# MySQL Prise en Main

# Plan du cour

- Introduction et prise en main
  - Versions, panorama des fonctionnalités et des outils.
  - Ressources et documentation.
  - Procédure d'installation.
  - Paramétrages de post-installation et premiers tests.
  - L'outil client ligne de commandes mysql.
  - L'outil graphique MySQL Query Browser.
- Modèle relationnel, conception et création d'une base
  - Eléments de conception d'un schéma de base de données.
  - Contraintes d'intégrité.
  - Types de données MySQL (numériques, chaînes, dates, types spécifiques...).
  - Fonctions intégrées de MySQL.
  - Types de tables (MyISAM, MEMORY, MERGE, InnoDB...).
  - Création de bases et de tables.
  - La base INFORMATION_SCHEMA.
  - Jeux de caractères, internationalisation.
- Pratique du SQL avec MySQL
  - Sélections simples, comparaisons, tris.
  - Sélections multitables, différents types de jointures.
  - Requêtes imbriquées, préparées.
  - Modifications et gestion des vues.
- Tables transactionnelles InnoDB
  - Notion de transaction, niveaux d'isolation.
  - Structure physique des tables.
  - Programmation des transactions (START TRANSACTION, COMMIT, ROLLBACK).
- SQL procédural
  - Procédures stockées et fonctions.
  - Définition des procédures. Déclencheurs (Triggers).
  - Gestion des erreurs.
  - Travaux pratiques
  - Ecriture de procédures stockées. Récupération de résultats à l'aide de curseurs. Création et utilisation de triggers.
- Connexions, droits d'accès, sécurité
  - Niveaux de privilèges et vérification des droits.
  - Gestion des utilisateurs et de leurs privilèges.
  - Sécurisation des procédures stockées et des vues.
  - Travaux pratiques
  - Gestion des privilèges et des mots de passe.
- Introduction à l'administration
  - Exportation de données.

- Sauvegardes, la commande mysqldump.
- Survol de l'installation de MySQL.
- Travaux pratiques
- Exportation de données.

# Difference Par rapport a oracle

Certaines particularités par rapport à Oracle
Les fichiers de journalisations Contrairement à Oracle, MySQL ne possède pas toujours des fichiers de journalisations. Cela dépend de son engin (voir plus bas) de tables. Ce qui veut dire qu'il est possible que les données soient perdues si la base de données plante. Ouch !

Les fichiers de données
Oracle groupe les fichiers de données selon leur tablespace. Par exemple USER01.DBF et USER02.DBF. Dans le cas de tablespaces permanents, ces fichiers peuvent contenir des index ou des tables.

MySQL regroupe les fichiers de données selon un schéma/base de données. Chaque fichier correspond à une table.

Exemple : nomBaseDeDonnées/table1.frm nomBaseDeDonnées/table2.frm

L'installation et l'administration
Le niveau de connaissance afin d'administrer et de gérer une base de données MySQL est moindre que pour Oracle. En effet, son installation est rapide et la courbe d'apprentissage du fonctionnement de MySQL est inférieure à Oracle.

La base de données MySQL est également moins lourde sur un ordinateur et son démarrage est presque instantané. Il prend seulement la RAM nécessaire à son bon fonctionnement, contrairement à Oracle.

Les clusters
Oracle (RAC – Real Application Cluster) - Chaque machine possède une instance, qui accède à la même base de données

MySQL - Il existe une version de MySQL pour les clusters : MySQL Cluster

La gestion des usagers
Les droits d'un usager sur une base de données varient dépendamment de l'endroit (machine) où il se connecte.

Par exemple :
CREATE USER 'usr_facturation'@'localhost' IDENTIFIED BY 'xyz_pwd'; GRANT ALL ON ma_bd_facturation.* TO 'usr_facturation'@'localhost' IDENTIFIED BY 'xyz_pwd';

Différences entre le serveur « community » et le « enterprise » Community (sous la licence GPL) - Ce que la compagnie dit sur l'utilisation du serveur MySQL Community : o Tu peux l'utiliser pour contribuer à son développement o Tu peux faire un projet dérivé et qui utilise MySQL, mais celui-ci doit être open source o Tu peux tester

- Ce que la licence GPL dit : o Si le projet est distribué avec MySQL, alors ce projet doit être open source sous la même licence. Il est également possible d'acheter une licence commerciale.

o Si le projet n'a pas à être distribué avec MySQL, alors il n'a pas besoin d'être open source.

Par exemple, un site Web qui se connecte à la base de données MySQL sur un serveur d'hébergement partagé. Celui-ci n'a pas à être open source. Enterprise - Pour les besoins commerciaux o Serveur plus stable o Meilleur support et mises à jour régulières

Les engines de tables de MySQL
innoDB

Engin par défaut de MySQL
Supporte les transactions (donc les clés étrangères également)
MyISAM

Jusqu'à tout récemment, c'était l'engin par défaut de MySQL
Ne supporte pas les transactions
Supporte les index sur plusieurs mots (FULL TEXT INDEXING)
MEMORY

Table dont le contenu est gardé en mémoire uniquement.
Comme toutes les données sont en mémoire (incluant les index), c'est extrêmement rapide. Il faut cependant faire attention. Si le serveur plante ou se ferme, les données seront perdues.
Quelques-unes des différences sur la syntaxe
LIMIT Afin de limiter les résultats d'une requête, la clause LIMIT est utilisée. Par exemple : <source lang='sql'>

SELECT
id, name FROM users ORDER BY name LIMIT 10, 20; </source> Ceci retourne les résultats à partir de la ligne #11 jusqu'à la ligne #35 Pour retourner les 10 premières lignes = LIMIT 0,10;

Pour faire identique avec Oracle, il faut plutôt faire : <source lang='sql'> SELECT * FROM (SELECT USERS_2.*, ROWNUM RNUM FROM (SELECT ID, NAME FROM USERS ORDER BY NAME) USERS_2 WHERE ROWNUM < 36) WHERE RNUM >= 10; AUTO_INCREMENT </source> Il n'est pas nécessaire de faire des séquences comme avec Oracle. AUTO_INCREMENT suffit ! <source lang='sql'> CREATE TABLE users ( id INT NOT NULL AUTO_INCREMENT, ... PRIMARY KEY pk_users (id) ) ENGINE = innoDB; </source>

Syntaxe
La syntaxe des commandes suivantes est en simplifiée. En effet, certaines clauses peuvent être ajoutées (optionnel).

Création d’une base de données
CREATE {DATABASE | SCHEMA } xyz_db CHARACTER SET = utf8 (ou ascii, greek, …)

Database et schema sont synonymes pour MySQL.

Création d’un usager
CREATE USER 'foo'@'192.168.0.1' IDENTIFIED BY 'pwd';

foo : Le nom de l’usager 192.168.0.1 : Nom de l’hôte d’où il se connecte. Pour n’importe quel hôte, on utilise « % ». pwd : Mot de passe de l’usager

Assignation de droits d’un usager à une base de données GRANT v ON w.x TO 'y'@'z';

V: Privilège donné - Exemple : o ALL - Tous les privilèges o INSERT - Privilège d’insertion o UPDATE - Privilège de mise à jour o SELECT - Privilège de SELECT o EXECUTE - Permet d’exécuter des procédures stockées o ALTER - Permet de modifier la structure d’une table avec ALTER TABLE W: Nom de la base de données (ex : xyz_db) X: Nom de la table, ou * pour toutes les tables Y: nom de l’usager (ex: foo) Z: hôte (ex: %)

Exemple complet : GRANT SELECT, INSERT ON xyz_db.* TO 'foo'@'192.168.0.1';

MySQL supporte également les rôles, mais pas les profiles.

Suppression de droits d’un usager sur une base de données REVOKE INSERT ONT xyz_db.* TO 'foo'@'192.168.0.1';

Exemple de création d’une table simple La colonne “status” permet une énumération de valeurs permises. <source lang='sql'> CREATE TABLE users (

id INT NOT NULL AUTO_INCREMENT,
status ENUM("pending", "inactive", "active") DEFAULT "pending",
password VARCHAR(40) NOT NULL,
email VARCHAR(70) NOT NULL,
PRIMARY KEY pk_users(id),
INDEX idx_users_email (email)
) ENGINE = innoDB; </source> Exemple de création d’une table avec une clef étrangère <source lang='sql'> CREATE TABLE forgot_passwords (

id INT NOT NULL AUTO_INCREMENT,
id_user INT NOT NULL,
access_key VARCHAR(70),
PRIMARY KEY pk_temporary_passwords (id),
CONSTRAINT fk_temporary_passwords_id_user FOREIGN KEY (id_user) REFERENCES users (id)
) ENGINE = innoDB; </source>

Le dictionnaire de données

Depuis la version 5.0 de MySQL, les informations sur les bases de données (tables, colonnes, accès, etc) se trouve dans le schéma/database « INFORMATION_SCHEMA ». C'est la version MySQL, du dictionnaire de données Oracle.

Par exemple, pour avoir le nombre de connexions, on utiliserait la vue processlist du schéma INFORMATION_SCHEMA. Par exemple : SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST;

Pour avoir la liste des colonnes des tables : SELECT * FROM INFORMATION_SCHEMA.COLUMNS;

MySQL Workbench SQL Development Est similaire à SQL Developer pour Oracle.

Pour afficher tous les schémas de MySQL (incluant le schéma du dictonnaire de données), il y a l'option Préférences SQL Editor Show Metadata schemata.

Data Modeling Permet de faire des diagrammes relationnels. Ces diagrammes peuvent par la suite être exportés en script SQL.

Il est également possible d'importer des schémas en script MySQL. MySQL Administration Permet de gérer la base de données MySQL.

Création de copies de sauvegarde
Les connexions actives à MySQL
Le CPU + RAM usage
- Etc.

# Exercice Data WareHouse

Le but de cette exercice est de construire un data warehouse pour une micro société d'informatique. Cette société se compose ainsi:

- Les personnes ont un nom,prenom et nom d'usage
- Les personnes ont aussi une adresse ainsi que des contacts a prevenir en cas d'urgence.
- Principalement, ils utilisent du matériel informatique qui peuvent être des PC.

Le schéma est le suivant :


Solution

```
 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;

SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL';


DROP SCHEMA IF EXISTS orsysformation ;

CREATE SCHEMA IF NOT EXISTS orsysformation DEFAULT CHARACTER SET latin1 ;

USE orsysformation ;


-- -----------------------------------------------------

-- Table orsysformation.ville

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.ville ;
```

```
CREATE TABLE IF NOT EXISTS orsysformation.ville (

 departement VARCHAR(3) NOT NULL ,
 INSEE VARCHAR(5) NOT NULL DEFAULT  ,
 commune VARCHAR(100) NOT NULL ,
 idville INT(11) NOT NULL AUTO_INCREMENT ,
 PRIMARY KEY (idville) ,
 INDEX IDX_DEPARTEMENT (departement ASC) )
ENGINE = InnoDB

AUTO_INCREMENT = 201

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.adresse

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.adresse ;


CREATE TABLE IF NOT EXISTS orsysformation.adresse (

 idadresse INT(11) NOT NULL AUTO_INCREMENT ,
 rue VARCHAR(50) NOT NULL COMMENT 'le nom de la rue avec le numéro' ,
 ville INT(11) NOT NULL DEFAULT '1' COMMENT 'Reference la table ville' ,
 PRIMARY KEY (idadresse) ,
 INDEX in_ville (ville ASC) ,
 CONSTRAINT fk_ville
   FOREIGN KEY (ville )
   REFERENCES orsysformation.ville (idville ))
ENGINE = InnoDB

AUTO_INCREMENT = 201
```

```
DEFAULT CHARACTER SET = latin1

COMMENT = 'Contient les informations sur les adresses des contact et du'  /* comment truncated
*/;



-- -----------------------------------------------------

-- Table orsysformation.application

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.application ;


CREATE TABLE IF NOT EXISTS orsysformation.application (

 idapplication INT(11) NOT NULL AUTO_INCREMENT ,
 name_application VARCHAR(20) NOT NULL ,
 type_application ENUM('bureautique','developpeur','financiere','autre') NOT NULL ,
 PRIMARY KEY (idapplication) )
ENGINE = InnoDB

AUTO_INCREMENT = 201

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.contacturgent

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.contacturgent ;
```

```
CREATE TABLE IF NOT EXISTS orsysformation.contacturgent (

 idContactUrgent INT(11) NOT NULL AUTO_INCREMENT ,
 nom VARCHAR(25) NOT NULL DEFAULT 'INCONNU' ,
 idadresse INT(11) NULL DEFAULT NULL ,
 telephone VARCHAR(10) NULL DEFAULT NULL ,
 relation ENUM('famille','autre') NULL DEFAULT NULL ,
 PRIMARY KEY (idContactUrgent) ,
 INDEX in_adresse (idadresse ASC) ,
 CONSTRAINT fk_adresse
   FOREIGN KEY (idadresse )
   REFERENCES orsysformation.adresse (idadresse )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
ENGINE = InnoDB

AUTO_INCREMENT = 201

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.materiel_informatique

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.materiel_informatique ;


CREATE TABLE IF NOT EXISTS orsysformation.materiel_informatique (

 idmateriel INT(11) NOT NULL AUTO_INCREMENT ,
 typemateriel ENUM('PC','Laptop','Portable') NULL DEFAULT NULL ,
 daterevision DATE NULL DEFAULT NULL ,
 PRIMARY KEY (idmateriel) )
ENGINE = InnoDB

AUTO_INCREMENT = 201
```

```
DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.materiel_pc

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.materiel_pc ;


CREATE TABLE IF NOT EXISTS orsysformation.materiel_pc (

 idmateriel INT(11) NOT NULL ,
 memoire INT(11) NOT NULL DEFAULT '0' ,
 disque INT(11) NOT NULL DEFAULT '0' ,
 processeur ENUM('pentium','core ix','autre') NOT NULL DEFAULT 'autre' ,
 ip VARCHAR(20) NULL DEFAULT NULL ,
 PRIMARY KEY (idmateriel) ,
 CONSTRAINT fk_materiel
   FOREIGN KEY (idmateriel )
   REFERENCES orsysformation.materiel_informatique (idmateriel ))
ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.personne

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.personne ;
```

```
CREATE TABLE IF NOT EXISTS orsysformation.personne (

 idPersonne INT(11) NOT NULL AUTO_INCREMENT ,
 nom VARCHAR(25) NOT NULL ,
 prenom VARCHAR(25) NOT NULL ,
 nom_usage VARCHAR(25) NOT NULL ,
 PRIMARY KEY (idPersonne) ,
 INDEX fk_personnerh (idPersonne ASC) )
ENGINE = InnoDB

AUTO_INCREMENT = 201

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.personnerh

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.personnerh ;


CREATE TABLE IF NOT EXISTS orsysformation.personnerh (

 idPersonne INT(11) NOT NULL ,
 date_naissance DATE NULL DEFAULT NULL ,
 adresse_personnel INT(11) NULL DEFAULT NULL ,
 bureau VARCHAR(25) NULL DEFAULT NULL ,
 telephone VARCHAR(14) NOT NULL DEFAULT '0033600000000' ,
 telephone_mobile VARCHAR(14) NULL DEFAULT NULL ,
 sexe ENUM('M','F','XX') NULL DEFAULT 'XX' ,
 PRIMARY KEY (idPersonne) ,
 INDEX in_contacturgent (idPersonne ASC) ,
 INDEX in_personne (idPersonne ASC) ,
 INDEX in_adresse (adresse_personnel ASC) ,
 CONSTRAINT personnerh_ibfk_1
   FOREIGN KEY (idPersonne )
```

```
   REFERENCES orsysformation.personne (idPersonne )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION,
 CONSTRAINT personnerh_ibfk_2
   FOREIGN KEY (adresse_personnel )
   REFERENCES orsysformation.adresse (idadresse )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.rel_materiel_personne

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.rel_materiel_personne ;


CREATE TABLE IF NOT EXISTS orsysformation.rel_materiel_personne (

 idpersonne INT(11) NOT NULL ,
 idmateriel INT(11) NOT NULL ,
 PRIMARY KEY (idmateriel) ,
 INDEX fk_materielinformatique (idmateriel ASC) ,
 INDEX in_personne (idpersonne ASC) ,
 CONSTRAINT fk_materielinformatique
   FOREIGN KEY (idmateriel )
   REFERENCES orsysformation.materiel_informatique (idmateriel )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION,
 CONSTRAINT fk_personne
   FOREIGN KEY (idpersonne )
   REFERENCES orsysformation.personne (idPersonne )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Table orsysformation.rel_personne_application

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.rel_personne_application ;


CREATE TABLE IF NOT EXISTS orsysformation.rel_personne_application (

 idpersonne INT(11) NOT NULL ,
 idapplication INT(11) NOT NULL ,
 login VARCHAR(20) NULL DEFAULT NULL ,
 password VARCHAR(20) NULL DEFAULT NULL ,
 PRIMARY KEY (idpersonne, idapplication) ,
 INDEX fk_rel_personne_application_personnerh1 (idpersonne ASC) ,
 INDEX in_applicatif (idapplication ASC) ,
 INDEX log_pass (login ASC, password ASC) ,
 CONSTRAINT fk_applicatif
   FOREIGN KEY (idapplication )
   REFERENCES orsysformation.application (idapplication )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION,
 CONSTRAINT fk_rel_personne_application_personnerh1
   FOREIGN KEY (idpersonne )
   REFERENCES orsysformation.personne (idPersonne )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;
```

```
-- -----------------------------------------------------

-- Table orsysformation.relpersonnerhcontacturgent

-- -----------------------------------------------------

DROP TABLE IF EXISTS orsysformation.relpersonnerhcontacturgent ;


CREATE TABLE IF NOT EXISTS orsysformation.relpersonnerhcontacturgent (

 idPersonneRH INT(11) NOT NULL ,
 idContactUrgent INT(11) NOT NULL ,
 PRIMARY KEY (idContactUrgent) ,
 INDEX in_personne (idPersonneRH ASC) ,
 CONSTRAINT fk_contacturgent
   FOREIGN KEY (idContactUrgent )
   REFERENCES orsysformation.contacturgent (idContactUrgent ),
 CONSTRAINT relpersonnerhcontacturgent_ibfk_1
   FOREIGN KEY (idPersonneRH )
   REFERENCES orsysformation.personnerh (idPersonne )
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
ENGINE = InnoDB

DEFAULT CHARACTER SET = latin1;



-- -----------------------------------------------------

-- Placeholder table for view orsysformation.parc_applicatif

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS orsysformation.parc_applicatif (nom INT, name_application INT,
type_application INT, login INT, password INT);
```

```
-- -----------------------------------------------------

-- Placeholder table for view orsysformation.vue_demenageur

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS orsysformation.vue_demenageur (idpersonne INT, nom INT, prenom INT,
nom_usage INT, bureau INT, telephone INT);


-- -----------------------------------------------------

-- View orsysformation.parc_applicatif

-- -----------------------------------------------------

DROP VIEW IF EXISTS orsysformation.parc_applicatif ;

DROP TABLE IF EXISTS orsysformation.parc_applicatif;

USE orsysformation;

CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=root@localhost SQL SECURITY DEFINER VIEW
orsysformation.parc_applicatif AS select orsysformation.personne.nom AS
nom,orsysformation.application.name_application AS
name_application,orsysformation.application.type_application AS
type_application,orsysformation.rel_personne_application.login AS
login,orsysformation.rel_personne_application.password AS password from
((orsysformation.personne join orsysformation.rel_personne_application
on((orsysformation.personne.idPersonne = orsysformation.rel_personne_application.idpersonne)))
join orsysformation.application on((orsysformation.rel_personne_application.idapplication =
orsysformation.application.idapplication)));


-- -----------------------------------------------------

-- View orsysformation.vue_demenageur

-- -----------------------------------------------------
```

```
DROP VIEW IF EXISTS orsysformation.vue_demenageur ;

DROP TABLE IF EXISTS orsysformation.vue_demenageur;

USE orsysformation;

CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=root@localhost SQL SECURITY DEFINER VIEW
orsysformation.vue_demenageur AS select orsysformation.personne.idPersonne AS
idpersonne,orsysformation.personne.nom AS nom,orsysformation.personne.prenom AS
prenom,orsysformation.personne.nom_usage AS nom_usage,orsysformation.personnerh.bureau AS
bureau,orsysformation.personnerh.telephone AS telephone from (orsysformation.personne left
join orsysformation.personnerh on((orsysformation.personne.idPersonne =
orsysformation.personnerh.idPersonne)));

USE orsysformation;


DELIMITER $$


USE orsysformation$$

DROP TRIGGER IF EXISTS orsysformation.insert_personne $$

USE orsysformation$$

CREATE

DEFINER=root@localhost

TRIGGER orsysformation.insert_personne

AFTER INSERT ON orsysformation.personne

FOR EACH ROW

BEGIN

DECLARE mycontactId INTEGER;
```

```
insert into personnerh(idPersonne) values (NEW.idPersonne);

insert into contacturgent values ();

select MAX(contacturgent.idContactUrgent) from contacturgent into mycontactId;

insert into relpersonnerhcontacturgent values(NEW.idPersonne,mycontactId);

END$$



DELIMITER ;



SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

# Excercice SQL

## Création des tables

```
CREATE TABLE Segment

(indIP varchar(11),
nomSegment varchar(20) NOT NULL,
 etage TINYINT(1),
  CONSTRAINT pk_Segment PRIMARY KEY (indIP));
CREATE TABLE Salle
(nSalle varchar(7),
  nomSalle varchar(20) NOT NULL,
nbPoste TINYINT(2),
indIP varchar(11),

 CONSTRAINT pk_salle PRIMARY KEY (nSalle));
CREATE TABLE Poste
(nPoste varchar(7),
nomPoste varchar(20) NOT NULL,
  indIP varchar(11),
  ad varchar(3),
  typePoste varchar(9),
nSalle varchar(7),
  CONSTRAINT pk_Poste PRIMARY KEY (nPoste),
 CONSTRAINT ck_ad CHECK (ad BETWEEN '000'  AND '255'));
 CREATE TABLE Logiciel
(nLog varchar(5),
  nomLog varchar(20) NOT NULL,
dateAch DATETIME,
version varchar(7), typeLog varchar(9),

prix DECIMAL(6,2),
```

```
  CONSTRAINT pk_Logiciel PRIMARY KEY (nLog),
 CONSTRAINT ck_prix CHECK (prix >= 0));
CREATE TABLE Installer
(nPoste varchar(7),
 nLog varchar(5),
 numIns INTEGER(5) AUTO_INCREMENT,
 dateIns TIMESTAMP DEFAULT NOW(),
 delai DECIMAL(8,2),
CONSTRAINT pk_Installer PRIMARY KEY(numIns));
 CREATE TABLE Types
(typeLP varchar(9),
nomType varchar(20),
CONSTRAINT pk_types PRIMARY KEY(typeLP));
```

# Création des données

```
 INSERT INTO Segment VALUES ('130.120.80','Brin RDC',NULL);
INSERT INTO Segment VALUES ('130.120.81','Brin 1er étage',NULL);
INSERT INTO Segment VALUES ('130.120.82','Brin 2ème étage',NULL);
INSERT INTO Salle VALUES ('s01','Salle 1',3,'130.120.80');
INSERT INTO Salle VALUES ('s02','Salle 2',2,'130.120.80');
INSERT INTO Salle VALUES ('s03','Salle 3',2,'130.120.80');
INSERT INTO Salle VALUES ('s11','Salle 11',2,'130.120.81');
INSERT INTO Salle VALUES ('s12','Salle 12',1,'130.120.81');
INSERT INTO Salle VALUES ('s21','Salle 21',2,'130.120.82');
INSERT INTO Salle VALUES ('s22','Salle 22',0,'130.120.83');
INSERT INTO Salle VALUES ('s23','Salle 23',0,'130.120.83');
INSERT INTO poste VALUES ('p1','Poste 1','130.120.80','01','TX','s01');
INSERT INTO poste VALUES ('p2','Poste 2','130.120.80','02','UNIX','s01');
INSERT INTO poste VALUES ('p3','Poste 3','130.120.80','03','TX','s01');
INSERT INTO poste VALUES ('p4','Poste 4','130.120.80','04','PCWS','s02');
INSERT INTO poste VALUES ('p5','Poste 5','130.120.80','05','PCWS','s02');
INSERT INTO poste VALUES ('p6','Poste 6','130.120.80','06','UNIX','s03');
INSERT INTO poste VALUES ('p7','Poste 7','130.120.80','07','TX','s03');
INSERT INTO poste VALUES ('p8','Poste 8','130.120.81','01','UNIX','s11');
INSERT INTO poste VALUES ('p9','Poste 9','130.120.81','02','TX','s11');
```

```
INSERT INTO poste VALUES ('p10','Poste 10','130.120.81','03','UNIX','s12');
INSERT INTO poste VALUES ('p11','Poste 11','130.120.82','01','PCNT','s21');
INSERT INTO poste VALUES ('p12','Poste 12','130.120.82','02','PCWS','s21');
INSERT INTO logiciel VALUES ('log1','Oracle 6', '1995-05-13','6.2','UNIX',3000);
INSERT INTO logiciel VALUES ('log2','Oracle 8', '1999-09-15','8i','UNIX',5600);
INSERT INTO logiciel VALUES ('log3','SQL Server', '1998-04-12','7','PCNT',3000);
INSERT INTO logiciel VALUES ('log4','Front Page', '1997-06-03','5','PCWS',500);
INSERT INTO logiciel VALUES ('log5','WinDev', '1997-05-12','5','PCWS',750);
INSERT INTO logiciel VALUES ('log6','SQL*Net', NULL, '2.0','UNIX',500);
INSERT INTO logiciel VALUES ('log7','I. I. S.', '2002-04-12','2','PCNT',900);
INSERT INTO logiciel VALUES ('log8','DreamWeaver','2003-09-21','2.0','BeOS',1400);
INSERT INTO Types VALUES ('TX', 'Terminal X-Window');
INSERT INTO Types VALUES ('UNIX','Système Unix');
INSERT INTO Types VALUES ('PCNT','PC Windows NT');
INSERT INTO Types VALUES ('PCWS','PC Windows');
INSERT INTO Types VALUES ('NC', 'Network Computer');
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p2', 'log1', '2003-05-15',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p2', 'log2', '2003-09-17',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p4', 'log5', NULL,NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p6', 'log6', '2003-05-20',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p6', 'log1', '2003-05-20',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p8', 'log2', '2003-05-19',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p8', 'log6', '2003-05-20',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p11','log3', '2003-04-20',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p12','log4', '2003-04-20',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p11','log7', '2003-04-20',NULL);
INSERT INTO installer (nPoste,nLog,dateIns,delai) VALUES ('p7', 'log7', '2002-04-01',NULL);
```

# Ajout de colonnes

- Rajouter à la table Segment une colonne nombre de salle et nombre de poste
- Rajouter une colonne nombre d'install dans la table Logiciel
- Rajouter une colonne nombre de type logiciel sur la table poste

```
ALTER TABLE Segment ADD (nbSalle TINYINT(2) DEFAULT 0, nbPoste TINYINT(2) DEFAULT 0);
ALTER TABLE Logiciel ADD nbInstall TINYINT(2) DEFAULT 0;
ALTER TABLE Poste ADD nbLog TINYINT(2) DEFAULT 0;
```

# Modification des colonnes

- Modifier la colonne nomSalle de la table Salle afin que cela soit un VARCHAR(30)
- Modifier la colonne nomSergment de la table Segment afin que cela soit un VARCHAR(15)

```
 ALTER TABLE Salle MODIFY nomSalle VARCHAR(30);
 DESC Salle;
 ALTER TABLE Segment MODIFY nomSegment VARCHAR(15);
 DESC Segment;
```

# Modification de donnée

Mettre a jout les étages avec les segments IP tel que :

- 130.120.80 est assignée à l'étage 0
- 130.120.81 est assignée à l'étage 1
- 130.120.82 est assignée à l'étage 2.

Le prix des PC sous NT (PCNT) à baisser de 10%.

```
UPDATE Segment SET etage=0 WHERE indIP = '130.120.80'; UPDATE Segment SET etage=1 WHERE indIP
= '130.120.81';
 UPDATE Segment SET etage=2 WHERE indIP = '130.120.82';
 SELECT * FROM Segment;
 UPDATE Logiciel

SET prix = prix*0.9 WHERE typeLog = 'PCNT';
 SELECT nLog, typeLog, prix FROM Logiciel;
```

# Selection de données

Trouvé les type de poste tel que le la colonne nPoste de la table Poste soit p8

```
 SELECT nPoste, typePoste FROM Poste WHERE nPoste = 'p8';
```

Trouver les noms des logiciels de type UNIX

```
SELECT nomLog FROM Logiciel WHERE typeLog = 'UNIX';
```

Trouver les Nom,Adresse, numéro de salle des poste de type UNIX ou PCWS

```
SELECT nomPoste, indIP, ad, nSalle FROM poste WHERE typePoste = 'UNIX' OR typePoste = 'PCWS';
```

Faites de même pour les postes du segment 130.120.80 en les triant par numéro de salle décroissant

```
SELECT nomPoste, indIP, ad, nSalle FROM poste WHERE (typePoste = 'UNIX' OR typePoste = 'PCWS') AND indIP = '130.120.80' ORDER BY nSalle DESC;
```

Trouvé les numéo de logiciel installés sur le poste p6

```
SELECT nLog FROM Installer WHERE nPoste = 'p6';
```

Trouver les Nom et adresse IP complète (ex : 130.120.80.01) des postes de type TX

```
SELECT nomPoste, CONCAT(indIP,'.',ad) FROM Poste WHERE typePoste = 'TX';
```

# Fonction et Groupements

Trouver les prix moyen des logiciel de type Unix

```
SELECT AVG(prix) FROM Logiciel WHERE typeLog = 'UNIX';
```

Afficher la liste des postes avec le nombre de logiciel.

```
SELECT nPoste, COUNT(nLog) FROM installer GROUP BY (nPoste);
```

Afficher la liste des logiciels avec le nombre de poste installé

```
SELECT nLog, COUNT(nPoste) FROM Installer GROUP BY (nLog);
```

Trouver la valeur maximal des date d'achats de logiciel

```
SELECT MAX(dateAch) FROM Logiciel;
```

Afficher les poste ayant 2 logiciel installé

```
SELECT nPoste FROM Installer GROUP BY nPoste HAVING COUNT(nLog)=2;
```

# Requetes multitable

Afficher les systèmes d'exploitation jamais installé sur les postes

```
SELECT DISTINCT typeLP FROM Types WHERE typeLP NOT IN (SELECT DISTINCT typePoste FROM Poste);
```

Afficher les logiciels installé sur les postes

```
SELECT DISTINCT typeLog FROM Logiciel WHERE typeLog IN (SELECT typePoste From Poste)
```

Renvoyer les adresses IP des postes ou Oracle 8 a été installé

```
SELECT CONCAT(indIP,'.',ad) FROM Poste WHERE nPoste IN

(SELECT nPoste FROM Installer WHERE nLog =  (SELECT nLog
 FROM Logiciel    WHERE nomLog = 'Oracle 8'));
```

# Jointures

Adresse IP des postes qui hébergent le logiciel de nom ‘Oracle 8’

```
 SELECT CONCAT(indIP,'.',ad) FROM Poste p, Installer i, Logiciel l WHERE p.nPoste = i.nPoste
AND l.nLog = i.nLog AND l.nomLog = 'Oracle 8';
```

```
SELECT CONCAT(indIP,'.',ad) FROM Poste NATURAL JOIN Installer

NATURAL JOIN Logiciel WHERE nomLog = 'Oracle 8';
```

Noms des salles ou l’on peut trouver au moins un poste hébergeant ‘Oracle 6’

```
 SELECT s.nomSalle FROM Salle s, Poste p, Installer i, Logiciel l WHERE s.nSalle = p.nSalle
AND p.nPoste = i.nPoste AND i.nLog = l.nLog AND l.nomLog = 'Oracle 6';
```

# Exercice Modelisation

L'idée de cet exercice est d'arriver a modeliser un logiciel de gestion de condition général d'assurance vie. Une assurance vie est un document ayant les caractéristiques suivantes:

- composé de titre/chapitre/sous-chapitre
- composé de paragraphes qui sont des textes plus ou moins long

L'idée de ce logiciel est double:

- permettre un partage entre les documents des titre/chapitres/sous chapitres/paragraphes.
- permettre de faire des mises a jour groupé de documents.

En effet le contexte est qu'un paragraphe de contrat est un objet complexe a créer, et il semble une bonne idée que tant qu'a l'avoir crée il puisse être partagé avec plusieurs contrats.

```
 CREATE DATABASE IF NOT EXISTS `legitheque` /*!40100 DEFAULT CHARACTER SET latin1 */; USE
`legitheque`; -- MySQL dump 10.13 Distrib 5.5.16, for Win32 (x86) -- -- Host: localhost
Database: legitheque -- ------------------------------------------------------ -- Server
version 5.5.24

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */; /*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */; /*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */; /*!40101 SET NAMES utf8 */; /*!40103 SET
@OLD_TIME_ZONE=@@TIME_ZONE */; /*!40103 SET TIME_ZONE='+00:00' */; /*!40014 SET
@OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */; /*!40014 SET
@OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */; /*!40101 SET
@OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */; /*!40111 SET
@OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

-- -- Table structure for table `node` --

DROP TABLE IF EXISTS `node`; /*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */; CREATE TABLE `node` (

 `ID` bigint(20) NOT NULL AUTO_INCREMENT,
 `IDCONTRAT` bigint(20) NOT NULL,
```

```
 `ID_Chapitre` bigint(20) DEFAULT NULL,
 `ID_Paragraphe` bigint(20) DEFAULT NULL,
 PRIMARY KEY (`ID`),
 KEY `idx_contrat` (`IDCONTRAT`) USING HASH,
 KEY `idx_para` (`ID_Paragraphe`) USING HASH,
 KEY `idx_chap` (`ID_Chapitre`) USING HASH,
 KEY `fk_contrat` (`IDCONTRAT`),
 KEY `fk_chapitre` (`ID_Chapitre`),
 KEY `fk_para` (`ID_Paragraphe`),
 CONSTRAINT `fk_para` FOREIGN KEY (`ID_Paragraphe`) REFERENCES `paragraphe` (`ID`) ON DELETE
NO ACTION ON UPDATE NO ACTION,
 CONSTRAINT `fk_chapitre` FOREIGN KEY (`ID_Chapitre`) REFERENCES `chapitre` (`ID`) ON DELETE
NO ACTION ON UPDATE NO ACTION,
 CONSTRAINT `fk_contrat` FOREIGN KEY (`IDCONTRAT`) REFERENCES `contrat` (`ID`) ON DELETE NO
ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1; /*!40101 SET character_set_client = @saved_cs_client
*/;

-- -- Table structure for table `paragraphe` --

DROP TABLE IF EXISTS `paragraphe`; /*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */; CREATE TABLE `paragraphe` (

 `ID` bigint(20) NOT NULL AUTO_INCREMENT,
 `VALUE` longtext NOT NULL,
 PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=latin1; /*!40101 SET character_set_client =
@saved_cs_client */;

-- -- Table structure for table `chapitre` --

DROP TABLE IF EXISTS `chapitre`; /*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */; CREATE TABLE `chapitre` (

 `ID` bigint(20) NOT NULL AUTO_INCREMENT,
 `VALUE` varchar(50) NOT NULL,
 PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=latin1; /*!40101 SET character_set_client =
@saved_cs_client */;
```

```
-- -- Table structure for table `contrat` --

DROP TABLE IF EXISTS `contrat`; /*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */; CREATE TABLE `contrat` (

 `ID` bigint(20) NOT NULL AUTO_INCREMENT,
 `NAME` varchar(50) NOT NULL,
 PRIMARY KEY (`ID`),
 UNIQUE KEY `NAME_UNIQUE` (`NAME`)
) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=latin1; /*!40101 SET character_set_client =
@saved_cs_client */;

-- -- Table structure for table `parentchildnode` --

DROP TABLE IF EXISTS `parentchildnode`; /*!40101 SET @saved_cs_client = @@character_set_client
*/; /*!40101 SET character_set_client = utf8 */; CREATE TABLE `parentchildnode` (

 `ID_parent` bigint(20) NOT NULL,
 `ID_child` bigint(20) NOT NULL,
 `order_node` int(11) NOT NULL DEFAULT '0',
 PRIMARY KEY (`ID_parent`,`ID_child`),
 KEY `idx_order` (`order_node`) USING BTREE,
 KEY `idx_node` (`ID_parent`) USING HASH,
 KEY `fk_parent` (`ID_parent`),
 KEY `fk_childe` (`ID_child`),
 CONSTRAINT `fk_childe` FOREIGN KEY (`ID_child`) REFERENCES `node` (`ID`) ON DELETE NO ACTION
ON UPDATE NO ACTION,
 CONSTRAINT `fk_parent` FOREIGN KEY (`ID_parent`) REFERENCES `node` (`ID`) ON DELETE NO ACTION
ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1; /*!40101 SET character_set_client = @saved_cs_client
*/; /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */; /*!40014 SET
FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */; /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS
*/; /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */; /*!40101 SET
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */; /*!40101 SET
COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */; /*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2013-09-16 22:12:22
```

# Exercice SQL 2

# Partie 1

Sur la table lpecom_livres

Quelle requête utiliser pour afficher l'ensemble des enregistrements de la table `lpecom_livres` ?

```
SELECT *
FROM lpecom_livres;
```

Quelle requête utiliser pour sélectionner uniquement les livres qui ont un **prix strictement supérieur à 20** de la table `lpecom_livres` ?

```
SELECT *
FROM lpecom_livres
WHERE prix > 20;
```

Quelle requête utiliser pour trier les enregistrements de la table `lpecom_livres` du prix le plus élevé aux prix le plus bas ?

```
SELECT *
FROM lpecom_livres
ORDER BY prix DESC;
```

Quelle requête utiliser pour récupérer le prix du livre le plus élevé de la table `lpecom_livres` ?

```
SELECT MAX(prix)
FROM lpecom_livres;
```

Quelle requête utiliser pour récupérer les livres de la table `lpecom_livres` qui ont un prix compris entre 20 et 22 ?

```
SELECT *
FROM lpecom_livres
WHERE prix BETWEEN 20 AND 22;
```

Quelle requête utiliser pour récupérer tous les livres de la table `lpecom_livres` à l'exception de celui portant la valeur pour la colonne **isbn_10** : **2092589547** ?

```
SELECT *
FROM lpecom_livres
WHERE isbn_10 != 2092589547;
```

Quelle requête utiliser pour récupérer le prix du livre le moins élevé de la table `lpecom_livres` en renommant la colonne dans les résultats par **minus** ?

```
SELECT MIN(prix) as minus
FROM lpecom_livres;
```

Quelle requête utiliser pour sélectionner uniquement les 3 premiers résultats sans le tout premier de la table `lpecom_livres` ?

```
SELECT *
FROM lpecom_livres
LIMIT 3 OFFSET 1;
```

# Partie 2

Les tables sont lpecom_etudiants et lpecom_examens

Quelle requête utiliser pour afficher l'id des étudiants qui ont participés à au moins un examen ?

```
SELECT DISTINCT id_etudiant
FROM lpecom_examens;
```

Quelle requête utiliser pour compter le nombre d'étudiants qui ont participés à au moins un examen ?

```
SELECT COUNT(DISTINCT id_etudiant)
FROM lpecom_examens;
```

Quelle requête utiliser pour calculer la moyenne de l'examen portant l'id : **45** ?

```
SELECT AVG(note)
FROM lpecom_examens
WHERE id_examen = 45;
```

Quelle requête utiliser pour récupérer la meilleure note de l'examen portant l'id : **87** ?

```
SELECT MAX(note)
FROM lpecom_examens
WHERE id_examen = 87;
```

Quelle requête utiliser pour afficher l'id des étudiants qui ont eu **plus de 11 à l'examen 45** ou **plus de 12 à l'examen 87** ?

```
SELECT DISTINCT id_etudiant
FROM lpecom_examens
WHERE (id_examen = 45 AND note > 11)
OR (id_examen = 87 AND note > 12);
```

Quelle requête utiliser pour afficher tous les enregistrement de la table `lpecom_examens` avec en plus, si c'est possible, le prenom et le nom de l'étudiant ?

```
SELECT ex.*, et.prenom, et.nom
FROM lpecom_examens ex
LEFT JOIN lpecom_etudiants et ON ex.id_etudiant = et.id_etudiant;
```

Quelle requête utiliser pour afficher les enregistrement de la table `lpecom_examens` avec le prenom et le nom de l'étudiant, uniquement quand les étudiants sont présents dans la table lpecom_etudiants ?

```
SELECT ex.*, et.prenom, et.nom
FROM lpecom_examens ex
INNER JOIN lpecom_etudiants et ON ex.id_etudiant = et.id_etudiant;
```

Quelle requête utiliser pour afficher uniquement le nom et le prenom de l'étudiant avec l'id : **30** avec la moyenne de ses deux examens dans une colonne **moyenne** ?

```
SELECT et.prenom, et.nom, AVG(ex.note) as moyenne
FROM lpecom_examens ex
INNER JOIN lpecom_etudiants et ON ex.id_etudiant = et.id_etudiant
WHERE et.id_etudiant = 30;
```

Quelle requête utiliser pour afficher les 3 meilleures examens, du meilleur au moins bon, avec le prenom et le nom de l'étudiant associé ?

```
SELECT *
FROM lpecom_examens ex
```

```
INNER JOIN lpecom_etudiants et ON ex.id_etudiant = et.id_etudiant
ORDER BY ex.note DESC
LIMIT 3;
```

# Partie 3

Tables lpecom_realisateurs, lpecom_films, lpecom_films_notes

Quel est le résultat de la requête ci-dessous ?

```
SELECT id, prenom, nom
FROM lpecom_realisateurs
WHERE nation = "us"
AND sexe = 1;
```

| id | prenom | nom |
|---|---|---|
| 47 | Patty | Jenkins |

Quel est le résultat de la requête ci-dessous ?

```
SELECT *
FROM lpecom_realisateurs
WHERE sexe = "0"
ORDER BY nom DESC
LIMIT 1;
```

| id | nom | prenom | sexe | nation |
|---|---|---|---|---|
| 16 | Scott | Ridley | 0 | uk |

Quel est le résultat de la requête ci-dessous ?

```
SELECT f.id, f.nom AS film, r.prenom, r.nom
FROM lpecom_films f
INNER JOIN lpecom_realisateurs r ON f.id_realisateur = r.id
ORDER BY f.id ASC;
```

| id | film | prenom | nom |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 121 | Requiem for a Dream | Darren | Aronofsky |
| 546 | Gladiator | Ridley | Scott |
| 775 | Blade Runner | Ridley | Scott |
| 984 | Seul sur Mars | Ridley | Scott |
| 986 | Black Swan | Darren | Aronofsky |
| 987 | Wonder Woman | Patty | Jenkins |

Quel est le résultat de la requête ci-dessous ?

```
SELECT f.id, f.nom AS film, r.prenom, r.nom
FROM lpecom_films f
LEFT JOIN lpecom_realisateurs r ON f.id_realisateur = r.id
ORDER BY f.id ASC;
```

| id | film | prenom | nom |
|---|---|---|---|
| 121 | Requiem for a Dream | Darren | Aronofsky |
| 546 | Gladiator | Ridley | Scott |
| 666 | Fight Club | | |
| 775 | Blade Runner | Ridley | Scott |
| 984 | Seul sur Mars | Ridley | Scott |
| 986 | Black Swan | Darren | Aronofsky |
| 987 | Wonder Woman | Patty | Jenkins |
| 988 | The Tomorrow Man | | |

Quel est le résultat de la requête ci-dessous ?

```
SELECT f.id, f.nom, fn.note
FROM lpecom_films f
LEFT JOIN lpecom_films_notes fn ON f.id = fn.id_film
ORDER BY f.id ASC;
```

| id | nom | note |
|---|---|---|
| 121 | Requiem for a Dream | 1 |
| 546 | Gladiator | 4.5 |
| 546 | Gladiator | 2.5 |
| 666 | Fight Club | 4.2 |
| 775 | Blade Runner | 5 |

| | | |
|---|---|---|
| 984 | Seul sur Mars | 3.5 |
| 986 | Black Swan | 4.3 |
| 986 | Black Swan | 3 |
| 987 | Wonder Woman | 3.1 |
| 988 | The Tomorrow Man | |

Quel est le résultat de la requête ci-dessous ?

```
SELECT f.nom, r.prenom AS realisateur_prenom, r.nom AS realisateur_nom, AVG(fn.note) AS
moyenne_note
FROM lpecom_films f
INNER JOIN lpecom_realisateurs r ON f.id_realisateur = r.id
INNER JOIN lpecom_films_notes fn ON f.id = fn.id_film
WHERE f.id = 546;
```

| nom | realisateur_prenom | realisateur_nom | moyenne_note |
|---|---|---|---|
| Gladiator | Ridley | Scott | 3.5 |

Quel est le résultat de la requête ci-dessous ?

```
SELECT r.nation, AVG(fn.note) AS moyenne_note
FROM lpecom_films f
INNER JOIN lpecom_realisateurs r ON f.id_realisateur = r.id
INNER JOIN lpecom_films_notes fn ON f.id = fn.id_film
WHERE r.nation = "us";
```

| nation | moyenne_note |
|---|---|
| us | 2.85 |

Quel est le résultat de la requête ci-dessous ?

```
SELECT r.nation, MAX(fn.note) AS max_note
FROM lpecom_films f
INNER JOIN lpecom_realisateurs r ON f.id_realisateur = r.id
INNER JOIN lpecom_films_notes fn ON f.id = fn.id_film
WHERE r.nation = "uk";
```

| nation | max_note |
|---|---|

| | |
|---|---|
| uk | 5 |

# Partie 4

Tables lpecom_cities, lpecom_departments, lpecom_regions

Quelle requête utiliser pour retrouver la ville qui possède les coordonnées GPS suivantes : **48.66913724637683, 1.87586057971015** ?

```
SELECT * FROM lpecom_cities WHERE gps_lat = 48.66913724637683 AND gps_lng = 1.87586057971015;
```

Sans jointure, quelle requête utiliser pour calculer le nombre de villes que compte le département de l'**Essonne** ?

```
SELECT COUNT(*) FROM lpecom_cities WHERE department_code = 91;
```

Sans jointure, quelle requête utiliser pour calculer le nombre de villes en Île-de-France se terminant par "**-le-Roi**" ?

```
SELECT COUNT(*) FROM lpecom_cities WHERE name LIKE "%-le-Roi";
```

Combien de villes possèdent le code postal (`zip_code`) 77320 ? Renommez la colonne de résultat `n_cities`.

```
SELECT COUNT(*) as n_cities FROM lpecom_cities WHERE zip_code = 77320;
```

Sans jointure, quelle requête utiliser pour calculer le nombre de villes commençant par "**Saint-**" en **Seine-et-Marne** ?

```
SELECT COUNT(*) FROM lpecom_cities WHERE name LIKE "SAINT-%" AND department_code = 77;
```

Sans jointure, quelles sont les deux villes de **Seine-et-Marne** à avoir le code postal (`zip_code`) le plus grand ?

```
SELECT * FROM lpecom_cities WHERE department_code = 77 ORDER BY zip_code DESC LIMIT 2;
```

Quel est le code postal (`zip_code`) le plus grand de la table `lpecom_cities` ?

```
SELECT MAX(zip_code) FROM lpecom_cities;
```

Avec un seul `WHERE` et aucun `OR`, quelle est la requête permettant d'afficher les départements des régions ayant le code suivant : 75, 27, 53, 84 et 93 ? Le résultat doit afficher le nom du département ainsi que le nom et le slug de la région associée.

```
SELECT d.name AS departement, r.name AS region, d.slug
FROM lpecom_departments d
INNER JOIN lpecom_regions r ON (d.region_code = r.code)
WHERE d.region_code IN (75, 27, 53, 84, 93);
```

Quelle requête utiliser pour obtenir en résultat, les noms de la région, du département et de chaque ville du département ayant pour `code` **77** ?

```
SELECT r.name as reg, d.name as dep, c.name as ville
FROM lpecom_cities c
INNER JOIN lpecom_departments d ON (c.department_code = d.code)
INNER JOIN lpecom_regions r ON (d.region_code = r.code)
WHERE d.code = 77;
```

# Partie 5

Tables lpecom_covid,lpecom_regions

Quelle requête utiliser pour afficher toutes les données de vaccination uniquement pour **le 1er avril 2021** ?

```
SELECT c.*
FROM lpecom_covid c
WHERE jour = '2021-04-01';
```

Quelle requête utiliser pour afficher toutes les données de vaccination uniquement pour **le 1er avril 2021** avec le nom de la région concernée ?

```
SELECT r.name, c.*
FROM lpecom_covid c
```

```
INNER JOIN lpecom_regions r ON c.id_region = r.code
WHERE jour = '2021-04-01';
```

Quelle requête utiliser pour afficher le nombre au cumulé de vaccination première dose toutes régions en **2020** ? Proposez également une solution pour les vaccination deuxième dose.

```
SELECT SUM(n_dose1)
FROM lpecom_covid c
WHERE jour <= '2020-12-31';
SELECT SUM(n_dose2)
FROM lpecom_covid c
WHERE jour <= '2020-12-31';
```

Quelle requête SQL utiliser pour afficher le nombre au cumulé de vaccination première dose pour la région avec le code `93` uniquement pour le mois de **mars 2021** ?

```
SELECT SUM(n_dose1)
FROM lpecom_covid c
WHERE id_region = '93'
AND jour BETWEEN '2021-03-01' AND '2021-03-31';
```

Quelle requête utiliser pour afficher le nombre au cumulé de vaccination deuxième dose pour la région avec le code `11` uniquement pour le mois de **mars 2021** ?

```
SELECT SUM(n_dose2)
FROM lpecom_covid c
WHERE id_region = '11'
AND jour BETWEEN '2021-03-01' AND '2021-03-31';
```

Quelle requête SQL utiliser pour afficher le record de vaccination **première dose** en une seule journée ? Avec une deuxième requête, afficher les informations de la région concernée, dont son nom, ainsi que le jour du record.

```
SELECT MAX(n_dose1)
FROM lpecom_covid c;

SELECT c.*, r.name
FROM lpecom_covid c
INNER JOIN lpecom_regions r ON c.id_region = r.code
WHERE c.n_dose1 >= 56661;
```

Quelle requête utiliser pour afficher le record de vaccination **deuxième dose** en une seule journée ? Avec une deuxième requête, afficher les informations de la région concernée, dont son nom, ainsi que le jour du record.

```
SELECT MAX(n_dose2)
FROM lpecom_covid c;

SELECT c.*, r.name
FROM lpecom_covid c
INNER JOIN lpecom_regions r ON c.id_region = r.code
WHERE c.n_dose2 >= 21524;
```

Quelles requêtes permettent de connaitre quelle région possède la plus grande couverture de vaccination avec une dose et deux doses ? Vous aurez besoin de 4 requêtes pour répondre aux deux questions. Vous aurez besoin du résultat de la première requête pour la deuxième.

```
SELECT MAX(couv_dose1)
FROM lpecom_covid c;

SELECT c.*, r.name
FROM lpecom_covid c
INNER JOIN lpecom_regions r ON c.id_region = r.code
WHERE c.couv_dose1 >= 19.7;

SELECT MAX(couv_dose2)
FROM lpecom_covid c;

SELECT c.*, r.name
FROM lpecom_covid c
INNER JOIN lpecom_regions r ON c.id_region = r.code
WHERE c.couv_dose2 >= 8;
```

Quelle requête utiliser pour afficher le nom de la région qui a le plus faible taux de couverture de vaccination avec une dose ? Vous aurez besoin de 2 requêtes pour répondre à la question.

```
SELECT MIN(c.couv_dose1)
FROM lpecom_covid c
WHERE c.jour = '2021-04-06';
```

```
SELECT c.*, r.name
FROM lpecom_covid c
INNER JOIN lpecom_regions r ON c.id_region = r.code
WHERE c.jour = '2021-04-06'
AND c.couv_dose1 <= 2.80;
```

Quelle requête utiliser pour calculer la couverture moyenne entre les différentes régions à la date la plus récente, pour les vaccinations une et deux doses ? Vous renommez les colonnes de résultats : `couverture_dose1_avg` et `couverture_dose2_avg`.

```
SELECT AVG(c.couv_dose1) AS couverture_dose1_avg, AVG(c.couv_dose2) AS couverture_dose2_avg
FROM lpecom_covid c
WHERE c.jour = '2021-04-06';
```

Quelle requête utiliser pour afficher les données de vaccination des régions (avec leur nom) qui possèdent une couveture vaccinale **supérieure à 15 %** pour la première dose et **supérieure à 5 %** pour la deuxième dose ?

```
SELECT c.*, r.name
FROM lpecom_covid c
INNER JOIN lpecom_regions r ON c.id_region = r.code
WHERE c.couv_dose1 >= 15
AND c.couv_dose2 >= 5
AND c.jour = '2021-04-06';
```

# Partie 6

table lpecom_departments,lpecom_covid_vaccin_type,lpecom_covid_vaccin

Sans jointure, quelle requête SQL utiliser pour afficher **toutes les données de vaccination** du **14 février 2021** uniquement, pour le département de **Seine-et-Marne (77)** ?

```
SELECT *
FROM lpecom_covid_vaccin v
WHERE v.jour = '2021-02-14'
AND v.dep_code = 77;
```

Sans jointure, quelle requête SQL utiliser pour afficher **le cumul de toutes les données de vaccination pour tous les vaccins** du **14 février 2021** uniquement, pour les départements de l'**Essonne (91)** et de la **Seine-et-Marne (77)** ?

```
SELECT *
FROM lpecom_covid_vaccin v
WHERE v.jour = '2021-02-14'
AND v.dep_code IN (77, 91)
AND v.vaccin = 0;
```

Sans jointure, quelle requête utiliser pour afficher la **somme des vaccinations première dose** réalisées uniquement avec le vaccin **AstraZeneka** pour le mois de **février 2021** pour le département de la **Seine-et-Marne (77)** ?

```
SELECT SUM(v.n_dose1)
FROM lpecom_covid_vaccin v
WHERE v.dep_code = 77
AND v.jour BETWEEN '2021-02-01' AND '2021-02-31'
AND v.vaccin = 3;
```

Sans jointure, quelle requête utiliser pour afficher la **somme des vaccinations deuxième dose** réalisées avec le vaccin **AstraZeneka** ou **Moderna** pour le mois de **mars 2021** pour le département de la **Seine-et-Marne (77)** ?

```
SELECT SUM(v.n_dose2)
FROM lpecom_covid_vaccin v
WHERE v.dep_code = 77
AND v.jour BETWEEN '2021-03-01' AND '2021-03-31'
AND v.vaccin IN (2, 3);
```

Sans jointure, quelle requête utiliser pour afficher le **record de vaccination première dose** avec un type de vaccin en **une seule journée** ? Avec une deuxième requête qui exploitera une jointure, afficher toutes les informations possibles pour cette journée record et sur le type de vaccin.

```
SELECT MAX(v.n_dose1)
FROM lpecom_covid_vaccin v
WHERE v.vaccin != 0;

SELECT *
FROM lpecom_covid_vaccin v
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
WHERE v.vaccin != 0
```

```
AND v.n_dose1 >= 7494;
```

Sans jointure, quelle requête utiliser pour afficher **le record de vaccination deuxième dose** avec un type de vaccin en une **seule journée** ? Avec une deuxième requête qui exploitera deux jointures, afficher toutes les informations possibles pour cette journée record, sur le type de vaccin et sur le département.

```
SELECT MAX(v.n_dose2)
FROM lpecom_covid_vaccin v
WHERE v.vaccin != 0;

SELECT *
FROM lpecom_covid_vaccin v
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
INNER JOIN lpecom_departments d ON d.code = v.dep_code
WHERE v.vaccin != 0
AND v.n_dose2 >= 5046;
```

Quelle requête permet de savoir quel département possède le **plus grand nombre d'injections première dose** pour le vaccin **AstraZeneka** ? Avec une deuxième requête, afficher uniquement les colonnes suivantes :

- le nom du vaccin ;
- le jour ;
- le nom et le code du département ;
- le nombre cumulé d'injections.

```
SELECT MAX(v.n_cum_dose1)
FROM lpecom_covid_vaccin v
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
INNER JOIN lpecom_departments d ON d.code = v.dep_code
WHERE jour = '2021-04-06'
AND v.vaccin = 3;

SELECT v.jour, t.nom, v.n_cum_dose1, d.code, d.name
FROM lpecom_covid_vaccin v
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
INNER JOIN lpecom_departments d ON d.code = v.dep_code
WHERE jour = '2021-04-06'
AND v.vaccin = 3
AND v.n_cum_dose1 >= 122709;
```

Quelle requête permet de savoir quel département a eu le **moins de vaccinations première dose** avec le vaccin **COMIRNATY Pfizer/BioNTech** ? Avec une deuxième requête, afficher uniquement les colonnes suivantes :

- le nom du vaccin ;
- le jour ;
- le nom et le code du département ;
- le nombre cumulé d'injections.

```
SELECT MIN(v.n_cum_dose1)
FROM lpecom_covid_vaccin v
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
INNER JOIN lpecom_departments d ON d.code = v.dep_code
WHERE jour = '2021-04-06'
AND vaccin = 1;


SELECT v.jour, t.nom, v.n_cum_dose1, d.code, d.name
FROM lpecom_covid_vaccin v
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
INNER JOIN lpecom_departments d ON d.code = v.dep_code
WHERE jour = '2021-04-06'
AND v.vaccin = 1
AND v.n_cum_dose1 <= 90832;
```

Quelle requête permet de connaître la moyenne de vaccinations **première dose** dans tous les départements pour le vaccin **Moderna** ? Renommer la colonne de résultat avec `avg_moderna`.

```
SELECT AVG(n_cum_dose1) AS avg_moderna
FROM lpecom_covid_vaccin v
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
INNER JOIN lpecom_departments d ON d.code = v.dep_code
WHERE v.jour = '2021-04-06'
AND v.vaccin = 2;
```

Quelle requête utiliser pour afficher les départements (avec leur nom) qui possèdent un nombre d'injections deuxième dose avec le vaccin **Moderna** supérieur à **9000** ou un nombre d'injections première dose avec le vaccin **COMIRNATY Pfizer/BioNTech** supérieur à **120000** ? Vous aurez besoin de deux jointures.

```
SELECT v.jour, t.nom, v.n_cum_dose1, d.code, d.name
FROM lpecom_covid_vaccin v
```

```
INNER JOIN lpecom_covid_vaccin_type t ON t.id = v.vaccin
INNER JOIN lpecom_departments d ON d.code = v.dep_code
WHERE (v.jour = '2021-04-06' AND v.vaccin = 1 AND v.n_cum_dose1 > 120000)
OR (v.jour = '2021-04-06' AND v.vaccin = 2 AND v.n_cum_dose2 > 9000);
```

# Partie 7

Table lpecom_rpps

Quelle requête SQL utiliser pour compter, sans doublons, le nombre de professionnels de santé en **Seine-et-Marne (77)** ?

```
SELECT COUNT(DISTINCT id_pp_nat)
FROM lpecom_rpps;
```

Quelle requête SQL utiliser pour afficher pour tous les professionnels de santé avec le code postal **77300** les colonnes suivantes : `id_pp_nat`, `prenom`, `nom`, `code_postal`, `ville`, `departement` et `région`. Vous aurez besoin de plusieurs jointures.

```
SELECT rpps.id_pp_nat, rpps.prenom, rpps.nom, rpps.code_postal, c.name as ville, d.name as
departement, r.name as region
FROM lpecom_rpps rpps
INNER JOIN lpecom_cities c ON (rpps.code_postal = c.zip_code)
INNER JOIN lpecom_departments d ON (c.department_code = d.code)
INNER JOIN lpecom_regions r ON (d.region_code = r.code)
WHERE rpps.code_postal = 77300;
```

# Exercice Optimization

```
CREATE TABLE employees (

   emp_no      INT             NOT NULL,
   birth_date  DATE            NOT NULL,
   first_name  VARCHAR(14)     NOT NULL,
   last_name   VARCHAR(16)     NOT NULL,
   gender      ENUM ('M','F')  NOT NULL,
   hire_date   DATE            NOT NULL,
);

CREATE TABLE departments (

   dept_no     CHAR(4)         NOT NULL,
   dept_name   VARCHAR(40)     NOT NULL
);

CREATE TABLE dept_manager (

  dept_no      CHAR(4)         NOT NULL,
  emp_no       INT             NOT NULL,
  from_date    DATE            NOT NULL,
  to_date      DATE            NOT NULL
);

CREATE TABLE dept_emp (

   emp_no      INT             NOT NULL,
   dept_no     CHAR(4)         NOT NULL,
   from_date   DATE            NOT NULL,
   to_date     DATE            NOT NULL
);

CREATE TABLE titles (

   emp_no      INT             NOT NULL,
```

```
   title       VARCHAR(50)     NOT NULL,
   from_date   DATE            NOT NULL,
   to_date     DATE);
CREATE TABLE salaries (

   emp_no      INT             NOT NULL,
   salary      INT             NOT NULL,
   from_date   DATE            NOT NULL,
   to_date     DATE            NOT NULL
);
```

- Requete pour avoir les employées feminin:

```
 select * from employees where last_name='titi'  OR gender='F'
```

- Requete RH pour faire de la BI

```
SELECT T1.emp_no, T1.gender,SUM(T1.salary), T1.hire_date, T1.birth_date from
(SELECT employees.emp_no, employees.gender, salaries.salary, employees.hire_date,
employees.birth_date
 FROM employees INNER JOIN salaries ON employees.emp_no = salaries.emp_no) AS T1 GROUP BY
T1.salary
```

- Requete les employées Feminines ainsi que les noms des département

```
SELECT departments.dept_name, employees.first_name, employees.last_name
FROM employees INNER JOIN dept_emp ON dept_emp.emp_no = employees.emp_no
INNER JOIN departments ON dept_emp.dept_no = departments.dept_no
WHERE employees.gender = 'F'
```

- Requete pour avoir les salaire des senior staff

```
SELECT employees.first_name, employees.last_name, titles.title, salaries.salary
FROM employees INNER JOIN titles ON titles.emp_no = employees.emp_no
INNER JOIN salaries ON salaries.emp_no = employees.emp_no WHERE titles.title LIKE '%staff%'
```

# Vue Materialise

# Quelle est une vue matérialisée?

Une vue matérialisée (MV) est le résultat pré-calculée (matérialisée) d'une requête. Contrairement à une vue simple le résultat d'une vue matérialisée est stocké quelque part, généralement dans un tableau. Vues matérialisées sont utilisées lorsque la réponse immédiate est nécessaire et que la requête d'où la vue sur les bases matérialisées prendrait trop longtemps pour produire un résultat. Vues matérialisées doivent être rafraîchi de temps en temps. Cela dépend de la façon dont les exigences souvent une vue matérialisée est rafraîchie et comment réelle de son contenu est. Fondamentalement, une vue matérialisée peut être rafraîchi immédiatement ou à terme, il peut être actualisé en totalité ou un certain point dans le temps. MySQL ne fournit pas vues matérialisées par lui-même. Mais il est facile de construire des vues matérialisées vous-même. Mettre en œuvre vos propres vues matérialisées

Un petit exemple de la façon dont cela pourrait être fait est la requête suivante:

```
SELECT COUNT(*)
 FROM MyISAM_table;
```

renvoie un résultat immédiat parce que le compteur est stockée dans l'en-tête de la table. La requête suivante peut prendre quelques secondes à minutes:

```
SELECT COUNT(*)
FROM innodb_huge;
```

Une solution possible serait de créer une table où tous les chefs de ligne, InnoDB sont stockés dans:

```
CREATE TABLE innodb_row_count (
  id          INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
 , schema_name VARCHAR(64)  NOT NULL
 , table_name  VARCHAR(64)  NOT NULL
 , row_count   INT UNSIGNED NOT NULL
 );
```

n fonction de la justesse nécessaire de cette information, le tableau peut être actualisée une fois par jour (moins utilisé les ressources du système, mais plus grosse erreur dans le résultat), une fois

par heure ou dans le cas le plus extrême, après chaque changement (le plus lent)!

Une autre possibilité serait d'obtenir les données à partir du schéma de l'information. Mais cette information peut être jusqu'à 20% tort!

```
SELECT table_schema, table_name, table_rows  FROM information_schema.tables
WHERE table_type = 'BASE TABLE';
```

# Actualiser les vues matérialisées

Vues matérialisées peuvent être actualisées de différentes manière:

- jamais (une seule fois au début, pour des données statiques seulement)
- sur demande (par exemple une fois par jour, par exemple après le chargement de nuit)
- immédiatement (après chaque instruction)

Une actualisation peut se faire par les moyens suivants:

- complètement (lent, plein de zéro)
- différée (rapide, par une table de journal)

En stockant les informations de modification dans une table de journalisation. En outre, certains clichés ou retardées temps, les États peuvent être produites:

- rafraîchir à jour
- rafraîchissement complet

# En pratique

Pour comprendre tout cela plus en détail, il est probablement plus facile de faire des exemples. Supposons que nous avons un chiffre d'affaires de table

```
CREATE TABLE sales (

   sales_id       INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
 , product_name   VARCHAR(128) NOT NULL
 , product_price  DECIMAL(8,2) NOT NULL
 , product_amount SMALLINT     NOT NULL

);
```

```
INSERT INTO sales VALUES

 (NULL, 'Apple', 1.25, 1), (NULL, 'Apple', 2.40, 2)

, (NULL, 'Apple', 4.05, 3), (NULL, 'Pear', 6.30, 2) , (NULL, 'Pear', 12.20, 4), (NULL, 'Plum', 4.85, 3)

SELECT * FROM sales;
```

Et maintenant, nous voulons connaître le prix vendu et l'argent gagné par produit:

```
EXPLAIN

SELECT product_name

    , SUM(product_price) AS price_sum, SUM(product_amount) AS amount_sum
    , AVG(product_price) AS price_avg, AVG(product_amount) amount_agg
    , COUNT(*)
 FROM sales
GROUP BY product_name
ORDER BY price_sum;

+-------------+-------+------+---------------+------+---------------------------------+
|select_type | table | type | possible_keys | rows | Extra |
+-------------+-------+------+---------------+------+---------------------------------+
| SIMPLE | sales | ALL | NULL | 6 | Using temporary; Using filesort |
+-------------+-------+------+---------------+------+---------------------------------+
```

# Créer votre propre vue matérialisée

```
DROP TABLE sales_mv; CREATE TABLE sales_mv (

   product_name VARCHAR(128)  NOT NULL
 , price_sum    DECIMAL(10,2) NOT NULL
 , amount_sum   INT           NOT NULL
```

```
 , price_avg    FLOAT         NOT NULL
 , amount_avg   FLOAT         NOT NULL
 , sales_cnt    INT           NOT NULL
 , UNIQUE INDEX product (product_name)

);

INSERT INTO sales_mv SELECT product_name

   , SUM(product_price), SUM(product_amount)
   , AVG(product_price), AVG(product_amount)
   , COUNT(*)
 FROM sales

GROUP BY product_name;
```

C'est jusqu'à présent la partie la plus facile! Et, comme prévu, nous obtenons le résultat correct:

```
 mysql> SELECT * FROM sales_mv;
 +--------------+-----------+------------+-----------+------------+-----------+
| product_name | price_sum | amount_sum | price_avg | amount_avg | sales_cnt
| +--------------+-----------+------------+-----------+------------+-----------+
| Apple | 7.70 | 6 | 2.56667 | 2 | 3 |
| Pear | 18.50 | 6 | 9.25 | 3 | 2 | | Plum | 4.85 | 3 | 4.85 | 3 | 1 |
 +--------------+-----------+------------+-----------+------------+-----------+
 3 rows in set (0.00 sec)
```

Cela couvre le mode de rafraîchissement "JAMAIS" Mais ce n'est pas ce que nous voulons généralement à faire. Actualiser vue matérialisée sur demande

L'actualisation de la vue matérialisée sur la demande peut être mis en œuvre avec une procédure stockée comme suit:

```
ROP PROCEDURE refresh_mv_now;

DELIMITER $$

CREATE PROCEDURE refresh_mv_now (

   OUT rc INT
```

```
) BEGIN

 TRUNCATE TABLE sales_mv;

 INSERT INTO sales_mv
 SELECT product_name
     , SUM(product_price), SUM(product_amount)
     , AVG(product_price), AVG(product_amount)
     , COUNT(*)
   FROM sales
 GROUP BY product_name;

 SET rc = 0;

END; $$

DELIMITER ;
```

Pour vérifier si cela fonctionne la déclaration suivante a été utilisée:

```
CALL refresh_mv_now(@rc);

SELECT * FROM sales_mv;

+-------------+----------+-----------+----------+-----------+----------+
| product_name | price_sum | amount_sum | price_avg | amount_avg | sales_cnt
| +-------------+----------+-----------+----------+-----------+----------+
| Apple | 7.70 | 6 | 2.56667 | 2 | 3 |
| Pear | 18.50 | 6 | 9.25 | 3 | 2 |
| Plum | 4.85 | 3 | 4.85 | 3 | 1 |
+-------------+----------+-----------+----------+-----------+----------+

INSERT INTO sales VALUES

 (NULL, 'Apple', 2.25, 3), (NULL, 'Plum', 3.35, 1)

, (NULL, 'Pear', 1.80, 2);

CALL refresh_mv_now(@rc);
```

```
SELECT * FROM sales_mv;

+-------------+----------+-----------+----------+-----------+----------+
| product_name | price_sum | amount_sum | price_avg | amount_avg | sales_cnt
| +-------------+----------+-----------+----------+-----------+----------+
| Apple | 9.95 | 9 | 2.4875 | 2.25 | 4 | | Pear | 20.30 | 8 | 6.76667 | 2.66667 | 3 |
| Plum | 8.20 | 4 | 4.1 | 2 | 2 |
+-------------+----------+-----------+----------+-----------+----------+
```

# Actualiser vue matérialisée immédiate

Pour faire un rafraîchissement complet après chaque déclaration n'a pas de sens. Mais nous aimerions toujours avoir à bon résultat. Pour ce faire, il est un peu plus compliqué.

Sur chaque INSERT sur la table de vente, nous devons mettre à jour notre vue matérialisée. Nous pouvons mettre en œuvre cette manière transparente par INSERT / UPDATE / DELETE déclencheurs sur la table sales:

Maintenant, laissez-nous créer les déclencheurs nécessaires:

```
 DELIMITER $$

CREATE TRIGGER sales_ins AFTER INSERT ON sales FOR EACH ROW BEGIN

 SET @old_price_sum = 0;
 SET @old_amount_sum = 0;
 SET @old_price_avg = 0;
 SET @old_amount_avg = 0;
 SET @old_sales_cnt = 0;

 SELECT IFNULL(price_sum, 0), IFNULL(amount_sum, 0), IFNULL(price_avg, 0)
      , IFNULL(amount_avg, 0), IFNULL(sales_cnt, 0)
   FROM sales_mv
  WHERE product_name = NEW.product_name
   INTO @old_price_sum, @old_amount_sum, @old_price_avg
      , @old_amount_avg, @old_sales_cnt
 ;
```

```
 SET @new_price_sum = @old_price_sum + NEW.product_price;
 SET @new_amount_sum = @old_amount_sum + NEW.product_amount;
 SET @new_sales_cnt = @old_sales_cnt + 1;
 SET @new_price_avg = @new_price_sum / @new_sales_cnt;
 SET @new_amount_avg = @new_amount_sum / @new_sales_cnt;

 REPLACE INTO sales_mv
 VALUES(NEW.product_name, @new_price_sum, @new_amount_sum, @new_price_avg
      , @new_amount_avg, @new_sales_cnt)
 ;

END; $$

DELIMITER ;
```