

BDBIO - Modélisation relationnelle - niveau physique

Fabien Duchateau

fabien.duchateau [at] univ-lyon1.fr

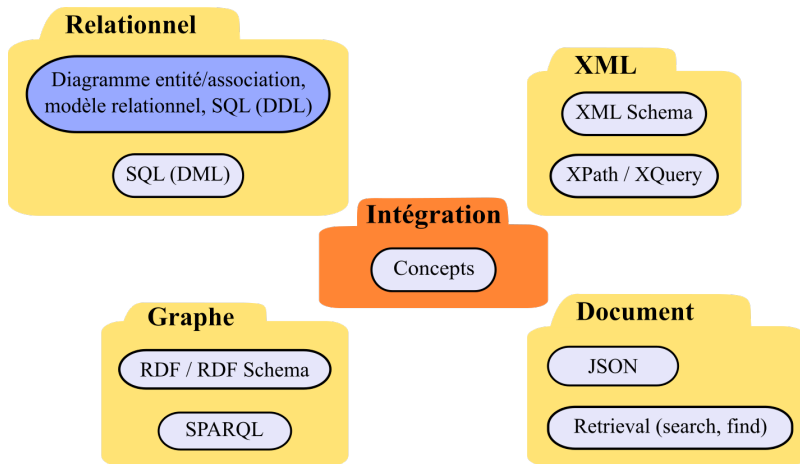
Université Claude Bernard Lyon 1

2023 - 2024



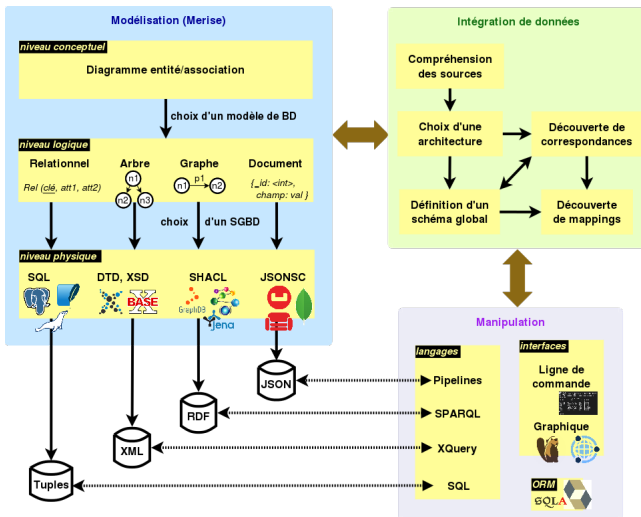
<https://perso.liris.cnrs.fr/fabien.duchateau/BDBIO/>

Positionnement dans BDBIO



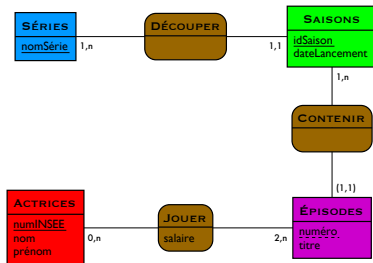
Ces diapositives utilisent **le genre féminin** (e.g., chercheuse, développeuses) plutôt que **l'écriture inclusive** (moins accessible, moins concise, et pas totalement inclusive)

Relations entre les notions étudiées en BDBIO



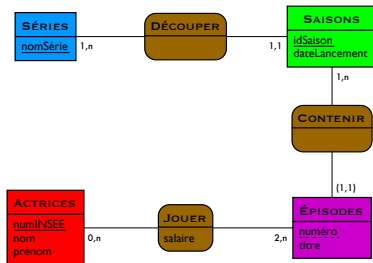
Rappels

Du diagramme entité / association (niveau conceptuel)...



Rappels

Du diagramme entité / association (niveau conceptuel)...



... à un modèle relationnel (niveau logique)

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
- ▶ Certains outils traduisent automatiquement (e.g., Mocodo)

Généralités sur le langage SQL

SQL = Structured Query Language

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

SQL comme langage de description de données (niveau physique) : chaque SGBD respecte plus ou moins bien la norme SQL lors de son implémentation.

Dans ce cours, syntaxe SQL du SGBD PostgreSQL

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

<https://www.postgresql.org/docs/current/ddl.html>

Plan

Création d'une BD

Création de tables

Insertion de n-uplets

Contraintes d'intégrité

Évolution

Syntaxe de création d'une base de données

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)

```
CREATE DATABASE nom_bd ;
```

- ▶ Création d'une base de données nommée *nom_bd*
- ▶ Notion de schéma sous PostgreSQL, pour organiser logiquement les tables d'une base de données. Par défaut, schéma *public* (accès transparent)

<https://www.postgresql.org/docs/current/sql-createdatabase.html>

<https://www.postgresql.org/docs/current/ddl-schemas.html>

Plan

Création d'une BD

Création de tables

Insertion de n-uplets

Contraintes d'intégrité

Évolution

Généralités

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

- ▶ Avec une requête SQL ou un client graphique (e.g., [DBeaver](#), [pgAdmin](#))
- ▶ Attention aux différences d'implémentation de SQL

À la création d'une table, on indique :

- ▶ Le nom des attributs
- ▶ Le type de chaque attribut
- ▶ Des contraintes d'intégrité

<https://www.postgresql.org/docs/current/ddl.html>

Syntaxe de création de table

```
CREATE TABLE nom_table (  
  att1 type1  
  [, att2 type2, ...]  
);
```

- ▶ 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 [...] représentent des éléments optionnels

<https://www.postgresql.org/docs/current/sql-createtable.html>

Quelques types d'attributs

- ▶ BOOLEAN
- ▶ VARCHAR(longueur) ou TEXT (pas de limite)
- ▶ INT ou INTEGER, SMALLINT, BIGINT
- ▶ REAL (type inexact) ou NUMERIC(préc, éch), un nombre exact de *préc* chiffres dont *éch* après la virgule
- ▶ DATE (format YYYY-MM-JJ), TIME, TIMESTAMP
- ▶ Variantes, énumération, types géométriques, tableaux, etc.

<https://www.postgresql.org/docs/current/datatype.html>

Exemple de création de tables

SÉRIES(nomSérie)

```

19 CREATE TABLE Series(
20   nomSérie VARCHAR(255)
21 );

```

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

SAISONS(idSaison, dateLancement, #nomSérie)

```

23 CREATE TABLE Saisons(
24   idSaison INT,
25   dateLancement DATE,
26   nomSérie VARCHAR(255)
27 );

```

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

Exemple de création de tables (2)

ÉPISODES(numéro, #idSaison, titre)

```

29 CREATE TABLE Episodes(
30     numero INT,
31     idSaison INT,
32     titre VARCHAR(255)
33 );

```

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

ACTRICES(numINSEE, nom, prénom)

```

35 CREATE TABLE Actrices(
36     numINSEE INT,
37     nom VARCHAR(255),
38     prenom VARCHAR(255)
39 );

```

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
41 CREATE TABLE Jouer(
42     numero INT,
43     idSaison INT,
44     numINSEE INT,
45     salaire REAL
  
```

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	

Colonne	Type	Modificateurs
numero	integer	
idsaison	integer	
numinsee	integer	
salaire	real	

En résumé

- ▶ Création (du schéma) d'une table avec CREATE TABLE
- ▶ Définition de chaque attribut par son nom et son type



Plan

Création d'une BD

Création de tables

Insertion de n-uplets

Contraintes d'intégrité

Évolution

Généralités

Le schéma de la table est créé, mais il n'y a aucune donnée !

SQL (comme LMD) permet la création (CRUD) d'instances

Deux méthodes pour ajouter des instances en SQL :

- ▶ Insertion d'un (ou plusieurs) n-uplets
- ▶ Copie de données existantes

<https://www.postgresql.org/docs/current/dml-insert.html>

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(att1, ..., attn)  
VALUES(val1, ..., valn);
```

- ▶ 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 non définie (NULL)

Insertion d'un seul n-uplet (2)

```
INSERT INTO nom_table  
VALUES(val1, ..., valn);
```

- ▶ 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

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

← T →		nomSerie		
<input type="checkbox"/>	Modifier	Copier	Effacer	The Big Bang Theory
<input type="checkbox"/>	Modifier	Copier	Effacer	Game of Thrones

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







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 Thrones

Copie de données existantes

```
INSERT INTO nom_table(att1, ..., attn)  
SELECT e1, ..., en  
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

		nomSerie		
<input type="checkbox"/>	 Modifier	 Copier	 Effacer	The Big Bang Theory
<input type="checkbox"/>	 Modifier	 Copier	 Effacer	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 Thrones

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


Exemple de copie de données existantes

		nomSerie		
<input type="checkbox"/>	 Modifier	 Copier	 Effacer	The Big Bang Theory
<input type="checkbox"/>	 Modifier	 Copier	 Effacer	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 Thrones

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

► Quel résultat obtient-on ?

Exemple de copie de données existantes

	nomSerie
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	The Big Bang Theory
<input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer	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 Thrones

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

► Quel résultat obtient-on ?

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

Exemple de copie de données existantes

		nomSerie
<input type="checkbox"/>	Modifier	The Big Bang Theory
<input type="checkbox"/>	Modifier	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 Thrones

```

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

```

- ▶ Quel résultat obtient-on ?
- ▶ Pourquoi peut-on avoir deux instances *The Big Bang Theory* dans la table Serie ?

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
- ▶ Insertion d'un n-uplet (INSERT INTO ... SELECT ...)

```
SELECT *
FROM 'Université'
LIMIT 0, 30
```

Afficher : Ligne de départ: 0 Nombre de lignes: 30

Trier sur l'index: Aucune

+ Options

	nomU	ville	effectif
<input type="checkbox"/> Modifier Copier Effacer	INSA	Lyon	36000
<input type="checkbox"/> Modifier Copier Effacer	UCB	Lyon	15000
<input type="checkbox"/> Modifier Copier Effacer	UJF	Grenoble	10000
<input type="checkbox"/> Modifier Copier Effacer	UJM	Saint-Etienne	21000

Plan

Création d'une BD

Création de tables

Insertion de n-uplets

Contraintes d'intégrité

Évolution

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, mises à jour ou suppressions 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 non définies
- ▶ Clé primaire (PK, pour Primary Key)
- ▶ Clé étrangère (FK, pour Foreign Key)
- ▶ Satisfaction d'une condition simple (CHECK)

<https://www.postgresql.org/docs/current/ddl-constraints.html>

Syntaxe de création de contrainte

```
CREATE TABLE nom_table (  
  att1 type1,  
  [...]   
  CONSTRAINT [nom1] def_contrainte1  
  [...]   
);
```

- ▶ Le mot-clé **CONSTRAINT** permet de spécifier une contrainte
- ▶ *nom₁* est le nom de la contrainte (optionnel)
- ▶ *def_contrainte₁* 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

```
104 CREATE TABLE TestUnique1(  
105   att1 INTEGER,  
106   att2 INTEGER,  
107   CONSTRAINT UNIQUE(att1, att2)  
108 );
```

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

```
110 CREATE TABLE TestUnique2(  
111   att1 INTEGER UNIQUE  
112 );
```

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

Le raccourci n'est valable que lorsque 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 non définies (représentées par NULL)
- ▶ Par défaut, les attributs acceptent les valeurs non définies

Exemple de contrainte [NOT] NULL

```
114 CREATE TABLE TestNotNull(  
115   att1 INTEGER NOT NULL,  
116   att2 INTEGER NOT NULL  
117 );
```

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



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 ambiguë
- ▶ 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 MySQL, `SERIAL` pour PostgreSQL, etc.)
- ▶ À l'insertion d'un n-uplet, l'attribut en clé primaire reçoit une valeur non définie (`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

```
119 CREATE TABLE TestPK1(  
120   att1 INTEGER,  
121   att2 INTEGER,  
122   CONSTRAINT pk_TestPK1 PRIMARY KEY(att1, att2)  
123 );
```

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

```
125 CREATE TABLE TestPK2(  
126   att1 INTEGER AUTO_INCREMENT PRIMARY KEY  
127 );
```

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 n'est valable que lorsque la contrainte porte sur un seul attribut

Contrainte de clé étrangère

```
CONSTRAINT nomc FOREIGN KEY(att1, ..., attk)  
REFERENCES table_cible(att'1, ..., att'k)
```

- ▶ Une clé étrangère est une **référence** vers la clé primaire d'une autre table
- ▶ Les valeurs pour (*att₁*, ..., *att_k*) doivent correspondre aux valeurs d'un des n-uplets de *table_cible* pour ses attributs (*att'₁*, ..., *att'_k*)

Exemple de contrainte de clé étrangère

```
129 CREATE TABLE TestFK1(  
130   att1 INTEGER,  
131   att2 INTEGER,  
132   CONSTRAINT fk_TestFK1_att1 FOREIGN KEY(att1) REFERENCES  
      TestPK1(att1)  
133 );
```

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)

```
135 CREATE TABLE TestFK2(  
136   att1 INTEGER REFERENCES TestPK2(att1)  
137 );
```

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

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

`SERIES(nomSerie)`

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

Exemple de création de tables avec contraintes

SERIES(nomSerie)

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

SAISONS(idSaison, dateLancement, #nomSerie)

```
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)

EPISODES(numero, #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 );
```

Exemple de création de tables avec contraintes (2)

EPISODES(numero, #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, prenom)

```

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)

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

```

173 CREATE TABLE Jouer(
174     numero INT,
175     idSaison INT,
176     numINSEE INT REFERENCES Actrices(numINSEE),
177     salaire REAL,
178     CONSTRAINT pk_Joue PRIMARY KEY(numero, idSaison, numINSEE),
179     CONSTRAINT fk_Episodes FOREIGN KEY(numero, idSaison)
180         REFERENCES Episodes(numero, idSaison),
181     CONSTRAINT check_salaire_positif CHECK(salaire > 0.0)
182 );

```

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=> \d+ 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)

```
[fduchateau=> select * from episodes ;
 numero | idsaison | titre
-----|-----|-----
      1 |      1 | The Skank Reflex Analysis
      1 |      2 | The Date Night Variable
      1 |      3 | Winter is coming
      2 |      3 | The Kingsroad
(4 rows)

[fduchateau=> select * from joue ;
 numero | idsaison | numinsee | salaire
-----|-----|-----|-----
      1 |      3 |      111 |
      1 |      3 |      222 |      6000
      2 |      3 |      111 | 5437.65
      2 |      3 |      222 |      6000
      1 |      1 |      333 |      3200
      1 |      1 |      444 |      3200
      1 |      2 |      333 |      3200
      1 |      2 |      444 |      3200
(8 rows)
```

Plan

Création d'une BD

Création de tables

Insertion de n-uplets

Contraintes d'intégrité

Évolution

Motivation

Une base de données est amenée à évoluer :

- ▶ Contraintes non implémentables à la création (e.g., deux tables qui se référencent)
- ▶ Nouveaux besoins ou fonctionnalités, qui nécessitent une modification de l'application et de son modèle de données

Opérations fréquentes :

- ▶ Ajout d'information ou modification du schéma d'une table
- ▶ Mise à jour (CRUD) et suppression (CRUD) d'instance

Commandes SQL pour l'évolution

- ▶ Base de données (ALTER DATABASE, DROP DATABASE)
- ▶ Table (ALTER TABLE, DROP TABLE)
- ▶ Attribut (ALTER TABLE)
- ▶ Contrainte (ALTER TABLE)
- ▶ N-uplet (UPDATE, DELETE)

<https://www.postgresql.org/docs/current/ddl-alter.html>

<https://www.postgresql.org/docs/current/sql-droptable.html>

<https://www.postgresql.org/docs/current/dml.html>