions] ...
    [ELSE instruction]
END IF
```

Pour la repetition

```
WHILE condition DO
    instructions
END WHILE
```

MySQL, en bref(suite)

REPEAT instructions

UNTIL condition

END REPEAT

Pour les autres instructions de contrôle, visitez https://dev.mysql.com/doc/refman/5.7/en/

MySQL, en bref(suite)

Exemple1:

```
delimiter |
create function verifier2(palias varchar(30))returns integer
begin
Declare
rep integer;
if palias in(select nom from etudiants where nom =palias)
Then set rep =1;
else  set rep=0;
 end if;
return rep;
end |
```

MySQL, en bref(suite)

Exemple2: vérifier que le user et le mot de passe sont corrects avec une procédure.

L'opérateur EXISTS s'utilise comme le IN dans les sous requêtes. S'il est utilisé dans le SELECT et non dans le WHERE, alors il retourne 1 ou 0 selon qu'il existe des enregistrements ou non.

```
delimiter |
CREATE  PROCEDURE verifier(in pseudo varchar(20),in mpasse varchar(20))
BEGIN
declare var int;
declare rep char(1);
select exists(select alias,Motdepasse from membres where alias = pseudo and mpasse =Motdepasse) into var;
if var=1  then set rep ='Y'  ; else set  rep = 'N'; end if;
select rep;
END
```

MySQL, en bref

- Les triggers ? Même principe avec les exeptions suivantes
- Il n'y a pas de raise_application_error. Il faudra écrire une procédure pour l'insertion des messages erreurs dans une table, puis lire le contenu
- On ne peut pas combiner plusieurs opérations DML dans un même trigger. Il faudra écrire un trigger pour chaque :
- Il n' y a pas de OF dans le update pour préciser la colonne.
- Il n'y a pas les deux points devant le NEWet le OLD
- Le declare est à l'intérieur du BEGIN

MySQL, en bref(suite)

Avec ORACLE

```
create or replace trigger empsal
before insert or update on employesclg
for each row
declare sal number;
begin
select avg(salaireemp) into sal from employesclg
group by codedep having codedep =:new.codedep;
if :new.salaireemp is null
then :new.salaireemp:=sal;
end if;
end;
```

MySQL, en bref(suite)

Avec MySQL, il faudra deux triggers: Un pour INSERT l'autre pour le UPDATE.

```
DELIMITER |;
create trigger empsal before insert on employes
for each row
begin
declare sal decimal;
select avg(salaire) into sal from employes
group by codep having codep =new.codep;
if new.salaire is null
then set new.salaire =sal;
end if;
end |;
```

MySQL, en bref(suite)

Avec Oracle:

```
CREATE  or REPLACE TRIGGER ctrlSalaire2
BEFORE  UPDATE OF salaire ON EMPLOYESBIDON
FOR EACH ROW
BEGIN
……..
```

Avec MySQL

```
CREATE  or REPLACE TRIGGER ctrlSalaire2
BEFORE  UPDATE ON EMPLOYESBIDON
FOR EACH ROW
BEGIN
……..
```

MySQL, en bref(suite)

Optimisation de requêtes: (pour tous les SGBDs).

- En principe, lorsqu'une requête SQL est envoyée au SGBD, celui-ci établit un plan d'exécution. Le module se charge d'établir un plan d'exécution s'appelle Optimizer.
- Le fonctionnement de l'Optimizer globalement similaire pour l'ensemble des SGBDs (Oracle et SQL Server), en utilisant les étapes suivantes :
  1. Validation syntaxique
  2. Validation sémantique
  3. Utilisation éventuelle d'un plan précédemment produit
  4. Réécriture/Simplification de la requête
  5. Exploration des chemins d'accès et estimation des coûts.
  6. Désignation du chemin le moins coûteux, génération du plan d'exécution et mise en cache de ce dernier.

MySQL, en bref(suite)

Rappel: Un index est un objet permettant d'accélérer l'accès aux données. La création d'un index sur une clé primaire se fait automatiquement par le système. Pour créer un index sur une colonne autre que la clé primaire on utilise la commande CREATE INDEX.

create index indexnom on joueurs(nom);

create index indexnomprenom on joueurs(nom,prenom);

Si la colonne sur laquelle l'index est créé a la contraint UNIQUE alors vous pouvez créer un index UNIQUE.

Create UNIQUE index indexAlias on joueurs(Pseudo);

Important:

- Même si les indexes permettent d'accélérer les recherches, trop d'indexe a l'effet inverse (ralentit les recherches)
- Un index ne peut être créé que sur une table (pas une vue).
- Les index UNIQUE applique les contraintes d'unicité
- Ne jamais créer trop d'index
- Créer des index sur une colonne ayant une petite plage de valeurs nulles.

MySQL, en bref(suite)

Si vous êtes développeur de bases de données vous devez connaître certaines règles qui permettent d'optimiser l'exécution de requêtes. En voici quelques-unes de ces règles (qui ne sont pas nécessairement d'ans l'ordre).

R1 : Évitez le SELECT * : écrire plutôt le nom des colonnes dont vous avez besoin pour la requête.

R2 : Créez des indexes sur les colonnes que vous utilisez dans la clause WHERE. Pour plus de performances, ces indexes doivent-être créés après l'insertion des données dans la table.

R3 : Lorsque c'est possible, utilisez le WHERE à la place du Having.

R4 : Évitez les jointures dans le WHERE, utilisez plutôt le INNER JOIN.

R5 : Lorsque c'est possible, utilisez une jointure à la place d'une sous-requête. Les jointures sont l'essentiel des SGBDRs alors ils sont optimisés pour l'écriture des jointures.

R6 : Utilisez le SELECT count(*) à la place de compter les colonnes. (select count(*) from etudiants à la place de SELECT count(numad) from etudiants

MySQL, en bref(suite)

R7 : Évitez les fourchettes < et >; utilisez le BETWEEN

Si possible, utilisez le BETWEEN à la place du Like

R8 : transformer l'INTERSECT en jointure

R9 : Utilisez le IN à la place du ANY et le <> ALL en NOT IN

R10 : Évitez la clause DISTINCT dans le SELECT sauf si c'est absolument nécessaire

R11 : Ordonnez par nom des colonnes plutôt que par les numéros