

CRUD – ou comment standardiser l'accès aux données

Dans le chapitre sur les procédures stockées, nous avons vu que celles-ci sont utiles pour des **opérations récurrentes** sur votre base de données.

Prenons l'exemple de la base de données sakila.

Voici quelques opérations que vous auriez envie de pouvoir effectuer en tant qu'utilisateur de l'application du vidéo-club :

- Enregistrer un nouveau client ou modifier un client existant
- Ajouter un nouveau film
- Ajouter ou retirer des exemplaires du film de votre inventaire
- Renseigner un nouvel acteur ou une nouvelle actrice
- Enregistrer une location ou un retour
- Radier un client
- Ajouter ou supprimer un point de vente

Toutes ces opérations peuvent être effectuées en incluant vos requêtes SQL dans le code de votre application avec les bons paramètres.

Alors pourquoi s'embêter avec des procédures stockées ?

Pour deux raisons principales :

- Le code est réutilisable. Les développeurs ont moins de code à produire, donc à débbuger. Qui plus est, on peut tester une procédure stockée unitairement. Si on inclut une requête dans le code, le scénario de test est beaucoup plus compliqué.
- La procédure stockée « cache » le code qu'elle contient à l'application. C'est une sorte de boîte noire. Si on sait quels paramètres il faut fournir en entrée et qu'on sait ce qu'elle retourne, peu importe le code à l'intérieur. C'est d'autant plus appréciable qu'il est théoriquement possible que la structure de la base de données soit modifiée, que le code de la procédure soit modifié, mais qu'on ne touche pas à l'application si les entrées et sorties de la procédure ne changent pas.

Pour que l'accès aux données soit standardisé dans la base, on crée des procédures stockées pour les opérations CRUD (**C**reate, **R**ead, **U**ppdate, **D**eleter).

Cela ne veut pas dire que l'on crée systématiquement 4 procédures pour les 4 opérations, parfois certains besoins spécifiques peuvent dicter d'en avoir plus que 4.

Par souci de simplicité, nous allons baser nos exemples sur la table film de la base sakila et en particulier.

Tout d'abord, il est nécessaire de savoir recenser le(s) champ(s) clé(s) étrangère(s) sur la table film et le(s) champ(s) clé(s) étrangère(s) d'autres tables référençant la clé primaire de la table film.

Pour cela, le plus simple est de faire un clic droit sur la table -> Table Inspector -> onglet Foreign Keys.

MySQL Workbench

Local instance 3306

File Edit View Query Database Server Tools Scripting Help

SCHEMAS

Filter objects

ASSURAUTO

sakila

Tables

actor

address

category

city

country

customer

film

Columns

Indexes

Foreign Keys

Triggers

film_actor

film_category

film_text

inventory

language

Object Info

Session

Table: film

Columns:

film_id smallint(5) UN AI PK

title varchar(128)

description text

release_year year(4)

language_id tinyint(3) UN

original_language_id tinyint(3) UN

rental_duration tinyint(3) UN

rental_rate decimal(4,2)

length smallint(5) UN

replacement_cost decimal(5,2)

rating enum('G','PG','PG-13','R','NC-17')

special_features set('Trailers','Commentaries')

last_update timestamp

SQL Editor Opened.

jeu. 09:50

MySQL Workbench

SQL File 17* SQL File 9* PL_Film_by_year_nonreg PL_Film_nonreg PSGet_Film_nonreg PI_Film_nonreg PU_Film_nonreg PD_Film_nonreg* sakila.film

Info Columns Indexes Triggers Foreign keys Partitions Grants

Name	Schema	Table	Column	Referenced Schema	Referenced Table	Referenced Column
fk_film_language	sakila	film	language_id	sakila	language	language_id
fk_film_language_original	sakila	film	original_language_id	sakila	language	language_id
fk_film_actor_film	sakila	film_actor	film_id	sakila	film	film_id
fk_film_category_film	sakila	film_category	film_id	sakila	film	film_id
fk_inventory_film	sakila	inventory	film_id	sakila	film	film_id

Count: 5

Refresh

Action Output

#	Time	Action	Message	Duration / Fetch
---	------	--------	---------	------------------

Les deux premières lignes signifient que d'une part on a 2 clés étrangères dans film:

- le champ language_id de la table film est une clé étrangère qui fait référence au champ language_id de la table language.
- le champ original_language_id de la table film est une clé étrangère qui fait référence au champ language_id de la table language.

Et d'autre part on a 3 clés étrangères sur d'autres tables qui pointent vers film:

- le champ film_id de la table film_actor est une clé étrangère qui fait référence au champ film_id de la table film.
- le champ film_id de la table film_category est une clé étrangère qui fait référence au champ film_id de la table film.
- le champ film_id de la table inventory est une clé étrangère qui fait référence au champ film_id de la table film.

1- CREATE

Pour la procédure d'insertion, il faut tout d'abord faire l'inventaire des champs de la table et de leurs types. Il faut aussi *recenser les champs dont les valeurs sont générées automatiquement*.

En l'occurrence, on a ceci pour la clé primaire :

```
CREATE TABLE film (  
  film_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
  title VARCHAR(128) NOT NULL,
```

On peut spécifier (ou pas) la valeur de film_id à l'insertion.

Il faut aussi recenser *les clés étrangères*.

Il y en a 2 sur cette table :

```
CONSTRAINT fk_film_language FOREIGN KEY (language_id) REFERENCES language (language_id) ON DELETE RESTRICT ON UPDATE CASCADE,  
CONSTRAINT fk_film_language_original FOREIGN KEY (original_language_id) REFERENCES language (language_id) ON DELETE RESTRICT ON UPDATE CASCADE
```

Lorsqu'on a une clé primaire qui se génère automatiquement et qu'on ne spécifie pas de valeur, il peut être pratique de faire en sorte que la procédure retourne la valeur clé.

```
USE sakila;
```

```
DROP PROCEDURE IF EXISTS `PI_Film`;
```

```
DELIMITER ||
```

```
CREATE PROCEDURE `PI_Film` (  
    IN    filmid smallint(5),  
          title varchar(128),  
          description text,  
          releaseyear year(4),  
          languageid tinyint(3),  
          originallanguageid tinyint(3),  
          rentalduration tinyint(3),  
          rentalrate decimal(4,2),  
          length smallint(5),  
          replacementcost decimal(5,2),  
          rating enum('G', 'PG', 'PG-13', 'R', 'NC-17'),  
          specialfeatures set('Trailers', 'Commentaries', 'Deleted Scenes', 'Behind the Scenes'),  
          lastupdate timestamp)
```

```
BEGIN
```

```
-- vérifier que le l'ID du film que l'on souhaite insérer n'est pas déjà pris  
-- inutile si on le laisse générer tout seul  
--IF EXISTS (SELECT * FROM film WHERE film_id = filmid) THEN  
-- SIGNAL SQLSTATE '50004' SET MESSAGE_TEXT = 'film_id already existing';  
--END IF;
```

```
-- vérifier l'existence des valeurs des paramètres destinés à des clés étrangères  
IF NOT EXISTS (SELECT * FROM language WHERE language_id = languageid) THEN  
    SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Language not found';  
END IF;
```

```
IF NOT EXISTS (SELECT * FROM language WHERE language_id = originallanguageid) THEN  
    SIGNAL SQLSTATE '50003' SET MESSAGE_TEXT = 'Original language not found';  
END IF;
```

```
-- insertion  
INSERT INTO film (film_id,title,description,release_year,language_id,original_language_id,rental_duration,  
    rental_rate,length,replacement_cost,rating,special_features,last_update)  
VALUES(filmid,title,description,releaseyear,languageid,originallanguageid,rentalduration,  
    rentalrate,length,replacement_cost,rating,specialfeatures,NOW());
```

```
END ||
```

```
DELIMITER ;
```

2- READ

Dans ce cas-là, il convient de distinguer les différentes opérations de lecture :

- Celles renvoyant une ligne (get)
- Celle renvoyant toutes les lignes (find)
- Celles renvoyant toutes les lignes correspondant aux critères spécifiés (find by)

Get

Il s'agit de la **recherche par clé primaire**. Elle peut renvoyer 0 ou 1 ligne.

```
USE sakila;
```

```
DROP PROCEDURE IF EXISTS `PSGet_Film`;
```

```
DELIMITER ||
```

```
CREATE PROCEDURE `PSGet_Film` (IN filmid smallint(5))
```

```
BEGIN
```

```
    SELECT      film_id, title, description, release_year, language_id, original_language_id, rental_duration,  
                rental_rate, length, replacement_cost, rating, special_features, last_update  
    FROM film  
    WHERE film_id = filmid;
```

```
END ||
```

```
DELIMITER ;
```

Find

Il s'agit de la procédure la plus simple. Elle renvoie la totalité des lignes de la table. Elle n'est utilisée que pour les tables dont on peut prévoir qu'elles stockeront peu d'enregistrements.

```
USE sakila;
```

```
DROP PROCEDURE IF EXISTS `PL_Film`;
```

```
DELIMITER ||
```

```
CREATE PROCEDURE `PL_Film` ()
```

```
BEGIN
```

```
    SELECT      film_id, title, description, release_year, language_id, original_language_id, rental_duration,  
                rental_rate, length, replacement_cost, rating, special_features, last_update  
    FROM film;
```

```
END ||
```

```
DELIMITER ;
```


Find by search criteria

Cette procédure renvoie toutes les lignes qui correspondent au(x) critère(s) spécifié(s) et n'est développée que pour répondre à un besoin de recherche spécifique.

```
USE sakila;
```

```
DROP PROCEDURE IF EXISTS `PL_Film_by_year`;
```

```
DELIMITER ||
```

```
CREATE PROCEDURE `PL_Film_by_year` (IN releaseyear year(4))
```

```
BEGIN
```

```
    SELECT      film_id, title, description, release_year, language_id, original_language_id, rental_duration,  
                rental_rate, length, replacement_cost, rating, special_features, last_update  
    FROM        film  
    WHERE       release_year = releaseyear;
```

```
END ||
```

```
DELIMITER ;
```

3- UPDATE

Les opérations de mise à jour ne se font que sur un enregistrement, identifié par sa clé primaire.

Tout comme pour l'insertion, il faut recenser les clés éventuelles étrangères.

A noter qu'**on ne modifie jamais la clé primaire**.

La procédure aura autant de paramètres que les champs de la table, sauf pour les valeurs générées automatiquement.

```

USE sakila;

DROP PROCEDURE IF EXISTS `PU_Film`;

DELIMITER ||

CREATE PROCEDURE `PU_Film` (
    IN filmid smallint(5),
    description text,
    releaseyear year(4),
    title varchar(128),
    languageid tinyint(3),
    originallanguageid tinyint(3),
    rentalduration tinyint(3),
    rentalrate decimal(4,2),
    length smallint(5),
    replacementcost decimal(5,2),
    rating enum('G', 'PG', 'PG-13', 'R', 'NC-17'),
    specialfeatures set('Trailers', 'Commentaries', 'Deleted Scenes', 'Behind the Scenes'))
BEGIN
    #-- vérifier que le film que l'on souhaite mettre à jour existe bien
    IF NOT EXISTS (SELECT * FROM film WHERE film_id = filmid) THEN
        SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Film not found';
    END IF;

    #-- vérifier l'existence des valeurs des paramètres destinés à des clés étrangères
    IF NOT EXISTS (SELECT * FROM language WHERE language_id = languageid) THEN
        SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Language not found';
    END IF;

    IF NOT EXISTS (SELECT * FROM language WHERE language_id = originallanguageid) THEN
        SIGNAL SQLSTATE '50003' SET MESSAGE_TEXT = 'Original language not found';
    END IF;

    #-- mise à jour
    UPDATE film
    SET title=title, description=description, release_year=releaseyear, language_id= languageid,
        original_language_id=originallanguageid, rental_duration=rentalduration, rental_rate=rentalrate,
        length=length, replacement_cost=replacement_cost, rating=rating, special_features=specialfeatures, last_update=NOW()
    WHERE film_id = filmid ;

END ||

DELIMITER ;

```

4- DELETE

Les enregistrements sont supprimés un par un, identifiés par leur clé primaire.

Cette fois-ci, il faut déterminer 2 choses :

- S'agit-il d'une suppression physique ou logique ?
- La clé primaire est-elle une clé étrangère d'une autre table, et si oui, quel est le comportement à prévoir ?

Suppression physique

Si la ligne peut être supprimée sans qu'il soit nécessaire de l'archiver on peut faire un delete sur la table, en vérifiant que la ligne que l'on supprime n'est pas référencée ailleurs.

```
USE sakila;
```

```
DROP PROCEDURE IF EXISTS `PD_Film`;
```

```
DELIMITER ||
```

```
CREATE PROCEDURE `PD_Film` (IN filmid smallint(5))
```

```
BEGIN
```

```
    #--verification des clés étrangères pointant sur film_id
    IF EXISTS (SELECT * FROM film_actor WHERE film_id = filmid) THEN
        SIGNAL SQLSTATE '50005' SET MESSAGE_TEXT = 'film_id still present in film_actor: delete all entries or allow delete
on cascade and remove this check';
    END IF;
    IF EXISTS (SELECT * FROM film_category WHERE film_id = filmid) THEN
        SIGNAL SQLSTATE '50006' SET MESSAGE_TEXT = 'film_id still present in film_category: delete all entries or allow
delete on cascade and remove this check';
    END IF;
    IF EXISTS (SELECT * FROM inventory WHERE film_id = filmid) THEN
        SIGNAL SQLSTATE '50007' SET MESSAGE_TEXT = 'film_id still present in inventory: delete all entries or allow delete
on cascade and remove this check';
    END IF;

    #--suppression
    DELETE
    FROM film
    WHERE film_id = filmid;
```

```
END ||
```

```
DELIMITER ;
```

Suppression logique

Parfois, pour conserver une valeur historique et conserver une référence, on souhaite “désactiver l’accès” à une ligne sans la supprimer.

Lorsque ce cas de figure est identifié, il est prévu un champ de type booléen dans la table qui va déterminer si la ligne est valide ou pas.

Prenons un exemple à titre d’illustration. Supposons que la table film ait un champ IsReferenced de type tinyint(1) et avec comme valeur par défaut 1 pour que tout film inséré soit référencé par défaut.

Au lieu de supprimer la ligne par un delete, il suffira de faire une update pour passer la valeur de 1 à 0.

Le code suivant ne fonctionnera pas sur la base sakila, c’est simplement pour illustrer le mécanisme.

```
USE sakila;
```

```
DROP PROCEDURE IF EXISTS `PD_Film_logique`;
```

```
DELIMITER ||
```

```
CREATE PROCEDURE `PD_Film_logique` (IN filmid smallint(5))  
BEGIN
```

```
    #-- mise à jour  
    UPDATE film  
    SET   IsReferenced=0, last_update=NOW()  
    Where film_id = filmid ;
```

```
END ||
```

```
DELIMITER ;
```

Tests unitaires

Les choses principales dont il faut se souvenir sont :

- le scénario de test unitaire doit pouvoir être exécuté par une personne qui n'est pas le développeur.
- il ne doit se reposer sur aucun jeu de données préexistantes car il pourrait y avoir une différence entre les données de l'environnement de développement et l'environnement de test, tout comme il y aura des différences entre l'environnement de test et la production!
- idéalement, après le test, on souhaite laisser les données dans le même état qu'elles étaient avant le test. Pour faire cela, on prévoit d'exécuter le test à l'intérieur d'une transaction qui sera ROLLBACK.
- les tests sont **unitaires**, ce qui signifie qu'on ne teste qu'une seule fonctionnalité à la fois.

CREATE

Le scénario est simple : vous souhaitez effectuer une requête SELECT dont vous savez qu'elle retourne 0 lignes, effectuer l'insertion, et répéter la même requête SELECT qui vous renverra cette fois-ci 1 ligne.

Pour cela on souhaite, pour la ligne insérée :

- être sûrs que les champs clé étrangère ont une correspondance dans leur(s) table(s) respective(s).
- être sûrs que la ligne que l'on souhaite insérer en guise de test n'est pas déjà présente
- être sûrs de pouvoir récupérer la valeur clé primaire de la ligne insérée

```
use sakila;
```

```
START TRANSACTION;
```

```
--il y a 2 clés étrangères pointant vers la table language : language_id et original_language_id
```

```
INSERT INTO language (name)
```

```
VALUES ('Slovakistanais');
```

```
--on récupère la clé primaire correspondante à la ligne que l'on vient d'insérer
```

```
SET @lid = (SELECT language_id from language where name = 'Slovakistanais') ;
```

```
-- on s'assure qu'un film ayant pour titre 'scary movie 2021' n'existe pas déjà
```

```
SET @fid = (SELECT film_id from film where title='scary movie 2021');
```

```
SELECT * FROM film where film_id = @fid ; --on soite voir 0 lignes retournées
```

```
--on choisit la 1ere valeur dispo pour film_id
```

```
SET @fid = (SELECT max(film_id) from film) +1;
```

```
--on insère une ligne de test en utilisant la procédure stockée
```

```
CALL PI_Film (@fid,'scary movie 2021', 'LE film du futur qui fait peur', 2021, @lid, @lid, 1,1, 121,5, 'R', 'Trailers', NOW());
```

```
-- on s'assure que la ligne a bien été insérée existe bien
```

```
SELECT * FROM film where film_id = @fid ;
```

```
ROLLBACK; -- on annule toutes les actions précédentes
```


READ

Get

Concernant les clés étrangères et la clé primaire on suit la même logique.

On insère une ligne témoin pour s'assurer que la table ne sera pas vide. On affiche la dernière ligne insérée.

On insère la ligne que l'on souhaite récupérer. On affiche la dernière ligne insérée et on vérifie que l'affichage a bien changé et est valide.

```
use sakila;
```

```
START TRANSACTION;
```

```
--il y a 2 clés étrangères pointant vers la table language : language_id et original_language_id
```

```
INSERT INTO language (name)
```

```
values ('Slovakistanaïs');
```

```
--on récupère la clé primaire correspondante à la ligne que l'on vient d'insérer
```

```
SET @lid = (SELECT language_id from language where name = 'Slovakistanaïs') ;
```

```
--on insère un premier film au cas où la table serait vide
```

```
INSERT INTO film (`title`,`description`,`release_year`,`language_id`,`original_language_id`,  
`rental_duration`,`rental_rate`,`length`,`replacement_cost`,`rating`,`special_features`)
```

```
values ('Wakanda forever','un film de science-fiction sur le cosplay animal',2020,@lid,@lid,1,1,120,10,'G','Trailers') ;
```

```
--on récupère l'id max de tous les films dans la table, en l'occurrence celui qui a été inséré
```

```
SET @fid = (SELECT max(film_id) from film);
```

```
CALL PSGet_Film(@fid) ; --renvoie les infos du film dont l'ID est spécifié
```

```
-- on insère la ligne que l'on souhaite afficher
```

```
INSERT INTO film (`title`,`description`,`release_year`,`language_id`,`original_language_id`,  
`rental_duration`,`rental_rate`,`length`,`replacement_cost`,`rating`,`special_features`)
```

```
values ('scary movie','un film qui fait peur',2020,@lid,@lid,1,1,120,10,'G','Trailers') ;
```

```
--on récupère l'id max de tous les films dans la table, en l'occurrence celui qui a été inséré en 2e
```

```
SET @fid = (SELECT max(film_id) from film);
```

```
CALL PSGet_Film(@fid) ; --renvoie les infos du film dont l'ID est spécifié
```

```
ROLLBACK; -- on annule toutes les actions précédentes
```

Find

C'est le scenario de test le plus simple.

Il faut s'assurer que la procédure renvoie la totalité des lignes de la table quel qu'en soit le nombre.

```
use sakila;
```

```
START TRANSACTION;
```

```
--il y a 2 clés étrangères pointant vers la table language : language_id et original_language_id
```

```
INSERT INTO language (name)
```

```
values ('Slovakistanais');
```

```
--on récupère la clé primaire correspondante à la ligne que l'on vient d'insérer
```

```
SET @lid = (SELECT language_id from language where name = 'Slovakistanais') ;
```

```
CALL PL_Film(); --on affiche la totalité des lignes de la table, peut renvoyer 0 lignes
```

```
SELECT count(*) FROM film; -- on souhaite compter le nombre de lignes avant (surtout s'il y en a beaucoup)
```

```
--on insère 2 films pour être sûrs que la procédure puisse renvoyer plus d'une ligne
```

```
INSERT INTO film (`title`, `description`, `release_year`, `language_id`, `original_language_id`,  
`rental_duration`, `rental_rate`, `length`, `replacement_cost`, `rating`, `special_features`)
```

```
values ('Wakanda forever', 'un film de science-fiction sur le cosplay animal', 2020, @lid, @lid, 1, 1, 120, 10, 'G', 'Trailers')  
;
```

```
INSERT INTO film (`title`, `description`, `release_year`, `language_id`, `original_language_id`,  
`rental_duration`, `rental_rate`, `length`, `replacement_cost`, `rating`, `special_features`)
```

```
values ('scary movie', 'un film qui fait peur', 2020, @lid, @lid, 1, 1, 120, 10, 'G', 'Trailers') ;
```

```
CALL PL_Film(); --on affiche la totalité des lignes de la table, doit renvoyer au moins 2 lignes
```

```
SELECT count(*) FROM film; -- on souhaite compter le nombre de lignes après (surtout s'il y en a beaucoup)
```

```
ROLLBACK; -- on annule toutes les actions précédentes
```

Find by search criteria

Il s'agit de la même logique que le cas précédent : il faut s'assurer que la procédure renvoie la totalité des lignes de la table **correspondant au critère souhaité** quel qu'en soit le nombre.

En l'occurrence on se focalise sur release_year.

```
use sakila;
```

```
START TRANSACTION;
```

```
--il y a 2 clés étrangères pointant vers la table language : language_id et original_language_id
```

```
INSERT INTO language (name)
```

```
VALUES ('Slovakistanais');
```

```
--on récupère la clé primaire correspondante à la ligne que l'on vient d'insérer
```

```
SET @lid = (SELECT language_id from language where name = 'Slovakistanais') ;
```

```
CALL PL_Film_by_year(2020) ; --on affiche la totalité des lignes de la table ayant release_year = 2020, peut renvoyer 0 lignes
```

```
-- on souhaite compter le nombre de lignes avant (surtout s'il y en a beaucoup)
```

```
SELECT count(*) FROM film where release_year = 2020;
```

```
--on insère 2 films pour être sûrs que la procédure puisse renvoyer plus d'une ligne
```

```
INSERT INTO film (`title`, `description`, `release_year`, `language_id`, `original_language_id`,  
`rental_duration`, `rental_rate`, `length`, `replacement_cost`, `rating`, `special_features`)
```

```
VALUES ('Wakanda forever', 'un film de science-fiction sur le cosplay animal', 2020, @lid, @lid, 1, 1, 120, 10, 'G',  
'Trailers');
```

```
INSERT INTO film (`title`, `description`, `release_year`, `language_id`, `original_language_id`,  
`rental_duration`, `rental_rate`, `length`, `replacement_cost`, `rating`, `special_features`)
```

```
VALUES ('scary movie', 'un film qui fait peur', 2020, @lid, @lid, 1, 1, 120, 10, 'G', 'Trailers') ;
```

```
--on affiche la totalité des lignes de la table ayant release_year = 2020, doit renvoyer au moins 2 lignes
```

```
CALL PL_Film_by_year(2020) ;
```

```
-- on souhaite compter le nombre de lignes après (surtout s'il y en a beaucoup)
```

```
SELECT count(*) FROM film WHERE release_year = 2020;
```

```
ROLLBACK; -- on annule toutes les actions précédentes
```

UPDATE

On souhaite insérer une nouvelle ligne, l'afficher, la modifier, l'afficher après modification et constater les différences.

```
use sakila;
```

```
START TRANSACTION;
```

```
--il y a 2 clés étrangères pointant vers la table language : language_id et original_language_id
```

```
INSERT INTO language (name)
```

```
VALUES ('Slovakistanais');
```

```
--on récupère la clé primaire correspondante à la ligne que l'on vient d'insérer
```

```
SET @lid = (SELECT language_id from language where name = 'Slovakistanais');
```

```
-- on insère une ligne témoin
```

```
INSERT INTO film (`title`, `description`, `release_year`, `language_id`, `original_language_id`,  
`rental_duration`, `rental_rate`, `length`, `replacement_cost`, `rating`, `special_features`)
```

```
VALUES ('scary movie', 'un film qui fait peur', 2020, @lid, @lid, 1, 1, 120, 10, 'G', 'Trailers') ;
```

```
--on récupère l'id max de tous les films dans la table, en l'occurrence celui qui a été inséré
```

```
SET @fid = (SELECT max(film_id) from film);
```

```
SELECT * FROM film where film_id = @fid ; --on affiche la ligne avant modification
```

```
--appel à la procédure d'update avec de nouvelles valeurs
```

```
CALL PU_Film (@fid, 'SCARY MOVIE 2021', 'LE film du futur qui fait peur', 2021, @lid, @lid, 1, 1, 121, 5, 'R', 'Trailers');
```

```
SELECT * FROM film where film_id = @fid ; --on affiche la ligne après modification
```

```
ROLLBACK; -- on annule toutes les actions précédentes
```

DELETE

Suppression physique

On souhaite insérer une ligne, l'afficher, la supprimer et constater qu'on ne peut plus l'afficher.

```
use sakila;
```

```
START TRANSACTION;
```

```
--il y a 2 clés étrangères pointant vers la table language : language_id et original_language_id
```

```
INSERT INTO language (name)
```

```
VALUES ('Slovakistanais');
```

```
--on récupère la clé primaire correspondante à la ligne que l'on vient d'insérer
```

```
SET @lid = (SELECT language_id from language where name = 'Slovakistanais') ;
```

```
-- on insère une ligne témoin
```

```
INSERT INTO film (`title`, `description`, `release_year`, `language_id`, `original_language_id`,  
`rental_duration`, `rental_rate`, `length`, `replacement_cost`, `rating`, `special_features`)
```

```
VALUES ('scary movie', 'un film qui fait peur', 2020, @lid, @lid, 1, 1, 120, 10, 'G', 'Trailers') ;
```

```
--on récupère l'id max de tous les films dans la table, en l'occurrence celui qui a été inséré
```

```
SET @fid = (SELECT max(film_id) from film);
```

```
SELECT * FROM film where film_id = @fid ; --on affiche la ligne insérée
```

```
CALL PD_Film (@fid); --on supprime la ligne
```

```
SELECT * FROM film where film_id = @fid ; --on essaye d'afficher la ligne et on doit constater la suppression
```

```
ROLLBACK;
```

Suppression logique

Pour une suppression logique le test est semblable à celui de l'update, puisqu'il s'agit d'une opération d'update sur un champ booleen.