# Transaction :

## Principe :

Une transaction, c'est un ensemble de requêtes qui sont exécutées en un seul bloc. Ainsi, si une des requêtes du bloc échoue, on peut décider d'annuler tout le bloc de requêtes (ou de quand même valider les requêtes qui ont réussi).

### Problématique :z

// permet de quitter le mode auto commit

SET autocommit=0;

Tant que l’on est en plus en autocommit, chaque modification de donnée devra être commitée pour prendre effet. Tant que vos modifications ne sont pas validées, vous pouvez à tout moment les annuler (faire un rollback).

Pour cela il faut :

COMMIT ; -- valide les requêtes

ROLLBACK ; -- annule les requêtes

### Solution en mode autocommit :

START TRANSACTION;

*// Traitement non gardé car rollback*

ROLLBACK ;

START TRANSACTION;

*// Traitement gardé car commit*

COMMIT ;

### Jalon de transaction :

Il s’agit d’un point de repère qui permet d’annuler toutes les requêtes exécutées depuis ce jalon seulement et non toutes les requêtes de la transaction.

START TRANSACTION;

// traitement 1

SAVEPOINT jalon1 ; (jalon1 = alias)

// Traitement 2

ROLLBACK TO SAVEPOINT jalon1 ;

// Traitement 3

COMMIT ;

Dans cette exemple, traitement 2 ne sera pas effectué

Cependant certaines requêtes ne peuvent pas être « annulées » via la commande ROLLBACK, il s’agit des commandes qui influent sur une structure de données, comme par exemple :

CREATE DATABASE, DROP DATABASE, CREATE TABLE, ALTER TABLE, RENAME TABLE, DROP TABLE, CREATE INDEX ou encore DROP INDEX.

# Trigger :

## Principe :

Les triggers (ou déclencheurs) sont des objets de la base de données. Attachés à une table, ils vont déclencher l'exécution d'une instruction, ou d'un bloc d'instructions, lorsqu'une ou plusieurs lignes sont insérées, supprimées ou modifiées dans la table à laquelle ils sont attachés.

Un trigger peut être déclenché par :

* INSERT
* DELETE
* UPDATE

Un trigger se déclenche une fois par instruction, c’est-à-dire que si l’on fait 5 INSERT, on aura 5 TRIGGER.

## Intérêt des triggers :

Intégrité des données : réalisé une suppression logique : on conserve les lignes « supprimées » en base de données, on ajoutera soit une date de fin, soit un booléen pour dire si la ligne est active ou non.

Historique des actions : qui a modifié et à quelle heure tel ou tel objet

## Syntaxe :

Voir schéma

Moment\_trigger : AFTER ou BEFORE

(Attention, on ne peut définir qu’un seul trigger par combinaison de moment\_trigger et evenement\_trigger par table !)

## Autre mot-clef :

OLD et NEW, ils représentent respectivement les valeurs des colonnes de ligne traitée par le trigger.

OLD correspond à l’ancienne valeur, c’est-à-dire avant que la ligne ne soit traitée par le trigger.

NEW correspond à la nouvelle valeur, soit la valeur modifiée après le trigger.

(Lors d’un INSERT on aura qu’un NEW, lors d’un DELETE on aura qu’un OLD et on aura les deux lors d’un UPDATE)

## Supprimer un trigger :

**DROP TRIGGER nom\_trigger;**

**DELIMITER :** nécessaire afin de permettre à SQL de lui indiquer d’interpréter toutes les lignes suivantes comme une seule commande, jusqu’à ce qu’il atteigne le délimiter personnalisé. (utilisé pour les commandes)

## Gestion des erreurs :

Table error avec id et message, et faire un insert dans le THEN

(NB : On ne peut pas effectuer de TRANSACTION dans un TRIGGER)

# Bonus

## Déclaration de variables dans les requêtes :

SET @priceMax = 49 ;

## Réaliser des requêtes précharger :

PREPARE select\_game\_on\_genre

FROM 'SELECT \* FROM games JOIN games\_genres ON games\_genres.games\_id = games.id JOIN genres ON genres.id = games\_genres.genres\_id WHERE genres.name = ?';

## Exécuter la requête :

EXECUTE select\_game\_on\_genre USING @genre

Function :

DELIMITER //

CREATE FUNCTION can\_buy\_game ( accountId INT, gameId INT )

RETURNS TINYINT

BEGIN

SET @return\_value = 0;

SET @gamePrice = (SELECT games.price FROM games WHERE games.id = gameId);

SET @accountWallet = (SELECT accounts.wallet FROM accounts WHERE accounts.id = accountId);

IF @accountWallet > @gamePrice THEN

SET @return\_value = 1;

END IF;

RETURN @return\_value;

END //

DELIMITER ;

PREPARE select\_model\_from\_brand

FROM 'SELECT \* FROM model JOIN brand ON model.brand\_id = brand.id WHERE brand.name = ?';

EXECUTE select\_model\_from\_brand USING 'Opel';

EXECUTE select\_model\_from\_brand USING 'Ferrari';

DELIMITER //

DROP PROCEDURE IF EXISTS getCA //

CREATE PROCEDURE

getCA( idBrand INT )

BEGIN

SELECT SUM(listing.price)

FROM listing

JOIN model ON model.id = listing.model\_id

JOIN brand ON model.brand\_id = brand.id

WHERE brand.id = idBrand;

END //

DELIMITER ;

[call](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/call.html) getBuyableGames ( 1 )