2021-2022

Université Clermont Auvergne
IUT Département Informatique
1ère année

Introduction aux Bases de Données

Pascale Brigoulet, Raphaël Delage, Anaïs Durand,
Franck Glaziou, Johanna Millet

Plan du cours

1. Conception et modélisation des données

Dans ce chapitre, nous abordons les premières étapes indispensables lors de la mise en place d'une base de données : la conception et la modélisation des données.

1.1. Méthode Merise et modèles de données

Concevoir une base de données est une tâche complexe qui requière de penser globalement l'ensemble du système et de se poser des questions sur la structure à mettre en place pour stocker les données de façon efficace. Il est donc important de s'appuyer sur une **méthode** de conception et de déploiement d'un système d'information.

La méthode **Merise**, inventée dans les années 70, permet d'analyser les besoins d'une organisation à partir d'un **cahier des charges** exprimé en langage naturel (en français par exemple) et de définir la structure des données à utiliser.

Cette méthode est composée de trois niveaux d'analyse, qui distinguent l'analyse des données (les informations à sauvegarder) et l'analyse des traitements (comment et pourquoi les données sont manipulées) :

Niveau / Analyse	**Données**	**Traitements**
Conceptuel	Quelles informations sont manipulées ?	Pourquoi ?
Logique	Comment les structurer ?	Qui fait quoi, où et quand ?
Physique	Où les stocker ?	Comment les stocker ?

La méthode Merise s'appuie sur quatres modèles :

— Modèle Conceptuel de Données (MCD)
— Modèle Conceptuel des Traitements (MCT)
— Modèle Logique de Données (MLD)
— Modèle Organisationnel des Traitements (MOT)

Dans ce cours, nous nous focaliserons sur les deux modèles les plus importants : le MCD et le MLD. Le MCD permet de répondre à la question "Quelles informations sont manipulées ?". Le MLD permet de répondre à la question "Comment les structurer ?".

Il est important de passer du temps sur cette phase de conception de la base de données, car les choix faits vont avoir un impact fort sur l'utilisation de la base. Une mauvaise conception peut par exemple mener à des incohérences.

1.2. Modèle Conceptuel de Données (MCD)

Le **Modèle Conceptuel de Données (MCD)** est aussi appelé **schéma Entités/Associations (E/A)**. Son objectif est de recenser et d'organiser les données manipulées dans le système d'information.

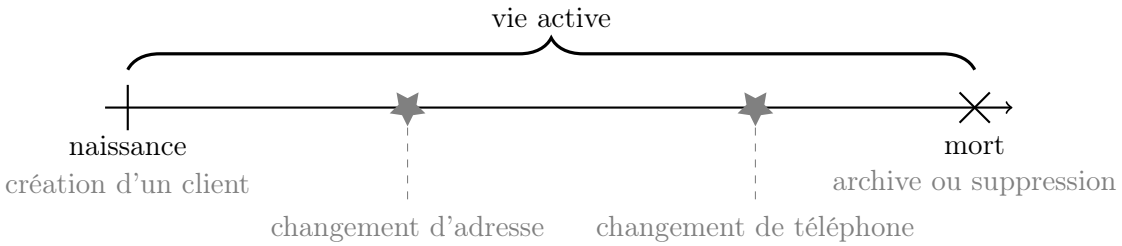
1.2.1. Entités

Une **entité** représente un élément du système d'information qui est un regroupement logique de données (par exemple, une personne, un produit, une commande, ...). C'est un objet concret ou abstrait qui est **identifié** de manière unique.

Une **classe d'entités** ou un **type d'entités** regroupe toutes les entités de même nature, c'est-à-dire un ensemble d'entités qui possèdent des propriétés communes. Par abus de langage, on parle souvent d'entité au lieu de classe d'entité car, à ce niveau d'abstraction, l'entité elle-même ne nous intéresse pas.

Exemple 1.1 : Jean Petit et Julie Dubois sont des entités appartenant à la classe d'entités Client.

▶ **Cycle de vie d'une entité.** Chaque entité à un cycle de vie marqué par différents événements représentés ci-dessous :



▶ **Attributs, propriétés, rubriques.** Les **attributs**, **propriétés** ou **rubriques** sont des données élémentaires qui décrivent les entités. Toutes les entités d'une même classe présentent les même caractéristiques. Chaque attribut a un **domaine**, l'ensemble (fini ou infini) des valeurs qu'il peut prendre.

Pour éviter la redondance et les incohérences, une information qui peut être calculée à partir des autres données n'est pas un attribut.

▶ **Identifiant, critère d'identification (CI), clé.** L'**identifiant**, le **critère d'identification** ou la **clé** est un attribut ou un ensemble d'attributs permettant d'identifier *une seule et unique* entité parmi les entités de même type. Par convention, l'identifiant est souligné pour le distinguer des autres attributs. Il s'agit souvent d'attributs codifiés.

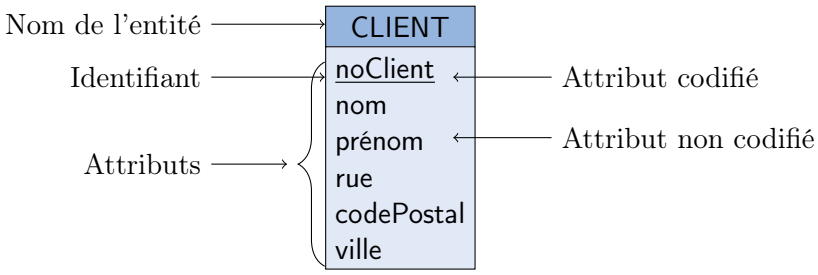
⚠ C'est une rubrique non modifiable. La valeur ne changera pas tout le long de la vie de l'entité.

Exercice 1

L'identifiant suivant est-il un bon identifiant pour le type d'entité Etudiant ?

n^o d'étudiant = n^o de département + n^o de groupe + n^o dans le groupe

Exemple 1.2 : Dans le système d'information d'une banque, la classe d'entité CLIENT représente les clients de la banque. Ils sont identifiés par un numéro unique noClient. Ils disposent tous d'un nom, d'un prénom et d'une adresse (rue, code postal, ville).

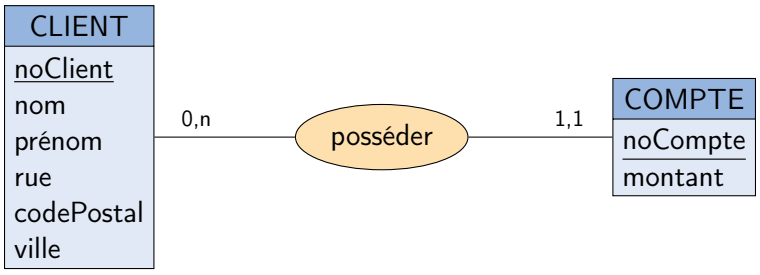


1.2.2. Associations

Une **association** représente le lien entre plusieurs entités. Une **classe d'associations** est un ensemble d'associations de même nature. Elle est nommée par un verbe (à l'infinitif), le nom de l'action représentée ou le couple d'entités associées. Une association peut relier plus de deux entités.

▶ **Cardinalités.** Dans chaque sens de lecture d'une association, il y a une **cardinalité** composée de deux entiers positifs : le nombre minimum et le nombre maximum de fois où une occurence d'une entité peut participer à une association. Si le nombre maximum exact n'est pas connu, la cardinalité est notée n.

Exemple 1.3 :



Un client peut posséder de 0 à n comptes, autrement dit un client peut n'avoir aucun compte, en avoir un seul, ou en avoir plusieurs sans limite maximum. Un compte a obligatoirement un et un seul propriétaire.

Exercice 2

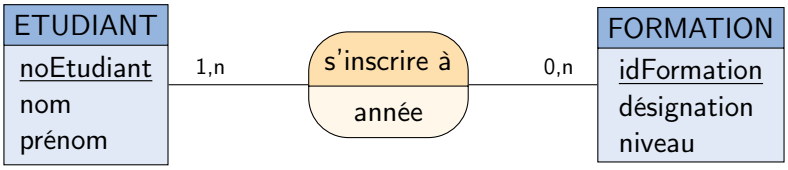
Modifier l'association posséder pour :

Q_1). accepter les compte-joints,

Q_2). enlever de la base les clients qui ne possèdent plus de compte.

▶ **Attributs et identifiants.** Implicitement, on retrouve toujours dans une association les identifiants des entités reliées. L'association peut également posséder des informations qui lui sont propres. On parle alors d'association **porteuse d'information**. Dans ce cas, l'attribut dépend de toutes les entités liées par le type d'association.

Exemple 1.4 :



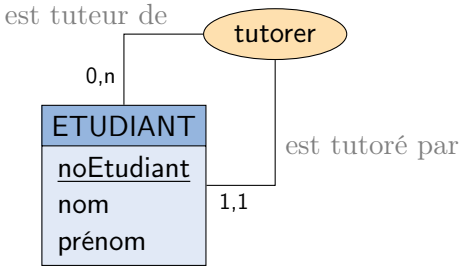
Par exemple, l'étudiant n^o*21012345* s'est inscrit en 2021 dans la formation *BUTINFO1A* correspondant à la 1ère année de BUT Informatique puis s'est inscrit en 2022 dans la formation *BUTINFO2A* correspondant à la 2ème année de BUT Informatique.

1.2.3. Associations particulières

▶ **Dépendance Fonctionnelle (DF).** Une association est une **dépendance fonctionnelle** si elle porte une cardinalité $1,1$. Elle ne peut pas avoir d'attributs propres.

▶ **Association réflexive.** Une association est **réflexive** lorsqu'elle relie deux entités de la même classe.

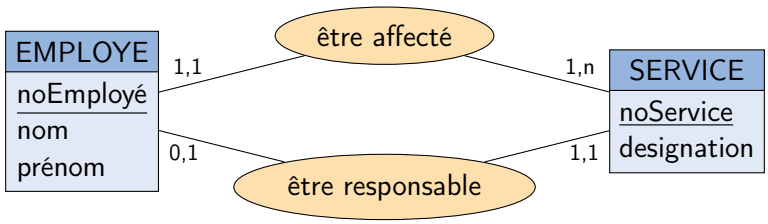
Exemple 1.5 : Un étudiant peut être le tuteur d'un autre.



▶ **Associations plurielles.** Plusieurs associations de nature différente peuvent relier les mêmes types d'entités. On parle alors d'**associations plurielles**.

Exercice 3

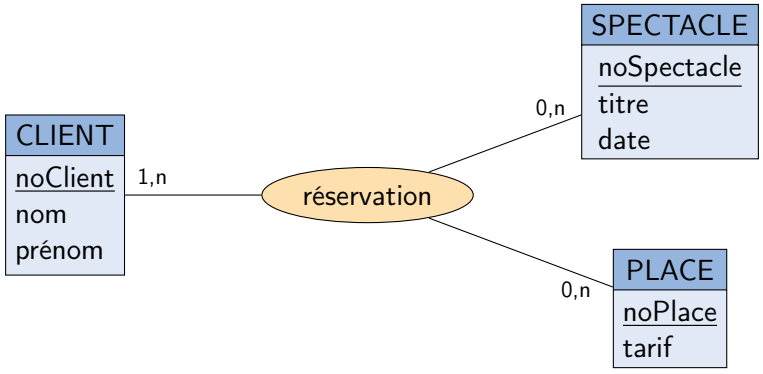
Voici un extrait du MCD modélisant le système d'information d'une entreprise. Lister les informations indiquées sur ce MCD.



▶ **Association n-aire.** Une association peut relier plus de deux entités. On parle d'association **ternaire** lorsqu'elle relie trois entités ou d'association **n-aire** de façon générale.

Exercice 4

Voici un extrait du MCD modélisant le système d'information d'un théâtre. Lister les informations indiquées sur ce MCD.



1.3. Passage au relationnel

1.3.1. *Modèle relationnel*

Le modèle relationnel est un modèle logique des données proposé par Edgar Codd en 1969. Il qcorrespond à la vision de la machine/des logiciels qui vont traiter les données et qui tient compte des contraintes de l'informatisation. Il représente l'organisation des données dans une base de données relationnelle.

Il existe différents modèles logiques (relationnel, hiérarchique, réseau, objet, *etc.*), mais le modèle relationnel reste le plus répandu actuellement.

▶ **Concepts du modèle relationnel.** Un modèle relationnel est composé de **relations** caractérisées par des **attributs** correspondant à une **table** et ses **colonnes**. Chaque **ligne** d'une table correspond à une occurence (ou tuple) de la relation. Toutes les lignes sont distinctes.

Par convention, le nom de le relation est noté en majuscules. L'identifiant d'une relation, appelé **clé primaire** est souligné. Cette clé peut être composée d'un ou plusieurs attributs.

Une relation contient parfois des **clés étrangères**. Une clé étrangère est un attribut de la relation qui fait référence à la clé primaire d'une autre relation. Il existe plusieurs notations pour les clés étrangères :

1. Mettre un # devant la clé étrangère :
   Exemple 1.6 : ETUDIANT(noEtudiant, nom, prenom, #noGroupe)

noEtudiant	nom	prenom	noGroupe
'21012345'	'Dupont'	'Julien'	'1A-G1'
'21987654'	'Petit'	'François'	'1A-G4'
'21010203'	'Clément'	'Marion'	'2A-G2'

2. Souligner en pointillé la clé étrangère :
   Exemple 1.7 : ETUDIANT(noEtudiant, nom, prenom, noGroupe)
3. Noter le détail des clés étrangères sous la relation :
   Exemple 1.8 : ETUDIANT(noEtudiant, nom, prenom, noGroupe)
   noGroupe clé étrangère référençant GROUPE

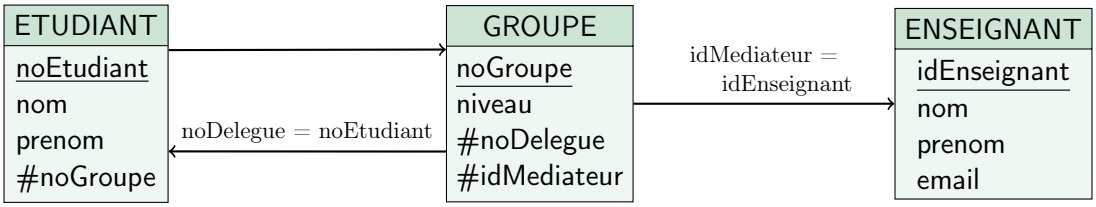
Dans ce cours, nous utiliserons la première notation (avec le #) sauf lorsque le nom de la clé étrangère ne permet pas de comprendre quelle table est référencée. Dans ce cas, nous privilégierons la troisième notation.

1.3.2. *Modèle Logique de Données (MLD)*

Le **Modèle Logique de Données (MLD)** est une représentation graphique de l'organisation des données.

Chaque table est représentée par une "boîte". Les flèches entre les tables représentent les clés étrangères. On précise sur la flèche le nom de la clé étrangère et le nom de l'attribut référencé par cette clé si nécessaire.

Exemple 1.9 :



1.3.3. Règles de traduction

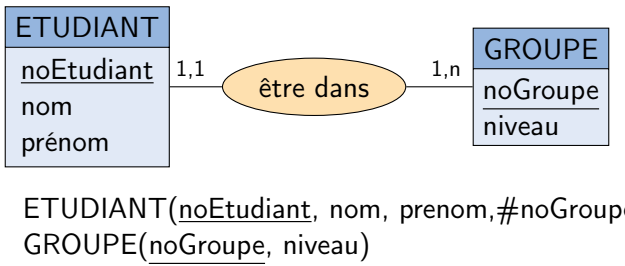
Pour passer du MCD au MLD, il faut suivre trois étapes.

▶ **Étape 1 :** Toute entité est transformée en relation dont la clé primaire est l'identifiant. Chaque propriété de l'entité devient un attribut de la relation. Les noms des attributs sont adaptés en supprimant les éventuels accents, espaces et mots inutiles mais ils doivent rester compréhensibles.

Exemple 1.10 :

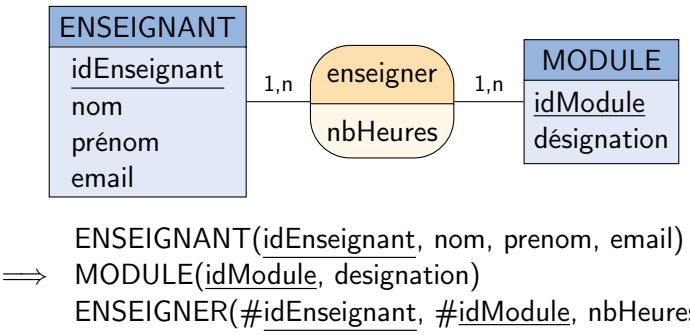
▶ **Étape 2 :** Toute association portant une cardinalité 1,1 ou 0,1 est transformée en clé étrangère. La relation représentant l'entité portant la cardinalité 1,1 ou 0,1 reçoit comme attribut supplémentaire la clé primaire de l'entité à l'autre bout de l'association.

Exemple 1.11 :



▶ **Étape 3 :** Toutes les autres associations (cardinalités 1,n, associations n-aires, associations porteuses d'information, ...) sont transformées en une nouvelle relation. La clé primaire de cette nouvelle relation est composée des identifiants des entités en relation. Si l'association est porteuse d'information, ses attributs propres deviennent des attributs de la nouvelle relation.

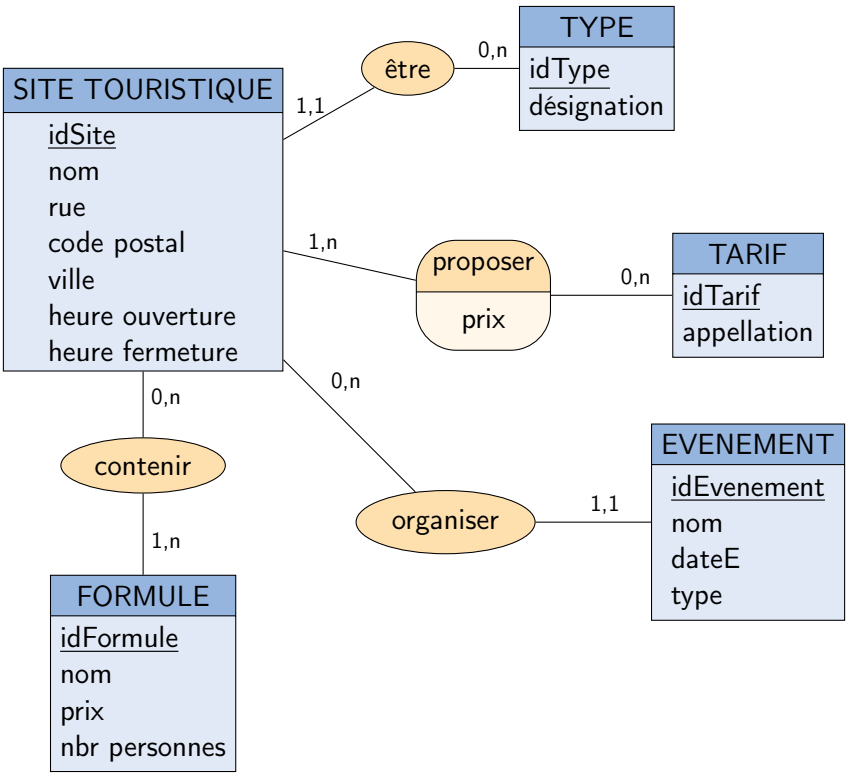
Exemple 1.12 :



⚠ Une association est traduite **une seule fois**. L'association "être dans" de l'exemple 1.1.3.3 est traduite à l'étape 2. Il est inutile (et faux !) de la traduire à nouveau lors de l'étape 3 en créant une table ETRE_DANS même si une de ses cardinalités est 1,n.

Notez que ces règles s'appliquent également pour les associations réflexives.

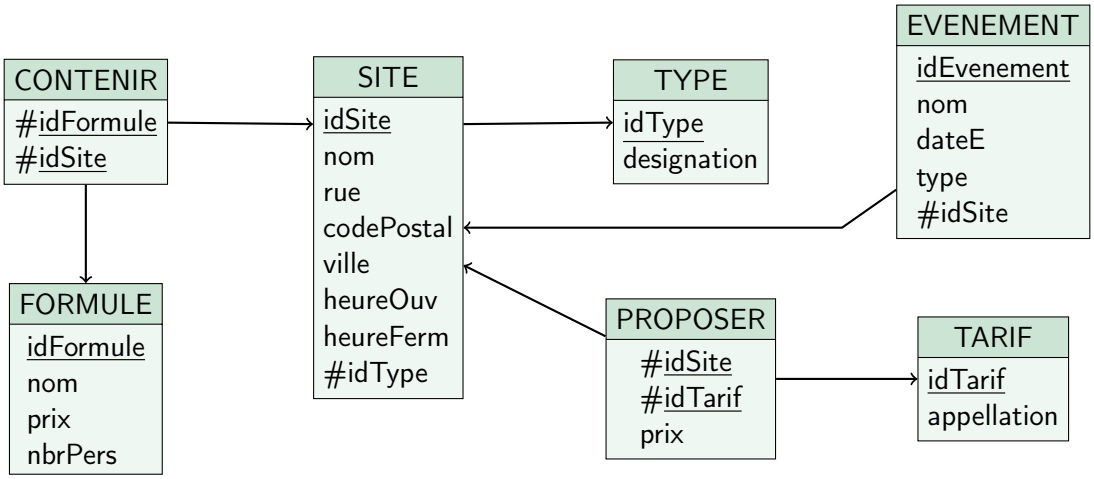
Exemple 1.13 : Le MCD suivant représente une partie de la base de données de l'office de tourisme de Clermont-Ferrand.



À partir de ce MCD, nous obtenons le modèle relationnel suivant :

SITE(idSite, nom, rue, codePostal, ville, heureOuv, heureFerm, #idType)
TYPE(idType, designation)
TARIF(idTarif, appellation)
FORMULE(idFormule, nom, prix, nbrPers)
EVENEMENT(idEvenement, nom, dateE, type, #idSite)
CONTENIR(#idFormule, #idSite)
PROPOSER(#idSite, #idTarif, prix)

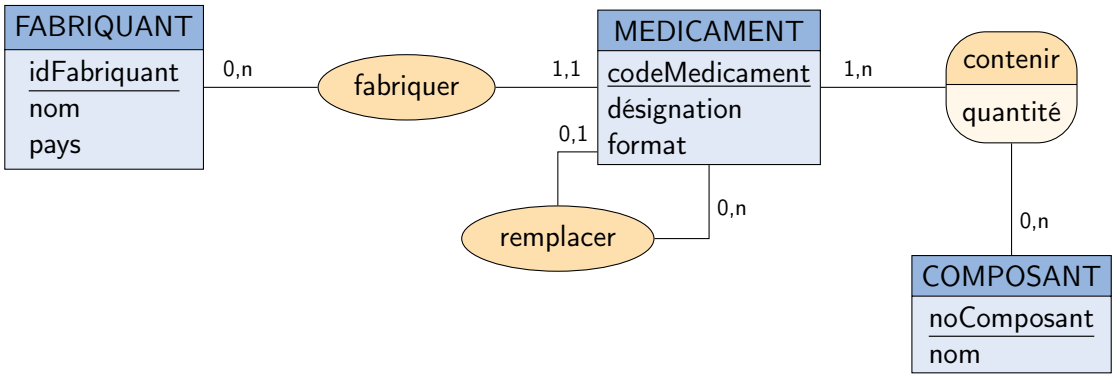
Le MLD correspondant est présenté ci-dessous :



Sauf indication contraire, ce MLD servira de base aux exemples et exercices dans la suite de ce cours.

Exercice 5

À partir du MCD suivant, donner le modèle relationnel et le MLD correspondant.



2. Introduction au SQL

SQL (Structured Query Language) est un langage normalisé (première norme en 1986, dernière norme en 2016) permettant d'exploiter des bases de données relationelles. Ce langage est reconnu et utilisé par de nombreux Systèmes de Gestion de Bases de Données (SGBD) comme par exemple Oracle, MySQL, PostgreSQL, MariaDB, SQLite, *etc.* Dans ce cours, nous utiliserons le Système de Gestion de Bases de Données (SGBD) **PostgreSQL**.

SQL est en réalité composé de plusieurs langages :

— le **Langage de Manipulation des Données (LMD)** : sélection, recherche, ajout, suppression ou modification des données en base

— le **Langage de Définition des Données (LDD)** : création, suppression ou modification des tables, autrement dit gestion de l'organisation des données

— le **Langage de Contrôle de Transaction (LCT)** : gestion des transactions (ensemble d'opérations à gérer d'un seul bloc)

— le **Langage de Contrôle des Données (LCD)** : gestion des accès aux données par les utilisateurs de la base

Nous nous focalisons dans ce chapitre sur les requêtes de bases permettant la création de tables et leur manipulation.

2.1. Création de tables

Pour pouvoir définir une nouvelle table dans une bases de données, il faut avoir le droit d'en créer. Les tables d'un utilisateur doivent porter des noms différents mais des tables d'utilisateurs différents peuvent porter le même nom.

Les colonnes d'une table doivent avoir des noms différents, mais deux tables peuvent avoir une colonne de même nom.

Exemple 2.1 :

```
CREATE TABLE Site(
    idSite      char(4),
    nom         varchar(30),
    rue         varchar(30),
    codePostal  char(5),
    ville       varchar(20),
    heureOuv    time,
    heureFerm   time,
    idType      char(4)
);
```

2.1.1. *Types de données*

Les principaux types de données d'une base PostgreSQL sont les suivants.

Types de données	Description
`char(n)`	Chaîne de caractères de longueur **fixe** `n`. Si la chaîne de caractère entrée est de taille inférieure à `n`, du *padding* est utilisé pour compléter la chaîne avec des espaces.
`varchar(n)`	Chaîne de caractères de longueur **variable**, au plus `n`. La chaîne de caractères entrée n'est donc pas complétée avec du padding.
`numeric(p,s)`	Nombre entier ou décimal. `p` correspond au nombre total de chiffres et `s` correspond au nombre de chiffres après la virgule. Il est possible d'utiliser `numeric` sans `s` ni `p`. `numeric(p)` correspond à un entier sur `p` chiffres.
`date`	Date
`time`	Heure
`timestamp`	Date et heure
`boolean`	Booléen `true` ou `false`

2.1.2. *Clé primaire*

Le mot-clé `PRIMARY KEY` permet de définir la **clé primaire** d'une table. Il ne peut y avoir qu'une seule clé primaire par table. La valeur d'une clé primaire doit toujours être définie (pas de valeur `NULL`) et être unique.

Il est possible de définir la clé primaire de deux façons.

▶ Contrainte de colonne

Exemple 2.2 :

```
CREATE TABLE Tarif(
    idTarif     char(4) PRIMARY KEY,
    designation varchar(30)
);
```

Une erreur est signalée lors de l'insertion d'un `idTarif` déjà présent dans la table ou si l'`idTarif` est indéfini.

▶ Contrainte de table

Exemple 2.3 :

```
CREATE TABLE Proposer(
    idSite  char(4),
    idTarif char(4),
    prix    numeric(4,2),
    PRIMARY KEY(idSite, idTarif)
);
```

Lorsque la clé primaire est composée de plusieurs attributs, il obligatoire d'utiliser cette écriture. En effet, il n'est pas possible de définir plusieurs `PRIMARY KEY`.

2.1.3. *Clé étrangère*

Le mot-clé `REFERENCES` permet de définir une **clé étrangère**. Les valeurs de la clé étrangère doivent être aussi présentes dans une autre colonne ou ensemble de colonnes. La clé référencée doit être définie

comme `PRIMARY KEY` ou `UNIQUE`. Cependant, plusieurs lignes d'une table peuvent référencer la même clé étrangère.

▶ Contrainte de colonne

Exemple 2.4 :

```
CREATE TABLE Proposer(
    idSite  char(4) REFERENCES Site,
    idTarif char(4) REFERENCES Tarif(idTarif),
    prix    numeric(4,2),
    PRIMARY KEY(idSite, idTarif)
);
```

▶ Contrainte de table

Exemple 2.5 :

```
CREATE TABLE Proposer(
    idSite  char(4),
    idTarif char(4),
    prix    numeric(4,2),
    PRIMARY KEY(idSite, idTarif),
    FOREIGN KEY (idSite) REFERENCES Site(idSite),
    FOREIGN KEY (idTarif) REFERENCES Tarif;
);
```

▶ **Ordre des opérations** Lorsqu'il y a des contraintes référentielles, autrement dit des clés étrangères, l'ordre des opérations sur les tables doit suivre un ordre précis tenant compte de ces références. Ici, une erreur est levée lors de manipulations des tables si :

— On veut supprimer la table `Site` ou `Tarif` alors que la table `Proposer` y fait référence. Il faut d'abord supprimer la table `Proposer`.

— On veut ajouter ou modifier une ligne dans `Proposer` en référençant un `idSite` ou un `idTarif` qui n'existe pas dans les tables correspondantes. Il faut d'abord insérer une nouvelle ligne dans la table `Site` ou `Tarif` avant de modifier la table `Proposer`.

— On veut supprimer une ligne dans `Site` ou `Tarif` alors que des lignes de `Proposer` y font référence. Il faut d'abord supprimer les lignes en question dans `Proposer`.

Exercice 6

En supposant que la table `Site` est déjà créée, écrire les requêtes permettant de créer les tables `Contenir` et `Formule`.

Types de données :

`idSite` :	4 caractères	`prix` :	nombre sur 5 chiffres dont 2 décimales
`idFormule` :	4 caractères	`nbrPers` :	entier sur 2 chiffres
`nom` :	$\leq$ 30 caractères		

2.2. Opérations utiles sur les tables

— Affichage de la liste des tables définies : `\d;`

— Affichage de la définition d'une table : `\d site;`

— Affichage du contenu d'une table : `SELECT * FROM site;`

— Suppression d'une table : `DROP TABLE site;`

2.3. Insertion de lignes

Exemple 2.6 :

```
INSERT INTO Site(idSite, codePostal, ville) VALUES ('S001', '63000',
    'Clermont-Ferrand');
```

Les chaînes de caractères et les dates sont données entre apostrophes ' '. Les valeurs doivent être données dans le même ordre que la liste de colonnes indiquée. La liste des colonnes est facultative si les valeurs de **toutes** les colonnes sont précisées. Dans ce cas, les valeurs doivent être données dans l'ordre de définition des colonnes lors de la création de la table. Lorsque la valeur d'une colonne n'est pas précisée, elle prend la valeur indéfinie `NULL`.

⚠ `NULL` ≠ `''` et `NULL` ≠ `0`

Exemple 2.7 :

```
INSERT INTO Site VALUES ('S001', 'Musée Henri Lecoq', NULL, '63000',
    'Clermont-Ferrand', NULL, NULL, 'T001');
```

2.4. Contraintes

Les contraintes permettent de préciser les valeurs possibles pour un attribut. Il y a deux types de contraintes :

— Une **contrainte de colonne** est définie dans la description de la colonne et ne concerne que cette colonne.

— Une **contrainte de table** concerne plusieurs colonnes.

Une colonne peut avoir plusieurs contraintes.

Notez que `PRIMARY KEY` et `REFERENCES` sont des contraintes.

2.4.1. *Clause DEFAULT*

La clause `DEFAULT` permet d'indiquer la valeur par défaut à mettre dans la colonne si une ligne est insérée dans la table sans valeur donnée pour cette colonne.

Exemple 2.8 :

```
CREATE TABLE Site(
    idSite      char(4) PRIMARY KEY,
    nom         varchar(30),
    rue         varchar(30),
    codePostal  char(5) DEFAULT '63000',
    ville       varchar(20),
    heureOuv    time,
    heureFerm   time,
    idType      char(4) REFERENCES Type
);
```

a).

```
INSERT INTO Site VALUES ('S001', 'Musée Henri Lecoq', NULL, NULL,
    'Clermont-Ferrand', NULL, NULL, 'T001');
```

Les colonnes `rue`, `codePostal`, `heureOuv` et `heureFerm` prennent la valeur indéfinie `NULL`.

b).

```
INSERT INTO Site(idSite, nom, ville, idType)
VALUES ('S001', 'Musée Henri Lecoq', 'Clermont-Ferrand', 'T001');
```

La colonne `codePostal` prend la valeur par défaut `'63000'`, alors que les colonnes `rue`, `heureOuv` et `heureFerm` prennent la valeur indéfinie `NULL`.

2.4.2. *Contrainte NOT NULL*

La contrainte `NOT NULL` sur une colonne interdit que la valeur de cette colonne soit non définie, autrement dit interdit la valeur `NULL`.

Exemple 2.9 :

```
CREATE TABLE Tarif(
    idTarif     char(4) PRIMARY KEY,
    designation varchar(30) NOT NULL
);
```

2.4.3. Contrainte UNIQUE

Les valeurs d'une colonne ou d'un ensemble de colonnes définies comme **UNIQUE** doivent être uniques. Mais, contrairement à la contrainte **PRIMARY KEY**, la valeur **NULL** est autorisée et plusieurs lignes peuvent avoir une valeur **NULL** dans la colonne définie comme **UNIQUE**.

▶ Contrainte de colonne

Exemple 2.10 :

```
CREATE TABLE Tarif(
    idTarif     char(4) PRIMARY KEY,
    designation varchar(30) UNIQUE
);
```

▶ Contrainte de table

Exemple 2.11 :

```
CREATE TABLE Site(
    idSite     char(4) PRIMARY KEY,
    nom        varchar(30),
    rue        varchar(30),
    codePostal char(5) DEFAULT '63000',
    ville      varchar(20),
    heureOuv   time,
    heureFerm  time,
    idType     char(4) REFERENCES Type,
    UNIQUE(rue, codePostal, ville)
);
```

2.4.4. Contrainte CHECK

Les contraintes **CHECK** permettent de vérifier que les valeurs d'une colonne ou d'un ensemble de colonnes respectent une condition.

▶ Contrainte de colonne

Exemple 2.12 :

```
CREATE TABLE Proposer(
    idSite  char(4) REFERENCES Site,
    idTarif char(4) REFERENCES Tarif(idTarif),
    prix    numeric(4,2) CHECK (prix >= 0),
    PRIMARY KEY(idSite, idTarif)
);
```

▶ Contrainte de table

Exemple 2.13 :

```
CREATE TABLE Site(
    idSite     char(4) PRIMARY KEY,
    nom        varchar(30),
    rue        varchar(30),
    codePostal char(5) DEFAULT '63000',
    ville      varchar(20),
    heureOuv   time,
```

```
    heureFerm  time,
    idType     char(4) REFERENCES Type,
    UNIQUE(rue, codePostal, ville),
    CHECK(heureOuv < heureFerm)
);
```

2.4.5. *Conditions*

Les conditions peuvent utiliser les opérateurs suivants.

▶ **Opérateurs logiques :** `NOT`, `AND`, `OR`

▶ **Opérateurs de comparaison :** =, !=, <>, >, <, >=, <=

Permettent de comparer :

— des valeurs numériques *Exemple :* `prix > 0`

— des chaînes de caractères *Exemple :* `codePostal = '63000'`

— des dates *Exemple :* `dateE > '09-JUL-2021'`

— ...

▶ **Liste de valeurs :** `expression [NOT] IN (liste_valeurs)`

Exemple 2.14 :

```
CREATE TABLE Evenement(
    idEvenement char(4) PRIMARY KEY,
    nom         varchar(30),
    dateE       date,
    type        char(4),
    idSite      char(4) REFERENCES Site,
    CHECK(type IN ('ANIM', 'CONC', 'EXPO'))
);
```

▶ **Colonnes non renseignées :** `colonne IS [NOT] NULL`

Exemple 2.15 :

```
CREATE TABLE Site(
    idSite     char(4) PRIMARY KEY,
    nom        varchar(30),
    rue        varchar(30),
    codePostal char(5) DEFAULT '63000',
    ville      varchar(20),
    heureOuv   time,
    heureFerm  time,
    idType     char(4) REFERENCES Type,
    UNIQUE(rue, codePostal, ville),
    CHECK(heureOuv IS NULL AND heureFerm IS NULL
        OR heureOuv IS NOT NULL AND heureFerm IS NOT NULL)
);
```

⚠ La condition `heureOuv = NULL` ne vaut ni vrai, ni faux, mais vaut une valeur inconnue. PostgreSQL considère que c'est toujours faux même si l'heure d'ouverture est vraiment `NULL`.

De même, les conditions `heureOuv = '10:00'` et `heureOuv != '10:00'` valent toutes les deux faux si l'heure d'ouverture est `NULL`.

▶ **Format de chaînes de caractères :** `colonne [NOT] LIKE 'format'`

Permet de comparer la valeur d'une colonne à un format. On peut utiliser des caractères génériques :

— `'%'` correspond à plusieurs caractères (0, 1, ou plus)

— `'_'` correspond à exactement un caractère

`'\%'` correspond au caractère % et n'est pas un caractère générique.

Exemple 2.16 :

```
CREATE TABLE Tarif(
    idTarif     char(4) PRIMARY KEY,
    designation varchar(30),
    CHECK (idTarif LIKE 'T\%')
);
```

2.4.6. *Nommage des contraintes*

Il est possible de nommer les contraintes (sauf les contraintes `NOT NULL`) sachant que deux contraintes d'une base ne peuvent pas avoir le même nom. Si une contrainte n'a pas été nommée, le SGBD lui attribut automatiquement un nom.

Exemple 2.17 :

```
CREATE TABLE Tarif(
    idTarif     char(4) CONSTRAINT pk_Tarif PRIMARY KEY,
    designation varchar(30),
    CONSTRAINT ck_Tarif CHECK (idTarif LIKE 'T\%')
);
```

Exercice 7

Définir la table `Formule` en s'assurant que :

— le `prix` et le `nbrPers` soient strictement positif,

— l'`idFormule` commence toujours par un `'F'`,

— le nom soit toujours renseigné et qu'il n'y ait pas de doublons,

— la valeur par défaut de `nbrPers` soit 1.

Exercice 8

Définir la table PERSONNE(noPers, nom, âge, catégorie) en respectant les contraintes suivantes :

- toutes les rubriques sont obligatoires,
- la catégorie est `'N'` (tarif normal) ou `'R'` (tarif réduit),
- il doit y avoir une cohérence entre l'âge et la catégorie (tarif réduit accordé uniquement aux mineurs et aux personnes de plus de 60 ans).

Exercice 9

Définir la table RESULTAT(noEtudiant, note, mention) en respectant les contraintes suivantes :

- toutes les rubriques sont obligatoires,
- la mention est `'P'` (présent), `'R'` (retard) ou `'A'` (absent),
- la note est comprise entre 0 et 20 et vaut 0 quand l'étudiant est absent.

2.5. Fichier de commandes SQL

Pour sauvegarder un ensemble de commandes à exécuter, il est possible de les placer dans un fichier. Pour exécuter un fichier nommé `fich.sql`, il faut exécuter la commande :

```
\i chemin_jusquau/fichier/fich.sql;
```

Exemple 2.18 :

```
/* commentaires
   sur plusieurs
   lignes */

DROP TABLE ...
CREATE TABLE ...

-- commentaire sur une ligne

INSERT ...
SELECT ...

...
```

3. Requêtes SQL simples

Pour consulter le contenu des tables, il faut utiliser une requête **SELECT**. Dans ce chapitre, nous abordons les requêtes de sélection simple, utilisant une seule table.

Syntaxe globale :

```
SELECT [ALL | DISTINCT] liste_de_sélection
FROM table
[WHERE condition]
[ORDER BY {expression | position} [ASC | DESC] [, ...]];
```

3.1. Liste de sélection

La liste de sélection permet de filtrer les colonnes et valeurs à afficher. Elle peut contenir :

- `*` : toutes les colonnes de la table listée dans le **FROM**
- `Table.*` : toutes les colonnes de la table `Table`
- des noms de colonnes
- des expressions sur les colonnes
- des constantes

Une table est désignée par son nom ou `nom_créateur.nom_table`. Une colonne est désignée par son nom ou `nom_table.nom_colonne`. Il est possible d'associer un alias à une colonne ou une expression qui s'affichera en tête de colonne.

Exemple 3.1 :

```
SELECT *
FROM Formule;
```

idFormule	nom	prix	nbrPers
'F001'	'Essentiel'	50	1
'F002'	'Essentiel Duo'	90	2
'F003'	'Essentiel Tribu'	150	4
'F004'	'Découverte'	90	1

```
SELECT nom, prix Montant
FROM Formule;
```

nom	Montant
'Essentiel'	50
'Essentiel Duo'	90
'Essentiel Tribu'	150
'Découverte'	80

3.2. Clause DISTINCT

La clause **DISTINCT** permet de supprimer les doublons parmi les lignes sélectionnées.

Exercice 10

Supposons que la requête `SELECT * FROM Formule;` donne le résultat suivant :

Formule

idFormule	nom	prix	nbrPers
'F001'	'Essentiel'	50	1
'F002'	'Essentiel Duo'	90	2
'F003'	'Essentiel Tribu'	150	4
'F004'	'Découverte'	90	1

Donner le résultat des requêtes suivantes :

Q_1). `SELECT nbrPers FROM Formule;`

Q_2). `SELECT DISTINCT nbrPers FROM Formule;`

Q_3). `SELECT DISTINCT nbrPers, prix FROM Formule;`

3.3. Conditions

Il est possible de filtrer les lignes affichées en fonction d'une condition exprimée dans le **WHERE**. Comme pour les conditions des contraintes **CHECK**, les conditions du **WHERE** peuvent utiliser des opérateurs logiques, des opérateurs de comparaison, tester si des colonnes sont renseignées ou non, tester des formats de chaînes de caractères ...

Exercice 11

Q_1). Lister les formules dont le prix est inférieur à 100€.

Q_2). Lister les autres formules.

Q_3). Lister les formules pour une personne ou plus de 3 personnes.

Exercice 12

Q_1). Lister les sites touristiques dont le nom contient `'parc'`.

Q_2). Lister les sites touristiques dans le Puy de Dôme (63).

Q_3). Lister les sites dont *(a)* le code postal est 63000 et qui sont de type `'TY02'` ou `'TY03'`, ou *(b)* le code postal n'est pas 63000 mais qui sont de type `'TY01'`.

3.4. ORDER BY

La clause **ORDER BY** permet de trier les lignes affichées en fonction d'une expression. Par défaut le tri est réalisé par ordre croissant (ou ordre alphabétique pour les chaînes de caractères). Il est possible de préciser l'ordre de tri : **ASC** par ordre croissant, **DESC** par ordre décroissant.

⚠ Toute rubrique utilisée pour trier les résultats doit apparaître dans la liste de sélection.

Exercice 13

Q_1). Lister les événements par ordre alphabétique.

Q_2). Lister les événements par date, les plus anciennes en premier et, à date égale, par leur nom dans l'ordre inverse de l'ordre alphabétique.

Q_3). Lister les formules par valeur par personne (prix / nbrPers) avec les prix les plus intéressants en premier.

3.5. Manipulation des chaînes de caractères

Nom	Définition
`c1 || c2`	Concaténation des chaînes `c1` et `c2` *Exemple :* `SELECT 'dra'||'gon';` ⟶ `'dragon'`
`char_length(c)`	Longueur de `c` *Exemple :* `SELECT char_length('dragon');` ⟶ `6`
`lower(c)`	Minuscules *Exemple :* `SELECT lower('DRAGON');` ⟶ `'dragon'`
`upper(c)`	Majuscules *Exemple :* `SELECT upper('Dragon');` ⟶ `'DRAGON'`
`initcap(c)`	Première lettre de chaque mot en majuscule *Exemple :* `SELECT initcap('dragon noir');` ⟶ `'Dragon Noir'`
`ltrim(c1,c2)`	Suppression des caractères de `c1` de gauche à droite appartenant à l'ensemble `c2`, tant qu'un caractère de `c1` est dans `c2` *Exemple :* `SELECT ltrim('_ _dragon_', '_ ');` ⟶ `'dragon_'`
`rtrim(c1,c2)`	Suppression des caractères de `c1` de droite à gauche appartenant à l'ensemble `c2`, tant qu'un caractère de `c1` est dans `c2` *Exemple :* `SELECT rtrim('_dragon_ _', '_ ');` ⟶ `'_dragon'`
`substr(c, i)`	Sous-chaîne de `c` à partir du `i`ème caractère *Exemple :* `SELECT substr('dragon', 3);` ⟶ `'agon'`
`substr(c, i, n)`	Sous-chaîne de `c` de longueur `n` à partir du `i`ème caractère *Exemple :* `SELECT substr('dragon', 2, 3);` ⟶ `'rag'`

Exercice 14

Q_1). Afficher toutes les formules selon le même format que l'exemple suivant : `'Formule ''DECOUVERTE'' : 90 euros'`. Le nom de la formule doit être en majuscules.

Q_2). Lister les noms des sites touristiques sans aucun espace avant ou après le nom.

Q_3). Même chose mais uniquement sur 12 caractères et avec une majuscule à chaque mot.

3.6. Manipulation des valeurs numériques

▶ **Fonctions.**

Nom	Définition
`abs(n)`	Valeur absolue de `n` *Exemple :* `SELECT abs(-2.73);` ⟶ `2.73`
`ceil(n)`	Valeur entière supérieure de `n` *Exemple :* `SELECT ceil(2.73);` ⟶ `3`
`floor(n)`	Valeur entière inférieure de `n` *Exemple :* `SELECT floor(2.73);` ⟶ `2`
`mod(n1, n2)`	Reste de la division entière `n1/n2` *Exemple :* `SELECT mod(9,4);` ⟶ `1`
`round(n, p)`	Arrondi de `n` à `p` chiffres en partie décimale *Exemple :* `SELECT round(2.76,1);` ⟶ `2.8`
`trunc(n, p)`	Tronque de `n` à `p` chiffres en partie décimale *Exemple :* `SELECT trunc(2.76,1);` ⟶ `2.7`
`sign(n)`	Signe de `n` (`-1` si négatif, `0` si nul, `1` si positif) *Exemple :* `SELECT sign(-2.76);` ⟶ `-1`

▶ **Conversion.**

Nom	Définition
`to_char(n,f)`	Converti le nombre `n` en chaîne de caractères selon le format `f` *Exemple :* `SELECT to_char(2.73,'09.999');` ⟶ `'02.730'`
`to_number(c,f)`	Converti la chaîne de caractères `c` en nombre selon le format `f` *Exemple :* `SELECT to_number('02.730','09.999');` ⟶ `2.73`

Caractères utilisables dans le format :

— `9` : chiffre à afficher, sauf si non significatif

— `0` : chiffre toujours affiché, même si non significatif

Exercice 15

Q_1). Afficher tous les tarifs arrondis à une décimale.

Q_2). Calculer une augmentation de 10% de tous les tarifs de plus de 20€.

Q_3). Afficher tous les tarifs comme dans l'exemple suivant : `'010.50'` pour 10,5€.

3.7. Manipulation des dates et heures

▶ **Opérations**

Nom	Définition
`d + nbj`	Ajoute `nbj` jours à la date `d` *Exemple :* `SELECT '2021-12-10'::date+3;` ⟶ `'2021-12-13'`
`d - nbj`	Retire `nbj` jours à la date `d` *Exemple :* `SELECT '2021-11-10'::date-3;` ⟶ `'2021-11-07'`
`d1 - d2`	Nombre de jours entre les dates `d1` et `d2` *Exemple :* `SELECT '2021-11-10'::date - '2021-11-06'::date;` ⟶ `4`
`d + t`	Timestamp correspondant à la date `d` et l'heure `t` *Exemple :* `SELECT '2021-11-10'::date + '10:39:51.2'::time;` ⟶ `'2021-11-10 10:39:51.2'`

▶ Fonctions

Nom	Définition
`CURRENT_DATE`	Date courante *Exemple :* `SELECT CURRENT_DATE;` ⟶ `'2021-07-21'`
`CURRENT_TIME`	Heure courante *Exemple :* `SELECT CURRENT_TIME;` ⟶ `'10:39:51.662522-05'`
`CURRENT_TIMESTAMP`	Date et heure courante *Exemple :* `SELECT CURRENT_TIMESTAMP;` ⟶ `'2021-07-21 10:39:51.662522-05'`
`date_trunc(p, t)`	Tronque le timestamp `t` à la précision `p` (parmi `'year'`, `'month'`, `'day'`, `'hour'`, `'minute'`, `'second'` ...) *Exemple :* `SELECT date_trunc('year', '2021-11-20'::date);` ⟶ `2021-01-01 00:00:00+01`

▶ Conversion

Nom	Définition
`make_date(y,m,d)`	Crée une date *Exemple :* `SELECT make_date(2021,7,12);` ⟶ `'2021-07-12'`
`make_time(h,mi,s)`	Crée une heure *Exemple :* `SELECT make_time(17,34,20.5);` ⟶ `'17:34:20.5'`
`make_timestamp(y, m,d,h,mi,s)`	Crée un timestamp *Exemple :* `SELECT make_timestamp(2021,7,12,17,34,20.5);` ⟶ `'2021-07-12 17:34:20.5'`
`to_char(d,f)`	Converti la date ou le timestamp `d` en chaîne de caractères selon le format `f` *Exemple :* `SELECT to_char(current_date,'DD MON YYYY');` ⟶ `'20 JUL 2021'`
`to_date(c,f)`	Converti la chaîne de caractères `c` en date selon le format `f` *Exemple :* `SELECT to_date('20 JUL 2021','DD MON YYYY');` ⟶ `'2021-07-20'`
`to_timestamp(c,f)`	Converti la chaîne `c` en timestamp selon le format `f` *Exemple :* `SELECT to_timestamp('20/07/21 13:42','DD/MM/YY HH24:MI');` ⟶ `'2021-07-20 13:42:00.0'`

Caractères utilisables dans le format :

- `HH` : heure (01-12)
- `HH24` : heure (01-24)
- `MI` : minute
- `SS` : seconde
- `YYYY` : année sur 4 chiffres
- `YY` : année sur 2 chiffres
- `MM` : numéro du mois
- `MON` : nom du mois sur 3 lettres
- `MONTH` : nom du mois
- `DD` : jour du mois
- `DAY` : nom du jour

Exercice 16

Q_1). Afficher la date dans 10 jours.

Q_2). Afficher la date de tous les événements au même format que l'exemple suivant : `'MONDAY 07 SEPTEMBER 2021'`

Q_3). Pour tous les événements à venir, afficher le nombre de jours restant avant l'événement.

4. Normalisation

La normalisation et les formes normales ont été proposées par Edgar Codd à partir de 1970. Le but est d'éviter les redondances et les pertes d'informations mais aussi de faciliter le passage à la mise en place de la base de données. La normalisation doit donc se faire au niveau conceptuel ou au niveau logique. Cela permettra d'obtenir des tables organisées, sans redondance de données et sans structures répétitives. C'est une démarche permettant d'améliorer la qualité du système d'information mis en place.

4.1. Dépendance Fonctionnelle (DF)

Les formes normales utilisent la notion de dépendance fonctionnelle. Un groupe d'attributs B est en **Dépendance Fonctionnelle (DF)** avec un groupe d'attributs A si, étant donnée une valeur de A, il n'est possible de lui associer qu'une seule valeur de B. Cette dépendance est notée $A \rightarrow B$.

Exemple 4.1 :

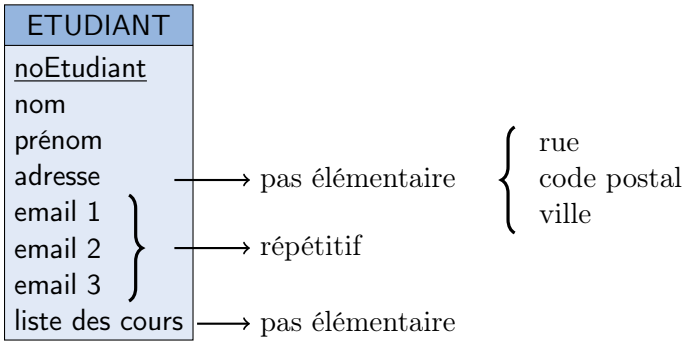
NSS → nom, prénom, dateDeNaissance
bâtiment, salle → typeSalle, nbPlaces

Notez qu'on doit toujours avoir : identifiant → autres propriétés de l'entité

4.2. Première forme normale (1NF)

Une relation/une entité est en **1NF** si et seulement si tout attribut est **élémentaire/atomique** (pas de liste ou de table, pas de valeur calculée) et si elle ne contient pas de structure répétitive.

Exemple 4.2 : La relation ETUDIANT n'est pas en 1NF.



4.3. Deuxième forme normale (2NF)

Une relation est en **2NF** si :

- elle est en 1NF
- il existe un ensemble d'attributs qui forme la clé primaire de la relation
- tout attribut ne faisant pas partie de la clé est en dépendance fonctionnelle avec l'ensemble de la clé primaire.

Exemple 4.3 :

SALLE
bâtiment
no salle
rue
type de salle
nbr places

L'entité SALLE n'est pas en 2NF car la clé primaire est bâtiment, no salle mais la rue ne dépend que du bâtiment :

$$\text{bâtiment} \rightarrow \text{rue}$$

4.4. Troisième forme normale (3NF)

Une relation est en **3NF** si :

- elle est en 2NF
- tout attribut ne faisant pas partie de la clé n'est pas en dépendance fonctionnelle avec un ensemble d'attributs non clé.

Exemple 4.4 :

MODULE
noModule
nom
filière
responsable filière

L'entité MODULE n'est pas en 3NF car l'attribut responsable filière ne fait pas partie de la clé primaire mais dépend de filière qui est aussi un attribut non clé :

$$\text{filière} \rightarrow \text{responsable filière}$$

Exercice 17

En quelle forme normale est l'entité EXAMEN ? Identifier les problèmes de cohérence des données que cela provoque dans l'exemple ci-dessous, puis corriger le modèle conceptuel pour éviter ces problèmes de cohérence.

EXAMEN
module
date
noEtudiant
nom
prénom
note

module	date	noEtudiant	nom	prénom	note
'BD1'	'15/11/2021'	'21123456'	'Dupont'	'Mathieu'	12
'BD1'	'15/11/2021'	'21987654'	'Petit'	'Jean'	16
'BD1'	'04/01/2022'	'21123456'	'Dupond'	'Mathieu'	11
'Algo'	'23/02/2022'	'21123456'	'Dupont'	'Mathilde'	17

5. Requêtes SQL avancées

Dans ce chapitre, nous abordons des requêtes SQL plus avancées : les requêtes de sélection sur plusieurs tables, les fonctions d'agrégation et la modification de la base.

Syntaxe globale du `SELECT` :

```
SELECT [ALL | DISTINCT] liste_de_sélection
FROM liste_de_tables
[WHERE condition]
[GROUP BY critère [HAVING condition]]
[ORDER BY {expression | position} [ASC | DESC] [, ...]];
```

5.1. Produit cartésien

Le **produit cartésien** de plusieurs tables T_1, T_2, ..., T_k permet d'obtenir toutes les combinaisons possibles entre une ligne de T_1, une ligne de T_2, ..., et une ligne de T_k.

Exemple 5.1 :

```
SELECT idSite, nom, Site.idType, Type.*
FROM Site, Type;
```

Exercice 18

En supposant que le contenu des tables `Site` et `Type` est le suivant :

`Site`

idSite	nom	rue	codePostal	ville	heureOuv	heureFerm	idType
'S001'	'Musée Henri Lecoq'		'63000'				'TY01'
'S002'	'Vulcania'		'63230'				'TY02'
'S003'	'Musée Bargoin'		'63000'				'TY01'

`Type`

idType	designation
'TY01'	'Musée'
'TY02'	'Parc à thême'

Donner le résultat de la requête de l'exemple 5.5.1 ci-dessus.

5.2. Jointure

Le produit cartésien associe toute combinaison de lignes possibles entre les tables indiquées dans le **FROM**, même s'il n'y a aucune cohérence entre ces lignes. En ajoutant des conditions de **jointure**, on peut filtrer le résultat du produit cartésien pour ne garder que lignes cohérentes.

Exemple 5.2 :

```
SELECT idSite, nom, Site.idType, Type.*
FROM Site, Type
WHERE Site.idType = Type.idType; /* condition de jointure */
```

Exercice 19

Avec le contenu des tables décrit dans l'exercice 5.1, donner le résultat de la requête de l'exemple 5.5.2 ci-dessus.

⚠ En général, il faut $n-1$ conditions de jointure si la requête porte sur n tables.

Les conditions de jointures peuvent être combinées avec des conditions locales pour filtrer le résultat.

Exercice 20

Avec le contenu des tables décrit dans l'exercice 5.1, donner le résultat de la requête suivante :

```
SELECT idSite, nom, Site.idType, Type.*
FROM Site, Type
WHERE Site.idType = Type.idType /* condition de jointure */
          AND idSite = 'S001'; /* condition locale */
```

▶ **Notation avec alias.** Lorsque plusieurs tables contiennent une colonne de même nom, il est obligatoire d'indiquer le nom de la table devant le nom de la colonne pour les différencier, comme `Site.idType` et `Type.idType` dans les exemples précédents.

Pour raccourcir l'écriture de telles requêtes et en faciliter la lecture, il est possible de renommer les tables avec un alias.

Exemple 5.3 :

```
SELECT idSite, nom, s.idType, t.*
FROM Site s, Type t
WHERE s.idType = t.idType AND idSite = 'S001';
```

Exercice 21

Q_1). Lister les événements à venir avec le nom et l'adresse du site touristique où ils ont lieu.

Q_2). Lister les types (avec la désignation) des sites touristiques dont le code postal est 63000. (⚠ aux doublons)

Exercice 22

Q_1). Lister les sites touristiques accessibles avec la formule `'Découverte'`.

Q_2). Lister les formules proposant des sites touristiques en dehors du Puy de Dôme.

Exercice 23

Lister les formules dont le prix est supérieur à celui de la formule `'F002'`.

5.3. Agrégation

Une fonction d'**agrégat** permet d'effectuer un calcul statistique sur toutes les lignes sélectionnées.

Nom	Définition
`count(*)`	Compte le nombre de lignes, `0` si aucune ligne.
`count(col | exp)`	Compte le nombre de lignes dont la valeur de la colonne `col` ou de l'expression `exp` ne vaut pas `NULL`.
`count(DISTINCT col | exp)`	Compte le nombre de valeurs distinctes (non `NULL`) de la colonne `col` ou de l'expression `exp`.
`sum(col | exp)`	Somme des lignes.
`avg(col | exp)`	Valeur moyenne des lignes.
`max(col | exp)`	Valeur maximum des lignes.
`min(col | exp)`	Valeur minimum des lignes.
`stddev(col | exp)`	Écart-type des lignes.
`variance(col | exp)`	Variance des lignes.

⚠ `COUNT` tient compte des valeurs `NULL`. Les autres fonctions les ignorent.

Exemple 5.4 : Nombre de sites touristiques de type `'TY01'` :

```
SELECT COUNT(*)
FROM Site
WHERE idType = 'TY01';
```

⚠ Il n'est pas possible d'utiliser une fonction d'agrégat dans les conditions du `WHERE`.

Exercice 24

Q_1). Compter les événements prévu cette année.

Q_2). Compter le nombre de formules dont le prix est supérieur à 100€.

Exercice 25

Q_1). Calculer le prix minimum des formules pour 4 personnes.

Q_2). Calculer le tarif moyen proposé par le `'Musée Bargoin'`.

5.4. Sous-requêtes

Une **sous-requête** permet de comparer la valeur d'une colonne, d'une expression ou d'une liste de colonnes ou d'expressions avec le résultat d'une autre requête.

5.4.1. *Sous-requête délivrant une seule ligne*

```
WHERE col|expr comparateur (SELECT ... FROM ...)
```

Exemple 5.5 : Événements ayant lieu après l'événement `'E010'`.

```
SELECT *
FROM Evenement
WHERE dateE > (SELECT dateE
               FROM Evenement
               WHERE idEvenement = 'E010');
```

Exercice 26

Q_1). Lister les formules dédiées au même nombre de personnes que la formule `'F003'`.

Q_2). Lister les évenements ayant lieu dans la ville où se trouve le site touristique `'S002'`.

5.4.2. *Sous-requête délivrant plusieurs lignes*

— Valeur correspondant à au moins une/aucune ligne de la sous-requête :

```
WHERE col|expr [NOT] IN (SELECT ... FROM ...)
```

— Comparaison évaluée à vrai avec toutes les lignes de la sous-requête :

```
WHERE col|expr comparateur ALL (SELECT ... FROM ...)
```

— Comparaison évaluée à vrai avec au moins une ligne de la sous-requête :

```
WHERE col|expr comparateur ANY|SOME (SELECT ... FROM ...)
```

Exemple 5.6 : Types de sites touristiques qui n'existent pas à Clermont-Ferrand.

```
SELECT *
FROM Type
WHERE idType NOT IN (SELECT idType
                     FROM Site
                     WHERE ville = 'Clermont-Ferrand');
```

Exercice 27

Q_1). Lister les formules les plus chères.

Q_2). Lister les sites touristiques qui proposent au moins un événement en 2022.

Q_3). Lister les sites touristiques qui ne proposent aucun événement en 2022.

Exercice 28

Q_1). Lister les formules proposant des sites touristiques dans le Puy de Dôme.

Q_2). Lister les tarifs des sites touristiques de Clermont-Ferrand.

Q_3). Lister les sites touristiques étant dans une formule qui propose le `'Musée Henri Lecoq'`.

5.4.3. *Tester si une sous-requête délivre des lignes*

```
WHERE EXISTS (SELECT ... FROM ...)
```

Exercice 29

Lister les formules de prix strictement supérieur à 150€, s'il existe des produits de prix strictement supérieur à 150€ pour une seule personne.

5.5. Modification de la base

5.5.1. *Modification des lignes d'une table*

▶ **Mise à jour de lignes.**

Exemple 5.7 : Mise à jour du prix et du nombre de personnes pour la formule `'F005'`.

```
UPDATE Formule
SET prix=180.5, nbrPers=3
WHERE idFormule = 'F005';
```

Exercice 30

Q_1). Augmenter de 5% le prix de toutes les formules.

Q_2). Diminuer de 5% le prix des formules de moins de 150€ et de 10% les autres.

▶ **Suppression de lignes.**

Exemple 5.8 : Suppression de toutes les lignes d'une table (sans supprimer la table elle-même).

```
DELETE FROM Formule;
```

Exemple 5.9 : Suppression des lignes d'une table selon un critère donné.

```
DELETE FROM Formule WHERE prix > 300;
```

Exercice 31

Supprimer les événements ayant lieu entre le 10 et le 16 janvier 2022.

▶ **Ordre des modifications.** Il n'est pas possible de supprimer (ou modifier) une valeur si une autre ligne y fait référence via une clé étrangère. Il faut d'abord supprimer les lignes faisant référence avant d'effectuer cette modification.

Il est cependant possible d'utiliser l'option `ON DELETE CASCADE` lors de la définition d'une clé étrangère.

Exemple 5.10 :

```
CREATE TABLE Proposer(
    idSite  char(4),
    idTarif char(4),
    prix    numeric(4,2),
    PRIMARY KEY(idSite, idTarif),
    FOREIGN KEY (idSite) REFERENCES Site(idSite) ON DELETE CASCADE,
    FOREIGN KEY (idTarif) REFERENCES Tarif ON DELETE CASCADE
);
```

Dans ce cas, supprimer une ligne de la table `Tarif` supprimera toutes les lignes de la table `Proposer` référençant le tarif supprimé. De même, supprimer une ligne de la table `Site` supprimera toutes les lignes de la table `Proposer` référençant le site touristique supprimé. Si d'autres tables font référence à

`Proposer` avec des clés étrangères définies avec `ON DELETE CASCADE`, ces suppressions vont se propager en cascade.

⚠ À manipuler avec précaution. Des suppressions en cascade peuvent rapidement vider votre base de données.

5.5.2. *Modification d'une table*

▶ **Ajout d'une colonne.** Il est possible d'ajouter une colonne à la définition d'une table *a posteriori*.

```
ALTER TABLE nom_table ADD nom_colonne type_donnees;
```

▶ **Ajout/suppression d'une contrainte.** Il est possible d'ajouter une contrainte à la définition d'une table *a posteriori*.

Exemple 5.11 :

```
ALTER TABLE Proposer ADD CONSTRAINT ck_proposer_prix CHECK(prix >= 0);
```

Lorsqu'une contrainte est ajoutée, il est possible que des lignes de la tables ne respectent pas cette nouvelle contrainte. Dans ce cas, PostgreSQL lève une erreur.

Nommer une contrainte est important, puisque cela permet de supprimer facilement une contrainte.

Exemple 5.12 :

```
ALTER TABLE Proposer DROP CONSTRAINT ck_proposer_prix;
```

▶ **Suppression d'une table.** La commande `DROP TABLE nom_table;` permet de supprimer une table et tout son contenu. Il n'est pas possible de supprimer une table si elle est référencée par une autre table via une clé étrangère. Il faut supprimer les tables dans l'ordre inverse des références.

Exercice 32

Donner un ordre possible pour la suppression des 7 tables de l'office du tourisme.

Il est possible d'ajouter l'option **CASCADE** lors d'un **DROP TABLE** pour supprimer les contraintes de clé étrangère référençant la table.

Exercice 33

Quelles contraintes sont supprimées lorsque l'on exécute la requête suivante :

```
DROP TABLE Site CASCADE;
```

Mémo SQL

▶ **Connexion à la base.**

1. Connexion au serveur `pretoria` : `ssh login@pretoria.iut-clermont.uca.fr`
2. Connexion à PostgreSQL : `psql -h berlin -d` database `-U` login `-W`
   login = `login_ENT`, par exemple `anaduran`
   database = `dblogin`, par exemple `dbanaduran`
   mot de passe = `achanger`

▶ **Changement de mot de passe.** Après s'être connecté à PostgreSQL :

```
ALTER USER login WITH PASSWORD 'mon nouveau mot de passe';
```

⚠ Ne pas oublier les `' '` (qui ne font pas partie du mot de passe) et ne pas oublier le `;` à la fin !

▶ **Commandes utiles.**

— `\?` : aide
— `\q` : quitter `psql`
— `\d` : liste des tables
— `\d Table` : description de la table `d`

▶ **Aide en ligne :** `https://s2i.iut.uca.fr/page/documentation/sgbd/`