

Découvrir un autre SGBD : PostgreSQL

Table des matières

I. Contexte	3
II. Présentation de PostgreSQL	3
III. Exercice : Appliquez la notion	4
IV. Installation de PostgreSQL	5
V. Exercice : Appliquez la notion	9
VI. Points communs et différences avec d'autres SGBD	10
VII. Exercice : Appliquez la notion	12
VIII. Les séquences	12
IX. Exercice : Appliquez la notion	14
X. Essentiel	14
XI. Auto-évaluation	14
A. Exercice final	14
B. Exercice : Défi	16
Solutions des exercices	17

I. Contexte

Durée : 45 min

Environnement de travail : PostgreSQL

Pré-requis : Avoir fait les cours pour MariaDB

Contexte

Il existe de nombreux systèmes de gestion de bases de données (SGBD) : chacun possède ses spécificités. Parfois, il s'agit de différences de syntaxe : deux SGBD peuvent disposer de la même fonction, mais sous un nom différent. Il pourra s'agir également de différences de fonctions, où un SGBD dispose d'une fonction qui n'existe pas chez un autre SGBD.

Parmi les SGBD open source, les plus connus sont MySQL/MariaDB et PostgreSQL. Ce cours présente PostgreSQL, ses particularités et ses différences par rapport à MariaDB.

II. Présentation de PostgreSQL

Objectif

- Découvrir PostgreSQL

Mise en situation

PostgreSQL est un système de gestion de **bases de données** relationnelles (SGBDR). Dans PostgreSQL, à l'instar des autres bases de données relationnelles, les données sont organisées en tableaux à deux dimensions, appelées tables.

Les colonnes de ces tables représentent la structure des données, tandis que les lignes de cette table représentent un enregistrement.

Remarque Licence

PostgreSQL est sous licence BSD¹. La licence BSD autorise la réutilisation du logiciel sans restriction, y compris pour un usage commercial. Il s'agit d'une licence libre. PostgreSQL est donc libre et gratuit, y compris pour un usage commercial.

Fonctionnalités

PostgreSQL est un SGBDR très complet. Il offre un grand nombre de fonctionnalités, parmi lesquelles :

Des objets tels que

- Les vues
- Les séquences
- Les triggers

¹ https://fr.wikipedia.org/wiki/Licence_BSD

Du requêtage complexe via

- Des jointures externes
- Des requêtes imbriquées
- Des requêtes UNION, UNION ALL et EXCEPT

De la programmation via

- Les procédures stockées
- PL/PGSQL, un langage de programmation
- Des extensions objet (héritage entre tables)
- La possibilité de rajouter des extensions dans d'autres langages

Des possibilités d'administration avec

- Des sauvegardes à chaud, complètes ou incrémentales
- Des restaurations complètes ou partielles

De plus, PostgreSQL est compatible avec

- Le XML
- L'Unicode

Cette liste n'est bien évidemment pas exhaustive.

Syntaxe À retenir

- PostgreSQL est un SGBD relationnel disposant d'un grand nombre de fonctionnalités.
- Il est disponible sous licence libre BSD et peut être utilisé pour tout usage personnel ou commercial.

Complément

Site de PostgreSQL¹

III. Exercice : Appliquez la notion

Question

Voici une requête permettant de créer une table très simple dans plusieurs SGBD :

```
1 /* MariaDB */
2 CREATE TABLE Ventes
3 (
4     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
5     NomVendeur varchar(50) NOT NULL,
6     MontantVente decimal(10,2) DEFAULT 0,
7     DateVente datetime DEFAULT NOW()
8 )
9 ;

1 /* SQL Server */
2 CREATE TABLE Ventes
3 (
4     Id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
5     NomVendeur nvarchar(50) NOT NULL,
```

¹ <https://www.postgresql.org/>

```
6   MontantVente decimal(10,2) DEFAULT 0,  
7   DateVente datetime DEFAULT NOW()  
8 )  
9 ;  
  
1 /* PostgreSQL */  
2 CREATE TABLE Ventes  
3 (  
4   Id Serial PRIMARY KEY,  
5   NomVendeur varchar(50) NOT NULL,  
6   MontantVente decimal(10,2) DEFAULT 0,  
7   DateVente timestamp DEFAULT NOW()  
8 )  
9 ;
```

Ces trois requêtes créent des tables similaires dans, respectivement, MariaDB, SQL Server et PostgreSQL.

Quelles sont les différences entre ces requêtes ?

Indice :

Il y a trois différences notables, portant sur la définition des champs `Id`, `NomVendeur` et `DateVente`.

IV. Installation de PostgreSQL

Objectif

- Installer PostgreSQL

Mise en situation

PostgreSQL est un SGBD multi plateformes, ce qui signifie qu'il est disponible sur de nombreux systèmes d'exploitation.

Ce cours n'a pas pour vocation de couvrir l'ensemble des possibilités d'installation, mais de fournir quelques informations importantes pour les différentes étapes.

Obtenir PostgreSQL

PostgreSQL est un logiciel open source. Il peut être librement téléchargé et installé. Le logiciel est disponible sur le site dédié¹.

Conseil Les différentes versions

PostgreSQL a été créé en 1985. Un logiciel aussi ancien a nécessairement connu de nombreuses versions.

Une version dite « bêta » ou « instable » correspond à une version en phase de test avant publication. Il s'agit généralement d'une version plutôt stable, néanmoins elle peut présenter certains bugs ou évoluer en fonctionnalités.

D'une manière générale, il est préférable de prendre la version stable la plus récente.

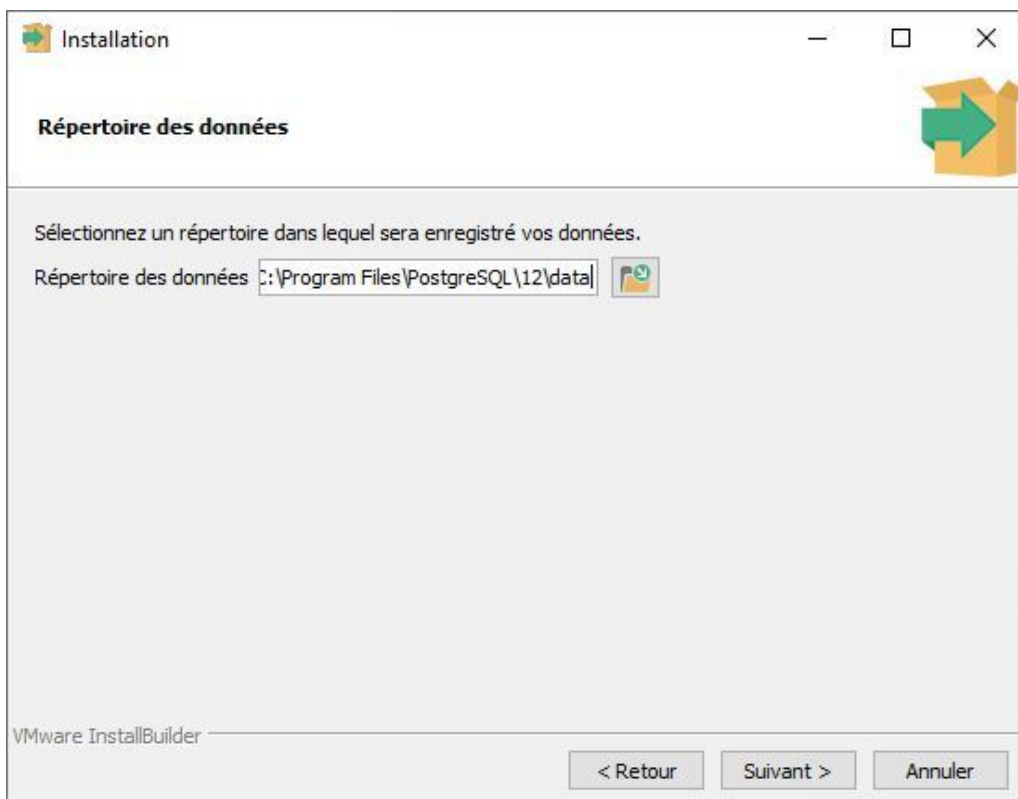
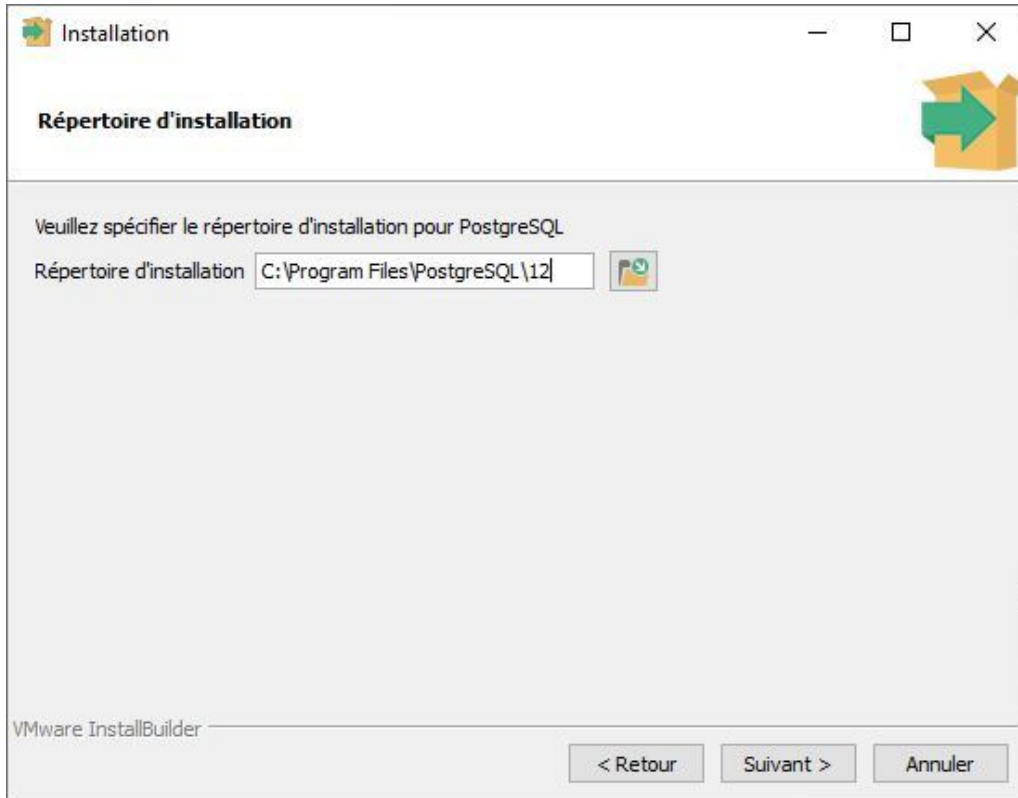
PostgreSQL peut être compilé depuis le code source, mais de nombreux installateurs ou paquets existent et simplifient grandement l'installation.

¹ <https://www.postgresql.org/>

Chemin de données

PostgreSQL permet de séparer les exécutable des données, c'est-à-dire que la base de données proprement dite peut être située ailleurs que dans le dossier contenant le logiciel lui-même.

C'est pour cela que PostgreSQL peut demander deux chemins : l'un pour l'installation proprement dite, et le second pour le chemin des données. Il s'agit, de façon générale, d'une bonne pratique.



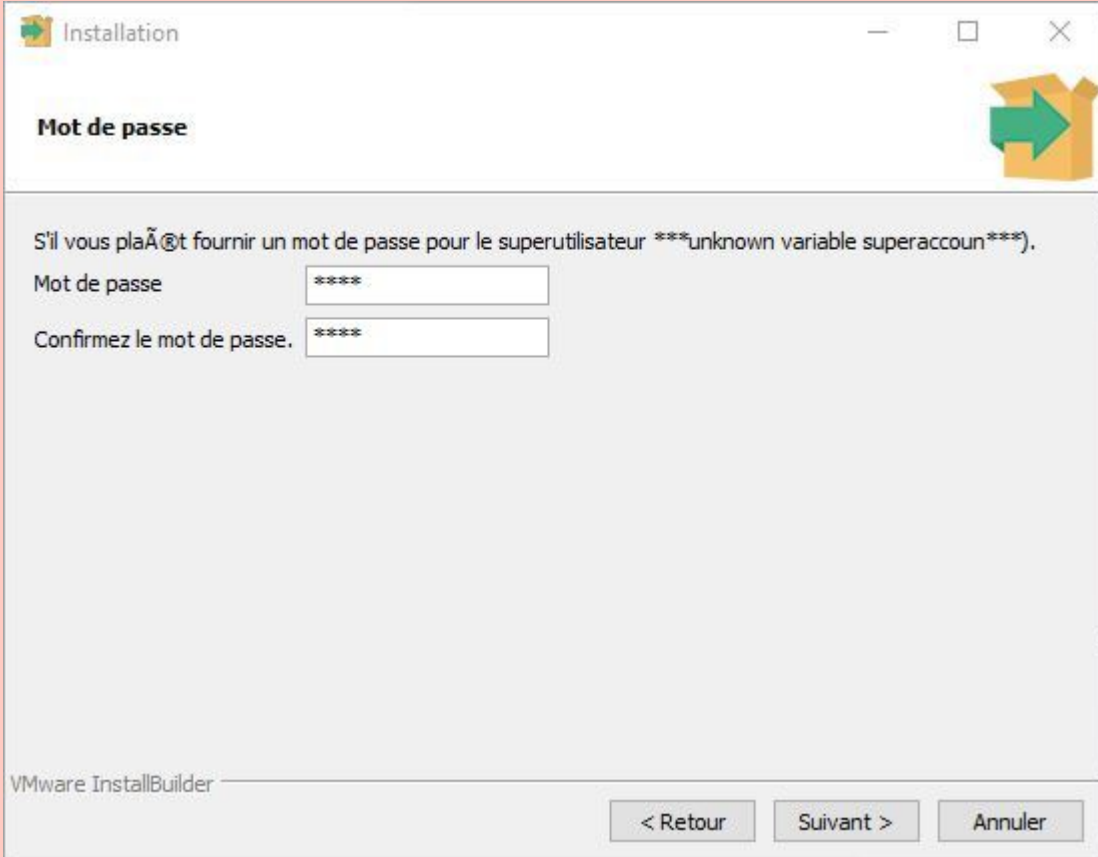
Mot de passe super utilisateur

Le **super utilisateur** est le compte ayant tous les droits sur le SGBD. En tant que tel, il peut absolument tout faire.

En pratique, le compte super utilisateur sera limité aux actions d'administration, et on créera des comptes utilisateurs pour l'utilisation courante de la base de données.

Attention

Au cours de l'installation, il faudra définir un mot de passe pour le super utilisateur. Ce mot de passe est très important, et il peut être difficile d'en recréer un en cas de perte. Il faudra choisir un mot de passe complexe, et ne pas l'oublier.



Installation

Mot de passe

S'il vous plaît fournir un mot de passe pour le superutilisateur (***unknown variable superaccoun***).

Mot de passe

Confirmez le mot de passe.

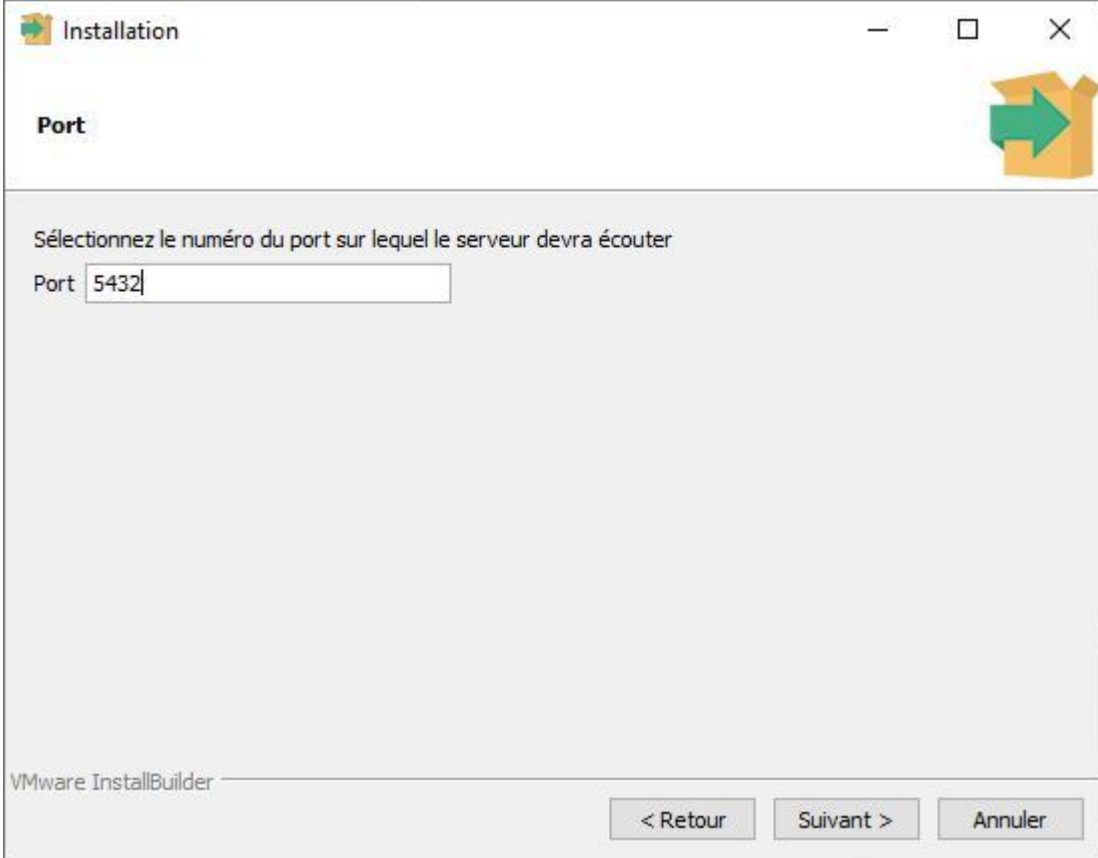
VMware InstallBuilder

< Retour Suivant > Annuler

Port d'écoute

Tout logiciel utilisant le réseau passera par un ou plusieurs ports. Chaque port est déterminé par un numéro entre 1 et 65535. Le port par défaut de PostgreSQL est le 5432. Il peut être gardé tel quel et il est possible de le changer plus tard si besoin.

N'oubliez pas d'autoriser ce port sur son pare-feu en cas de problème !



The screenshot shows a window titled "Installation" with a standard Windows-style title bar (minimize, maximize, close buttons). In the top right corner of the window, there is a large green arrow pointing right, superimposed on a yellow box icon. Below the title bar, the word "Port" is displayed in bold. The main area of the window contains the instruction "Sélectionnez le numéro du port sur lequel le serveur devra écouter" (Select the port number on which the server will listen). Below this instruction, there is a text input field labeled "Port" containing the value "5432". At the bottom left of the window, the text "VMware InstallBuilder" is visible. At the bottom right, there are three buttons: "< Retour", "Suivant >", and "Annuler".

Installation

Port

Sélectionnez le numéro du port sur lequel le serveur devra écouter

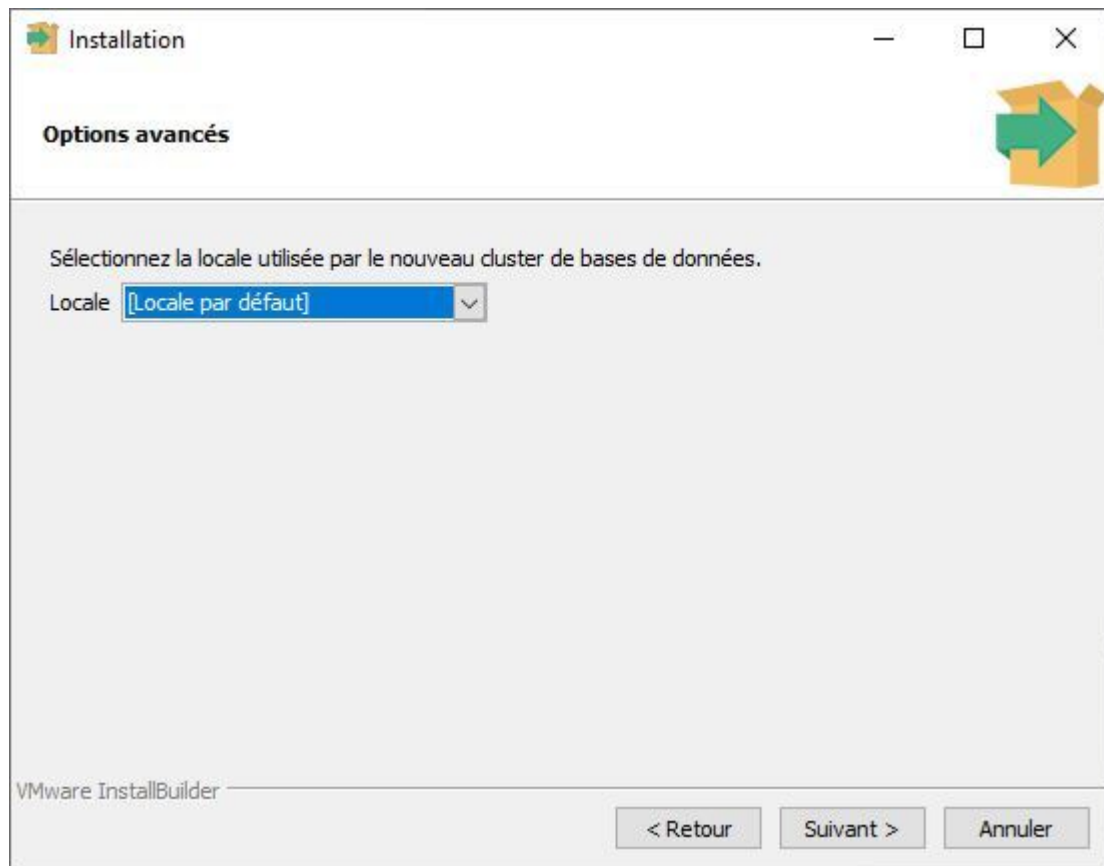
Port

VMware InstallBuilder

< Retour Suivant > Annuler

Locale

Le choix de la locale correspond aux choix des paramètres régionaux pour les langues, le format par défaut des nombres, des dates... Au besoin, ce paramètre peut être modifié plus tard.



Syntaxe À retenir

- L'installation de PostgreSQL est simplifiée par la disponibilité de nombreux installateurs ou packages pour les systèmes d'exploitation les plus courants.
- Il faut bien renseigner le mot de passe super utilisateur et ne pas l'oublier.

Complément

Site de PostgreSQL¹

V. Exercice : Appliquez la notion

Question

PostgreSQL possède une interface d'administration appelée **pgadmin** : cette interface est disponible en mode web une fois que le SGBD est démarré. Elle se présente sous la forme d'un exécutable qui est installé en même temps que le SGBD.

Pour cet exercice, installez et lancez PostgreSQL, puis accédez à **pgadmin**.

¹ <https://www.postgresql.org>

Une base de données du nom de PostgreSQL est déjà présente. Il s'agit d'une base de données système : **il est très fortement recommandé de ne pas y toucher.**

Créez une nouvelle base de données depuis l'interface graphique, puis utilisez ensuite l'onglet SQL pour observer le code généré.

Indice :

Il est possible d'ouvrir un éditeur de requêtes et de saisir une instruction `CREATE DATABASE`, ou tout simplement de faire un clic droit sur *Databases*, puis *Create database* et se laisser guider par l'assistant.

VI. Points communs et différences avec d'autres SGBD

Objectif

- Avoir un aperçu des différences entre SGBD

Mise en situation

On pourrait penser que les SGBD relationnels sont homogènes entre eux et facilement interchangeables, mais il n'en est rien : il existe de nombreuses différences à prendre en compte en passant d'un SGBD à un autre.

La norme SQL

Afin d'essayer d'harmoniser la syntaxe et les fonctionnalités des bases de données, il existe une norme à laquelle les SGBD doivent se soumettre : la **norme SQL**. En théorie, l'existence de cette norme devrait permettre de garantir une constance entre les différents moteurs, mais, dans la pratique, il n'en est rien. En effet, cette norme ne décrit pas l'ensemble des fonctionnalités attendues d'un SGBD relationnel, mais seulement les fonctionnalités d'un socle.

Certaines règles de la norme peuvent également ne pas être respectées par un SGBD. Par exemple, MySQL version 5.0 ne gère pas le `FULL OUTER JOIN` ni les CTE (*Common Table Expression*), un outil disponible dans d'autres SGBD. Oracle, un autre SGBD, gère les CTE, mais n'offre aucun équivalent à la clause `LIMIT` de MySQL.

D'autres fonctionnalités peuvent également avoir été implémentées avant leur normalisation par l'ISO et avoir été conservées par souci de compatibilité ascendante. C'est pourquoi certains SGBD peuvent disposer de plusieurs alias pour une même fonction, comme par exemple `SUBSTR`, `SUBSTRING` ou `MID`, qui servent toutes les trois à extraire une sous-chaîne d'un texte.

Fondamental	Requêtes CRUD
--------------------	----------------------

La structure de base d'une requête SQL reste la même dans tous les SGBD. Les `SELECT`, `UPDATE`, `INSERT` et `DELETE` ne changent pas d'une base à l'autre.

Types de données

Bien que les SGBD soient destinés à conserver des données de type numérique, texte ou date, leur façon de les conserver diffère d'un SGBD à un autre. Il est donc préférable de s'assurer de la compatibilité des types de données en passant d'une base à une autre.

Les types les plus simples, comme `int` ou `char`, existent dans toutes les bases de données. Par contre, les types plus complexes peuvent avoir des particularités, c'est pourquoi il faut toujours se référer à la documentation du SGBD en cas de problème.

Exemple

Le type `datetime`, permettant de stocker des dates, n'existe pas forcément dans tous les SGBDR : il est présent dans MariaDB, mais pas dans PostgreSQL.

Fonctions

La plupart des fonctions usuelles de manipulation de texte ou de date sont disponibles dans les SGBDR. Toutefois, il est fortement recommandé de se référer à la documentation du SGBDR utilisé pour vérification. En effet, certains SGBDR peuvent ne pas disposer de la fonction, ou bien la proposent sous un nom différent ou avec une syntaxe différente.

Exemple

Reprenons l'exemple des fonctions `SUBSTR`, `SUBSTRING` et `MID`. Le principe de ces fonctions est d'extraire une sous-chaîne à partir d'un texte, en fournissant la position de départ de la sous-chaîne et sa longueur, mais leur existence dépend du SGBD utilisé :

- Les dernières versions de MySQL et MariaDB proposent les trois, mais des versions plus anciennes peuvent ne gérer que `SUBSTRING`.
- PostgreSQL et Oracle ne proposent que `SUBSTR` et `SUBSTRING`.
- SQL Serveur ne propose que `SUBSTRING`.

Pour s'assurer de la compatibilité d'une requête avec un maximum de SGBD, il est donc préférable d'utiliser `SUBSTRING`.

Attention

Certains SGBDR peuvent même avoir des fonctions ayant le même nom, mais avec des effets différents !

- Dans MySQL, `ISNULL (Valeur)` est une fonction qui permet de tester si une valeur est nulle, et de renvoyer **vrai** ou **faux** selon le résultat.
- Dans SQL Server, `ISNULL (Valeur, Défaut)` est une fonction qui, si `Valeur` n'est pas null, renvoie `Valeur`, sinon renvoie `Défaut`.

La fonction `ISNULL` est donc une fonction booléenne dans MySQL, mais une fonction de transformation de données dans SQL Server !

Fonctionnalités

Les SGBDR proposent souvent de nombreuses fonctionnalités. Malheureusement, la plupart d'entre elles ne font pas l'objet de normes SQL ou sont fréquemment implémentées de façon différente d'un SGBDR à un autre.

Par exemple, certains SGBDR comme Oracle et PostgreSQL permettent d'utiliser des séquences, qui sont des compteurs indépendants d'une table. Une séquence permet d'utiliser un compteur commun à toute une base de données, ce qui peut être utilisé pour de gros projets. Sur d'autres SGBDR, il faudra passer par une solution différente.

Syntaxe **À retenir**

- Les différents SGBDR possèdent un socle commun, la norme SQL. Cependant, de nombreuses fonctionnalités, formats de données ou fonctions peuvent être différents.
- Il est important de toujours se référer à la documentation du SGBD utilisé.

Complément

 Site de PostgreSQL¹

VII. Exercice : Appliquez la notion

Question

Il est très courant de vouloir ajouter une chaîne de texte à la suite d'une autre, par exemple pour obtenir *Prénom Nom*. Cette opération s'appelle la concaténation.

Le code suivant permet de créer et d'alimenter une table `ListeNoms` :

```
1 CREATE TABLE ListeNoms
2 (
3     Nom varchar(50) NOT NULL,
4     Prenom varchar(50) NOT NULL
5 )
6 ;
7 INSERT INTO ListeNoms VALUES ('Dupont', 'Jean');
8 INSERT INTO ListeNoms VALUES ('Dupond', 'Jeanne');
9 INSERT INTO ListeNoms VALUES ('Duperre', 'Jeremy');
```

Pour obtenir la liste des *Prénom Nom* des valeurs dans cette table, il est possible d'écrire la requête suivante en MariaDB :

```
1 SELECT
2     CONCAT(Prenom, ' ', Nom)
3 FROM ListeNoms;
```

Bien que cette requête fonctionne également en PostgreSQL, ce SGBD propose une alternative beaucoup plus courte à écrire. En vous aidant de la documentation PostgreSQL, réécrivez cette requête en utilisant les outils de concaténation du moteur.

Indice :

Le but ici est de ne pas utiliser une fonction, mais plutôt un opérateur.

VIII. Les séquences

Objectif

- Voir l'exemple d'une fonctionnalité propre à certains SGBDR

Mise en situation

PostgreSQL possède donc certaines différences de syntaxe et de nommage, comparé à MariaDB. Mais il dispose également de fonctionnalités supplémentaires qui peuvent parfois faire la différence dans le choix d'un SGBD. Pour illustrer ce cas, approfondissons un exemple mentionné précédemment : les **séquences**.

Définition **Les séquences**

Une séquence est un compteur géré par la base de données. Elle est définie par un nom, une valeur initiale et un pas, et stocke sa valeur actuelle.

¹ <https://www.postgresql.org>

Exemple

Un très bon exemple de séquence dans la vie courante est le système de ticket mis en place dans certains guichets. Si le numéro de ticket commence à 100 et s'incrémente de 1 à 1, le premier ticket tiré sera 100, le suivant sera 101, puis 102, etc.

Méthode **Syntaxe**

En PostgreSQL, une séquence est créée via la syntaxe suivante :

```
1 CREATE SEQUENCE NomDeSequence INCREMENT BY x MINVALUE Minimum MAXVALUE Maximum CYCLE
```

- MINVALUE est optionnel. En cas d'absence, la séquence commencera à 1.
- MAXVALUE est optionnel. En cas d'absence, le maximum sera $2^{63} - 1$.
- CYCLE est optionnel. S'il est utilisé, lorsque la séquence atteint la valeur maximale, elle repartira de MINVALUE. Attention, s'il n'est pas utilisé et que la séquence atteint la valeur maximale, un message d'erreur sera envoyé à chaque nouvel appel.

La séquence fournit usuellement deux fonctions :

- `currval` : renvoie la valeur actuelle de la séquence.
- `nextval` : incrémente la séquence du pas choisi, puis renvoie la nouvelle valeur.

Ces deux fonctions prennent le nom de la séquence en paramètre.

```
1 currval('NomDeSequence')
2 nextval('NomDeSequence')
```

Exemple

L'exemple de ticket repris plus haut pourrait être écrit comme suit :

```
1 CREATE SEQUENCE Ticket INCREMENT BY 1 MINVALUE 100 MAXVALUE 999 CYCLE
```

Le numéro de ticket commencera donc à 100, montera de 1 en 1, et s'il est à 999, la prochaine valeur sera à nouveau 100. Pour incrémenter le compteur, il suffit d'utiliser la requête suivante :

```
1 SELECT NextVal('Ticket')
```

Comparaison avec MySQL et MariaDB

Dans PostgreSQL, les séquences sont disponibles depuis au moins la version 7.1, qui date de 2001. MariaDB propose les séquences depuis la version 10.3, qui date de 2018. MySQL par contre ne propose pas l'objet `Sequence`.

Syntaxe **À retenir**

- Les séquences permettent de créer un compteur indépendant dans la base de données.
- Ce compteur peut ensuite être utilisé via du code SQL par les fonctions `currval` et `nextval`.

Complément

Syntaxe du `CREATE`¹

Séquences en PostgreSQL²

¹ <https://www.postgresql.org/docs/9.5/sql-createsequence.html>

² <https://www.postgresql.org/docs/8.2/functions-sequence.html>

XII. Exercice : Appliquez la notion

Question 1

Une entreprise de fabrication de véhicules possède plusieurs usines, une par type de véhicules à construire : des voitures, des motos, des camions et des bus. Chacun de ces véhicules, quel que soit son type, est associé à une plaque d'immatriculation sous la forme XX-111-YY. Pour simplifier le problème, nous n'allons nous concentrer que sur la partie du milieu, qui est un nombre entre 1 et 999, qui s'incrémente à chaque nouveau véhicule et revient à 1 au 1000^{ème} véhicule.

Écrivez une séquence permettant de gérer cette partie d'une plaque d'immatriculation.

Question 2

Sachant que la table contenant les voitures est sous cette forme :

```
1 CREATE TABLE Voitures
2 (
3     Id Serial PRIMARY KEY,
4     Immatriculation integer NOT NULL,
5     BoiteAuto boolean NOT NULL,
6     TailleCoffre integer NOT NULL
7 );
```

Insérez une voiture qui n'a pas de boîte automatique et a un coffre de 520 litres, puis une voiture qui a une boîte automatique et un coffre de 450 litres. À noter que le type `boolean` de PostgreSQL permet d'utiliser `true` et `false` dans les requêtes.

Question 3

La table contenant les motos, quant à elle, est sous cette forme :

```
1 CREATE TABLE Motos
2 (
3     Id Serial PRIMARY KEY,
4     Immatriculation integer NOT NULL,
5     Cylindree integer NOT NULL
6 );
```

Insérez une moto de 1050 cm³. Attention, le numéro d'immatriculation doit suivre avec les insertions dans la table des voitures.

XIII. Essentiel

XIV. Auto-évaluation

A. Exercice final

Exercice

Exercice

PostgreSQL est...

- ☐ Une base de données
- ☐ Un système de base de données relationnel
- ☐ Un système de base de données non relationnel

Exercice

PostgreSQL est...

- ☐ Un logiciel gratuit
- ☐ Un logiciel payant
- ☐ Un logiciel libre

Exercice

PostgreSQL est disponible...

- ☐ Sur un grand nombre de plateformes
- ☐ Exclusivement sous Windows
- ☐ Exclusivement dans le Cloud

Exercice

Le super utilisateur est...

- ☐ Un compte disposant de droits très étendus, normalement réservé à des opérations d'administration
- ☐ Un compte protégé, sécurisant l'accès à toutes les applications nécessitant d'utiliser la base
- ☐ Un programme anti-intrusion vérifiant les mots de passe des utilisateurs se connectant à la base

Exercice

La locale est...

- ☐ Un raccourci pour désigner le langage local, c'est-à-dire la langue utilisée pour l'interface de PostgreSQL
- ☐ Un raccourci pour parler des règles locales propres à une seule base de données
- ☐ L'ensemble des paramètres régionaux utilisés par le serveur pour toutes les bases de données, comme le format des nombres, des dates...

Exercice

Une requête SQL peut être utilisée telle quelle sur tous les SGBDR du marché.

- ☐ Vrai, car le SQL est une norme réglementant précisément la structure d'une requête
- ☐ Pas toujours, la base est similaire pour tous les SGBDR, mais certaines fonctions ou mots-clés peuvent varier
- ☐ Faux, chaque SGBDR utilise une syntaxe unique

Exercice

D'un SGBDR à l'autre, les types de données sont...

- ☐ Pour la plupart similaires, mais il peut y avoir certaines différences (par exemple, un type peut ne pas exister, mais un autre joue le même rôle)
- ☐ Totalement différents
- ☐ Identiques dans leurs fonctions (parfois, ils n'ont pas le même nom, mais il suffit d'utiliser le bon nom)

Exercice

Tous les SGBDR proposent les mêmes fonctionnalités.

- ☐ Vrai, mais il est possible de rajouter des fonctionnalités personnelles via un langage de programmation
- ☐ Faux, mais ils proposent tous les fonctions de base décrites par la norme SQL, et certains peuvent en proposer davantage
- ☐ Faux, certains proposent plus que la norme SQL, mais d'autres peuvent ne même pas proposer toutes les fonctions de la norme SQL

Exercice

Une séquence est...

- ☐ Un alias pour créer une clé primaire auto-incrémentée dans une table
- ☐ Un objet de la base de données permettant de fournir un compteur
- ☐ Une fonction d'analyse renvoyant une séquence de nombres à partir d'une colonne source

Exercice

Pour obtenir le prochain numéro d'une séquence, il faut utiliser la fonction...

- ☐ NEXTVAL pour incrémenter, puis CURRVAL pour récupérer le numéro
- ☐ INCREMENT BY X pour indiquer à la séquence de s'incrémenter de X
- ☐ NEXTVAL, qui incrémente et renvoie le nouveau numéro

B. Exercice : Défi

Une migration de SGBDR est toujours un gros chantier, à cause des multiples différences entre chaque SGBDR. Cela demande d'adapter le code à tous les niveaux. Ce sont souvent de petites différences, mais qui, mises bout à bout, représentent un travail long et minutieux. En voici un petit exemple.

La table suivante est utilisée pour analyser les ventes d'un magasin :

```

1 CREATE TABLE AnalyseVentes
2 (
3     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     NomVendeur varchar(50) NOT NULL,
5     PrenomVendeur varchar(50) NOT NULL,
6     Exercice int,
7     TotalVentes decimal(15,2),
8     ObjectifAnnuel decimal(15,2)
9 )
10 ;

```

Voici un exemple de données pour alimenter cette table :

```

1 INSERT INTO AnalyseVentes VALUES (1, 'Dupont', 'Jean', 2020, 157007.50, 150000);
2 INSERT INTO AnalyseVentes VALUES (2, 'Dupond', 'Jeanne', 2020, 188045.14, 160000);
3 INSERT INTO AnalyseVentes VALUES (3, 'Duperre', 'Jeremy', 2020, 138790.8, 140000);

```

Cette requête permet d'obtenir le résultat des analyses de ventes comme suit :

- dans une seule colonne, le prénom suivi d'un espace, puis le nom en majuscules (par exemple, *Jean DUPONT*)
- la date de début de l'exercice, c'est-à-dire le 1^{er} janvier de l'année donnée dans la colonne *Exercice*
- la date de fin de l'exercice, c'est-à-dire le 31 décembre de l'année donnée dans la colonne *Exercice*
- le total des ventes
- la différence entre le total des ventes et l'objectif annuel
- un texte indiquant *Oui* si l'objectif a été atteint, ou *Non* dans le cas contraire


```
1 SELECT
2     CONCAT(PrenomVendeur, ' ', UPPER(NomVendeur)),
3     MAKEDATE(Exercice, 1),
4     DATE_ADD(MAKEDATE(Exercice + 1, 1), INTERVAL -1 DAY),
5     TotalVentes,
6     TotalVentes - ObjectifAnnuel,
7     CASE
8         WHEN TotalVentes > ObjectifAnnuel THEN 'Oui'
9         ELSE 'Non'
10    END
11 FROM AnalyseVentes;
```

Question

Comment transformer la requête `CREATE` et la requête `SELECT` pour les rendre compatibles avec PostgreSQL ?

Indice :

Par chance, la requête `CREATE` est pratiquement compatible avec PostgreSQL. La seule chose à faire est de changer la définition de la colonne `Id`.

Indice :

Le pseudo type de données `serial` est un alias pour `int not null auto_increment`.

Indice :

`MAKEDATE` n'existe pas en PostgreSQL. Il y a bien une fonction `MAKE_DATE`, mais elle est un petit peu différente...

Indice :

Il n'y a pas de `DATE_ADD` en PostgreSQL. Il faut effectuer une opération du genre **MaDate + 1 jour**. Vérifiez dans la documentation pour la syntaxe exacte.

Solutions des exercices

Exercice p. Solution n°1

Bien que ces trois tables soient compatibles entre elles, le code permettant de les générer est différent selon les bases de données.

Il s'agit d'une constante : **chaque SGBD possède ses particularités.**

La différence la plus évidente porte sur la colonne `Id`, qui est un compteur auto-incrémenté. Les définitions sont totalement différentes selon les SGBD, mais le résultat sera le même : un compteur qui commence à 1 et qui s'incrémente de 1 à chaque enregistrement.

La seconde différence porte sur le nom du vendeur, qui est décrit comme `nvarchar` dans SQL Server et `varchar` pour les autres. `nvarchar` est un type de données spécifique à SQL Server qui n'a pas de correspondance exacte en MariaDB ou PostgreSQL, mais le comportement le plus proche est de choisir un type `varchar` sur une base configurée pour utiliser l'Unicode.

La dernière différence est la colonne `DateVente`. Le type `datetime` n'existe pas en PostgreSQL, il faut utiliser `timestamp`. À noter que le type `timestamp` existe en MariaDB et en SQL Server.

Exercice p. Solution n°2

Le code suivant est celui qui a été généré par PostgreSQL suite à un clic droit sur *Databases > Create database*. La base de données a été appelée `Exemple` : aucun autre paramètre n'a été touché.

```
1 CREATE DATABASE "Exemple"
2   WITH
3   OWNER = postgres
4   ENCODING = 'UTF8'
5   LC_COLLATE = 'French_France.1252'
6   LC_CTYPE = 'French_France.1252'
7   TABLESPACE = pg_default
8   CONNECTION LIMIT = -1;
```

- `OWNER = postgres` signifie que la base `Exemple` appartient à l'utilisateur par défaut `postgres`.
- `ENCODING = 'UTF8'` vient du paramétrage par défaut du serveur. UTF8 est une très bonne idée.
- `LC_COLLATE` et `LC_CTYPE` sont des paramètres régionaux. Ici, ils correspondent aux paramètres usuels en France.
- `TABLESPACE` a sa valeur par défaut.
- `CONNECTION LIMIT = -1` signifie qu'il n'y a pas de limite en nombre de connexions.

Dans la pratique, ces valeurs sont rarement renseignées au moment d'un `CREATE TABLE` : ce sont les valeurs par défaut.

Exercice p. Solution n°3

En PostgreSQL, la concaténation peut se faire via l'opérateur `||` :

```
1 SELECT
2   Prenom || ' ' || Nom
3 FROM ListeNoms;
```

Cependant, il reste préférable d'utiliser la fonction `CONCAT` en PostgreSQL afin d'assurer la compatibilité de notre application avec n'importe quelle base de données. Cet opérateur n'est à utiliser que si l'on est certain de ne jamais changer de SGBD.

Exercice p. Solution n°4

Pour créer cette séquence, il faut utiliser la requête :

```
1 CREATE SEQUENCE Immatriculation INCREMENT BY 1 MINVALUE 1 MAXVALUE 999 CYCLE
```

Exercice p. Solution n°5

En utilisant la fonction `NextVal`, on peut récupérer la valeur de l'immatriculation une fois incrémentée :

```
1 INSERT INTO Voitures (Immatriculation, BoiteAuto, TailleCoffre) VALUES
  (NextVal('Immatriculation'), false, 520);
2 INSERT INTO Voitures (Immatriculation, BoiteAuto, TailleCoffre) VALUES
  (NextVal('Immatriculation'), true, 450);
```

Exercice p. Solution n°6


Grâce à la séquence, le numéro d'immatriculation est indépendant de la table : on peut donc utiliser `NextVal` également sur la table des motos.

```
1 INSERT INTO Motos (Immatriculation, Cylindree) VALUES (NextVal('Immatriculation'), 1050);
```

Exercice p. 14 Solution n°7


Exercice

PostgreSQL est...

- ☐ Une base de données
- ☒ Un système de base de données relationnel
- ☐ Un système de base de données non relationnel
-  PostgreSQL appartient à la famille des systèmes de gestion de base de données relationnels.

Exercice

PostgreSQL est...

- ☒ Un logiciel gratuit
- ☐ Un logiciel payant
- ☒ Un logiciel libre
-  PostgreSQL n'est pas seulement un outil gratuit, il est avant tout libre sous licence BSD. Il peut être utilisé, y compris pour un usage commercial.

Exercice

PostgreSQL est disponible...

- ☒ Sur un grand nombre de plateformes
- ☐ Exclusivement sous Windows
- ☐ Exclusivement dans le Cloud

- Q PostgreSQL est multi plateformes. Il est notamment disponible sous Windows et Linux, mais également sur bien d'autres systèmes d'exploitation. Attention, « libre » ne veut pas forcément dire « gratuit » : il existe de nombreux logiciels libres dont les sources sont disponibles, mais qui nécessitent de payer pour un hébergement (quand l'auto-hébergement n'est pas possible). C'est le cas de GitLab, par exemple.

Exercice

Le super utilisateur est...

- ☒ Un compte disposant de droits très étendus, normalement réservé à des opérations d'administration
- ☐ Un compte protégé, sécurisant l'accès à toutes les applications nécessitant d'utiliser la base
- ☐ Un programme anti-intrusion vérifiant les mots de passe des utilisateurs se connectant à la base

- Q Le super utilisateur dispose de tous les droits sur toutes les bases de données. Avec autant de pouvoir, il est très vivement recommandé de ne l'utiliser qu'en cas de nécessité. Le super utilisateur ne doit pas être utilisé pour les usages ordinaires !

Exercice

La locale est...

- ☐ Un raccourci pour désigner le langage local, c'est-à-dire la langue utilisée pour l'interface de PostgreSQL
 - ☐ Un raccourci pour parler des règles locales propres à une seule base de données
 - ☒ L'ensemble des paramètres régionaux utilisés par le serveur pour toutes les bases de données, comme le format des nombres, des dates...
- Q Il s'agit des paramètres régionaux. Ces paramètres affectent toutes les bases de données du serveur, mais peuvent ensuite être changés base par base si besoin.

Exercice

Une requête SQL peut être utilisée telle quelle sur tous les SGBDR du marché.

- ☐ Vrai, car le SQL est une norme réglementant précisément la structure d'une requête
 - ☒ Pas toujours, la base est similaire pour tous les SGBDR, mais certaines fonctions ou mots-clés peuvent varier
 - ☐ Faux, chaque SGBDR utilise une syntaxe unique
- Q Malheureusement, la norme SQL ne définit pas aussi précisément le langage SQL. Certains ajustements peuvent être nécessaires, même si les SGBDR ont une syntaxe similaire.


Exercice

D'un SGBDR à l'autre, les types de données sont...

- ☒ Pour la plupart similaires, mais il peut y avoir certaines différences (par exemple, un type peut ne pas exister, mais un autre joue le même rôle)
 - ☐ Totalement différents
 - ☐ Identiques dans leurs fonctions (parfois, ils n'ont pas le même nom, mais il suffit d'utiliser le bon nom)
- Q Les types de données sont souvent similaires, voire identiques. Ce n'est pas toujours vrai, cependant : un type peut ne pas exister ou exister sous le même nom, mais avec des propriétés différentes. Il est toujours bon de consulter la documentation.


Exercice

Tous les SGBDR proposent les mêmes fonctionnalités.

- ☐ Vrai, mais il est possible de rajouter des fonctionnalités personnelles via un langage de programmation
- ☐ Faux, mais ils proposent tous les fonctions de base décrites par la norme SQL, et certains peuvent en proposer davantage
- ☒ Faux, certains proposent plus que la norme SQL, mais d'autres peuvent ne même pas proposer toutes les fonctions de la norme SQL
-  Malheureusement, il n'est pas rare que certaines fonctionnalités décrites dans la norme SQL ne soient pas implémentées, ou seulement partiellement, tandis que d'autres fonctionnalités non normées sont présentes.


Exercice

Une séquence est...

- ☐ Un alias pour créer une clé primaire auto-incrémentée dans une table
- ☒ Un objet de la base de données permettant de fournir un compteur
- ☐ Une fonction d'analyse renvoyant une séquence de nombres à partir d'une colonne source
-  Une séquence est un compteur indépendant de toute table.

Exercice

Pour obtenir le prochain numéro d'une séquence, il faut utiliser la fonction...

- ☐ NEXTVAL pour incrémenter, puis CURRVAL pour récupérer le numéro
- ☐ INCREMENT BY X pour indiquer à la séquence de s'incrémenter de X
- ☒ NEXTVAL, qui incrémente et renvoie le nouveau numéro
-  CURRVAL renvoie la valeur actuelle. NEXTVAL procède à l'incrémentation et renvoie la nouvelle valeur, elle n'a pas besoin d'être suivie de CURRVAL.

Exercice p. Solution n°8

Pour le CREATE, il suffit de définir Id en tant que serial PRIMARY KEY.

```

1 CREATE TABLE AnalyseVentes
2 (
3     Id serial PRIMARY KEY,
4     NomVendeur varchar(50) NOT NULL,
5     PrenomVendeur varchar(50) NOT NULL,
6     Exercice int,
7     TotalVentes decimal(15,2),
8     ObjectifAnnuel decimal(15,2)
9 )
10 ;

```

Pour le SELECT, il y a un peu plus de travail.

MAKE_DATE (et non pas MAKEDATE) demande 3 paramètres : année, mois et jour. Il faut donc réécrire ces lignes, d'autant plus que DATE_ADD n'existe pas.

```

1 SELECT
2     CONCAT(PrenomVendeur, ' ', UPPER(NomVendeur)),
3     MAKE_DATE(Exercice, 1, 1),
4     MAKE_DATE(Exercice + 1, 1, 1) - INTERVAL '1 day',
5     TotalVentes,
6     TotalVentes - ObjectifAnnuel,

```

```
7      CASE
8          WHEN TotalVentes > ObjectifAnnuel THEN 'Oui'
9          ELSE 'Non'
10     END
11 FROM AnalyseVentes;
```

Cependant, en considérant que le but souhaité est d'avoir les dates du 1^{er} janvier et du 31 décembre de l'année d'exercice, il y a plus simple.

De manière générale lors d'une migration, il est préférable de comprendre ce qui a été fait et pourquoi, plutôt que de chercher à « traduire mot à mot » une requête.

```
1 SELECT
2     CONCAT(PrenomVendeur, ' ', UPPER(NomVendeur)),
3     MAKE_DATE(Exercice, 1, 1),
4     MAKE_DATE(Exercice, 12, 31),
5     TotalVentes,
6     TotalVentes - ObjectifAnnuel,
7     CASE
8         WHEN TotalVentes > ObjectifAnnuel THEN 'Oui'
9         ELSE 'Non'
10    END
11 FROM AnalyseVentes;
```