# SQL Approfondissement

# Plan de l'intervention

- ✓ Retours sur la conception
- ✓ Les transactions
- ✓ L’ agrégation
- ✓ Les fonctions
- ✓ Automatisation

# Conception avancée

# Clé primaire

```
CREATE TABLE [nom_de_la_table] (
        id INT PRIMARY KEY AUTO_INCREMENT,
        field type NULL,
        ...
);
```

**OU**

```
CREATE TABLE [nom_de_la_table] (
        id INT AUTO_INCREMENT,
        field type NULL,
        ...
        PRIMARY KEY (id)
);
```

# Clé primaire

- Garantit l'unicité des enregistrements en table
- Peut être composée
- **Sans signification fonctionnelle**

# Clé étrangère

```
CREATE TABLE [nom_de_la_table] (
        id INT PRIMARY KEY AUTO_INCREMENT,
        parent_id INT NOT NULL,
        FOREIGN KEY (parent_id)
        REFERENCES parent(id)
);

ALTER TABLE [nom_de_la_table]
        ADD CONSTRAINT FK_child_parent
        FOREIGN KEY child (parent_id)
        REFERENCES parent (id);
```

# Clé étrangère

**Définition des actions en chaîne :**

**… ON UPDATE [*value*] ON DELETE [*value*]**

- CASCADE : Supprime les enfants si le parent est supprimé
- SET NULL : Rend les enfants orphelins si le parent est supprimé
- SET DEFAULT [value] : Donne un parent par défaut si le parent est supprimé
- RESTRICT (mode par défaut)

# Index

```
CREATE TABLE [nom_de_la_table] (
        id INT PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(200) NOT NULL,
        INDEX (name)
);


CREATE INDEX [nom_index]
ON [nom_de_la_table] (colonne1 , colonne2, ...);
```

# Index

- Permet de changer l'ordre d'interrogation des éléments de la table
- Performant sur les tables volumineuses
- Ralentit l'écriture sur les tables

# Commandes simples

## Créations d'une table

**CREATE TABLE** [*nomTable*] **(**

[*nomChamp*] [*type*] [NULL] [*options*],

PRIMARY KEY ([nomChamp]) **);**

**NULL** : champ facultatif

**NOT NULL** : champ obligatoire

*Options* : AUTO_INCREMENT, DEFAULT,UNIQUE, CHECK

# Les transactions

# Utilisation avancée

## Les procédures stockées (*approche*)

- **Exemple de procédure stockée :**

```
DELIMITER //
CREATE PROCEDURE nomProcedure()
BEGIN
    ...Requête SQL
END//
```

# Les transactions

*« Une transaction est une unité de travail. Lorsqu'une transaction aboutit, toutes les modifications de données apportées lors de la transaction sont validées et intégrées de façon permanente à la base de données. Si une transaction rencontre des erreurs et doit être annulée ou restaurée, toutes les modifications de données sont supprimées. »*

Les transactions sont un moyen d'exécuter une série de requête SQL dans un environnement **bac à sable** ou toutes les modifications peuvent être soit validées, soit annulées.

# Règles de transaction

**Une transaction** est caractérisée par le mot l'acronyme **ACID (Atomic Consistency Isolation Durability)** :

- **Atomique** car la transaction constitue une unité indivisible de travail pour le serveur.
- **Consistance** car à la fin d'une transaction, les données montrées sont soit celles d'avant transaction (dans le cas d'une annulation de la transaction) soit celle d'après transaction (dans le cas d'une validation).
- **Isolation**, car il est possible de verrouiller (isoler) les données pendant l'exécution de la transaction (verrouillage en lecture, en écriture, …).
- **Durée** car les changements apportés sur des données par une transaction sont durables (non volatiles).

# Transaction

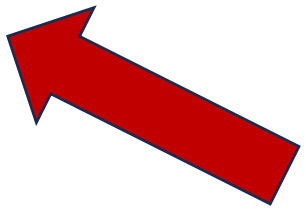
- **Exemple de transaction:**

```
START TRANSACTION;
SELECT @A:=SUM(amount) FROM table;
UPDATE table2 SET total = @A WHERE id = 1;
COMMIT;
```

# Rollback

- **Exemple de transaction:**

```
START TRANSACTION;
SELECT @A:=AVG(amount) FROM table;
DELETE FROM table WHERE amount < @A;
ROLLBACK;
```



Annule toute la transaction

# Limites du Rollback

Attention il y a des limites au Rollback :

- Certaines fonctions ont des autocommits implicites : https://dev.mysql.com/doc/refman/8.0/en/implicit-commit.html
- CREATE, DROP DATABASE
- CREATE, DROP, ALTER TABLE

# Savepoint

SAVEPOINT *identifier;*

ROLLBACK TO *identifier;*

*ou*

RELEASE SAVEPOINT *identifier;*

N’interromps pas la transaction.

# Agrégations avancés

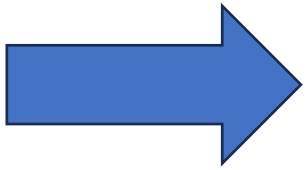
# ROLLUP

```
SELECT
    productLine,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline WITH ROLLUP;
```

# GROUPING

La fonction GROUPING() retourne le résultat 1 quand on est sur une ligne de ROLLUP

```
SELECT
    orderYear,
    productLine,
    SUM(orderValue)
totalOrderValue,
    GROUPING(orderYear),
    GROUPING(productLine)
FROM
    sales
GROUP BY
    orderYear,
    productline
WITH ROLLUP;
```



| orderYear | productLine | totalOrderValue | GROUPING(orderYear) | GROUPING(productLine) |
|---|---|---|---|---|
| 2003 | Classic Cars | 5571.80 | 0 | 0 |
| 2003 | Motorcycles | 2440.50 | 0 | 0 |
| 2003 | Planes | 4825.44 | 0 | 0 |
| 2003 | Ships | 5072.71 | 0 | 0 |
| 2003 | Trains | 2770.95 | 0 | 0 |
| 2003 | Trucks and Buses | 3284.28 | 0 | 0 |
| 2003 | Vintage Cars | 4080.00 | 0 | 0 |
| 2003 | NULL | 28045.68 | 0 | 1 |
| 2004 | Classic Cars | 8124.98 | 0 | 0 |
| 2004 | Motorcycles | 2598.77 | 0 | 0 |
| 2004 | Planes | 2857.35 | 0 | 0 |
| 2004 | Ships | 4301.15 | 0 | 0 |
| 2004 | Trains | 4646.88 | 0 | 0 |
| 2004 | Trucks and Buses | 4615.64 | 0 | 0 |
| 2004 | Vintage Cars | 2819.28 | 0 | 0 |
| 2004 | NULL | 29964.05 | 0 | 1 |
| 2005 | Classic Cars | 5971.35 | 0 | 0 |
| 2005 | Motorcycles | 4004.88 | 0 | 0 |
| 2005 | Planes | 4018.00 | 0 | 0 |
| 2005 | Ships | 3774.00 | 0 | 0 |
| 2005 | Trains | 1603.20 | 0 | 0 |
| 2005 | Trucks and Buses | 6295.03 | 0 | 0 |
| 2005 | Vintage Cars | 5346.50 | 0 | 0 |
| 2005 | NULL | 31012.96 | 0 | 1 |
| NULL | NULL | 89022.69 | 1 | 1 |

# Algorithmie

# Les variables

```
SET @x = 8;

SELECT @x as result;
```

# Conditions

```
CASE case_value
        WHEN when_value THEN statement_list
        [WHEN when_value THEN statement_list] ...
                                [ELSE statement_list]
END CASE
```

# Conditions

```
IF search_condition
        THEN statement_list
        [ELSEIF search_condition THEN
statement_list] ...      [ELSE statement_list]
END IF
```

# Boucles

```
BEGIN
        label1: LOOP
                SET p1 = p1 + 1;
                IF p1 < 10 THEN
                        ITERATE label1;
                END IF;
                LEAVE label1;
        END LOOP label1;
        SET @x = p1;
END
```

# Boucles

```
BEGIN
        SET @x = 0;
        REPEAT
                SET @x = @x + 1;
        UNTIL @x > p1 END REPEAT;
END
```

# Curseurs

```
BEGIN
 DECLARE done INT DEFAULT FALSE;
 DECLARE a CHAR(16);
 DECLARE b, c INT;
 DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
 DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

 OPEN cur1;
 OPEN cur2;
 read_loop: LOOP
                FETCH cur1 INTO a, b;
                FETCH cur2 INTO c;
                IF done THEN
                                LEAVE read_loop;
                END IF;
                IF b < c THEN
                                INSERT INTO test.t3 VALUES (a,b);
                ELSE
                                INSERT INTO test.t3 VALUES (a,c);
                END IF;
 END LOOP;
 CLOSE cur1;
 CLOSE cur2;
END
```

# SQL


Plus d'informations sur http://www.dawan.fr
Contactez notre service commercial au **0800.10.10.97**(prix d'un appel local)