Le modèle relationnel (partie I)

Génie Biologique et Alimentaire 4ème année

Olivier Caron[1]
http://ocaron.polytech-lille.net

[1]École d'ingénieurs Polytech Lille
Université de Lille

30 septembre 2024

Plan

Partie I

Plan

Partie I

- Le modèle relationnel

Plan

Partie I

- Le modèle relationnel
- Le langage SQL, partie DDL (Data Definition Language)

Plan

Partie I

- Le modèle relationnel
- Le langage SQL, partie DDL (Data Definition Language)
- Passage d'un schéma conceptuel à un schéma relationnel

Plan

Partie I

- Le modèle relationnel
- Le langage SQL, partie DDL (Data Definition Language)
- Passage d'un schéma conceptuel à un schéma relationnel

Partie II

Plan

Partie I

- Le modèle relationnel
- Le langage SQL, partie DDL (Data Definition Language)
- Passage d'un schéma conceptuel à un schéma relationnel

Partie II

- L'algèbre relationnelle (base de requêtes)

Plan

Partie I

- Le modèle relationnel
- Le langage SQL, partie DDL (Data Definition Language)
- Passage d'un schéma conceptuel à un schéma relationnel

Partie II

- L'algèbre relationnelle (base de requêtes)
- Le langage SQL, partie requêtes et modification de données

Le modèle relationnel

- Un schéma conceptuel (UML, MCD, ...) est très pratique pour la phase d'analyse et conception

Le modèle relationnel

- Un schéma conceptuel (UML, MCD, ...) est très pratique pour la phase d'analyse et conception
- Par contre, un schéma conceptuel présente des limites pour une implémentation ou effectuer des requêtes.

Le modèle relationnel

- Un schéma conceptuel (UML, MCD, ...) est très pratique pour la phase d'analyse et conception
- Par contre, un schéma conceptuel présente des limites pour une implémentation ou effectuer des requêtes.
- L'universitaire Codd (1970) a inventé le modèle relationnel basé sur des concepts simples :

Le modèle relationnel

- Un schéma conceptuel (UML, MCD, ...) est très pratique pour la phase d'analyse et conception
- Par contre, un schéma conceptuel présente des limites pour une implémentation ou effectuer des requêtes.
- L'universitaire Codd (1970) a inventé le modèle relationnel basé sur des concepts simples :
  - Facilement implémentable sur un ordinateur

Le modèle relationnel

- Un schéma conceptuel (UML, MCD, ...) est très pratique pour la phase d'analyse et conception
- Par contre, un schéma conceptuel présente des limites pour une implémentation ou effectuer des requêtes.
- L'universitaire Codd (1970) a inventé le modèle relationnel basé sur des concepts simples :
  - Facilement implémentable sur un ordinateur
  - Facilité pour poser des requêtes

Le modèle relationnel

- Un schéma conceptuel (UML, MCD, ...) est très pratique pour la phase d'analyse et conception
- Par contre, un schéma conceptuel présente des limites pour une implémentation ou effectuer des requêtes.
- L'universitaire Codd (1970) a inventé le modèle relationnel basé sur des concepts simples :
  - Facilement implémentable sur un ordinateur
  - Facilité pour poser des requêtes
  - Passage simple : schéma conceptuel $\longrightarrow$ schéma relationnel

Concepts de base

Domaine

ensemble de valeurs

Concepts de base

Domaine

ensemble de valeurs

- Exemples :
  le domaine des entiers
  le domaine des couleurs d'un drapeau $Dcoul = \{bleu, blanc, rouge\}$
  le domaine des booléens $Dbool = \{0, 1\}$

Rappel : le produit cartésien

Produit cartésien

le produit cartésien d'un ensemble de domaines $D_1, D_2 \ldots D_n$ noté $D_1 \times D_2 \times \ldots D_n$ est l'ensemble des n-uplets ou tuples $< v_1, v_2, \ldots, v_n >$ tel que $v_i \in D_i$

Rappel : le produit cartésien

Produit cartésien

le produit cartésien d'un ensemble de domaines $D_1, D_2 \ldots D_n$ noté $D_1 \times D_2 \times \ldots D_n$ est l'ensemble des n-uplets ou tuples $< v_1, v_2, \ldots, v_n >$ tel que $v_i \in D_i$

- Exemple :
  $Dcoul \times Dbool = \{$
  $< bleu, 0 >, < bleu, 1 >, < blanc, 0 >, < blanc, 1 >,$
  $< rouge, 0 >, < rouge, 1 >\}$

Relation

Relation ou Table

sous-ensemble du produit cartésien d'une liste de domaine

Relation

Relation ou Table

sous-ensemble du produit cartésien d'une liste de domaine

- Une relation est caractérisée par un nom

Relation

Relation ou Table

sous-ensemble du produit cartésien d'une liste de domaine

- Une relation est caractérisée par un nom
- Il n'y a pas deux lignes (t-uples) égale (théorie des ensembles)

Attribut

Attribut

Colonne d'une relation caractérisée par un nom

Attribut

Attribut

Colonne d'une relation caractérisée par un nom

- L'ordre des attributs n'a aucune importance

Clé primaire de relation

Clé primaire

Une clé primaire de la relation R est un ensemble minimal des attributs de R dont les valeurs identifient à coup sûr une occurrence.

Clé primaire de relation

Clé primaire

Une clé primaire de la relation R est un ensemble minimal des attributs de R dont les valeurs identifient à coup sûr une occurrence.

- Toute relation dispose d'une clé primaire

Clé primaire de relation

Clé primaire

Une clé primaire de la relation R est un ensemble minimal des attributs de R dont les valeurs identifient à coup sûr une occurrence.

- Toute relation dispose d'une clé primaire
- Une clé primaire ne peut avoir de valeurs nulles.

Clé primaire de relation

Clé primaire

Une clé primaire de la relation R est un ensemble minimal des attributs de R dont les valeurs identifient à coup sûr une occurrence.

- Toute relation dispose d'une clé primaire
- Une clé primaire ne peut avoir de valeurs nulles.
- Pour signaler la clé primaire, ses attributs sont généralement soulignés.

Clé primaire de relation

Clé primaire

Une clé primaire de la relation R est un ensemble minimal des attributs de R dont les valeurs identifient à coup sûr une occurrence.

- Toute relation dispose d'une clé primaire
- Une clé primaire ne peut avoir de valeurs nulles.
- Pour signaler la clé primaire, ses attributs sont généralement soulignés.
- Quelques exemples :

Clé primaire de relation

Clé primaire

Une clé primaire de la relation R est un ensemble minimal des attributs de R dont les valeurs identifient à coup sûr une occurrence.

- Toute relation dispose d'une clé primaire
- Une clé primaire ne peut avoir de valeurs nulles.
- Pour signaler la clé primaire, ses attributs sont généralement soulignés.
- Quelques exemples :
  - Le numéro `insee` pour la relation `Individu`

Clé primaire de relation

Clé primaire

Une clé primaire de la relation R est un ensemble minimal des attributs de R dont les valeurs identifient à coup sûr une occurrence.

- Toute relation dispose d'une clé primaire
- Une clé primaire ne peut avoir de valeurs nulles.
- Pour signaler la clé primaire, ses attributs sont généralement soulignés.
- Quelques exemples :
  - Le numéro `insee` pour la relation `Individu`
  - Le numéro `isbn` pour la relation `Livre`

Clé étrangère de relation

Clé étrangère

Une clé étrangère dans une relation est formée d'un ou plusieurs attributs qui constituent une clé primaire dans une autre relation.

Clé étrangère de relation

Clé étrangère

Une clé étrangère dans une relation est formée d'un ou plusieurs attributs qui constituent une clé primaire dans une autre relation.

- Il peut y avoir plusieurs clés étrangères dans une relation

Clé étrangère de relation

Clé étrangère

Une clé étrangère dans une relation est formée d'un ou plusieurs attributs qui constituent une clé primaire dans une autre relation.

- Il peut y avoir plusieurs clés étrangères dans une relation
- En général, on utilise le caractère "♯" pour désigner une clé étrangère

Schéma relationnel

schéma relationnel

Un *schéma de relation* est le nom de la relation suivi de la liste des attributs avec leur domaine.

Schéma relationnel

schéma relationnel

Un *schéma de relation* est le nom de la relation suivi de la liste des attributs avec leur domaine.

- Un schéma de relation représente *l'intention* de la relation.

Schéma relationnel

schéma relationnel

Un *schéma de relation* est le nom de la relation suivi de la liste des attributs avec leur domaine.

- Un schéma de relation représente *l'intention* de la relation.
- L'ensemble des tuples d'une relation représente une *extension* possible

Schéma relationnel

schéma relationnel

Un *schéma de relation* est le nom de la relation suivi de la liste des attributs avec leur domaine.

- Un schéma de relation représente *l'intention* de la relation.
- L'ensemble des tuples d'une relation représente une *extension* possible
- Exemple : intention relation

```
Voiture (immatriculation: chaîne(8), marque: chaîne(60), puissance: entier)

Proprietaire (code: entier, nom: chaîne(40), ♯voiture(Voiture): chaîne(8))
```

Base de données relationnelle

Base de données

Ensemble de schémas de relation et dont les occurrences sont des tuples de ces relations

Implantation dans les SGBDR

- 80% des SGBD (Systèmes de Gestion de Bases de données) sont des SGBD relationnels (SGBDR)

Implantation dans les SGBDR

- 80% des SGBD (Systèmes de Gestion de Bases de données) sont des SGBD relationnels (SGBDR)
- Tous désormais exploitent un même langage : le langage SQL (Structured Query Language)

Implantation dans les SGBDR

- 80% des SGBD (Systèmes de Gestion de Bases de données) sont des SGBD relationnels (SGBDR)
- Tous désormais exploitent un même langage : le langage SQL (Structured Query Language)
- SQL sert pour la consultation de la base (les requêtes)

Implantation dans les SGBDR

- 80% des SGBD (Systèmes de Gestion de Bases de données) sont des SGBD relationnels (SGBDR)
- Tous désormais exploitent un même langage : le langage SQL (Structured Query Language)
- SQL sert pour la consultation de la base (les requêtes)
- SQL sert pour la création et modification de la base

Implantation dans les SGBDR

- 80% des SGBD (Systèmes de Gestion de Bases de données) sont des SGBD relationnels (SGBDR)
- Tous désormais exploitent un même langage : le langage SQL (Structured Query Language)
- SQL sert pour la consultation de la base (les requêtes)
- SQL sert pour la création et modification de la base
- Standard de fait, puis véritable standard - Norme ISO

Implantation dans les SGBDR

- 80% des SGBD (Systèmes de Gestion de Bases de données) sont des SGBD relationnels (SGBDR)
- Tous désormais exploitent un même langage : le langage SQL (Structured Query Language)
- SQL sert pour la consultation de la base (les requêtes)
- SQL sert pour la création et modification de la base
- Standard de fait, puis véritable standard - Norme ISO
- Langage en constante évolution : SQL-86, SQL-89 (SQL 1), SQL-92 (SQL 2), SQL-99 (SQL 3), SQL 2003, . . .SQL 2016, SQL 2023

Syntaxe générale des commandes SQL

- Pas de distinction minuscule et majuscule.

Syntaxe générale des commandes SQL

- Pas de distinction minuscule et majuscule.
- Les commandes commencent par un mot clé servant à nommer l'opération de base à exécuter.

Syntaxe générale des commandes SQL

- Pas de distinction minuscule et majuscule.
- Les commandes commencent par un mot clé servant à nommer l'opération de base à exécuter.
- Chaque commande SQL doit remplir deux exigences :

Syntaxe générale des commandes SQL

- Pas de distinction minuscule et majuscule.
- Les commandes commencent par un mot clé servant à nommer l'opération de base à exécuter.
- Chaque commande SQL doit remplir deux exigences :
  - Indiquer les données sur lesquelles elle opère (un ensemble de lignes stockées dans une ou plusieurs classes)

Syntaxe générale des commandes SQL

- Pas de distinction minuscule et majuscule.
- Les commandes commencent par un mot clé servant à nommer l'opération de base à exécuter.
- Chaque commande SQL doit remplir deux exigences :
  - Indiquer les données sur lesquelles elle opère (un ensemble de lignes stockées dans une ou plusieurs classes)
  - Indiquer l'opération à exécuter sur ces données

Les domaines principaux par défaut

- Les numériques : `integer`, `smallint`, `double`, `float`,...
  exemple : `23`, `-122`, `3.4`, `-6.7`,...

Les domaines principaux par défaut

- Les numériques : `integer, smallint, double, float,...`
  exemple : `23, -122, 3.4, -6.7,...`
- Les alphanumériques : `char, varchar(n), char(n), text`
  exemples : `'v', 'la vie est un long fleuve ...'`

Les domaines principaux par défaut

- Les numériques : `integer, smallint, double, float,...`
  exemple : `23, -122, 3.4, -6.7,...`
- Les alphanumériques : `char, varchar(n), char(n), text`
  exemples : `'v', 'la vie est un long fleuve ...'`
- Le type date (format configurable)
  exemple : `'2002-02-28'`

Les domaines principaux par défaut

- Les numériques : `integer, smallint, double, float,...`
  exemple : `23, -122, 3.4, -6.7,...`
- Les alphanumériques : `char, varchar(n), char(n), text`
  exemples : `'v', 'la vie est un long fleuve ...'`
- Le type date (format configurable)
  exemple : `'2002-02-28'`
- Le type boolean : `boolean` ou `bool`
  exemples : `true` ou `'t'` et `false` ou `'f'`

Les domaines principaux par défaut

- Les numériques : `integer, smallint, double, float,...`
  exemple : `23, -122, 3.4, -6.7,...`
- Les alphanumériques : `char, varchar(n), char(n), text`
  exemples : `'v', 'la vie est un long fleuve ...'`
- Le type date (format configurable)
  exemple : `'2002-02-28'`
- Le type boolean : `boolean` ou `bool`
  exemples : `true` ou `'t'` et `false` ou `'f'`
- Et bien d'autres encore (timestamp, ...)

Quelques mots sur Postgres

- Travaux de Stonebraker (Univ. Berkeley), `http://www.postgresql.org/`

Quelques mots sur Postgres

- Travaux de Stonebraker (Univ. Berkeley), `http://www.postgresql.org/`
- Fonctionne sous Unix (linux, Mac OS X) et Windows

Quelques mots sur Postgres

- Travaux de Stonebraker (Univ. Berkeley), `http://www.postgresql.org/`
- Fonctionne sous Unix (linux, Mac OS X) et Windows
- Gratuit, mais il existe des versions commerciales

Quelques mots sur Postgres

- Travaux de Stonebraker (Univ. Berkeley), `http://www.postgresql.org/`
- Fonctionne sous Unix (linux, Mac OS X) et Windows
- Gratuit, mais il existe des versions commerciales
- Architecture client/serveur (processus, TCP-IP)

Quelques mots sur Postgres

- Travaux de Stonebraker (Univ. Berkeley), `http://www.postgresql.org/`
- Fonctionne sous Unix (linux, Mac OS X) et Windows
- Gratuit, mais il existe des versions commerciales
- Architecture client/serveur (processus, TCP-IP)
- Gère plusieurs utilisateurs (sécurité), les accès concurrents

Les utilisateurs Postgres

- L'utilisateur **"postgres"** dispose de tous les droits (root), création bases, utilisateurs, destruction,...

Les utilisateurs Postgres

- L'utilisateur **"postgres"** dispose de tous les droits (root), création bases, utilisateurs, destruction,...
- Les **utilisateurs-administrateurs** : peuvent créer des bases, peuvent autoriser d'autres utilisateurs à accéder à ces bases.
  Certains utilisateurs-administrateurs peuvent créer d'autres utilisateurs.

Les utilisateurs Postgres

- L'utilisateur **"postgres"** dispose de tous les droits (root), création bases, utilisateurs, destruction,...
- Les **utilisateurs-administrateurs** : peuvent créer des bases, peuvent autoriser d'autres utilisateurs à accéder à ces bases.
  Certains utilisateurs-administrateurs peuvent créer d'autres utilisateurs.
- Les **utilisateurs** peuvent accéder à des bases (selon les droits)

Le réseau Polytech Lille

- Un serveur postgres situé sur le serveur `serveur-etu.polytech-lille.fr`

Le réseau Polytech Lille

- Un serveur postgres situé sur le serveur `serveur-etu.polytech-lille.fr`
- Chaque étudiant dispose d'un compte **utilisateur-administrateur**

Le réseau Polytech Lille

- Un serveur postgres situé sur le serveur `serveur-etu.polytech-lille.fr`
- Chaque étudiant dispose d'un compte **utilisateur-administrateur**
- Accès disponible à distance sur toutes les machines Polytech

Le réseau Polytech Lille

- Un serveur postgres situé sur le serveur `serveur-etu.polytech-lille.fr`
- Chaque étudiant dispose d'un compte **utilisateur-administrateur**
- Accès disponible à distance sur toutes les machines Polytech
- Configuration initiale (`.bashrc`) :

```
export PGHOST=serveur-etu.polytech-lille.fr
```

Le réseau Polytech Lille

- Un serveur postgres situé sur le serveur `serveur-etu.polytech-lille.fr`
- Chaque étudiant dispose d'un compte **utilisateur-administrateur**
- Accès disponible à distance sur toutes les machines Polytech
- Configuration initiale (`.bashrc`) :

  ```
  export PGHOST=serveur-etu.polytech-lille.fr
  ```

- Par défaut : login Postgres = login Unix (sauf pour login avec '-'), password ("postgres")

Le réseau Polytech Lille

- Un serveur postgres situé sur le serveur `serveur-etu.polytech-lille.fr`
- Chaque étudiant dispose d'un compte **utilisateur-administrateur**
- Accès disponible à distance sur toutes les machines Polytech
- Configuration initiale (`.bashrc`) :

  ```
  export PGHOST=serveur-etu.polytech-lille.fr
  ```

- Par défaut : login Postgres = login Unix (sauf pour login avec '-'), password ("postgres")
- Modifier le mot de passe (sous psql) :

  ```
  ALTER USER loginPostgres WITH password 'motdepasse'
  ```

Les commandes de base POSTGRES

- Commandes sous unix

Les commandes de base POSTGRES

- Commandes sous unix
- Création d'une base ([] facultatif) :
  `createdb  nomBase [ -U comptePostgres ]`

Les commandes de base POSTGRES

- Commandes sous unix
- Création d'une base ([] facultatif) :
  `createdb  nomBase [ -U comptePostgres ]`
- Destruction d'une base :
  `dropdb nomBase [ -U comptePostgres ]`

Les commandes de base POSTGRES

- Commandes sous unix
- Création d'une base ([] facultatif) :
  `createdb  nomBase [ -U comptePostgres ]`
- Destruction d'une base :
  `dropdb nomBase [ -U comptePostgres ]`
- Accès à une base :
  `psql nomBase [ -U comptePostgres ]`

Environnement psql

- Interface texte :-) ou :-(

Environnement psql

- Interface texte :-) ou :-(
- Reconnaît toutes les requêtes SQL !

Environnement psql

- Interface texte :-) ou :-(
- Reconnaît toutes les requêtes SQL !
- Et bien plus encore (aide en ligne,. . .)

Création de tables

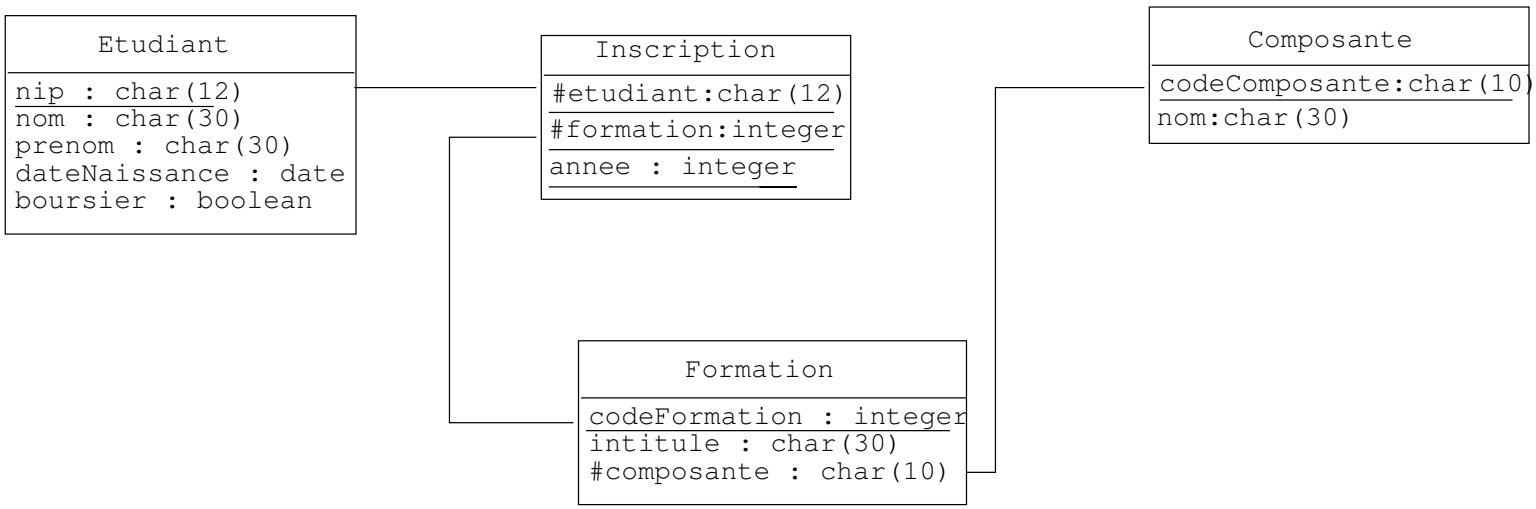
- Syntaxe POSTGRES (très très simplifiée !)
  ([] : 0 ou 1 occurrence, {} 0 ou plusieurs occurrences, | :alternative) :

```
CREATE TABLE table_name (
    { column_name type [ column_constraint [ ... ] ]
      | table_constraint } )
column_constraint ::
[ CONSTRAINT constraint_name ]
  PRIMARY KEY | REFERENCES table_name [ ( column [, ... ] ) ]
table_constraint::
[ CONSTRAINT constraint_name ]
  PRIMARY KEY ( column_name [, ... ] ) |
  FOREIGN KEY ( column_name [, ... ] )
     REFERENCES table_name [ ( column [, ... ] ) ]
```

Exemple schéma relationnel (1/2)

- Soit le schéma relationnel suivant :
  Etudiant(nip:char(12), nom:char(30), prenom:char(30), dateNaissance:date, boursier:boolean)
  Formation(codeFormation:integer,intitule:char(30), ♯composante(Composante):char(10))
  Composante(codeComposante:char(10),nom:char(30))
  Inscription(♯etudiant(Etudiant):char(12), ♯formation(Formation):integer,annee:integer)

Exemple schéma relationnel (2/2)



- Attention : schéma conceptuel **différent** de schéma relationnel

Représentation SQL d'un schéma relationnel (1/3)

```
-- alternative 1 : un nom pour la contrainte
CREATE TABLE Etudiant (
   nip CHAR(12),
   nom CHAR(30), prenom CHAR(30),
   dateNaissance DATE, boursier BOOLEAN,
   CONSTRAINT pkey_etudiant PRIMARY KEY (nip)
) ;
-- alternative 2 : pas de nom pour la contrainte
CREATE TABLE Etudiant (
   nip CHAR(12),
   nom CHAR(30), prenom CHAR(30),
   dateNaissance DATE, boursier BOOLEAN,
   PRIMARY KEY (nip)
) ;
```

Représentation SQL d'un schéma relationnel (2/3)

```
-- alternative 3 : contrainte au niveau de la colonne
CREATE TABLE Etudiant (
   nip CHAR(12) PRIMARY KEY,
   nom CHAR(30), prenom CHAR(30),
   dateNaissance DATE, boursier BOOLEAN
) ;

-- clé primaire définie au niveau table:
CREATE TABLE Composante (
   codeComposante CHAR(10),
   nom CHAR(30),
   CONSTRAINT pkey_composante PRIMARY KEY (codeComposante)
) ;
```

Représentation SQL d'un schéma relationnel (3/3)

```
-- clés primaire et étrangère définies au niveau colonne
CREATE TABLE Formation (
   codeFormation INTEGER PRIMARY KEY,
   intitule CHAR(30),
   composante CHAR(10) REFERENCES Composante(codeComposante),
) ;

-- clés définies au niveau table avec nommage des contraintes
CREATE TABLE Inscription (
   etudiant CHAR(12) ,
   formation INTEGER, annee INTEGER,
   CONSTRAINT pkey_inscription PRIMARY KEY (etudiant, formation,
   CONSTRAINT fkey_refEtudiant
     FOREIGN KEY (etudiant) REFERENCES Etudiant(nip),
   CONSTRAINT fkey_refFormation FOREIGN KEY (formation)
     REFERENCES Formation(codeFormation)
) ;
```

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)
- Mots-clefs en majuscules ou minuscules.

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)
- Mots-clefs en majuscules ou minuscules.
- `primary key` : valeur unique et non nulle au sein de la table

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)
- Mots-clefs en majuscules ou minuscules.
- `primary key` : valeur unique et non nulle au sein de la table
- Possibilité de définir la clé au niveau de l'attribut ou au niveau de la table (obligatoire si clé composée)

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)
- Mots-clefs en majuscules ou minuscules.

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)
- Mots-clefs en majuscules ou minuscules.
- `primary key` : valeur unique et non nulle au sein de la table

Schéma relationnel en SQL

- Possibilité de nommer ou pas les contraintes (maintenance)
- Mots-clefs en majuscules ou minuscules.
- `primary key` : valeur unique et non nulle au sein de la table
- Possibilité de définir la clé au niveau de l'attribut ou au niveau de la table (obligatoire si clé composée)

Suppression de tables, d'index

- A chaque commande `CREATE`, une commande `DROP`

Suppression de tables, d'index

- A chaque commande `CREATE`, une commande `DROP`
- Syntaxes :

```
DROP TABLE name {, name}
DROP INDEX index_name [, ...]
```

Insertion de lignes

- Parfois considéré comme une commande purement administrateur

Insertion de lignes

- Parfois considéré comme une commande purement administrateur
- Syntaxe (simplifiée) :

```
INSERT INTO table [ ( column {, ...} ) ]
    VALUES  ( expression {, ...} ) {, ( ...) } | SELECT query
```

Exemples insertion

- On insère une ligne complète :

  ```
  INSERT INTO composante VALUES ('EPU', 'Polytech') ;
  ```

  Respecter l'ordre des colonnes définies à la création (et le type)

Exemples insertion

- On insère une ligne complète :

  ```
  INSERT INTO composante VALUES ('EPU', 'Polytech') ;
  ```

  Respecter l'ordre des colonnes définies à la création (et le type)
- On insère plusieurs lignes :

  ```
  INSERT INTO composante(codeComposante,nom) VALUES
  ('IUT', 'Institut Universitaire'),
  ('FST', 'Faculté des Sciences et Technologies') ;
  ```

Exemples insertion

- On insère une ligne complète :

```
INSERT INTO composante VALUES ('EPU', 'Polytech') ;
```

  Respecter l'ordre des colonnes définies à la création (et le type)

- On insère plusieurs lignes :

```
INSERT INTO composante(codeComposante,nom) VALUES
('IUT', 'Institut Universitaire'),
('FST', 'Faculté des Sciences et Technologies') ;
```

- On insère une ligne complète en spécifiant les colonnes :

```
insert into etudiant (nip,nom,prenom,dateNaissance,boursier)
values ('cr0123dz', 'dupont', 'paul' , '27/02/1991', true) ;
```

Exemples insertion

- On insère une ligne complète :
  ```
  INSERT INTO composante VALUES ('EPU', 'Polytech') ;
  ```
  Respecter l'ordre des colonnes définies à la création (et le type)
- On insère plusieurs lignes :
  ```
  INSERT INTO composante(codeComposante,nom) VALUES
  ('IUT', 'Institut Universitaire'),
  ('FST', 'Faculté des Sciences et Technologies') ;
  ```
- On insère une ligne complète en spécifiant les colonnes :
  ```
  insert into etudiant (nip,nom,prenom,dateNaissance,boursier)
  values ('cr0123dz', 'dupont', 'paul' , '27/02/1991', true) ;
  ```
- On insère une ligne incomplète :
  ```
  insert into etudiant (nip,nom,prenom) values
  ('cr0123dz', 'dupont', 'paul') ;
  ```

Exemples insertion

- On insère une ligne complète :
  ```
  INSERT INTO composante VALUES ('EPU', 'Polytech') ;
  ```
  Respecter l'ordre des colonnes définies à la création (et le type)
- On insère plusieurs lignes :
  ```
  INSERT INTO composante(codeComposante,nom) VALUES
  ('IUT', 'Institut Universitaire'),
  ('FST', 'Faculté des Sciences et Technologies') ;
  ```
- On insère une ligne complète en spécifiant les colonnes :
  ```
  insert into etudiant (nip,nom,prenom,dateNaissance,boursier)
  values ('cr0123dz', 'dupont', 'paul' , '27/02/1991', true) ;
  ```
- On insère une ligne incomplète :
  ```
  insert into etudiant (nip,nom,prenom) values
  ('cr0123dz', 'dupont', 'paul') ;
  ```
- Valeurs nulles (`NULL`) ou valeurs par défaut

Valeurs de clés entière générée automatiquement

- Aucune colonne ne peut jouer le rôle de clé.

Valeurs de clés entière générée automatiquement

- Aucune colonne ne peut jouer le rôle de clé.
- On définit alors une colonne de type **serial** et on laisse le soin au SGBD de générer des valeurs lors de la création.

Valeurs de clés entière générée automatiquement

- Aucune colonne ne peut jouer le rôle de clé.
- On définit alors une colonne de type **serial** et on laisse le soin au SGBD de générer des valeurs lors de la création.
- Exemple :

```
create table article (
  id_article serial primary key,
  nom varchar(100), prix float
) ;

insert into article(nom, prix) values ('pull vert',23.0);
insert into article(nom, prix) values ('tshirt rouge',7.5);

select * from article;
 id_article | nom           | prix
 1          | pull vert     | 23.0
 2          | tshirt rouge  | 7.5
```

Suppression de lignes

- Syntaxe : `DELETE FROM nom_table [ WHERE condition ]`

Suppression de lignes

- Syntaxe : `DELETE FROM nom_table [ WHERE condition ]`
- Exemples :

```
DELETE FROM composante ;
DELETE FROM Etudiant where nip='cr0245zs' ;
```

Modification de lignes

- Syntaxe :
  ```
  UPDATE nom_table SET col = expression {, col = expression}
  [ WHERE condition ]
  ```

Modification de lignes

- Syntaxe :
  `UPDATE nom_table SET col = expression {, col = expression}`
  `[ WHERE condition ]`
- Exemples :

Modification de lignes

- Syntaxe :
  `UPDATE nom_table SET col = expression {, col = expression}`
  `[ WHERE condition ]`
- Exemples :
  - Modifier une colonne, pour une ligne :
    `UPDATE Composante SET nom='Polytech Lille'`
    `WHERE codeComposante='EPU'`

Modification de lignes

- Syntaxe :
  `UPDATE nom_table SET col = expression {, col = expression}`
  `[ WHERE condition ]`
- Exemples :
  - Modifier une colonne, pour une ligne :
    `UPDATE Composante SET nom='Polytech Lille'`
    `WHERE codeComposante='EPU'`
  - Modifier plusieurs colonnes pour une ligne :
    `UPDATE Etudiant SET dateNaissance='27/02/1991',`
    `boursier=true where nip='cr0123dz'`

Modification de lignes

- Syntaxe :
  `UPDATE nom_table SET col = expression {, col = expression}`
  `[ WHERE condition ]`
- Exemples :
  - Modifier une colonne, pour une ligne :
    ```
    UPDATE Composante SET nom='Polytech Lille'
    WHERE codeComposante='EPU'
    ```
  - Modifier plusieurs colonnes pour une ligne :
    ```
    UPDATE Etudiant SET dateNaissance='27/02/1991',
    boursier=true where nip='cr0123dz'
    ```
  - Modifier toutes les lignes :
    ```
    UPDATE personnel SET salaire=salaire+0.10*salaire
    ```

Modifier une table

- De plus en plus de possibilités au cours des versions (ex : supprimer une colonne)

Modifier une table

- De plus en plus de possibilités au cours des versions (ex : supprimer une colonne)
- Syntaxes :

```
ALTER TABLE nom_table ADD [ COLUMN ] nom_colonne type

ALTER TABLE nom_table RENAME [ COLUMN ] nom_colonne
              TO nouveau_nom
ALTER TABLE nom_table RENAME TO nouveau_nom_table

ALTER TABLE nom_table DROP [ COLUMN] nom_colonne
ALTER TABLE nom_table OWNER to nouveau_proprietaire
...
```