BDW1 : Bases de données et Programmation Web
SQL
Partie 3 - Fonctions diverses et commandes SQL

Nicolas Lumineau

nicolas.lumineau@univ-lyon1.fr

Université Claude Bernard Lyon 1

Licence 2^e année - 2016/2017

Enseignements de BDW1

1. [BD & Web] Introduction
2. [BD]La syntaxe SQL
   - SQL : de la projection à la jointure
   - SQL : sous-requêtes et regroupements
   - Fonctions diverses et commandes SQL
3. [BD]Conception de base de données
   - Schéma Entité/Association
   - Du modèle conceptuel au modèle relationnel
4. [Web] Programmation Web
   - Le langage HTML/CSS
   - Le langage PHP
     - Interrogation d'une BD via PHP
5. [BD] Optimisation de requête
   - Algèbre Relationnelle
   - Transformation d'arbres algébriques

Partie 3 : Le langage SQL

1. Fonctions diverses
   - Pour les valeurs de type numérique
   - Pour les valeurs de type chaînes de caractères
   - Pour les valeurs de type dates
   - Conversion
2. Commandes SQL pour créer, modifier et supprimer
   - Créer
   - Modifier
   - Supprimer
3. Contraintes d'intégrité
   - Attribut non vide
   - Unicité des valeurs
   - Contrainte de clé primaire
   - Contrainte de clé étrangère

Remarque

Certains besoins d'utilisateurs peuvent être de la forme :

Pour chaque artiste dont on précisera le nom et le prénom, donner :

- le montant de leur salaire concaténé par ' euros' si ce dernier est une valeur paire,
- la chaîne de caractères 'Reste : ' concaténée au reste de la division entière du salaire par 10 si la valeur du montant est impaire .

Le résultat sera présenté sous le schéma (PrenomNom, Montant) où le prénom et le nom seront concaténés dans l'attribut PrenomNom, avec le prénom affiché en majuscule.

Remarque

Ou de la forme :

- Pour chaque nom et prénom d'artiste donner un code de la forme suivante XX-YYY-ZZZZ-AA-BB-CC, où :
  - XX : le code du pays de nationalité de l'artiste
  - YYY : les 3 premiers caractères du nom de l'artiste
  - ZZZZ : les 4 premiers caractères du prénom de l'artiste dans lesquels la lettre 'A' est remplacé par un 'O'
  - AA-BB-CC : la date du jour à laquelle 1 an, 6 mois et 12 jours ont été ajouté.

Ce que nous avons appris de la syntaxe SQL jusqu'à présent, ne nous permet pas de répondre à ces besoins.

Fonctions diverses

Chaque SGBD propose des fonctions permettant d'exprimer des opérations ou des transformations des valeurs d'un attribut. Comme l'implémentation (nom + définition) de ces fonctions est propre à chaque SGBD, il est important de se référer à la documentation spécifique du SGBD utilisé !

- MySQL : `http://dev.mysql.com/doc/`
- POSGRESQL : `http://docs.postgresqlfr.org/`
- ORACLE : `http://www.oracle.com/pls/db102/`
- ...

Fonctions diverses

Il est possible d'utiliser des expressions plus complexes que de simples attributs dans le select

Entre autres :

- Fonctions et expressions arithmétiques
- Fonctions sur les chaînes de caractères
- Fonctions sur les dates
- Fonctions de conversion

Fonctions sur les valeurs numériques

Quelques fonctions que l'on peut appliquer à des valeurs numériques :

- $att_1 + att_2$: somme de att_1 et de att_2.
- $att_1 - att_2$: soustraction de att_2 à la valeur de att_1 .
- $att_1 * att_2$: produit de att_1 et de att_2.
- att_1 / att_2 : division de att_1 par att_2.
- $ABS(att)$: valeur absolue de att
- $COS(att)$: cosinus de att avec att en radians
- $SQRT(att)$: racine carrée de att
- $MOD(m, n)$: reste de la division entière de m par n, vaut 0 si $n = 0$;

Fonctions sur les valeurs numériques

- *ROUND*(*att*, *n*) : valeur arrondie de *att* à *n* chiffres après la virgule, *n* optionnel et vaut 0 par défaut
- *TRUNC*(*att*, *n*) : valeur tronquée de *att* à *n* chiffres après la virgule, *n* optionnel et vaut 0 par défaut

Pour ROUND et TRUNC, si *n* est négatif cela indique des chiffres avant la virgule

Exemple

Donner les titres de films et une valeur nommée 'Euhhh' qui vaut la somme du budget du film auquel on ajoute le reste de la division entière de l'année, auquel on aura retranché la racine carrée de l'identifiant du réalisateur multiplié par 3, par 10. Le résultat sera arrondi à 2 chiffres après la virgule :

```
SELECT titre,
    ROUND(budget + MOD( annee - SQRT(3 * real) ,10)
                 ,2) AS Euhhh
FROM Film
```

Résultat :

titre	Euhhh
Edge of tomorrow	187.42
Lucy	49.10
The Artist	14.80
OSS 117 : Le Caire, Nid d'espions	14.80
Bienvenue chez les Ch'tis	15.13

Fonctions sur les chaînes de caractères

Quelques fonctions que l'on peut appliquer à des chaînes de caractères :

- $CONCAT(v_1, v_2, v_3, ...)$: concaténation des valeurs v_1, v_2, v_3,
  - Si le SGBD ne gère le $CONCAT$ avec plus de deux arguments, il est nécessaire d'imbriquer les $CONCAT$: $CONCAT(v_1, CONCAT(v_2, CONCAT(v_3, ...)))$
- $REPLACE(att, v\_old, v\_new)$ renvoie att dans laquelle les occurrences de v_old ont été remplacées par v_new
- $UPPER(att)$ convertit att en majuscule
- $LOWER(att)$ convertit att en minuscule
- $LENGTH(att)$: longueur en nombre de caractères de att

Fonctions sur les chaînes de caractères

Quelques fonctions que l'on peut appliquer à des chaînes de caractères :

- $INSTR(att, val)$: position de la 1^{ere} occurrence de val dans att
- $SUBSTR(att, n, l)$ ou $SUBSTRING(att, n, l)$ renvoie la sous-chaîne de att commençant au caractère n et de longueur l
  - si l n'est pas précisé, on prend la sous-chaîne du caractères n jusqu'à la fin de e

Exemple

Donner le nom et le prénom des artistes en affichant le nom en minuscule et en ayant remplacé les caractères 'ON' par 'OUNNE' dans le nom. Le résultat s'affichera dans un attribut nommé 'PNOUNNE' :

```
SELECT CONCAT(prenom, ' ', LOWER(
    REPLACE(nom,'ON', 'OUNNE'))) AS PNOUNNE
FROM Artiste
```

Résultat :

PNOUNE
Jean dujardin
Scarlett johanssounne
Tom cruise
Bérénice bejo
Dany bounne
Emily blunt
Doug liman
Luc bessounne
Michel hazanavicius

Fonctions sur les valeurs de type date

Quelques fonctions que l'on peut appliquer à des dates :

- $YEAR(d)$:retourne l'année de la date d
- $MONTH(d)$:retourne le mois de la date d
- $DAY(d)$:retourne le jour de la date d
- $ADDDATE(d, INTERVAL\ n\ DAY)$: ajoute n jours à d
  - DAY peut être remplacé par $SECOND$, $MINUTE$, $HOUR$, $MONTH$, ou $YEAR$

Fonctions sur les valeurs de type date

Quelques fonctions que l'on peut appliquer à des dates :

- $SUBDATE(d, INTERVAL\ n\ DAY)$: similaire à $ADDDATE$, mais effectue une soustraction
- $DATEDIFF(d_1, d_2)$: nombre de jour entre d_1 et d_2
- $SYSDATE()$, $NOW()$: date courante

Remarque: : les fonctions sont vraiment différentes sur ORACLE, POSGRES $\Rightarrow$ cf documentation

Exemple

Pour chaque nom d'artiste, donner la date correspondant à la date du jour augmenté de *n* mois, où *n* correspond au nombre de lettres composant le prénom de l'artiste. Le résultat s'affichera dans un attribut nommé 'DesDates' :

```
SELECT nom,
 ADDDATE( SYSDATE(),
                INTERVAL (LENGTH(prenom)) MONTH)
        AS DesDAtes
FROM Artiste
```

Fonction de conversion

- *CAST*(*e AS type*) ou *CONVERT*(*e*, *type*) : convertit *e* en *type*
  - *type* peut être BINARY, CHAR, DATE, TIME, DATETIME, SIGNED, UNSIGNED
- *ASCII*(*e*) renvoie le code ASCII du premier caractère de *e*

Remarque: : les fonctions sont vraiment différentes sur ORACLE, POSGRES $\Rightarrow$ cf documentation

Retour au besoin

Pour chaque artiste dont on précisera le nom et le prénom, donner :

- le montant de leur salaire concaténé par ' euros' si ce dernier est une valeur paire,
- la chaîne de caractères 'Reste : ' concaténée au reste de la division entière du salaire par 10 si la valeur du montant est impaire .

Le résultat sera présenté sous le schéma (PrenomNom, Montant) où le prénom et le nom seront concaténés dans l'attribut PrenomNom, où le prénom sera affiché en majuscule.

Est-il possible de construire cette requête maintenant ?

Retour au besoin

Réponse : NON, pas encore !
En effet, il y a une notion d'alternative dans le résultat que nous ne savons pas encore exprimer.
$\Rightarrow$ Enrichissement de la clause **SELECT**
avec le **CASE** ...**WHEN** ...**END** .

Expression du **CASE**

SELECT CASE WHEN $< cond_1 >$ **THEN** $< cons_1 >$
WHEN $< cond_2 >$ **THEN** $< cons_2 >$
ELSE *alter* **END**
FROM *nomTable*
...;

- Le **CASE** permet d'introduire des conditions ($< cond_i >$) et les conséquences ($< cons_i >$) qui seront exécutées si la condition est vérifiée.
- La valeur alternative ($< alter >$) retournée si aucune condition n'est vérifiée est introduite par le **ELSE**
- Le **CASE** est finalisé par le **END**

Exemple

Pour chaque film, donner le titre et l'information 'OLD' si l'année de sortie est inférieure strictement à 2010, 'RECENT' si l'année est comprise entre 2010 et 2016 et 'NEW' si l'année est 2017. Dans tous les autres cas, 'N/A' sera affiché. L'information s'affichera dans le résultat sous le nom 'TypeFilm' :

```
SELECT titre, CASE WHEN annee < 2010 THEN 'OLD'
  WHEN annee BETWEEN 2010 AND 2016 THEN 'RECENT'
  WHEN annee = 2017 THEN 'NEW'
  ELSE 'N/A' END AS TypeFilm
FROM Film
```

Résultat :

titre	TypeFilm
Edge of tomorrow	RECENT
Lucy	RECENT
The Artist	RECENT
OSS 117 : Le Caire, Nid d'espions	OLD
Bienvenue chez les Ch'tis	OLD

Partie 3 : Le langage SQL

1. Fonctions diverses
   - Pour les valeurs de type numérique
   - Pour les valeurs de type chaînes de caractères
   - Pour les valeurs de type dates
   - Conversion
2. Commandes SQL pour créer, modifier et supprimer
   - Créer
   - Modifier
   - Supprimer
3. Contraintes d'intégrité
   - Attribut non vide
   - Unicité des valeurs
   - Contrainte de clé primaire
   - Contrainte de clé étrangère

Au commencement...

L'informaticien souhaite implémenter la relation

Personne(idp, nom, prenom, dateNaiss)

dans une base MySQL .

Pour l'utilisation d'autres SGBD il sera nécessaire de se référer à la documentation.

Création d'une table

L'informaticien, qui dispose d'un SGBD relationnel, commence par créer une table selon la syntaxe

```
CREATE TABLE nomTable ( att1 type1, att2 type2, ... ) ;
```

avec

- nomTable le nom de la table,
- att_i le i^e attribut,
- $type_i$ le type du i^e attribut.

Ce qui donne :

```
CREATE TABLE personne( idp int(5), nom varchar(50), prenom varchar(50), dateNaiss date()) ;
```

Création d'une table

Pour l'instant nous avons une table vide.

Personne

idp	nom	prenom	dateNaiss

Création d'un tuple

L'informaticien insère dans la table personne le tuple (18, 'Thaie', 'Marjorie', 1999-01-23) en respectant la syntaxe :

INSERT INTO `nomTable` (att_1, att_2,...) VALUES (val_1, val_2,...) ;

avec *nomTable* le nom de la table, att_i le i^e attribut et val_i la valeur associée au i^e attribut.

Il est important de noter que :

- c'est l'ordre des attributs et des valeurs qui font que la valeur val_i est associée à l'attribut att_i.
- il n'est pas nécessaire de spécifier tous les attributs (uniquement ceux dont on souhaite associer une valeur), par contre le nombre d'attributs spécifier à la gauche de VALUES doit correspondre au nombre de valeurs spécifiée à la droite de VALUES.

Création d'un tuple

Ce qui donne :

```
INSERT INTO personne (idp, nom, prenom, dateNaiss) VALUES
(18, 'Thaie', 'Marjorie', 1999-01-23) ;
```

ou de manière équivalente :

```
INSERT INTO personne (nom, prenom, idp, dateNaiss) VALUES
('Thaie', 'Marjorie', 18, 1999-01-23) ;
```

NB : dans le cas où l'informaticien souhaite insérer une personne dont il ne connaît pas l'âge, il pourrait écrire :

```
INSERT INTO personne (idp, nom, prenom) VALUES (18,
'Thaie', 'Marjorie') ;
```

Création d'une table

Pour l'instant nous avons une table avec un tuple.

Personne

idp	nom	prenom	dateNaiss
18	Thaie	Marjorie	1999-01-23

Petit bilan...

sur les mots clés de la syntaxe.

	Niveau schéma (table)	**Niveau instance (tuple)**
Créer	CREATE TABLE	INSERT INTO ... VALUES

Modification d'une table

L'informaticien décide d'effectuer les modifications suivantes :

- renommer l'attribut *dateNaiss* en *dateN*
- d'ajouter un attribut *numTel* de type chaînes d'au plus 15 caractères.

Modifier la table consiste à altérer son schéma.

Renommage/changement

```
ALTER TABLE nomTable CHANGE oldAtt newAtt typeNewAtt ;
```

avec nomTable le nom de la table, `oldAtt` l'attribut à renommer, `newAtt` le nouveau nom de l'attribut et `typeNewAtt` le type potentiellement modifié de l'attribut.

Modification d'une table

Ajout

```
ALTER TABLE nomTable ADD COLUMN newAtt typeNewAtt ;
```

avec nomTable le nom de la table, `newAtt` le nouveau nom de l'attribut et `typeNewAtt` le type du nouvel attribut.

Ce qui donne :

Renommage/changement

```
ALTER TABLE personne CHANGE dateNaiss dateN date ;
```

Ajout

```
ALTER TABLE nomTable ADD COLUMN numTel varchar(15) ;
```

Modification d'une table

La table ainsi modifiée ressemble à ça :

Personne

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL

Il est important de noter que l'attribut *numTel* qui a été créé après l'insertion du tuple, n'a pas de valeur. C'est pourquoi, il est à NULL.

Modification d'un tuple

L'informaticien souhaite attribuer le numéro 0400112233 au tuple dont l'idp vaut 18.

Mise à jour d'un tuple

UPDATE `nomTable` SET $\texttt{att}_i = \texttt{val}_i$, $\texttt{att}_j = \texttt{val}_j$ WHERE `condition` ;

avec nomTable le nom de la table, $\texttt{att}_i$ et $\texttt{att}_j$ les attributs modifiés et $\texttt{val}_i$ et $\texttt{val}_j$ les valeurs qui seront attribuées respectivement à $\texttt{att}_i$ et $\texttt{att}_j$. `condition` exprime la condition qui sera vérifiée par les tuples modifiés.
Ce qui donne :

```
UPDATE personne SET numTel = '0400112233'
WHERE idp = 18 ;
```

Modification d'un tuple

La table ainsi modifiée ressemble à ça :

Personne

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	0400112233

Petit bilan...

sur les mots clés de la syntaxe.

	Niveau schéma (table)	Niveau instance (tuple)
Créer	CREATE TABLE	INSERT INTO ... VALUES
Modifier	ALTER TABLE	UPDATE ...SET

Suppression d'un tuple

L'informaticien souhaite supprimer le tuple dont l'idp vaut 18.

Suppression d'un tuple

DELETE FROM `nomTable` WHERE `condition` ;

avec nomTable le nom de la table dans laquelle sera supprimé le (ou les tuples), et `condition` exprime la condition qui sera vérifiée par les tuples à supprimer.

Ce qui donne :

```
DELETE FROM personne
WHERE idp = 18 ;
```

Suppression d'un tuple

La table est ainsi modifiée :

Personne

idp	nom	prenom	dateNais	numTel

Suppression d'une table

L'informaticien décide de supprimer la table PERSONNE (qui existe encore même si elle est vide).

Renommage/changement

```
DROP TABLE nomTable ;
```

avec nomTable le nom de la table à supprimer.

Ce qui donne :

```
DROP TABLE personne ;
```

Petit bilan...

sur les mots clés de la syntaxe.

	Niveau schéma (table)	**Niveau instance (tuple)**
Créer	CREATE TABLE	INSERT INTO ... VALUES
Modifier	ALTER TABLE	UPDATE ...SET
Supprimer	DROP TABLE	DELETE

Et les contraintes d'intégrité dans tout ça ?

L'informaticien(ne) dispose des tables :
Il/elle souhaite déclarer que :

- Personne(<u>idp</u>, nom, prenom, dateBaiss, tel)
- Voiture(<u>immatriculation</u>, marque, modele, # proprio)

- une personne a forcément un nom
- il n'y a pas d'homonyme dans la base (unicité du couple prenom/nom)
- l'attribut *idp* de la table 'Personne' identifie de manière unique et sans ambiguïté un tuple
- l'attribut *proporio* de la table 'Voiture' référence un *idp* de la table 'Personne'

NOT NULL

L'informaticien(ne) souhaite déclarer que le nom d'une personne ne peut pas être vide.

Ajout d'un NOT NULL pour un attribut

ALTER TABLE `nomTable` MODIFY att_i $type_i$ NOT NULL ;

Ce qui donne :

ALTER TABLE `nomTable` MODIFY nom varchar(50) NOT NULL ;

NOT NULL

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL

Avec l'ajout du NOT NULL,

- il est possible d'exécuter :

Possible

```
INSERT INTO personne (idp, nom, prenom, dateNaiss, tel)
VALUES (2, 'LECENT', 'Aldo', 205-02-24 , '123456789' );
```

- il n'est pas possible d'exécuter (en tout cas une violation de contrainte d'intégrité sera levée)

Pas possible

```
INSERT INTO personne (idp, nom, prenom, dateNaiss, tel)
VALUES (2, NULL, 'Aldo', 205-02-24 , '123456789' );
```

UNIQUE

L'informaticien(ne) souhaite déclarer que le couple nom/prénom est unique.

Ajout d'un UNIQUE

ALTER TABLE `nomTable` ADD CONSTRAINT uk_nomTable UNIQUE (att_i,att_j,...) ;

Ce qui donne :

ALTER TABLE `Personne` ADD CONSTRAINT uk_personne UNIQUE (nom,prenom) ;

UNIQUE

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL
2	Lecent	Aldo	2005-02-24	123456789

Avec l'ajout du UNIQUE,

- il est possible d'exécuter :

Possible

```
INSERT INTO personne (idp, nom, prenom, dateNaiss, tel)
VALUES (25, 'Lecent', 'Marjorie', 1998-03-25 , NULL );
```

- il n'est pas possible d'exécuter (en tout cas une violation de contrainte d'intégrité sera levée)

Pas possible

```
INSERT INTO personne (idp, nom, prenom, dateNaiss, tel)
VALUES (26, 'Lecent', 'Aldo', 2004-04-26 , NULL );
```

PRIMARY KEY

L'informaticien(ne) souhaite déclarer l'attribut *idp* comme clé primaire pour s'assurer que chaque tuple aura une valeur de *idp* (NOT NULL) et que cette valeur sera unique.

ALTER TABLE `nomTable` ADD CONSTRAINT pk_nomTable PRIMARY KEY (att_i) ;

avec *nomTable* le nom de la table, *pk_nomTable* le nom de la contrainte de clé primaire, *att* le ou les noms des attributs de la table qui serviront de clé primaire.

PRIMARY KEY

Dans le cas où plusieurs attributs forment la clé primaire, la clé est dite **clé primaire composite** et s'écrit :

ALTER TABLE `nomTable` ADD CONSTRAINT pk_nomTable PRIMARY KEY (att_i, att_j, att_k) ;

La convention est que le nom de contraintes de clé primaire commence par *pk_*.

PRIMARY KEY

Ce qui donne :

```
ALTER TABLE Personne ADD CONSTRAINT pk_personne
PRIMARY KEY (idp);
```

Remarque : Si nous avions voulu mettre le couple nom et prénom comme clé primaire *à la place de idp* (car il ne peut y avoir qu'une seule et unique clé primaire par table !), il faudrait alors écrire :

```
ALTER TABLE personne ADD CONSTRAINT pk_personne
PRIMARY KEY (nom,prenom);
```

PRIMARY KEY

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL
2	Lecent	Aldo	2005-02-24	123456789
25	'Lecent'	'Marjorie'	'1998-03-25'	NULL

Avec l'ajout de la clé primaire,

- il est possible d'exécuter :

Possible

```
INSERT INTO personne (idp, nom, prenom, dateNaiss, tel)
VALUES (30, 'Hisson', 'Nour', 2016-11-02 , NULL ) ;
```

PRIMARY KEY

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL
2	Lecent	Aldo	2005-02-24	123456789
25	Lecent	Marjorie	1998-03-25	NULL

Avec l'ajout de la clé primaire,

- il n'est pas possible d'exécuter (en tout cas une violation de contrainte d'intégrité sera levée)

Pas possible

```
INSERT INTO personne (idp, nom, prenom, dateNaiss, tel)
VALUES (2, 'Hisson', 'Nour', 2016-11-02 , NULL );
```

Pas possible

```
INSERT INTO personne (idp, nom, prenom, dateNaiss, tel)
VALUES (NULL, 'Hisson', 'Nour', 2016-11-02 , NULL );
```

FOREIGN KEY

Nous considérons :

Personne

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL
2	Lecent	Aldo	2005-02-24	123456789
25	Lecent	Marjorie	1998-03-25	NULL

Voiture

immatriculation	marque	modele	prorio
AZ-123-ER	Peugeot	308	25

L'informaticien(ne) souhaite déclarer que l'attribut *proprio*, s'il a une valeur, référence une valeur de *idp* dans la table 'Personne'.

ALTER TABLE `nomTableSRC` ADD CONSTRAINT fk_tableSRC_TableDEST FOREIGN KEY (att_i) REFERENCES `nomTableDEST` (att_j) ;

FOREIGN KEY

Ce qui donne :

```
ALTER TABLE Voiture ADD CONSTRAINT
fk_personne_voiture FOREING KEY (proprio) REFERENCES
Personne (idp);
```

FOREIGN KEY

Personne

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL
2	Lecent	Aldo	2005-02-24	123456789
25	Lecent	Marjorie	1998-03-25	NULL

Voiture

immatriculation	marque	modele	prorio
AZ-123-ER	Peugeot	308	25

Avec l'ajout de la clé étrangère,

- il est possible d'exécuter :

Possible

```
INSERT INTO Voiture (immatriculation, marque, modele,
proprio)
VALUES ('QSD-456-FG', 'Citroën', 'C1', 18 );
```

Possible

```
INSERT INTO Voiture (immatriculation, marque, modele,
proprio)
```

FOREIGN KEY

Personne

idp	nom	prenom	dateNais	numTel
18	Thaie	Marjorie	1999-01-23	NULL
2	Lecent	Aldo	2005-02-24	123456789
25	Lecent	Marjorie	1998-03-25	NULL

Voiture

immatriculation	marque	modele	prorio
AZ-123-ER	Peugeot	308	25
QSD-456-FG	Citroën	C1	18

Avec l'ajout de la clé étrangère,

- il n'est pas possible d'exécuter (en tout cas une violation de contrainte d'intégrité sera levée)

Pas possible

INSERT INTO `Voiture` (immatriculation, marque, modele, proprio)
VALUES ('AQW-159-AS', 'Opel', 'Corsa', 9) ;

Scénario

Nous allons considéré nos artistes, films et rôle qui ont servi pour illustrer le cours.
Nous supposons que les commandes suivantes ont été exécutées :

Pour Artiste

CREATE TABLE Artiste (ida int(5), prenom varchar(50), nom varchar(50), nation varchar(10)) ;
ALTER TABLE Artiste ADD CONSTRAINT pk_artiste PRIMARY KEY (ida) ;

```
INSERT INTO Artiste VALUES (1,'Jean', 'DUJARDIN', 'FR'),
(2 , 'Scarlett', 'JOHANSSON', 'USA'),
(3 , 'Tom' , 'CRUISE' , 'USA'),
(4 , 'Bérénice' , 'BEJO' , 'FR-AR'),
(5 , 'Dany' , 'BOON' , 'FR'),
(6 , 'Emily' , 'BLUNT' , 'UK'),
(7 , 'Doug' , 'LIMAN' , 'USA'),
(8 , 'Luc' , 'BESSON' , 'FR'),
(9 , 'Michel' , 'HAZANAVICIUS' , 'FR') ;
```

Scénario

Pour Film

```
CREATE TABLE Film (idf int(5), titre varchar(50), annee int(4),
genre varchar(50),
reali int(5), budget float(5,2)) ;
ALTER TABLE Film ADD CONSTRAINT pk_film PRIMARY KEY
(idf) ;
ALTER TABLE Film ADD CONSTRAINT fk_film_artiste
FOREIGN KEY (reali)
REFERENCES Artiste(ida) ;
```

```
INSERT INTO Film VALUES (1 , 'Edge of tomorrow' , 2014 , 'SF' , 7 , 178),
(2 , 'Lucy' , 2014 , 'SF' , 8 , 40 ),
(3 , 'The Artist' , 2011 , 'Drame' , 9 , 9),
(4 , 'OSS 117 : Le Caire, Nid déspions' , 2006 , NULL , 9 , 14),
(5 , 'Bienvenue chez les Chtis' , 2008 , 'Comédie' , 5 , 11) ;
```

Scénario

Pour JoueDans

```
CREATE TABLE JoueDans (ida int(5), idf int(5), nom varchar(50),
sal float(5,2) ) ;

ALTER TABLE JoueDans ADD CONSTRAINT pk_joueDans
PRIMARY KEY (ida,idf) ;

ALTER TABLE JoueDans ADD CONSTRAINT
fk_jouedans_artiste
FOREIGN KEY (ida) REFERENCES Artiste(ida) ;

ALTER TABLE JoueDans ADD CONSTRAINT fk_jouedans_film
FOREIGN KEY (idf) REFERENCES Film(idf) ;
```

Scénario

Pour JoueDans

```
INSERT INTO JoueDans VALUES (1 , 3 , 'George Valentin' , 0.3),
(1 , 4 , 'Hubert Bonisseur de La Bath' , 0.2 ),
(2 , 2 , 'Lucy Miller' , 14),
(3 , 1 , 'Bill Cage' , 20),
(4 , 4 , 'Larmina El Akmar Betouche' , NULL),
(4 , 3 , 'Peppy Miller' , NULL),
(5 , 5 , 'Antoine Bailleul' , 1.3),
(6 , 1 , 'Rita Vrataski' , NULL);
```

Scénario

Certains besoins d'utilisateurs peuvent être de la forme :

- Créer la table Acteur à partir de la table Artistes. La table Acteur sera peuplé des tuples d'artistes qui ne concernent que les acteurs ayant déjà eu un rôle dans un film
- Ajouter les contraintes d'intégrité permettant de substituer Acteur à Artiste.
- Créer la table Realisateur à partir de la table Artistes. La table Realisateur sera peuplé des tuples d'artistes qui ne concernent que les acteurs ayant déjà réalisé un film
- Fusionner les tables Acteur et Realisateur en la table MesArtistes en vous assurant de ne pas insérer deux fois les acteur-réalisateur.

Scénario

- Créer la table Acteur à partir de la table Artistes. La table Acteur sera peuplés des tuples d'artistes qui ne concernent que les acteurs ayant déjà eu un rôle dans un film

```
CREATE TABLE Acteur AS
       (SELECT DISTINCT A.*
        FROM Artiste A JOIN JoueDans JD ON A.ida = JD.ida
       );
```

Syntaxe alternative pour le CREATE TABLE

Création d'une table à partir d'une requête

CREATE TABLE `nomTable` AS (SELECT $att_1, att_2, att_3, \ldots$ FROM...) ;

- Le schéma de la table créée correspond aux attributs $(att_1, att_2, att_3, \ldots)$ retournés par la requête.
- Les contraintes d'intégrité ne sont pas 'transmises' à la nouvelle table.

Scénario

- Ajouter les contraintes d'intégrité permettant de substituer Acteur à Artiste.

Ajout d'une clé primaire à Acteur

```
ALTER TABLE Acteur ADD CONSTRAINT pk_acteur
PRIMARY KEY (ida) ;
```

Suppression de la clé étrangère de JoueDans vers Artiste

```
ALTER TABLE JoueDans DROP CONSTRAINT
fk_jouedans_artiste ;
```

Scénario

- Ajouter les contraintes d'intégrité permettant de substituer Acteur à Artiste.

Ajout d'une clé étrangère de JoueDans vers Acteur

```
ALTER TABLE JoueDans ADD CONSTRAINT
fk_jouedans_acteur FOREIGN KEY (ida) REFERENCES
acteur(ida);
```

Scénario

- Créer la table Realisateur à partir de la table Artistes. La table Realisateur sera peuplé des tuples d'artistes qui ne concernent que les acteurs ayant déjà réalisé un film

```
CREATE TABLE Realisateur AS
        (SELECT DISTINCT A.*
         FROM Artiste A JOIN Film JD ON A.ida = F.real
        );
```

Scénario

Ajout d'une clé primaire à Réalisateur

```
ALTER TABLE Realisateur ADD CONSTRAINT pk_acteur
PRIMARY KEY (ida);
```

Suppression de la clé étrangère de Film vers Artiste

```
ALTER TABLE Film DROP CONSTRAINT fk_film_artiste;
```

Ajout d'une clé étrangère de Film vers Realisateur

```
ALTER TABLE Film ADD CONSTRAINT fk_film_real FOREIGN
KEY (real) REFERENCES realisateur(ida);
```

Scénario

- Fusionner les tables Acteur et Realisateur en la table MesArtistes en vous assurant de ne pas insérer deux fois les acteur-réalisateur.

```
CREATE TABLE MesArtistes AS
        (SELECT * FROM Acteur) ;
```

```
ALTER TABLE MesArtistes ADD CONSTRAINT pk_mesartistes
PRIMARY KEY (ida) ;
```

```
INSERT IGNORE INTO MesArtistes SELECT * FROM
Realisateur ;
```

Syntaxe alternative pour le INSERT INTO

Insertion à partir d'une requête

INSERT INTO `nomTable` SELECT ... FROM... ;

Fusion de deux tables avec suppression des doublons

INSERT IGNORE INTO `nomTable` SELECT ... FROM ... ;

Avec IGNORE, la commande d'insertion s'exécutera complètement même si des violations de contraintes de clé primaires sont soulevées.