

# Base de données : SQL – Partie 1

---

Cours et exercices

# SQL : Un standard

- Rétrocompatible
- ANSI/ISO
  - SQL-86 – intersection of IBM SQL implementations
  - SQL-89 – small revision, integrity constraints
  - SQL-92 – schema modification, transactions, set operators, new data types, cursors, referential integrity actions, ...
  - SQL:1999 – recursive queries, triggers, object-relational features, regular expressions, types for full-text, images, spatial data, ...
  - SQL:2003 – SQL/XML, sequence generators
  - SQL:2006 – other extensions of XML, integration of XQuery
  - SQL:2008
  - SQL:2011 – temporal databases

# Catégories des instructions SQL

- **Définition** des éléments d'une base de données
  - *Data Definition Language, soit DDL*
- **Manipulation** des données
  - *Data Manipulation Language, soit DML*
- **Gestion des droits d'accès** aux données
  - *Data Control Language, soit DCL*
- **Gestion des transactions**
  - *Transaction Control Language, soit TCL*
- **SQL intégré**
  - *Embedded SQL*

# SQL : Langage de définition de données

- **CREATE**
- **ALTER**
- **DROP**
- Et d'autres instructions :
  - AUDIT, NOAUDIT, ANALYZE, RENAME, TRUNCATE

# SQL : Langage de manipulation de données

- **INSERT**
- **UPDATE**
- **DELETE**
- **SELECT**
- Et d'autres instructions :
  - EXPLAIN, PLAN, LOCK TABLE

# Langage de définition de données

---

CREATE, ALTER,  
DROP

# CREATE : Création d'une BDD

<https://dev.mysql.com/doc/refman/8.0/en/charset-database.html>

```
CREATE DATABASE [IF NOT EXISTS] database_name  
[CHARACTER SET charset_name]  
[COLLATE collation_name]
```

# Charset et Collation

<http://www.dynamic-mess.com/sql/comprendre-charset-et-collation/>

- **Jeux de caractères (Character set)** : Un ensemble de symboles et d'encodage
- **Interclassement (Collation)** : Un ensemble de règles comparant les caractères dans un jeu de caractères
- Un jeu peut avoir plusieurs interclassemements, un par langue généralement.
  - Cela permet par ex de classer par ordre alphabétique les caractères => le SGBD saura interpréter des requêtes de tri, mais également savoir si un caractère est équivalent à un autre, par exemple le 'a' et le 'à'.

Tapez la commande SQL suivante pour connaître tous les jeux de caractères de Mysql avec leur interclassement par défaut:  
**SHOW CHARACTER SET;**



# CREATE : Suffixes de l'interclassement

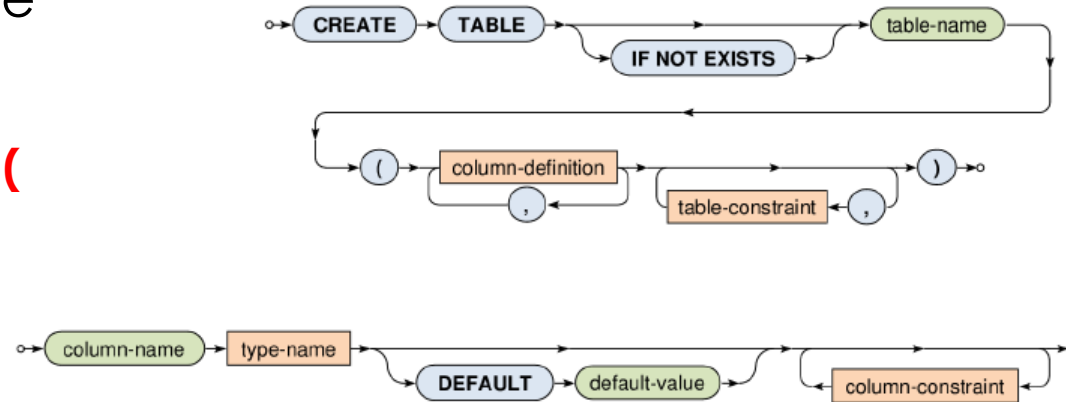
|      |                    |
|------|--------------------|
| _ai  | Accent insensitive |
| _as  | Accent sensitive   |
| _ci  | Case insensitive   |
| _cs  | Case sensitive     |
| _ks  | Kana sensitive     |
| _bin | Binary             |

- latin1\_general\_ci est explicitement case insensitive et implicitement accent insensitive
- latin1\_general\_cs est explicitement case sensitive et implicitement accent sensitive
- utf8mb4\_0900\_ai\_ci est explicitement case and accent insensitive.

# CREATE : Création simple d'une table et des colonnes associées

- Construction d'une table
  - Nom de la table
  - Définition de chaque colonne
    - Nom de la colonne
    - Type de données de la colonne
    - Valeur par défaut

**CREATE** TABLE nom\_de\_la\_table (  
colonne1 type\_donnees,  
colonne2 type\_donnees,  
colonne3 type\_donnees,  
colonne4 type\_donnees )



# CREATE : Types de données

- **INTEGER** : Ce type permet de stocker des entiers signés codés sur 4 octets
- **VARCHAR(longueur)** : Ce type de données permet de stocker des chaînes de caractères de longueur variable. longueur doit être inférieur à 2000, il n'y a pas de valeur par défaut
- **DATE** : Ce type de données permet de stocker des données constituées d'une date
- **TIMESTAMP** : Ce type de données permet de stocker des données constituées d'une date et d'une heure
- **BOOLEAN** : Ce type de données permet de stocker des valeurs Booléenne
- **CHAR(longueur)** : Ce type de données permet de stocker des chaînes de caractères de longueur fixe. longueur doit être inférieur à 255, sa valeur par défaut est 1

# CREATE : Types de données (suite)

- BIGINT : Ce type permet de stocker des entiers signés codés sur 8 octets.
- REAL : Ce type permet de stocker des réels comportant 6 chiffres significatifs codés sur 4 octets.
- DOUBLE PRÉCISION : Ce type permet de stocker des réels comportant 15 chiffres significatifs codés sur 8 octets.
- NUMERIC[(précision, [longueur])] : Ce type de données permet de stocker des données numériques à la fois entières et réelles avec une précision de 1000 chiffres significatifs.
  - longueur précise le nombre maximum de chiffres significatifs stockés et précision donne le nombre maximum de chiffres après la virgule.
- MONEY : Ce type de données permet de stocker des valeurs monétaires.
- TEXT : Ce type de données permet de stocker des chaînes de caractères de longueur variable.

# CREATE: exemple sans contraintes

**Product**(*id, name, price, produced, available, weight*)

```
CREATE TABLE Product (  
    id INTEGER,  
    name VARCHAR(128) ,  
    price DECIMAL(6,2) ,  
    produced DATE ,  
    available BOOLEAN DEFAULT TRUE ,  
    weight FLOAT  
);
```

# CREATE : Création avec contraintes d'intégrité

- Construction d'une table
  - Nom de la table
  - Définition de chaque colonne
    - Nom de la colonne
    - Type de données de la colonne
    - Valeur par défaut
    - **Contraintes d'intégrité des colonnes**
  - **Contraintes d'intégrité de la table**

# CREATE : Contraintes d'intégrité de **colonne**

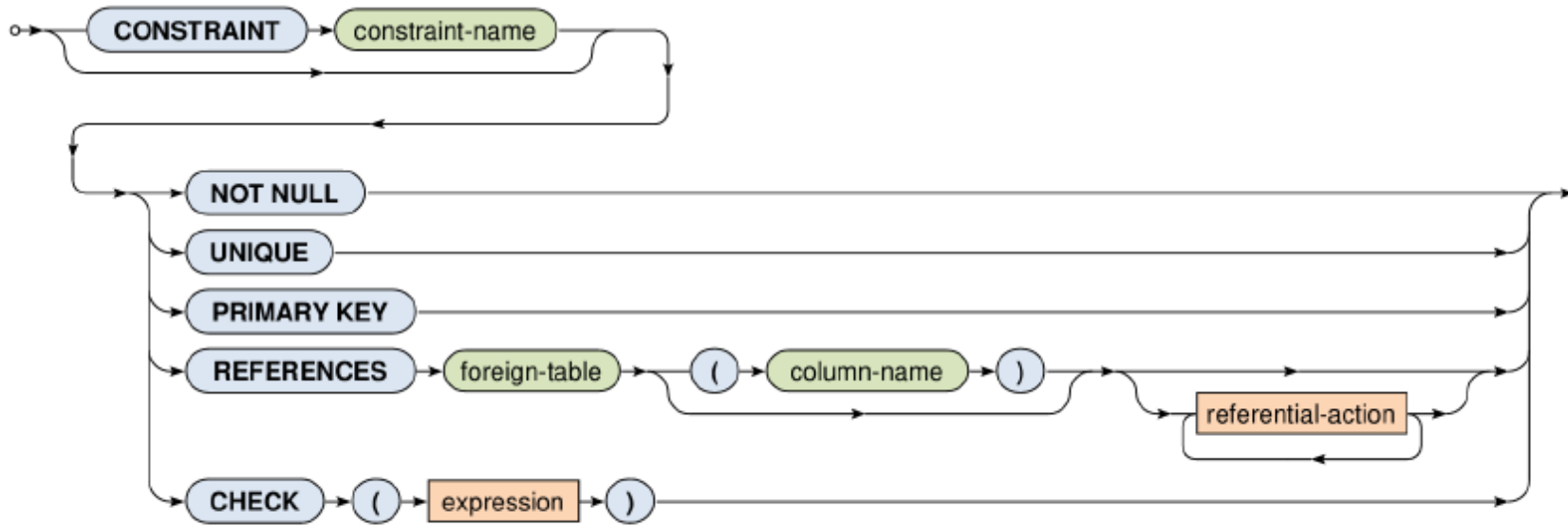
|                |   |
|----------------|---|
| NOT NULL       | Empêche d'enregistrer une valeur nulle pour une colonne   |
| NULL           | Autorise d'enregistrer une valeur nulle pour une colonne  |
| UNIQUE         | Désigne l'attribut comme clé secondaire de la table. Deux n-uplets ne peuvent recevoir des valeurs identiques pour cette colonne, mais l'insertion de valeur NULL est toutefois autorisée |
| DEFAULT valeur | Attribuer une valeur par défaut si aucune données n'est indiquée pour cette colonne lors de l'ajout d'une ligne dans la table   |
| PRIMARY KEY    | Désigne l'attribut comme clé primaire de la table. Equivalente à la contrainte UNIQUE NOT NULL  |

# CREATE : Contraintes d'intégrité de **colonne** (suite)

|   |  |
|---|--|
| <i>FOREIGN KEY [colonne]</i><br><b>REFERENCES</b> table<br><b>[(colonne)]</b><br><i>[ON DELETE CASCADE]</i> | Contrainte d'intégrité référentielle pour l'attribut de la table en cours de définition. Les valeurs prises par cet attribut doivent exister dans la colonne de la table étrangère 'table', cette colonne possède une contrainte PRIMARY KEY ou UNIQUE<br>En l'absence de précision d'attribut colonne, l'attribut retenu est celui correspondant à la clé primaire de la table 'table'. |
| <b>CHECK (condition)</b>  | Vérifie lors de l'insertion de n-uplets que l'attribut réalise la condition condition.   |



# CREATE : Contraintes d'intégrité de **colonne** (résumé)



# CREATE : Contraintes d'intégrité de **colonne** (exemples)



**Person**(id, personalNumber, address, age, serialNumber, color)

```
CREATE TABLE Person(  
    id INTEGER PRIMARY KEY,  
    personalNumber INTEGER,  
    adress VARCHAR(256),  
    age INTEGER,  
    serialNumber INTEGER NOT NULL,  
    color STRING  
);
```

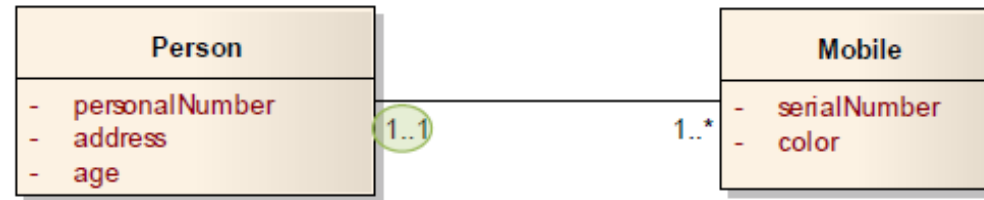
# CREATE : Contraintes d'intégrité de **colonne** (exemples)

```
CREATE TABLE Producer (  
    id INTEGER PRIMARY KEY,  
    name VARCHAR(128),  
    country VARCHAR(64)  
);
```

Nomme une contrainte, **pas obligatoire** dans le cas des contraintes de **colonne**

```
CREATE TABLE Product (  
    id INTEGER CONSTRAINT IC_Product_PK PRIMARY KEY,  
    name VARCHAR(128) UNIQUE,  
    price DECIMAL(6,2) CONSTRAINT IC_Product_Price NOT NULL,  
    produced DATE CHECK (produced >= '2015-01-01'),  
    available BOOLEAN DEFAULT TRUE NOT NULL,  
    weight FLOAT,  
    producer INTEGER,  
    FOREIGN KEY(producer) REFERENCES Producer(id)  
);
```

# CREATE : Contraintes d'intégrité de **colonne** (exemples)



```
CREATE TABLE Person(  
    personalNumber INTEGER PRIMARY KEY,  
    adress VARCHAR(256),  
    age INTEGER,  
);
```

```
CREATE TABLE Mobile(  
    serialNumber INTEGER PRIMARY KEY,  
    color STRING,  
    personalNumber INTEGER,  
    FOREIGN KEY (personalNumber) REFERENCES Person (personalNumber)  
);
```

**Person**(personalNumber, address, age)  
**Mobile**(serialNumber, color, personalNumber)  
 $\text{Mobile.personalNumber} \subseteq \text{Person.personalNumber}$

# CREATE : Contraintes d'intégrité de **table**

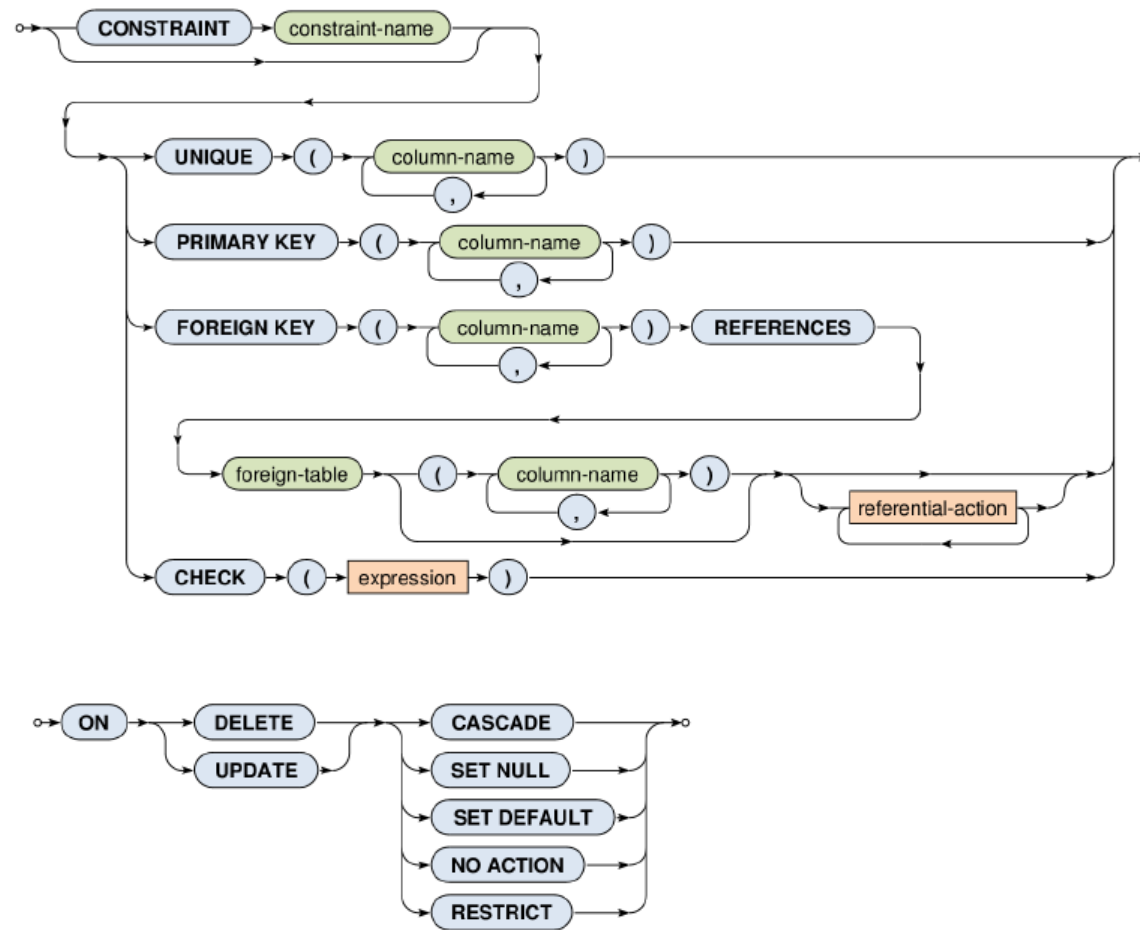
Analogue aux contraintes des colonnes mais pour les colonnes multiples

|                |   |
|----------------|---|
| NOT NULL       | Empêche d'enregistrer une valeur nulle pour une colonne   |
| NULL           | Autorise d'enregistrer une valeur nulle pour une colonne  |
| UNIQUE         | Désigne l'attribut comme clé secondaire de la table. Deux n-uplets ne peuvent recevoir des valeurs identiques pour cette colonne, mais l'insertion de valeur NULL est toutefois autorisée |
| DEFAULT valeur | Attribuer une valeur par défaut si aucune données n'est indiquée pour cette colonne lors de l'ajout d'une ligne dans la table   |
| PRIMARY KEY    | Désigne l'attribut comme clé primaire de la table. Equivalente à la contrainte UNIQUE NOT NULL  |

# CREATE : Contraintes d'intégrité de **table** (suite)

|  |  |
|--|--|
| <p>FOREIGN KEY (colonne...)<br/>REFERENCES table<br/>[(colonne)]*<br/><i>[ON DELETE CASCADE]</i></p> | <p>Contrainte d'intégrité référentielle pour l'attribut de la table en cours de définition. Les valeurs prises par cet attribut doivent exister dans l'attribut colonne qui possède une contrainte PRIMARY KEY ou UNIQUE dans la table 'table'. En l'absence de précision d'attribut colonne, l'attribut retenu est celui correspondant à la clé primaire de la table table spécifiée.</p> |
| <p>CHECK (condition)</p>   | <p>Vérifie lors de l'insertion de n-uplets que l'attribut réalise la condition condition.</p>  |

# CREATE : Contraintes d'intégrité de **table** (résumé)



# CREATE : Contraintes d'intégrité de **table** (**exemples**)

**Producer**(name, country)

**Product**(id, ..., producerName, producerCountry)

$\text{Product}(\text{producerName}, \text{producerCountry}) \subseteq \text{Producer}(\text{name}, \text{country})$

```
CREATE TABLE Producer (  
    name VARCHAR(128),  
    country VARCHAR(3),  
    CONSTRAINT IC_Producer_PK PRIMARY KEY (name,  
        country)  
);
```

```
CREATE TABLE Product (  
    id INTEGER PRIMARY KEY,  
    ...  
    producerName VARCHAR(128),  
    producerCountry VARCHAR(3),  
    CONSTRAINT IC_Product_Producer_FK FOREIGN KEY (producerName,   
        producerCountry)  
    REFERENCES Producer (name,  
        country)  
);
```



# CREATE : Contraintes d'intégrité de **table** (exemples)



**Person**(personalNumber, address, age)

**Mobile**(serialNumber, color)

**Ownership**(personalNumber, serialNumber)

Ownership.personalNumber  $\subseteq$  Person.personalNumber

Ownership.serialNumber  $\subseteq$  Mobile.serialNumber

```
CREATE TABLE Person(  
    personalNumber INTEGER PRIMARY KEY,  
    adress VARCHAR(256),  
    age INTEGER,  
);  
  
CREATE TABLE Mobile(  
    serialNumber INTEGER PRIMARY KEY,  
    color STRING,  
);
```

```
CREATE TABLE Ownership(  
    personalNumber INTEGER NOT NULL  
        REFERENCES Person(personalNumber),  
    serialNumber INTEGER NOT NULL  
        REFERENCES Mobile(serialNumber),  
    CONSTRAINT  
        PRIMARY  
    KEY(personalNumber, serialNumber)  
);
```

# CREATE: intégrité référentielle

- **l'intégrité référentielle** est une **contrainte que le concepteur de bases de données s'impose afin de garantir l'intégrité des données** stockées.
- Vérifier qu'une valeur de clé étrangère existe bien en tant que valeur de clé primaire dans une autre table.
- Exemple : Des factures sont reliées à un client. L'intégrité référentielle empêchera la suppression d'un client si des factures sont déjà stockées dans la base de données. Aussi, si un client est supprimé alors toutes ses factures sont supprimées également

# CREATE: intégrité référentielle (exemple)

```
CREATE TABLE Producer (  
    id INTEGER PRIMARY KEY,  
    name VARCHAR(128),  
    country VARCHAR(64)  
);
```

```
CREATE TABLE Product (  
    id INTEGER PRIMARY KEY,  
    ...  
    producer INTEGER  
    REFERENCES Producer (id) ON DELETE CASCADE  
);
```

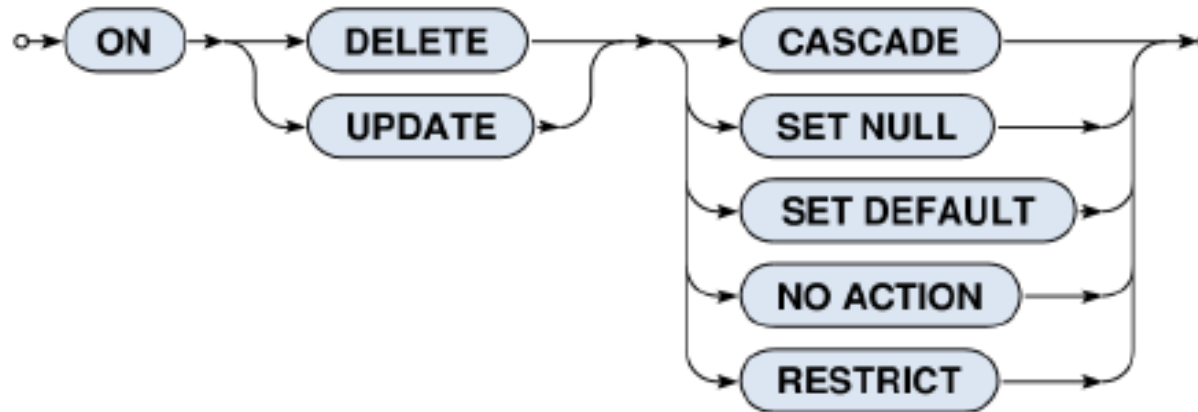
Si un Producteur est supprimé Alors toutes les lignes (tuples) de la table Product où ce producteur apparaît sont supprimées



# CREATE: intégrité référentielle

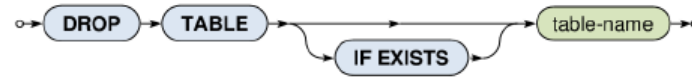
- Triggering situations
  - **ON UPDATE, ON DELETE**
    - When the action is triggered
    - Once again, these are considered to be operations on the referenced table
- Referential actions
  - **CASCADE**
    - Row with the referencing value is updated / deleted as well
  - **SET NULL** – referencing value is set to NULL
  - **SET DEFAULT** – referencing value is set to its default
  - **NO ACTION** – default – no action takes place
    - I.e. as if no referential action would be defined at all

# CREATE: intégrité référentielle (résumé)



# DROP

- Complementary to the table creation
  - I.e. table definition as well as table content are deleted



# ALTER

- Addition/change/removal of table columns/Integrity Constraint

