BDW - Conception des BD - modélisation physique

Fabien Duchateau

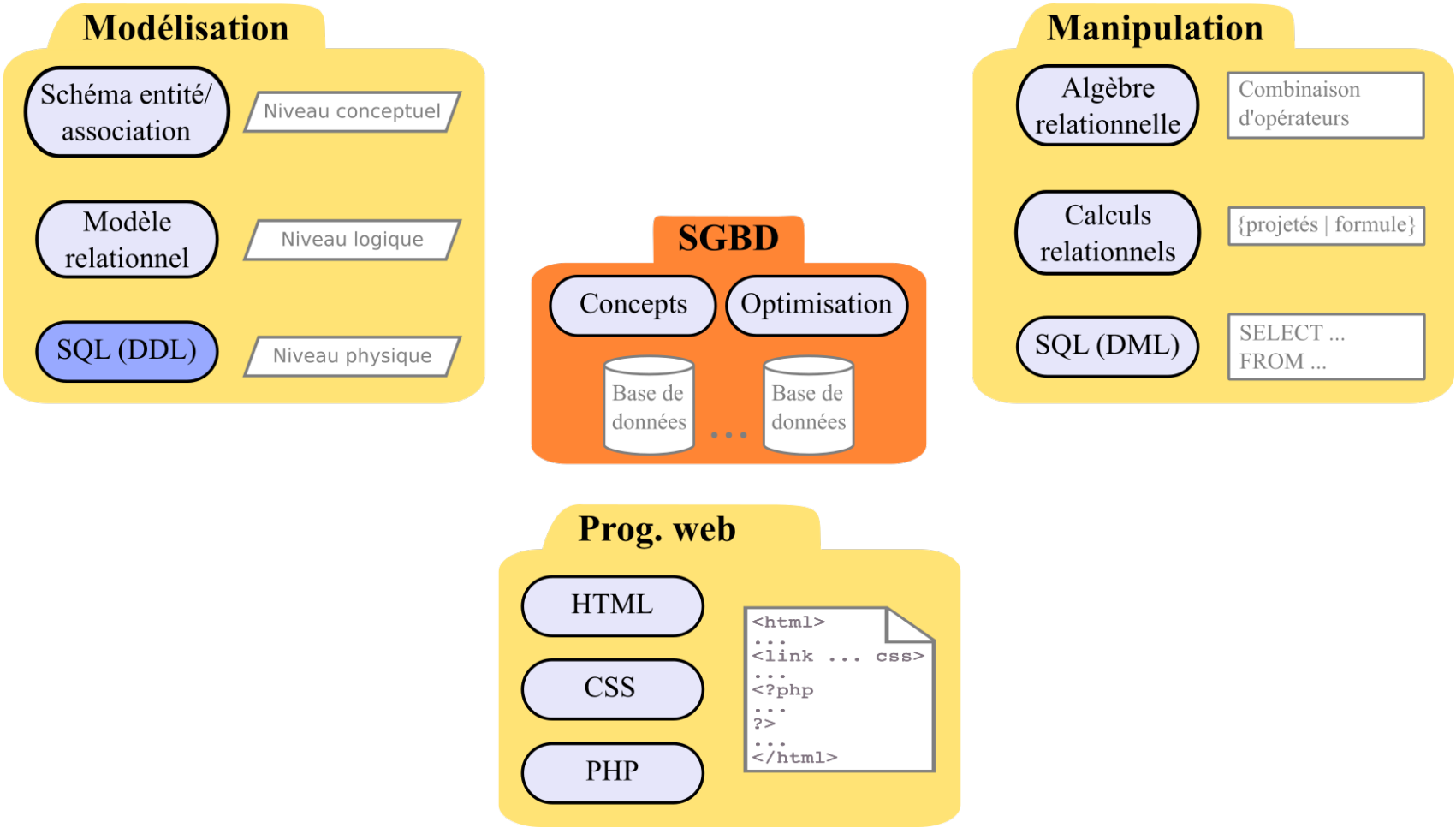
fabien.duchateau [at] univ-lyon1.fr

Université Claude Bernard Lyon 1

2023 - 2024

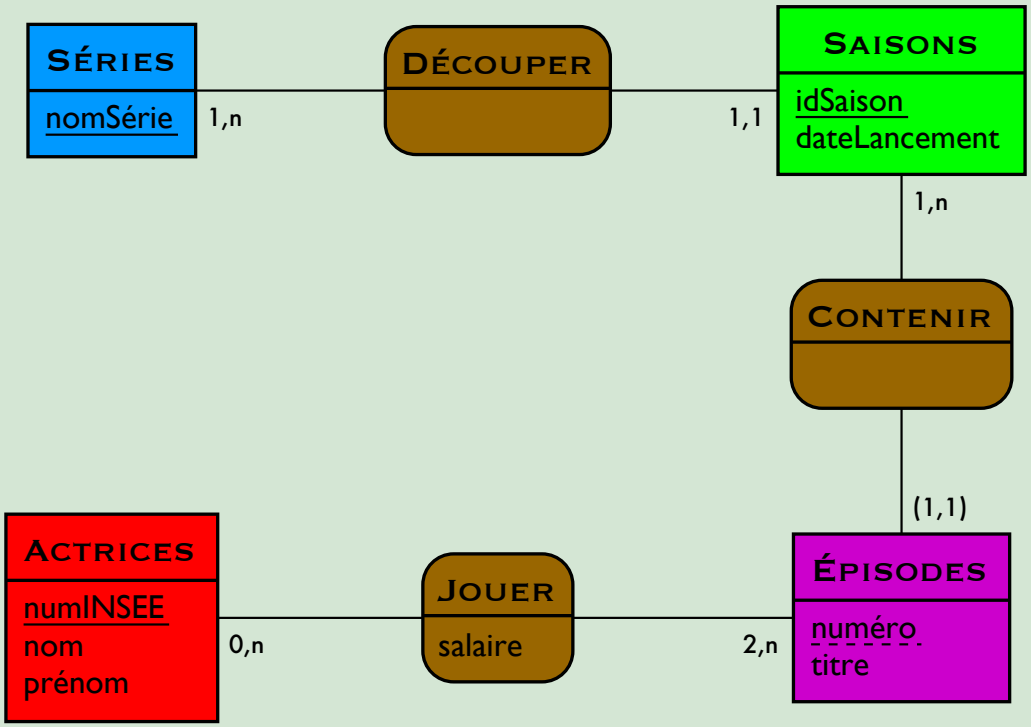
https://perso.liris.cnrs.fr/fabien.duchateau/BDW/

Positionnement dans BDW



Ces diapositives utilisent le genre féminin (e.g., chercheuse, développeuses) plutôt que l'écriture inclusive (moins accessible, moins concise, et pas totalement inclusive)

D'un diagramme E/A à un schéma relationnel



SÉRIES(nomSérie)
SAISONS(idSaison, dateLancement, #nomSérie)
ÉPISODES(numéro, #idSaison, titre)
ACTRICES(numINSEE, nom, prénom)
JOUER(#numéro, #idSaison, #numINSEE, salaire)

Motivation

Dernière étape de la modélisation (niveau physique) :

- Implique le choix d'un SGBD pour stocker la BD
- Besoin de traduire le modèle logique en modèle physique
- Dans notre cas, transformation du schéma relationnel en tables SQL pour le SGBD MariaDB

Le langage SQL permet la création (**C**RUD) à la fois de tables et contraintes (LDD) et d'instances (LMD)

Plan

Qu'est ce que SQL ?

SQL = Structured Query Language

Un langage concret pour interagir avec le modèle relationnel :

- Un langage de manipulation de données
- Un langage de description de données
- Un langage pour administrer la base, gérer les contrôles d'accès

Langage de manipulation de données **déclaratif** ⇒ description du résultat escompté

Cours et tutoriels SQL sur http://sql.sh

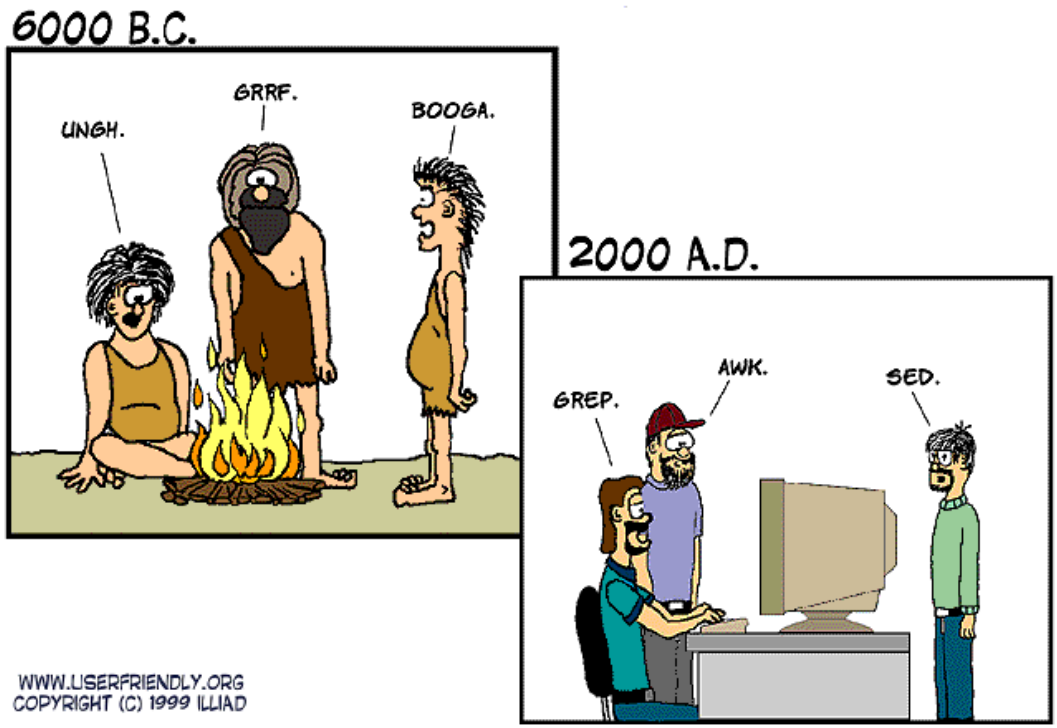
Pour aller plus loin, http://use-the-index-luke.com/ et http://modern-sql.com/

Historique de SQL

Origine : IBM en 1974

Standards :

- SQL-87 : 1987 (ISO)
- SQL-2 : 1992
- SQL-3 : 1999
- ...
- SQL-2016



http://fr.wikipedia.org/wiki/SQL

SQL dans les SGBD

De nombreux SGBD utilisent le langage SQL :

- MySQL / MariaDB
- PostgreSQL
- Oracle
- SQL Server
- LibreOffice Base, Access
- ...

Chaque SGBD implémente plus ou moins respectueusement le standard SQL (3 niveaux de conformité)

---

https://tdan.com/is-sql-a-real-standard-anymore/4923

Plan

Généralités

Dans un SGBD, les tables sont stockées dans une base de données, qu'il faut créer :

- Avec une requête SQL ou un client graphique (e.g., PHPMyAdmin)
- En option :
  - spécifier le jeu de caractères utilisé pour la BD (e.g., *latin1*, *utf8*)
  - spécifier la collation, i.e., la méthode de comparaison et de tri pour un jeu de caractères (e.g., *utf8_general_ci*, *utf8_bin*)

---

http://mariadb.com/kb/en/mariadb/create-database/
http://mariadb.com/kb/en/mariadb/use/
http://mariadb.com/kb/en/mariadb/character-set-and-collation-overview/

Syntaxe de création de base de données

CREATE OR REPLACE DATABASE *nom_bd* **;**
USE *nom_bd* **;**

- Création d'une base de données nommée *nom_bd*
- L'option OR REPLACE permet de remplacer (détruire puis recréer) une base de données *nom_bd* existante
- La commande USE spécifie que l'on va utiliser la base *nom_bd* (e.g., pour y créer des tables)

Exemple de création de base de données

```
7 /* Creation d'une base de donnees Séries */
8 CREATE OR REPLACE DATABASE BD_Series;
```

Création ou remplacement d'une base de données Series. La commande USE *permet de sélectionner (travailler) avec la base mentionnée*

Plan

Généralités

Pour chaque relation du modèle relationnel, création de la table correspondante

- Avec une requête SQL ou un client graphique (e.g., PHPMyAdmin)
- Attention aux différences d'implémentation de SQL dans les SGBD (MariaDB pour cet enseignement)

À la création d'une table (schéma), on indique :

- Le nom des attributs
- Le type de chaque attribut
- De manière optionnelle :
  - certaines contraintes d'intégrité
  - des caractéristiques de stockage

Syntaxe de création de table

CREATE TABLE *nom_table* **(**
att_1 $type_1$
$[, att_2$ $type_2, \ldots]$
);

- Création simple d'une table de nom *nom_table*, avec un attribut att_1 de type $type_1$, un attribut att_2 de type $type_2$, etc.
- Les crochets $[\ldots]$ représentent des éléments optionnels

---

http://mariadb.com/kb/en/mariadb/create-table/
http://mariadb.com/kb/en/mariadb/data-types/

Types d'attributs

Type pour les **booléens** :

- `BOOLEAN` ou `TINYINT(1)`

Types pour les **chaînes de caractères** :

- `CHAR(longueur)`
  - taille fixe égale à *longueur* (complétion par des espaces), mais limitée à 255 caractères
- `VARCHAR(longueur)` et `TEXT`
  - taille **variable**, mais limitée à *65365* caractères
  - un `varchar` peut contenir jusqu'à *longueur* caractères
  - autres capacités avec `TINYTEXT`, `MEDIUMTEXT`, `LONGTEXT`

---

http://mariadb.com/kb/en/mariadb/boolean/
http://mariadb.com/kb/en/mariadb/string-data-types/

Types d'attributs (2)

Types **numériques** :

- `DECIMAL(longueur, échelle)`
  - un nombre codé exactement sur *longueur* chiffres, dont *échelle* chiffres après la virgule
  - *longueur* et *échelle* sont optionnels
- `DOUBLE(longueur, échelle)`
  - un nombre à virgule flottante (double précision)
  - *longueur* et *échelle* sont optionnels
- Raccourcis d'écriture :
  - `INT` ou `INTEGER`, `TINYINT`, `MEDIUMINT`, `BIGINT` (entier)
  - `FLOAT` (nombre à virgule flottante, mais à simple précision)

---

http://mariadb.com/kb/en/mariadb/data-types-numeric-data-types/

Types d'attributs (3)

Types pour les **dates** :

- `YEAR`, une année sur 4 chiffres
- `DATE`, une date au jour près, stockée au format *YYYY-MM-JJ*
  - d'autres formats sont acceptés (chaîne, *YY-MM-JJ*, etc.)
- `TIME`, un temps stocké au format *HH :MM :SS.ssssss*
  - d'autres formats sont acceptés (*HH :MM :SS*, *HHMMSS*, etc.)
- `DATETIME`, une date avec temps, au format *YYYY-MM-DD HH :MM :SS*
- `TIMESTAMP`, le nombre de secondes écoulées depuis le 1^{er} janvier 1970

---

http://mariadb.com/kb/en/mariadb/date-and-time-data-types/

Types d'attributs (4)

Types pour les **objets binaires** (e.g., images, fichiers) :

- `TINYBLOB`, un objet contenant jusqu'à 255 octets de données binaires
- `BLOB`, un objet contenant jusqu'à 64 Ko de données binaires
- `MEDIUMBLOB`, un objet jusqu'à 16 Mo
- `LONGBLOB`, un objet jusqu'à 4 Go

---

http://mariadb.com/kb/en/mariadb/blob/

Types d'attributs (5)

Types **énumérés** :

- `ENUM`('val_1', 'val_2', ...)
  - définition d'une liste de valeurs autorisées pour un attribut
  - en théorie 65535 valeurs maximales dans l'énumération

- `SET`('val_1', 'val_2', ...)
  - similaire à `ENUM`, mais limité à 64 valeurs

---

http://mariadb.com/kb/en/mariadb/enum/
http://mariadb.com/kb/en/mariadb/set-data-type/

Exemple de création de tables

SÉRIES(nomSérie)

```
18 CREATE TABLE Series(
19  nomSerie VARCHAR(255)
20 );
```

Field	Type	Null	Key	Default	Extra
nomSerie	varchar(255)	YES		*NULL*	

SAISONS(idSaison, dateLancement, #nomSérie)

```
22 CREATE TABLE Saisons(
23  idSaison INT,
24  dateLancement DATE,
25  nomSerie VARCHAR(255)
26 );
```

Field	Type	Null	Key	Default	Extra
idSaison	int(11)	YES		*NULL*	
dateLancement	date	YES		*NULL*	
nomSerie	varchar(255)	YES		*NULL*	

Exemple de création de tables (2)

ÉPISODES(<u>numéro, #idSaison</u>, titre)

```
28 CREATE TABLE Episodes(
29  numero INT,
30  idSaison INT,
31  titre VARCHAR(255)
32 );
```

Field	Type	Null	Key	Default	Extra
numero	int(11)	YES		*NULL*	
idSaison	int(11)	YES		*NULL*	
titre	varchar(255)	YES		*NULL*	

ACTRICES(<u>numINSEE</u>, nom, prénom)

```
34 CREATE TABLE Actrices(
35  numINSEE INT,
36  nom VARCHAR(255),
37  prenom VARCHAR(255)
38 );
```

Field	Type	Null	Key	Default	Extra
numINSEE	int(11)	YES		*NULL*	
nom	varchar(255)	YES		*NULL*	
prenom	varchar(255)	YES		*NULL*	

Exemple de création de tables (3)

JOUER(#numero, #idSaison, #numINSEE, salaire)

```
40 CREATE TABLE Jouer(
41  numero INT,
42  idSaison INT,
43  numINSEE INT,
44  salaire DOUBLE
45 );
```

Field	Type	Null	Key	Default	Extra
numero	int(11)	YES		*NULL*	
idSaison	int(11)	YES		*NULL*	
numINSEE	int(11)	YES		*NULL*	
salaire	double	YES		*NULL*	

Visualisation du schéma d'une table

- Avec un client graphique (e.g., PHPMyAdmin)
- En ligne de commande, avec l'instruction DESC :
  - noms d'attributs, types, certaines contraintes d'intégrité

DESC *nom_table* **;**

```
      Table « public.joue »
 Colonne  |  Type   | Modificateurs
----------+---------+---------------
 numero   | integer |
 idsaison | integer |
 numinsee | integer |
 salaire  | real    |
```

Affichage des informations de la table Joue dans un terminal

Moteurs de stockage

Un moteur de stockage peut être spécifié lors de la création d'une table. Un moteur de stockage regroupe l'ensemble des algorithmes permettant de stocker et d'accéder aux données.

Sous MariaDB, les moteurs disponibles sont :

- **InnoDB / XtraDB** = gestion des clés étrangères et des transactions, verrouillage sur n-uplet
- **MyISAM / Aria** = non transactionnel, pas de gestion des clés étrangères, verrouillage sur table
- **Memory** = en mémoire (données temporaires), performant
- ...

---

http://mariadb.com/kb/en/mariadb/storage-engines/

http://openclassrooms.com/courses/les-moteurs-de-stockage-de-mysql-2

Moteurs de stockage (2)

CREATE TABLE *nom_table* (
...
) **ENGINE** = moteur ;

- Syntaxe de création d'une table avec un moteur de type *moteur*
- *moteur* peut prendre les valeurs MyISAM, InnoDB, etc.
- Commande optionnelle, par défaut XtraDB est utilisé
- Voir la documentation pour les moteurs disponibles sur un SGBD donné et choisir le plus adapté

En résumé

- Création (du schéma) d'une table avec CREATE TABLE
- Définition de chaque attribut par son nom et son type
- Possibilité de choisir un moteur de stockage pour la table

Plan

Généralités

Le schéma de la table est créé, mais il ne contient aucune donnée !

Deux méthodes pour ajouter des instances en SQL :

- Insertion d'un seul n-uplet (valeur pour chaque attribut du n-uplet)
- Copie de données existantes

Ces deux méthodes sont aussi réalisables via un client graphique (e.g., PHPMyAdmin) ou un langage de programmation

Insertion d'un seul n-uplet

INSERT INTO *nom_table*(att_1, ..., att_n)
VALUES(val_1, ..., val_n) **;**

- Insertion d'un n-uplet dans la table *nom_table*
- Chaque attribut de att_1, ..., att_n aura la valeur val_1, ..., val_n correspondante (i.e., val_i pour l'attribut att_i)
- Si un attribut de la table *nom_table* n'apparaît pas dans att_1, ..., att_n, alors la valeur du n-uplet pour cet attribut est NULL

Insertion d'un seul n-uplet (2)

INSERT INTO *nom_table*
VALUES(val_1, ..., val_n) **;**

- Insertion simplifiée d'un n-uplet dans la table *nom_table* (sans spécifier les attributs)
- Obligation de donner une valeur à chaque attribut de *nom_table*
- L'ordre des valeurs val_1, ..., val_n est l'ordre des attributs dans la définition de la table *nom_table*

Exemple d'insertion d'un seul n-uplet

```
52 INSERT INTO Series VALUES('The Big Bang Theory');
53 INSERT INTO Series VALUES('Game of Thrones');
```

←⊤→	nomSerie
Modifier Copier Effacer	The Big Bang Theory
Modifier Copier Effacer	Game of Thones

```
54 INSERT INTO Saisons VALUES(1, '2011-09-22', 'The Big
       Bang Theory');
55 INSERT INTO Saisons VALUES(2, '2012-09-27', 'The Big
       Bang Theory');
56 INSERT INTO Saisons VALUES(3, '2011-04-17', 'Game of
       Thrones');
```

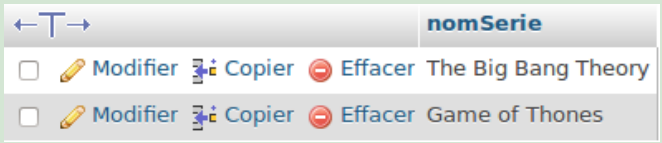
idSaison	dateLancement	nomSerie
1	2010-09-22	The Big Bang Theory
2	2011-09-27	The Big Bang Theory
3	2011-04-17	Game of Thones

Copie de données existantes

INSERT INTO *nom_table*$(att_1, \ldots, att_n)$
SELECT $e_1, \ldots, e_n$
FROM ... **;**

- Insertion de n-uplets dans la table *nom_table* à partir de données récupérées par la requête SELECT ... FROM
- La requête ne peut pas contenir de ORDER BY (le SGBD détermine l'ordre de stockage des n-uplets)
- Le nom des attributs dans le résultat de la requête n'est pas important : c'est l'ordre des expressions qui compte

Exemple de copie de données existantes



idSaison	dateLancement	nomSerie
1	2010-09-22	The Big Bang Theory
2	2011-09-27	The Big Bang Theory
3	2011-04-17	Game of Thones

```
81 INSERT INTO Series(nomSerie)
82 SELECT nomSerie
83 FROM Saisons
84 WHERE idSaison = 2;
```

- Quel résultat obtient-on ?
- Comment visualiser le résultat en SQL ?
- Pourquoi peut-on avoir deux instances *The Big Bang Theory* dans la table Series ?

nomSerie
The Big Bang Theory
Game of Thrones
The Big Bang Theory

En résumé

- Insertion d'un n-uplet (INSERT INTO ... VALUES ...)
  - une version compacte de cette requête permet d'insérer plusieurs n-uplets en même temps (cf "export" sous MariaDB)
- Insertion d'un n-uplet (INSERT INTO ... SELECT ...)

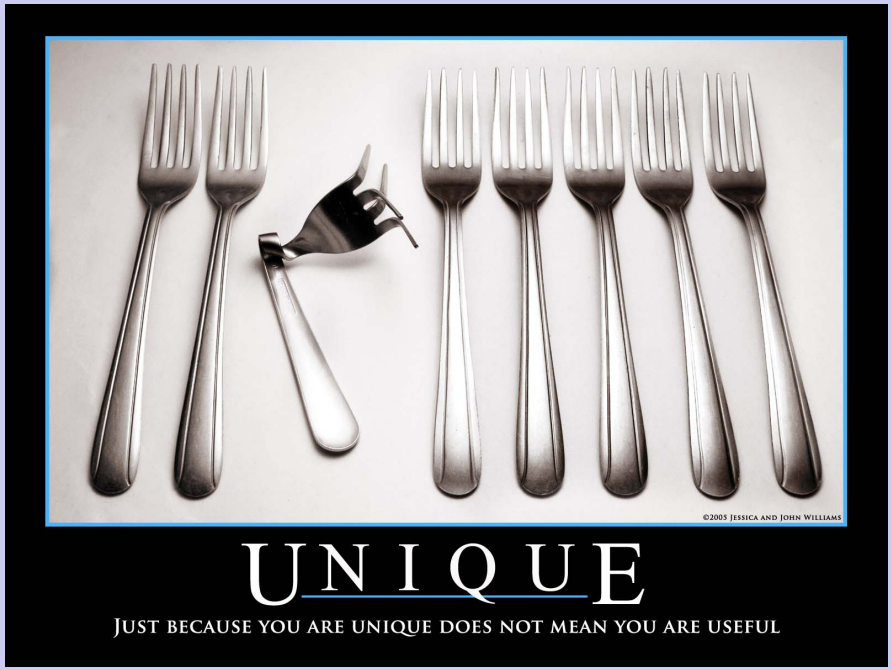
Démo avec MariaDB (script SQL en ligne)

Plan

Généralités

Dans l'exemple précédent, aucune contrainte sur les tables : il est donc possible de créer des épisodes identiques, d'avoir des épisodes sans actrice, d'insérer des salaires négatifs, etc.

Besoin de spécifier des contraintes d'intégrité !



Généralités (2)

Le SGBD vérifie les contraintes lors des insertions, suppressions ou mises à jour de n-uplets

Quand créer les contraintes ?

- Lors de la création de la table
- Lors d'une évolution ultérieure du schéma de la table

Quels types de contraintes ?

- Unicité
- Autorisation des valeurs nulles
- Clé primaire (PK, pour Primary Key)
- Clé étrangère (FK, pour Foreign Key)
- Valeurs autorisées (CHECK)

Syntaxe de création de contrainte

CREATE TABLE *nom_table* (
att_1 $type_1$,
[...]
CONSTRAINT [nom_1] *def_contrainte*$_1$
[...]
);

- Le mot-clé CONSTRAINT permet de spécifier une contrainte
- nom_1 est le nom de la contrainte (optionnel)
- *def_contrainte*$_1$ définit la contrainte (selon son type)
- Certaines contraintes ont une syntaxe "raccourcie" (contrainte indiquée après l'attribut concerné)

Contrainte UNIQUE

CONSTRAINT nom_c **UNIQUE**(att_i, att_j, ...)

- Avec la contrainte **unique**, chaque n-uplet doit avoir une combinaison de valeurs différente pour les attributs att_i, att_j, ...
- Il est par contre possible d'avoir deux fois la même valeur pour un attribut att_i
- Si une des valeurs pour att_i, att_j, ... est NULL, la contrainte ne s'applique pas sur le n-uplet concerné

Exemple de contrainte UNIQUE

```
103 CREATE TABLE TestUnique1(
104  att1 INTEGER,
105  att2 INTEGER,
106  CONSTRAINT UNIQUE(att1, att2)
107 );
```

Création d'une table TestUnique1 avec deux attributs, et une contrainte d'unicité sur ces deux attributs

```
109 CREATE TABLE TestUnique2(
110  att1 INTEGER UNIQUE
111 );
```

Création d'une table TestUnique2 avec un seul attribut unique (raccourci d'écriture)

Le raccourci d'écriture n'est valable que si la contrainte porte sur un seul attribut

Contrainte NOT NULL

att$_1$ *type*$_1$ [**NOT**] **NULL**

- Contrainte [NOT] NULL uniquement spécifiée lors de la définition d'un attribut
- Indique que l'attribut peut / ne peut pas recevoir de valeurs nulles
- Par défaut, les attributs acceptent les valeurs nulles

Exemple de contrainte NOT NULL

```
113 CREATE TABLE TestNotNull(
114  att1 INTEGER NOT NULL,
115  att2 INTEGER NOT NULL
116 );
```

Création d'une table TestNotNull avec deux attributs dont les valeurs ne peuvent pas être nulles

Calvin et Hobbes (Bill Watterson)

Contrainte de clé primaire

CONSTRAINT nom_c **PRIMARY KEY**(att_i, att_j, ...)

- Une contrainte de clé primaire indique que l'ensemble d'attributs (att_i, att_j, ...) sert d'identifiant principal pour les n-uplets de la table
- La valeur de la clé primaire permet donc de retrouver un n-uplet de manière non ambigüe
- Implique NOT NULL et UNIQUE sur chacun des (att_i, att_j, ...)
- Au maximum une contrainte PRIMARY KEY par table

Contrainte de clé primaire - auto-incrément

Il est souvent préférable d'utiliser un entier (sans signification) en clé primaire plutôt qu'un attribut (e.g., le nom de la série, les nom/prénom d'une actrice)

- Un SGBD peut gérer automatiquement ces clés primaires par auto-incrément (AUTO_INCREMENT pour MariaDB et MySQL, SERIAL pour PostgreSQL, etc.)
- Dans la requête d'insertion, l'attribut auto-incrémenté a une valeur NULL
- Le SGBD affecte automatiquement à la clé primaire la plus grande valeur positive de la colonne $+1$
- La première valeur d'un attribut auto-incrémenté est 1

Exemple de contrainte de clé primaire

```
118 CREATE TABLE TestPK1(
119  att1 INTEGER,
120  att2 INTEGER,
121  CONSTRAINT pk_TestPK1 PRIMARY KEY(att1, att2)
122 );
```

Création d'une table TestPK1 avec deux attributs qui forment la clé primaire de la table

```
124 CREATE TABLE TestPK2(
125  att1 INTEGER AUTO_INCREMENT PRIMARY KEY
126 );
```

Création d'une table TestPK2 avec un seul attribut qui est clé primaire (raccourci d'écriture), et dont les valeurs sont gérées par le SGBD (auto-increment)

Le raccourci d'écriture n'est valable que si la contrainte porte sur un seul attribut

Contrainte de clé étrangère

CONSTRAINT nom_c **FOREIGN KEY**(att_1, ..., att_k)
REFERENCES $table\_cible(att'_1,...,att'_k)$

- Une clé étrangère est une **référence** vers la clé primaire d'une autre table
- Les valeurs pour (att_1, ..., att_k) doivent correspondre aux valeurs d'un des n-uplets de $table\_cible$ pour ses attributs $(att'_1,...,att'_k)$

Certains moteurs de stockage ne gèrent pas les clés étrangères (e.g., MyISAM)

Exemple de contrainte de clé étrangère

```
128 CREATE TABLE TestFK1(
129  att1 INTEGER,
130  att2 INTEGER,
131  CONSTRAINT fk_TestFK1_att1 FOREIGN KEY(att1) REFERENCES
         TestPK1(att1)
132 );
```

Création d'une table TestFK1 avec deux attributs, parmi lesquels le premier est clé étrangère. Il ne peut donc avoir que des valeurs existantes dans l'attribut référencé (ici att1 de la table TestPK1)

```
134 CREATE TABLE TestFK2(
135  att1 INTEGER REFERENCES TestPK2(att1)
136 );
```

Création d'une table TestFK2 avec un attribut qui est clé étrangère (raccourci d'écriture). Il ne peut donc avoir que des valeurs existantes dans l'attribut référencé (ici att1 de la table TestPK2)

Contrainte CHECK

CONSTRAINT nom_c **CHECK**(*condition*)

- La contrainte CHECK permet de spécifier une condition (booléenne) qui sera vérifiée lors d'insertions ou modification de n-uplets
- La forme CHECK(*att* IN ('val_1', 'val_2', ...)), utilisée pour les types énumérés est un cas particulier de cette contrainte

Dans MariaDB/MySQL, les contraintes CHECK ne sont pas vérifiées

Exemple de contrainte CHECK

```
139 CREATE TABLE TestCheck1(
140  att1 INTEGER,
141  att2 INTEGER,
142  CONSTRAINT CHECK(att1 > 0)
143 );
```

Création d'une table TestCheck1 avec deux attributs. Le premier a une contrainte qui force ses valeurs à être positives

```
145 CREATE TABLE TestCheck2(
146  att1 INTEGER CHECK(att1 > 0)
147 );
```

Création d'une table TestCheck2 avec un seul attribut. Celui-ci a une contrainte qui force ses valeurs à être positives (raccourci d'écriture)

Exemple de création de tables avec contraintes

SÉRIES(nomSérie)

```
150 CREATE TABLE Series(
151  nomSerie VARCHAR(255) PRIMARY KEY
152 );
```

SAISONS(idSaison, dateLancement, #nomSérie)

```
154 CREATE TABLE Saisons(
155  idSaison INT PRIMARY KEY,
156  dateLancement DATE,
157  nomSerie VARCHAR(255) REFERENCES Series(nomSerie)
158 );
```

Field	Type	Null	Key	Default	Extra
idSaison	int(11)	NO	PRI	*NULL*	
dateLancement	date	YES		*NULL*	
nomSerie	varchar(255)	YES		*NULL*	

Exemple de création de tables avec contraintes (2)

ÉPISODES(numéro, #idSaison, titre)

```
160 CREATE TABLE Episodes(
161  numero INT,
162  idSaison INT REFERENCES Saisons(idSaison),
163  titre VARCHAR(255),
164  PRIMARY KEY(numero, idSaison)
165 );
```

ACTRICES(numINSEE, nom, prénom)

```
167 CREATE TABLE Actrices(
168  numINSEE INT PRIMARY KEY,
169  nom VARCHAR(255),
170  prenom VARCHAR(255)
171 );
```

Field	Type	Null	Key	Default	Extra
numINSEE	int(11)	NO	PRI	*NULL*	
nom	varchar(255)	YES		*NULL*	
prenom	varchar(255)	YES		*NULL*	

Exemple de création de tables avec contraintes (3)

JOUER(#numero, #idSaison, #numINSEE, salaire)

```
173 CREATE TABLE Jouer(
174  numero INT,
175  idSaison INT,
176  numINSEE INT REFERENCES Actrices(numINSEE),
177  salaire DOUBLE,
178  CONSTRAINT pk_Joue PRIMARY KEY(numero, idSaison, numINSEE),
179  CONSTRAINT fk_episodes FOREIGN KEY(numero, idSaison)
         REFERENCES Episodes(numero, idSaison),
180  CONSTRAINT check_salaire_positif CHECK(salaire > 0.0)
181 );
```

Field	Type	Null	Key	Default	Extra
numero	int(11)	NO	PRI	0	
idSaison	int(11)	NO	PRI	0	
numINSEE	int(11)	NO	PRI	0	
salaire	double	YES		*NULL*	

```
fduchateau=> \dS+ series.joue
                                        Table "series.joue"
  Column   |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
-----------+---------+-----------+----------+---------+---------+--------------+-------------
 numero    | integer |           | not null |         | plain   |              |
 idsaison  | integer |           | not null |         | plain   |              |
 numinsee  | integer |           | not null |         | plain   |              |
 salaire   | real    |           |          |         | plain   |              |
Indexes:
    "pk_joue" PRIMARY KEY, btree (numero, idsaison, numinsee)
Check constraints:
    "check_salaire_positif" CHECK (salaire > 0.0::double precision)
Foreign-key constraints:
    "fk_episodes" FOREIGN KEY (numero, idsaison) REFERENCES episodes(numero, idsaison)
    "joue_numinsee_fkey" FOREIGN KEY (numinsee) REFERENCES actrices(numinsee)
Access method: heap
```

En résumé

- Contraintes de clés (primaires et étrangères)
- Contraintes sur les valeurs autorisées (UNIQUE, [NOT] NULL, et CHECK)

Démo avec MariaDB (script SQL en ligne)