




Ressourcesinformatiques

 + QUIZ

Version en ligne

OFFERTE !


pendant 1 an

SQL

Les fondamentaux du langage

(avec exercices et corrigés)

En téléchargement

 scripts

4^e édition

Anne-Christine BISSON



PRÉSENTATION

SQL

Les fondamentaux du langage (avec exercices et corrigés) - (4e édition)

Ce livre sur les fondamentaux du langage SQL s'adresse aux **développeurs** et **informaticiens débutants** appelés à travailler avec un Système de Gestion de Bases de Données Relationnelles (SGBDR) pour stocker et manipuler des données. Son objectif est de décrire les **ordres principaux les plus utilisés du langage SQL** (indépendamment des déclinaisons réalisées par les éditeurs de SGBDR) pour permettre au lecteur de prendre en main rapidement une base de données relationnelle et être capable de **créer des tables**, de **les interroger**, de **les modifier**, **d'insérer** et **de supprimer des lignes**.

Le livre débute par un bref historique sur la création de la norme SQL puis présente quelques notions sur le **modèle relationnel**. Ensuite, chaque chapitre présente une subdivision de SQL : la **création** et la **manipulation des tables** puis la **gestion des données** dans ces tables en incluant les dernières évolutions comme les **fonctions de fenêtrage**. L'auteur enchaîne avec la **sécurité des données** et quelques notions de **transactions**, puis présente la programmation avec quelques éléments de **PL/SQL** et l'étude des **déclencheurs**. Le livre se termine en abordant des thèmes un peu plus complexes comme les **chargements en masse**, les **imports** et **exports de tables**, les notions de performances ou encore les **objets système**.

Les exemples utilisés dans ce livre ont été réalisés avec la version Oracle 19c DB Developer VM, SQL Server 2019 SP1 Developer Edition, MySQL version 8, PostgreSQL en version 12.2 et sont en téléchargement sur cette page.

Auteur(s)

Anne-Christine BISSON

Anne-Christine BISSON est consultante indépendante en informatique décisionnelle. Cette experte conseille sur la conception de bases et entrepôts de données de différents SGDB. A ce titre, elle manipule et agrège des données à partir de sources diverses pour les restituer de façon synthétique dans des entreprises et administrations de différents secteurs. Egalement formatrice, elle prolonge avec ce livre sa volonté de partager ses connaissances sur SQL auprès des lecteurs.

TABLE DES MATIÈRES

Introduction

La définition des données (LDD)

La manipulation des données (LMD)

Les fonctions

La sécurité des données (DCL)

Le contrôle des transactions (TCL)

La programmation

Approfondissement

Les erreurs les plus couramment rencontrées

Annexes

Avant-propos

Préambule

Cet ouvrage s'adresse aux développeurs et aux non-informaticiens qui sont amenés à travailler avec un système de gestion de bases de données relationnelles (SGBDR).

Le but est de décrire le fonctionnement du langage SQL indépendamment des dérivations réalisées par les éditeurs de SGBDR.

Le livre présente les ordres principaux les plus utilisés du langage SQL. Il reste généraliste et n'a pas comme objectif de présenter toutes les options du langage, l'accent est mis sur l'aspect pratique et concret.

Nous démarrons par un bref historique sur la création du langage et les normes SQL, puis quelques notions sur le modèle relationnel. Une description plus détaillée du modèle relationnel existe déjà dans d'autres livres parus aux Éditions ENI.

Les deux chapitres suivants abordent chacun une subdivision de SQL : tout d'abord la création et la manipulation des tables, puis la gestion des données dans ces tables, avant de compléter avec des fonctions.

On enchaîne avec la sécurité des données et quelques notions de transactions, puis la programmation avec quelques concepts de PL/SQL et les déclencheurs, et on termine en abordant des thèmes un peu plus complexes comme les chargements en masse, les imports et exports de tables, des notions de performances et les objets système.

L'objectif de cet ouvrage est de fournir les clés de départ permettant d'être autonome dans la manipulation d'une base de données de type relationnel.

Chaque SGBDR a développé ses propres extensions à la norme SQL, ou a ajouté des fonctions spécifiques. Néanmoins, en maîtrisant les bases du langage SQL, on peut répondre à la majorité des besoins dans l'utilisation d'une base de données relationnelle.

Les exemples indiqués dans ce livre ont été réalisés avec la version Oracle 19c DB Developer VM, SQL Server 2019 SP1 Developer Édition, la version 8 de MySQL et PostgreSQL version 12.2.

Introduction

Un peu d'histoire

Les bases de données sont indispensables dans tous développements informatiques. Toutes les données sont stockées dans la majorité des cas dans une structure de base de données.

On parle de BDD pour désigner le stockage des données et de SGBD pour désigner les éléments qui sont mis à disposition du développeur pour manipuler ces données.

Il existe plusieurs types de base de données.

Il y a les bases de type hiérarchique comme IMS/DL1 que l'on rencontre majoritairement sur les Mainframes. Ces éléments sont organisés comme un arbre avec un niveau de hiérarchie et des pointeurs entre les enregistrements.

Il existe également les bases en réseau (ou Codasyl) comme IDS2 ou SOCRATE qui ne sont pratiquement plus utilisées actuellement, et qui reprennent un peu le modèle hiérarchique mais en permettant de naviguer entre les éléments, pas uniquement de manière descendante.

Depuis quelques années, d'autres types de bases de données sont apparus avec les sites Internet utilisés par des millions d'utilisateurs :

- Base de données qui range les données en colonnes et non en lignes pour gérer des volumes de données énormes comme Bigtable, développée par Google, ou Cassandra, utilisée par Facebook.
- Base de données NoSQL, comme Redis, plus souple avec un schéma défini de façon imprécise, qui se préoccupe de la cohérence finale.
- Base de données de documents, similaire au type de base NoSQL. La plus connue et utilisée est MongoDB.

Le site db-engines.com actualise régulièrement les informations sur la popularité des bases de données, globalement et par catégorie. Il explique sa méthodologie de classement.

Les bases de données de type relationnelles sont apparues dans les années 1980. Elles s'appuient sur les travaux développés par un chercheur, Mr Edgar Codd, travaillant chez IBM sur le modèle relationnel au début des années 1970. Les données sont organisées en tables distinctes sans niveau de hiérarchie. Il n'y a plus de pointeurs mais les données contenues dans les tables permettent de réaliser des liens entre les tables.

La tendance actuelle est aux systèmes qui utilisent plusieurs types de bases de données, comme SQL Server 2016. Il est ainsi possible d'écrire des requêtes qui portent sur des bases relationnelles et NoSQL.

Le langage SQL (*Structured Query Language*) signifie langage d'interrogation structuré. Il a été créé au début des années 1970 par IBM. C'est une start-up nommée Relational Software qui produira la première version commercialisable en 1979. Cette start-up est depuis devenue Oracle Corp.

Le langage SQL se décompose en plusieurs sous-ensembles :

- Le **DDL** pour *Data Definition Language*, qui regroupe les ordres utilisés pour créer, modifier ou supprimer les structures de la base (tables, index, vues, etc.). Il s'agit principalement des ordres CREATE, ALTER et DROP.
- Le **DML** pour *Data Manipulation Language*, qui regroupe les ordres utilisés pour manipuler les données contenues dans la base. Il s'agit principalement des ordres SELECT, INSERT, DELETE et UPDATE.
- Le **DCL** pour *Data Control Language*, qui regroupe les ordres utilisés pour gérer la sécurité des accès aux données. Il s'agit principalement des ordres GRANT et REVOKE.

- Le **TCL** pour *Transaction Control Language*, qui regroupe les ordres utilisés pour gérer la validation ou non des mises à jour effectuées sur la base. Il s'agit principalement des ordres COMMIT et ROLLBACK.

Les normes SQL

La première version de SQL normalisée par l'ANSI date de 1986.

Elle sera suivie de plusieurs versions plus ou moins importantes.

La norme SQL2 ou SQL92 est la plus importante. La majorité des SGBDR existants implémentent cette version.

Puis suivront plusieurs autres évolutions SQL-3, SQL:2003, SQL:2008 et SQL:2011 qui apportent chacune quelques fonctions complémentaires.

Chaque fournisseur de SGBDR a implémenté à sa façon le langage SQL et a ajouté ses propres extensions. Les exemples qui sont proposés dans cet ouvrage pour illustrer les propos ne sont donc pas totalement compatibles avec tous les SGBDR.

Les exemples présentés dans cet ouvrage sont conçus principalement pour les bases les plus utilisées : Oracle, SQL Server, PostgreSQL et MySQL.

Norme	Nom courant	Explications
ISO/CEI 9075:1986	SQL-86 ou SQL-87	Édité par l'ANSI puis adopté par l'ISO en 1987.
ISO/CEI 9075:1989	SQL-89 ou SQL-1	Révision mineure.
ISO/CEI 9075:1992	SQL-92 ou SQL2	Révision majeure.
ISO/CEI 9075:1999	SQL-99 ou SQL3	Expressions rationnelles, requêtes récursives, déclencheurs, types non scalaires et quelques fonctions orientées objets.
ISO/CEI 9075:2003	SQL:2003	Introduction de fonctions pour la manipulation XML, « window functions », ordres standardisés et colonnes avec valeurs autoproduites (y compris colonnes d'identité).
ISO/CEI 9075:2008	SQL:2008	Ajout de quelques fonctions de fenêtrage (ntile, lead, lag, first value, last value, nth value), limitation du nombre de ligne (OFFSET / FETCH), amélioration mineure sur les types distincts, curseurs et mécanismes d'auto-incréments.
ISO/CEI 9075:2011	SQL:2011	Ajout des types de données temporels et des tables temporelles.

Description rapide du modèle relationnel

Le modèle relationnel a été créé, comme indiqué précédemment, par un chercheur, Edgar Codd, travaillant chez IBM au début des années 1970. Celui-ci a travaillé à partir de principes mathématiques simples, la théorie des ensembles et la logique de prédicats.

Le modèle relationnel repose sur la notion d'ensemble. Schématiquement, un ensemble peut être représenté par une table (une table peut également être appelée une relation).

Cet ensemble a des attributs (les colonnes) et des lignes contenant des valeurs (les tuples). La forme la plus couramment utilisée pour représenter une table est celle-ci :

Attributs (colonnes)				
idTypeChambre	NombreLit	TypeLit	TypeSDB	Description
1	1	lit simple	D	1 lit simple avec douche
2	2	lit simple	D	2 lits simples avec douche
3	2	lit simple	DW	2 lits simples avec douche et WC séparés
4	1	lit double	D	1 lit double avec douche
5	1	lit double	DW	1 lit double avec douche et WC séparés
6	1	lit double	BW	1 lit double avec bain et WC séparés
7	1	lit XL	BW	1 lit double large avec bain et WC séparés

Table TypesChambre = Ensemble

Le modèle relationnel présente les données sous forme logique, il est totalement indépendant du modèle physique. C'est le fournisseur qui décide du mode de stockage physique des tables. C'est l'avantage majeur des bases de données relationnelles, l'indépendance entre le logique et le physique.

Une fois les tables définies, il faut disposer d'un langage pour les manipuler, il s'agit de l'algèbre relationnelle. Celui-ci a également été inventé par Edgar Codd. À l'aide de ces opérateurs, on peut interroger les relations existantes et créer de nouvelles relations. On parle d'opérateurs ensemblistes : union, intersection, différence, produit cartésien, division et jointure.

L'algèbre relationnelle est mise en œuvre par le SQL et les systèmes de gestion de bases de données relationnelles (SGBDR) implémentent le modèle relationnel.

1. Principaux concepts du modèle relationnel

Les trois principaux concepts du modèle relationnel sont le domaine, le produit cartésien et les relations.

Domaine

C'est un ensemble de valeurs caractérisé par un nom.

Par exemple :

Le type de salle de bains est un domaine qui comprend les valeurs D, DW ou BW pour les douches, douches avec WC séparés ou baignoires avec WC séparés.

Le type de lit est un autre domaine (lit simple, lit double, lit XL...).

Le nombre d'occurrences de chacun des domaines donne la cardinalité.

Pour les types de lit, la cardinalité est de 3.

Produit cartésien

Celui-ci représente la jonction entre deux domaines. Si par exemple l'on effectue le produit cartésien des types de salle de bains et des types de lit, on obtient des tuples D1,D2.

Dans notre exemple, le produit cartésien du domaine 1 (nombre de lits) et du domaine 2 (type de lit) donne :

(DW, lit simple), (DW, lit double), (DW, lit XL), (BW, lit simple), (BW, lit double), etc.

Relation

La notion de relation est le fondement du modèle relationnel. La relation permet de mettre en relation les domaines selon certains critères.

Par exemple, si l'on veut créer une relation nommée TYPE_SDB_TYPE_LIT, on indiquera que l'on veut associer tous les types de salle de bains du domaine Type_SDB avec le seul élément « Lit double » du domaine Type_Lit.

La représentation de cette relation se fait sous forme de tableau à deux dimensions.

RELATION : TYPE_SDB_TYPE_LIT

TYPE_SDB	TYPE_LIT
D	Lit double
DW	Lit double
BW	Lit double

TYPE_LIT et TYPE_SDB sont des **attributs**.

Chaque ligne est unique et représente un objet de la relation.

Le **degré** est le nombre d'attributs d'une relation (ici = 2).

2. Principales règles

Le modèle relationnel gère donc un objet principal, la relation, associée aux concepts de domaine et d'attribut.

Des règles s'appliquent à cette relation afin de respecter les contraintes liées à l'analyse. Voici quelques-unes de ces règles :

Cohérence

Toute valeur prise par un attribut doit appartenir au domaine sur lequel il est défini.

Unicité

Tous les éléments d'une relation doivent être distincts.

Identifiant

Attribut ou ensemble d'attributs permettant de caractériser de manière unique chaque élément de la relation.

Clé primaire

Identifiant minimum d'une relation.

Clés secondaires

Autres identifiants de la relation.

Intégrité référentielle

Cette règle impose qu'un attribut ou ensemble d'attributs d'une relation apparaisse comme clé primaire dans une autre relation.

Clé étrangère

Attribut ou ensemble d'attributs vérifiant la règle d'intégrité référentielle.

Exemples

RELATION TYPECHAMBRE

NUMERO	NOMBRE_LIT	TYPE_LIT	TYPE_SDB	DESCRIPTION
1	1	lit simple	D	1 lit simple avec douche
2	2	lit simple	D	2 lits simples avec douche
3	2	lit simple	DW	2 lits simples avec douche et WC séparés
4	1	lit double	D	1 lit double avec douche
5	1	lit double	DW	1 lit double avec douche et WC séparés

NUMERO est l'identifiant primaire.

NOMBRE_LIT et DESCRIPTION sont des clés secondaires.

TYPE_LIT et TYPE_SDB sont des clés étrangères référençant les clés primaires des relations de TYPELIT et TYPESDB.

RELATION HOTEL

HOTEL	LIB_HOTEL	ETOILE
1	Ski Hotel	*
2	Art Hotel	**
3	Rose Hotel	***
4	Lions Hotel	****

HOTEL est l'identifiant primaire.

ETOILE est l'identifiant secondaire.

Valeur nulle

Dans le modèle relationnel, la notion de nullité est admise. C'est une valeur représentant une information inconnue ou inapplicable dans une colonne. Elle est notée , ^ ou NULL.

Contrainte d'entité

Toute valeur participant à une clé primaire doit être non NULL.

Exemple

RELATION CHAMBRE

NUMERO	HOTEL	TYPE_CHAMBRE	COMMENTAIRE
1	1	1	
2	1	1	En travaux
3	2	3	

NUMERO	HOTEL	TYPE_CHAMBRE	COMMENTAIRE
4	2	4	
5	3	5	Vue sur lac

Le commentaire peut être non renseigné alors que le numéro, qui est la clé primaire, est obligatoire.

L'algèbre relationnelle

1. Généralités

L'algèbre relationnelle a conduit à la mise au point du SQL qui est devenu le standard en ce qui concerne la gestion des données.

C'est une méthode d'extraction permettant la manipulation des tables et des colonnes. Son principe repose sur la création de nouvelles tables (tables résultantes) à partir des tables existantes, ces nouvelles tables devenant des objets utilisables immédiatement.

Les opérateurs de l'algèbre relationnelle permettant de créer les tables résultantes sont basés sur la théorie des ensembles.

La syntaxe et les éléments de notations retenus ici sont les plus couramment utilisés.

2. Les opérateurs

a. Union

L'union entre deux relations de même structure (degré et domaines) donne une table résultante de même structure ayant comme éléments l'ensemble des éléments distincts des deux relations initiales.

Notation : $R_x = R_1 \cup R_2$

Exemples

Soit les relations *HOTELS_EUROPE* et *HOTELS_AFRIQUE*

RELATION HOTELS_EUROPE

idHotel	Libelle	Etoile
1	Ski Hotel	*
2	Art Hotel	**

RELATION HOTELS_AFRIQUE

idHotel	Libelle	Etoile
1	Ski Hotel	*
2	Lions Hotel	****

UNION DES DEUX RELATIONS

idHotel	Libelle	Etoile
1	Ski Hotel	*

idHotel	Libelle	Etoile
2	Art Hotel	**
4	Lions Hotel	****

b. Intersection

L'intersection entre deux relations de même structure (degré et domaines) donne une table résultante de même structure ayant comme éléments l'ensemble des éléments communs aux deux relations initiales.

Notation : $R_x = R_1 \cap R_2$

Exemple

RELATION SKIHOTEL_TYPECHAMBRE

NumChambre	Description
1	1 lit simple avec douche
4	1 lit double avec douche
6	1 lit double avec bain et WC séparés

RELATION ARTHOTEL_TYPECHAMBRE

NumChambre	Description
1	1 lit simple avec douche
4	1 lit double avec douche
6	1 lit double avec bain et WC séparés
7	1 lit double large avec bain et WC séparés

TYPECHAMBRE communes aux deux relations :

NumChambre	Description
1	1 lit simple avec douche
4	1 lit double avec douche
6	1 lit double avec bain et WC séparés

c. Différence

La différence entre deux relations de même structure (degré et domaines) donne une table résultante de même structure ayant comme éléments l'ensemble des éléments de la première relation qui ne sont pas dans la deuxième.

Notation : $R_x = R_2 - R_1$

Exemple

TYPECHAMBRE présente dans la relation 2 et pas dans la relation 1 :

NumChambre	Description
7	1 lit double large avec bain et WC séparés

d. Division

La division entre deux relations est possible à condition que la relation diviseur soit totalement incluse dans la relation dividende. Le quotient de la division correspond à l'information qui, présente dans le dividende, n'est pas présente dans le diviseur.

Il est également possible de définir la division de la façon suivante : soit R1 et R2 des relations telles que R2 soit totalement inclus dans R1. Le quotient $R1 \div R2$ est constitué des tuples t tels que pour tout tuple t' défini sur R2, il existe le tuple t.t' défini sur R1.

Notation : $R_x = R2 \div R1$

Exemple

RELATION SKIHOTEL_TYPECHAMBRE

NumChambre	Description
1	1 lit simple avec douche
4	1 lit double avec douche
6	1 lit double avec bain et WC séparés

RELATION ARTHOTEL_TYPECHAMBRE

NumChambre	Description	Prix
1	1 lit simple avec douche	57.49
4	1 lit double avec douche	68.99
6	1 lit double avec bain et WC séparés	91.99
4	1 lit double large avec bain et WC séparés	103.49

La division entre les deux relations permet d'isoler l'information complémentaire à la relation SKIHOTEL_TYPECHAMBRE et présente dans la relation ARTHOTEL_TYPECHAMBRE :

Prix
57.49
68.99
91.99

e. Restriction

La restriction repose sur une condition. Elle produit, à partir d'une relation, une relation de même schéma n'ayant que les éléments de la relation initiale qui répondent à la condition.

Notation : $R_x = s(\text{condition}) R1$

La condition s'exprime sous la forme :

[NON] [(| attribut opérateur valeur |)] [{ET/OU}condition]

opérateur : un opérateur de comparaison (=, <>, >, <, >=, <=).

valeur : une constante ou un autre attribut.

Exemple

Hôtels avec un type de chambre = 7

TypesChambre7= σ (idTypeChambre=7)Hotels

idHotel	Libelle	idTypeChambre	Description
2	Art Hotel	7	1 lit double large avec bain et WC séparés
3	Rose Hotel	7	1 lit double large avec bain et WC séparés
4	Lions Hotel	7	1 lit double large avec bain et WC séparés

f. Projection

La projection d'une relation sur un groupe d'attributs donne une relation résultante ayant comme schéma uniquement ces attributs, et comme éléments les n-uplets distincts composés par les valeurs associées de ces attributs.

Notation : $R_x = \rho R (A_1, A_2, \dots, A_n)$

Exemple sur NombreLit et Description de la table TypesChambre :

LISTTypCham=? TypesChambre(NombreLit,Description)

NombreLit	Description
1	1 lit simple avec douche
2	2 lits simples avec douche
3	3 lits simples avec douche et WC séparés
1	1 lit double avec douche

S'il y avait eu des doublons, ils auraient été supprimés du résultat.

g. Produit cartésien

Le produit cartésien entre deux relations produit une relation ayant comme schéma tous les attributs des deux relations existantes et comme éléments l'association de chaque ligne de la première table avec chaque ligne de la deuxième.

Notation : $R_x = S_1 \times S_2$

Exemple

RELATION Hotels

IdHotel	Libelle
1	Ski Hotel
2	Art Hotel

IdHotel	Libelle
4	Lions Hotel

RELATION TypesChambre

idTypeChambre	Description
1	1 lit simple avec douche
2	2 lits simples avec douche
3	3 lits simples avec douche et WC séparés

HOTELTypeCh = Hotels X TypesChambre

IdHotel	Libelle	idTypeChambre	Libelle
1	Ski Hotel	1	1 lit simple avec douche
2	Art Hotel	1	1 lit simple avec douche
3	Rose Hotel	1	1 lit simple avec douche
1	Ski Hotel	2	2 lits simples avec douche
2	Art Hotel	2	2 lits simples avec douche
3	Rose Hotel	2	2 lits simples avec douche
1	Ski Hotel	3	3 lits simples avec douche et WC séparés
2	Art Hotel	3	3 lits simples avec douche et WC séparés
3	Rose Hotel	3	3 lits simples avec douche et WC séparés

h. Jointure

La jointure entre deux relations est produite par la restriction sur le produit cartésien.

Notation : $R_x = S_1 \text{ JOIN (condition) } S_2$.

Exemple

RELATION Chambres

IdChambre	TypeChambre
1	1
2	2
3	3

RELATION TypesChambre

TypeChambre	Description
1	1 lit simple avec douche
2	2 lits simples avec douche
3	3 lits simples avec douche et WC séparés

JOINTURE entre ces deux relations :

ChambreTypeChambre = Chambres JOIN (Chambres.IdChambre=TypesChambre.idTypeChambre)

TypesChambre.

IdChambre	TypeChambre	idType Chambre	Description
1	1	1	1 lit simple avec douche
2	2	2	2 lits simples avec douche
3	3	3	3 lits simples avec douche et WC séparés

Les différents types de jointure sont :

Thêta-jointure

La condition est une comparaison entre deux attributs.

Équi-jointure

La condition porte sur l'égalité entre deux attributs.

Jointure naturelle

Équi-jointure entre les attributs portant le même nom.

i. Calculs élémentaires

Projection sur une relation associée à un calcul portant sur chaque ligne pour créer un ou plusieurs nouveaux attributs.

Notation : $R_x = p \ S \ (A_1, \dots, N_1 = \text{expression calculée}, \dots)$

L'expression calculée peut être :

- une opération arithmétique,
- une fonction mathématique,
- une fonction portant sur une chaîne.

j. Calcul d'agrégats

Projection sur une relation associée à un ou des calculs statistiques portant sur un attribut pour tous les éléments de la relation ou du regroupement lié à la projection afin de créer un ou plusieurs nouveaux attributs.

Notation : $R_x = p \ S \ (A_1, \dots, N_1 = \text{fonction statistique } (A_x), \dots)$

Les principales fonctions statistiques sont :

- **COUNT (*)** : nombre de lignes.
- **COUNT (attribut)** : nombre de valeurs non nulles.

- **SUM (attribut)** : somme des valeurs non nulles.
- **AVG (attribut)** : moyenne des valeurs non nulles.
- **MAX (attribut)** : valeur maximum (non nulle).
- **MIN (attribut)** : valeur minimum (non nulle).

Les systèmes de gestion de bases de données utilisant SQL

Les SGBDR sur le marché sont assez nombreux. Les plus utilisés sont Oracle, MySQL, SQL Server (Microsoft), PostgreSQL et DB2 (IBM) notamment dans les entreprises qui ont des volumes de données importants et des contraintes de performance transactionnelles fortes.

Nous n'allons pas ici réaliser un comparatif de tous ces SGBDR, chacun a ses spécificités, ses avantages et ses inconvénients. Le choix d'une base de données est une décision qui doit être bien réfléchie, en fonction de critères tels que le volume des données, les types de données, les contraintes de performance, les coûts d'achat et de maintenance, les connaissances des développeurs, etc.

La définition des données (LDD)

Les types de données

Nous abordons dans cette section les types de données les plus utilisés pour la description des colonnes d'une table. Il existe trois grandes familles de données : numérique, caractère (ou alphanumérique) et temporelle (dates et heures).

Chaque SGBDR a décliné des types spécifiques pour un besoin précis comme les types géographiques, ou pour des problématiques de stockage. Le type choisi dépendra donc de la précision recherchée, tout en tenant compte de la taille nécessaire pour stocker la donnée. Un bon compromis permet d'accéder rapidement à la valeur.

1. Numériques

Les types numériques permettent de définir si l'on souhaite un entier, un décimal ou un nombre à virgule flottante.

Les nombres entiers :

Type	Précision	Stockage (octets)	BDD
TINYINT	0 à 255	1	SQL Server
TINYINT	0 à 255 ou -128 à 127	1	MySQL
SMALLINT	-32 768 à 32 768 ou 0 à 65 535	2	SQL Server, PostgreSQL
SMALLINT	-32 768 à 32 768 ou 0 à 65 535 pour MySQL	2	MySQL
SMALLSERIAL	1 à 32 767. Entier à incrémentation automatique	2	PostgreSQL
MEDIUMINT	-8 388 608 à 8 388 607 ou 0 à 16 777 215	3	MySQL
INT ou INTEGER	-2 147 483 648 à 2 147 483 647 ou 0 à 4 294 967 295	4	SQL Server, PostgreSQL
INT(p)	-2 147 483 648 à 2 147 483 647 ou 0 à 4 294 967 295 où p désigne le nombre de chiffres maximum	4	MySQL, Oracle
SERIAL	1 à 2 147 483 647. Entier à incrémentation automatique ou 1 à 9 223 372 036 854 775 807 pour MySQL	4	PostgreSQL, MySQL
BIGINT	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 ou 0 à 18 446 744 073 709 551 615 pour MySQL	8	SQL Server, PostgreSQL, MySQL
BIGSERIAL	1 à 9 223 372 036 854 775 807. Entier à incrémentation automatique	8	PostgreSQL
BIT	1, 0 ou NULL	1	SQL Server

Les nombres décimaux et flottants :

DECIMAL(p,s) ou NUMERIC(p,s)	-10 ³⁸ +1 à 10 ³⁸ -1. p représente la précision, c'est-à-dire le nombre total et maximum de chiffres à gauche et à droite de la virgule. La précision par défaut est 18. s représente l'échelle, c'est-à-dire le nombre de chiffres maximum après la virgule. L'échelle est optionnelle. La valeur par défaut est 0. Il peut être utilisé pour les devises.	5 à 17 selon la précision et l'échelle	SQL Server, PostgreSQL, MySQL
------------------------------	---	--	-------------------------------

FIXED (p[,s])	Idem DECIMAL	Idem DECIMAL	MySQL
NUMBER(p,s)	Entier ou décimal. p représente la précision et peut aller jusqu'à 40. s représente l'échelle de -84 à 127 où le chiffre négatif correspond à l'arrondi.	1 à 22 selon la précision et l'échelle	Oracle
FLOAT(n)	-1,79E+308 à -2,23E-308, 0 et 2,23E-308 à 1,79E+308. n représente le nombre de bits utilisés pour stocker la donnée. Sa valeur par défaut est 53. Il peut être utilisé pour les calculs scientifiques.	4 pour n entre 1 et 24 ou 8 pour n entre 25 et 53. 22 maxi pour Oracle	SQL Server, Oracle, MySQL
DOUBLE(n) ou DOUBLE PRECISION(n)	n est le nombre maximum de décimaux utilisés de 1 à 15.	8	PostgreSQL, MySQL
REAL(n)	-3,40E+38 à -1,18E-38, 0 et 1,18E-38 à 3,40E+38. n représente le nombre de bits utilisés pour stocker la donnée. Sa valeur par défaut est 24.	4	SQL Server, MySQL, PostgreSQL

Les données

monétaires :

MONEY	-922 337 203 685 477,5808 922 337 203 685 477,5807	à 8	SQL Server, PostgreSQL
SMALLMONEY	-214 748,3648 à 214 748,3647	4	SQL Server

Selon le type de base de données, il existe des propriétés qui complètent le type de données comme :

- IDENTITY(s, i) dans SQL Server, où s représente la valeur de la première ligne insérée et i le pas entre les lignes suivantes, IDENTITY dans Oracle, AUTO_INCREMENT dans MySQL pour incrémenter automatiquement un entier.
- UNSIGNED dans MySQL pour préciser si l'on accepte ou non les nombres négatifs.
- ZEROFILL dans MySQL permet de remplacer les espaces par des 0. L'attribut UNSIGNED est automatiquement ajouté avec ZEROFILL.

Exemple Oracle

PRIX NUMBER (8,2) ou PRIX DECIMAL (8,2) sont identiques.

IDCHAMBRE NUMBER (5) ou IDCHAMBRE NUMBER (5,0) sont identiques.

Dans le premier exemple, on décrit un type décimal de 8 chiffres de longueur dont deux décimales, comme 123456,12.

Dans le deuxième exemple, on ne précise pas de longueur après la virgule donc il s'agit d'un entier.

MySQL, de son côté, a également fusionné les deux types numériques dans le type DECIMAL.

Exemple MySQL

PRIX DECIMAL (8,2)

IDCHAMBRE DECIMAL (5)

NUMCHAMBRE SMALLINT

2. Caractères

Le type alphanumérique classique se note CHAR pour les données de taille fixe et VARCHAR pour les chaînes de caractères de longueur variable.

La majorité du temps, il est conseillé d'utiliser VARCHAR. En effet, une donnée déclarée sur 50 par exemple et qui ne contient que deux caractères sera effectivement stockée sur deux caractères avec VARCHAR alors qu'elle sera stockée sur 50 caractères avec CHAR. CHAR complète avec des espaces jusqu'à la longueur

maximum. NCHAR ou NVARCHAR sont utilisés pour stocker des chaînes de caractères comprenant des caractères spéciaux comme le chinois, l'hébreu, le russe... Le nombre d'octets est doublé.

Dans le cas de VARCHAR, la longueur effective de la chaîne est stockée sur un octet, permettant ainsi au SGBDR de restituer correctement la donnée.

Il est possible de trouver le type TEXT, mais ce type est amené à disparaître. Ce n'est pas un standard SQL.

Dernière remarque, Oracle préconise d'utiliser le type VARCHAR2 à la place du VARCHAR. A priori, ces deux types sont identiques, mais Oracle pourrait utiliser le VARCHAR différemment dans ses versions futures.

Type	Précision	Stockage (octets)	BDD
CHAR(n)	Jusqu'à 255 caractères pour MySQL.	n octets (dépend de la sémantique dans Oracle et limité à 2 000 octets).	SQL Server, MySQL, PostgreSQL, Oracle
NCHAR(n)	Type de données unicode dans Oracle limité à 1 000 ou 2 000 selon le jeu de caractères.	n octets * 2 (selon le codage dans Oracle)	SQL Server, Oracle
CHARACTER(n)	Chaîne de caractères à longueur variable.	x octets	PostgreSQL
VARCHAR(n)	0 à 65 535 caractères pour MySQL. (n) peut être remplacé par (max) dans SQL Server.	Longueur chaîne + 1 octet	SQL Server, MySQL, PostgreSQL, Oracle
NVARCHAR(n)	Chaîne de caractères à longueur variable.	(n * 2) + 2 - 2 Go maxi	SQL Server
CHARACTER VARYING	(n) est optionnel. S'il n'est pas précisé, toute taille de chaîne est acceptée.	Longueur chaîne + 1 à 4 octets selon la longueur, limité à 1 Go	PostgreSQL
VARCHAR2(n) CHAR ou BYTE)	Chaîne de caractères à longueur variable.	4 000 ou 32 767	Oracle
NVARCHAR2	Type de données Unicode dans Oracle limité à 16 383 ou 32 736 selon le jeu de caractères.	Limité à 2 000 ou 16 383 octets - 4 000 ou 32 736 selon le jeu de caractères ou la propriété MAX_STRING_SIZE de la base (STANDARD ou EXTENDED).	Oracle

Selon le type de base de données, il existe des propriétés qui complètent le type de données.

Par exemple, dans MySQL, sur le type CHAR, il est possible d'ajouter des options :

- BINARY est utilisé pour tenir compte de la casse.
- NATIONAL spécifie que le jeu de caractères par défaut de MySQL doit être utilisé.
- ASCII spécifie le jeu de caractères latin1 à utiliser.

- UNICODE spécifie le jeu de caractères à utiliser.

Exemple

```
TYPE      CHAR(2),
COULEUR   VARCHAR(25));
```

3. Dates et heures

Les types de format temporels sont principalement DATE, TIME et TIMESTAMP.

Les formats par défaut des dates ou des heures varient d'une base de données à l'autre. Pour connaître le format par défaut il faut utiliser :

- Pour SQL Server :

```
SELECT GETDATE();
```

- Pour PostgreSQL :

```
SELECT current_date;
```

- Pour Oracle :

```
SELECT SYSDATE FROM DUAL;
```

- Pour MySQL :

```
SELECT NOW();
```

qui donnent la date du jour au format utilisé par la base.

Il est préférable de ne pas s'appuyer sur un format défini pour une base de données. En effet, ce format peut évoluer ou la séquence de code s'appliquer sur une autre base qui n'aura pas le même format de date par défaut. Nous verrons dans la suite de cet ouvrage que la manipulation des dates et des heures est très variable d'un SGBDR à l'autre, les fonctions de conversion notamment sont différentes (cf. Les fonctions de gestion des dates et heures - Les différents formats d'affichage des dates du chapitre Les fonctions).

Lors de la création de vos tables, il n'est pas utile de connaître le format de stockage, il faut déclarer la colonne DATE, TIME ou DATETIME selon la précision attendue.

- DATE : format date, norme ISO AAAA-MM-JJ
- TIME : format heure, norme ISO HH:MM:ss.nnnnnnn
- TIMESTAMP : format date et heure, norme ISO AAAA-MM-JJ HH:MM:ss .nnn

Attention : Oracle ne connaît pas le type TIME, il faut utiliser le type TIMESTAMP, ou stocker l'heure dans un format CHAR(10) puis utiliser les formats de conversion ou stocker les heures en les convertissant en secondes dans un champ de type NUMBER.

Type	Précision	Stockage (octets)	BDD
DATETIME	Milliseconde (du 01/01/1753 au 31/12/9999)	8 octets	SQL Server
DATETIME	Nanoseconde (du 01/01/1000 au 31/12/9999)	8 octets	MySQL
SMALLDATETIME	Minute (du 01/01/1900 au 06/06/2079)	4 octets	SQL Server
DATETIME2	Nanoseconde (du 01/01/0001 au 31/12/9999)	6 à 8 octets	SQL Server
DATE	Jour (du 01/01/0001 au 31/12/9999)	3 octets	SQL Server, Oracle, MySQL
DATE	Jour (4713 avant JC - 31/12/5874897)	4 octets	PostgreSQL
TIME	Nanoseconde	3 à 5 octets	SQL Server, MySQL
TIME	Microseconde	8 ou 12 octets (sans ou avec timezone)	PostgreSQL

Type	Précision	Stockage (octets)	BDD
TIMESTAMP	Date et heure courantes	8 octets	MySQL
TIMESTAMP	Microseconde (4713 avant JC - 31/12/294276)	8 octets (avec ou sans timezone)	PostgreSQL
DATETIMEOFFSET	Nanoseconde (du 01/01/0001 au 31/12/9999 - affiche le fuseau horaire)	8 à 10 octets	SQL Server
INTERVAL	Microseconde (intervalles de temps 178000000 années)	16 octets	PostgreSQL

Exemple

DATE_ACHAT DATE,
DATE_MAJ TIMESTAMP

4. Les types binaires

Oracle utilise les types suivants :

Types numériques	Valeur min	Valeur max	Remarque
RAW	1	32 767 octets	Entrée en hexadécimal
LONG RAW	1	2 Go	Entrée en hexadécimal
BLOB	1	4 Go	Gros objet binaire
BFILE	1	4 Go	Pointeur vers un fichier binaire externe

Cette liste n'est pas exhaustive, consultez la documentation de la base de données pour détailler tous les types possibles.

Données de type BLOB :

Type	Précision	Stockage (octets)	BDD
TINYBLOB	1 à 255 caractères (sensible à la casse)	Longueur chaîne + 1 octet	MySQL
BLOB	1 à 65 535 caractères (sensible à la casse)	Longueur chaîne + 1 octet	MySQL
MEDIUMBLOB	1 à 16 777 215 caractères (sensible à la casse)	Longueur chaîne + 3 octets	MySQL
LOB	1 à 4 294 967 295 caractères (sensible à la casse)	Longueur chaîne + 4 octets	MySQL

5. Autres types de données

Selon les bases de données, il existe d'autres types de données comme :

- géographiques,
- spatiales,
- spécifiques comme URL.

Il est aussi possible de créer son propre type de données.

La création de tables

Dans cette section, nous allons voir comment créer une table, ajouter ou supprimer des colonnes, mettre des commentaires sur les colonnes et les tables, mais également la méthode pour copier une table dans une autre puis comment attribuer un synonyme à un nom de table.

1. L'ordre CREATE

CREATE est l'ordre de base en langage SQL pour créer un élément. Celui-ci sert à créer une table, un index, une vue ou un synonyme. En fonction du besoin, la syntaxe est différente.

Dans cette section, nous allons traiter de la création de tables. Une table se définit principalement par les colonnes qui la composent et les règles qui s'appliquent à ces colonnes.

Nous n'aborderons pas les aspects de stockage physique de la table. En effet, chaque SGBDR a sa propre syntaxe dans ce domaine. Dans la majorité des cas, ce sont les DBA (*Database Administrator*, administrateur de base de données) qui spécifient les normes de stockage et les options à appliquer sur les tables. Le stockage des tables est un élément déterminant en termes de performance et de sécurité de la base de données. Il est donc fortement conseillé de se rapprocher d'un administrateur avant de se lancer dans la création d'une table.

Pour supprimer une table, on utilise l'ordre DROP TABLE.

Avant de créer une table, il est préférable de se poser quelques questions et de respecter certaines règles.

Essayez de donner des noms parlants aux tables et aux colonnes afin de retrouver ensuite facilement une information. Les DBA définissent la plupart du temps les règles de nommage des tables et des colonnes. Ils fournissent quelquefois des scripts standards pour créer une table. Il est donc impératif de consulter les normes en vigueur dans l'entreprise avant toute création.

Les règles minimums à respecter dans le nommage d'une table ne sont pas très nombreuses et se résument ainsi : un nom de table commence par une lettre, il est unique et il ne doit pas dépasser 256 caractères.

Quand une colonne a la même signification dans plusieurs tables, il est conseillé de garder le même nom. En effet, si la colonne se retrouve dans plusieurs tables et que le modèle relationnel a été respecté, cela signifie que la colonne est la clé d'une table. Donc, en gardant le même nom, on simplifie ensuite les jointures entre ces deux tables, les colonnes de clé seront facilement identifiables. Cela permet à certains outils, comme Power BI, de proposer les relations lorsqu'elles n'existent pas.

Nous verrons, dans le chapitre La manipulation des données (LMD) - section L'utilisation des alias, comment distinguer deux colonnes de même nom dans une même sélection.

Syntaxe

```
CREATE TABLE nom_de_table Nom_colonne Type_colonne,  
                Nom_colonne Type_colonne,  
                Nom_colonne Type_colonne,  
                ... );
```

Voici trois structures de tables.

Table TARIFS

Tarifs
IdTarif
Hotel
TypeChambre
DateDebut
DateFin
Prix

Table HOTELS

Hotels
idHotel
Libelle
Etoile

Table TypesChambre

TypesChambre
idTypeChambre
NombreLit
TypeLit

TypesChambre
Description

Les scripts nécessaires pour créer les tables ci-dessus sont :

Syntaxe SQL Server et PostgreSQL

```
CREATE TABLE TARIFS (idTarif    INTEGER,
                    hotel        INTEGER,
                    typeChambre INTEGER,
                    DateDebut    DATE,
                    DateFin      DATE,
                    Prix          MONEY);
```

Syntaxe MySQL (remplacement du type MONEY par DECIMAL)

```
CREATE TABLE TARIFS (idTarif    INTEGER,
                    hotel        INTEGER,
                    typeChambre INTEGER,
                    DateDebut    DATE,
                    DateFin      DATE,
                    Prix          DECIMAL(7,3));
```

Syntaxe Oracle

```
CREATE TABLE TARIFS (idTarif    NUMBER(38,0),
                    hotel        NUMBER(38,0),
                    typeChambre NUMBER(38,0),
                    DateDebut    DATE,
                    DateFin      DATE,
                    Prix          DECIMAL(7,3));
```

Syntaxe SQL Server, PostgreSQL et MySQL

```
CREATE TABLE Hotels (idHotel    INTEGER,
                    Libelle      VARCHAR(50),
                    Etoile       VARCHAR(5));

CREATE TABLE TypesChambre (idTypeChambre INTEGER,
                    NombreLit    INTEGER,
                    TypeLit      VARCHAR(20),
                    Description   VARCHAR(255));
```

Il s'agit d'exemples simples. Aucune colonne n'a de restriction ou de valeur par défaut. Aucune clé n'a été définie.

Nous pouvons constater que les colonnes idHotel et idTypeChambre sont les clés des tables Hotels et TypesChambre et que l'on retrouve ces colonnes sous le nom hotel et typeChambre dans la table Tarifs. Ainsi, si nous avons des jointures à réaliser entre ces tables, nous utiliserons ces colonnes.

2. Tables temporaires

Il est possible de créer des tables temporaires. Comme leur nom l'indique, ces tables ont une durée de vie limitée qui correspond à la session dans laquelle elles sont créées. Elles sont supprimées automatiquement à la fermeture de la session. Elles sont utilisées dans les fonctions ou procédures stockées, comme une table. La syntaxe est quasi identique à celle de la création de tables. Les tables temporaires globales peuvent être utilisées par plusieurs sessions.

Syntaxe SQL Server

Il faut juste ajouter un # devant le libellé de la table locale et deux ## pour les globales.

```
CREATE TABLE #TARIFS2021 (idTarif    INTEGER,
                    hotel        INTEGER,
                    typeChambre INTEGER,
                    DateDebut    DATE,
```

```
DateFin    DATE,  
Prix       MONEY);
```

Syntaxe MySQL

```
CREATE TEMPORARY TABLE TMP_TARIFS2021 (idTarif    INTEGER,  
    hotel    INTEGER,  
    typeChambre INTEGER,  
    DateDebut DATE,  
    DateFin   DATE,  
    Prix      DECIMAL(7,3));
```

Syntaxe Oracle

```
CREATE LOCAL TEMPORARY TABLE TMP_TARIFS2021 (idTarif  
NUMBER(38,0),  
    hotel    NUMBER(38,0),  
    typeChambre NUMBER(38,0),  
    DateDebut DATE,  
    DateFin   DATE,  
    Prix      DECIMAL(7,3))
```

ON COMMIT PRESERVE ROWS;

Pour créer une table temporaire globale, il suffit de remplacer LOCAL par GLOBAL dans la syntaxe ci-dessus.

Pour que les lignes soient supprimées à chaque validation, remplacer l'ordre PRESERVE par DELETE.

Syntaxe PostgreSQL

```
CREATE TEMPORARY TABLE TMP_TARIFS2021 (idTarif    INTEGER,  
    hotel    INTEGER,  
    typeChambre INTEGER,  
    DateDebut DATE,  
    DateFin   DATE,  
    Prix      MONEY);
```

3. Les commentaires (COMMENT)

La clause COMMENT permet d'ajouter des commentaires sur les colonnes et les tables. Cette option est très utile pour les développeurs futurs qui pourront ainsi connaître la fonction d'une table et le contenu de chaque colonne en affichant sa description.

Cela ne dispense pas d'avoir une documentation papier décrivant chaque table dans la base de données.

Néanmoins, en prenant l'habitude de documenter toutes les tables et les colonnes directement dans la base au moment de la création de celles-ci, automatiquement la maintenance et l'évolution de la base s'en trouvent simplifiées.

Une base de données a une durée de vie conséquente et va donc être utilisée par une population importante et hétéroclite.

Les commentaires seront d'autant plus utiles que la base est ancienne et que la documentation papier n'est plus toujours disponible, notamment en cas de migration de base de données ou de migration d'un système à un autre.

La syntaxe de cette clause diffère d'un SGBDR à un autre.

Dans Oracle et PostgreSQL, la syntaxe est la suivante :

```
COMMENT ON <'COLUMN' ou 'TABLE'> <Nom colonne ou nom de table> IS  
'libellé libre';
```

- Ajouter un commentaire sur une table :

```
COMMENT ON TABLE Tarifs IS 'Tarif par hotel et type de chambre';
```

- Ajouter un commentaire sur une colonne :

```
COMMENT ON COLUMN Hotels.idHotel IS 'Numéro 'Numéro de l"hotel";
```

Les commentaires sont stockés dans deux tables Oracle :

- user_tab_comments : les commentaires sur les tables.

- user_col_comments : les commentaires sur les colonnes.

Dans MySQL, les commentaires seront mis au moment de la création de la table en ajoutant l'ordre COMMENT ainsi :

```
CREATE TABLE Hotels(
  idHotel int NOT NULL PRIMARY KEY COMMENT 'Numéro de l"hotel',
  Libelle varchar(50) NULL COMMENT 'Dénomination de l"hotel',
  Etoile varchar(5) NULL COMMENT 'Nombre d"étoiles obtenues par
l"hotel'
);
```

Pour afficher les colonnes et les commentaires, il faut utiliser cette commande :

```
SHOW FULL COLUMNS FROM Hotels;
```

Dans SQL Server, il n'existe pas de propriété spécifique pour ajouter un commentaire. Il est possible de contourner ce manque en spécifiant une propriété étendue avec la syntaxe suivante :

```
EXEC sys.sp_addextendedproperty @name=N'Description', @value=N'Liste des
chambres', @level0type=N'SHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Chambres'
```

Pour les colonnes, la propriété étendue Description existe et peut être incrémentée en utilisant cette commande :

```
EXEC sys.sp_addextendedproperty @name=N'MS_Description',
@value=N'N° identifiant de chambre indépendant de l"hôtel',
@level0type=N'SHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Chambres',
@level2type=N'COLUMN',@level2name=N'idChambre' ;
```

Dans PostgreSQL, il existe une commande COMMENT pour ajouter un commentaire sur une table ou tout autre objet de la base comme la colonne.

```
COMMENT ON TABLE matable IS 'Ceci est ma table.';
```

```
COMMENT ON COLUMN matable.macolonne IS 'Ceci est ma colonne';
```

4. Créer une table à partir d'une sélection de données

Une autre méthode pour créer une table consiste à dupliquer ou à s'inspirer de tables existantes.

Cette méthode est souvent utilisée par les développeurs pour créer des tables de test et des jeux d'essais. Il n'est pas souhaitable d'utiliser cette méthode pour la définition initiale de la base de données.

La création de table associée à un ordre de sélection de colonne d'une autre table ou de plusieurs tables permet de générer rapidement la structure d'une table spécifique et, dans le même ordre, de remplir cette table.

Avec cette syntaxe, les possibilités sont très étendues, étant donné que l'on peut utiliser toutes les possibilités de l'ordre SELECT.

Il est possible de récupérer certaines colonnes, de filtrer les données, mais également de copier uniquement la structure sans les données en ajoutant une restriction qui n'est jamais vérifiée.

Syntaxe

```
CREATE TABLE <Nouvelle table> AS SELECT <nom colonne1>,<nom
colonne2>... ou <*> FROM <Table à copier> WHERE ... ... ;
```

Exemple : copie de structure et de données

Avec Oracle et PostgreSQL, il faut utiliser la syntaxe suivante :

```
CREATE TABLE SAUV_Chambres AS SELECT * FROM Chambres;
```

Avec cet ordre, on copie toute la structure de la table Chambres ainsi que tout son contenu dans la table SAUV_Chambres.

Attention : avec les très grosses tables, il existe un risque de rencontrer des problèmes d'espace disque ou de temps de réponse importants. Il est préférable de tester la requête SELECT avant de lancer la création proprement dite.

Par exemple, il est possible de compter le nombre de lignes de la table Chambres par un SELECT COUNT(*) FROM Chambres qui permet de connaître le volume à transférer.

Avec MySQL, la syntaxe est la suivante (le AS disparaît) :

```
CREATE TABLE SAUV_Chambres SELECT * FROM Chambres;
```

Avec SQL Server, la syntaxe est la suivante :

```
SELECT * INTO Chambres_SAUV
```

```
FROM Chambres;
```

Pour récupérer uniquement la structure de la table d'origine, il faut ajouter une clause qui n'est jamais vérifiée. Nous verrons, dans le chapitre La manipulation des données (LMD) - section La sélection de données, la description détaillée de la clause SELECT.

Exemple : copie de structure

```
CREATE TABLE SAUV_Chambres AS SELECT * FROM Chambres WHERE 1=0;
```

La clause WHERE n'étant jamais vraie, le SELECT ramène zéro ligne.

Il est également possible de ne sélectionner que quelques colonnes de la table d'origine.

Exemple : copie d'une partie de la structure

```
CREATE TABLE SAUV_Chambres AS SELECT idChambre, Hotel FROM  
Chambres WHERE 1=0;
```

La table SAUV_Chambres contient seulement deux colonnes et sera vide.

SAUV_Chambres
idChambre
Hotel

Avec MySQL, il faut copier la structure d'une table sur une autre avec l'utilisation d'un LIKE.

```
CREATE TABLE SAUV_Chambres LIKE Chambres;
```

Il est également possible de ne sélectionner que quelques lignes de la table d'origine.

Exemple : copie d'une partie de la structure et sélection de lignes (syntaxe Oracle)

```
CREATE TABLE SAUV_Chambres AS SELECT idChambre, Hotel,  
TypeChambre FROM Chambres WHERE TypeChambre = 3;
```

Ainsi, si la table Chambres contient les données suivantes :

idChambre	Hotel	TypeChambre	NumChambre	Commentaire
23	4	2	2	NULL
16	3	2	2	NULL
9	2	2	2	NULL
2	1	2	2	NULL
3	1	3	3	NULL
10	2	3	3	NULL
17	3	3	3	NULL
24	4	3	3	NULL
25	4	4	4	NULL

La table SAUV_Chambres contient les éléments suivants :

idChambre	Hotel	TypeChambre
3	1	3
10	2	3
17	3	3
24	4	3

Seules les quatre lignes concernées de la table Chambres ont été sélectionnées. Elles correspondent au critère demandé TypeChambre = 3 et la table SAUV_Chambres contient seulement trois colonnes.

Avec ce type de sélection, il est possible de créer très facilement un jeu d'essais pour son propre compte ou à destination d'un utilisateur. Nous verrons, dans la section Les vues de ce chapitre, le mécanisme des vues, qui est assez semblable à cette méthode.

5. Utilisation des synonymes

Pour donner un nom différent à une table, il faut utiliser la commande CREATE SYNONYM.

On peut utiliser également un synonyme pour simplifier un nom de table qui est normalisé, ou pour pointer sur une archive dont le nom n'a pas à être connu par les utilisateurs.

Cette fonction n'est pas implémentée avec MySQL ni dans PostgreSQL.

Syntaxe (Oracle, SQL Server)

```
CREATE SYNONYM <Nom synonyme> FOR <Nom table>;
```

Exemple

```
CREATE SYNONYM PRIX FOR Tarifs;
```

Ainsi, lorsqu'un développeur utilise la table, il peut indiquer PRIX comme ceci :

```
SELECT * FROM PRIX;
```

6. Les séquences

Dans une table, il y a souvent une colonne qui fait office d'identifiant unique et qui est la plupart du temps de type numérique.

Pour laisser le SGBDR gérer l'incrémentation de cette colonne, on peut utiliser un objet de type séquence et l'associer à une colonne.

Lors de chaque ajout de ligne dans la table, il n'est pas nécessaire d'attribuer une valeur à cette colonne, le système s'en occupe. Ainsi, on est certain de ne jamais avoir de doublons sur cette colonne.

Une séquence peut être créée indépendamment d'une colonne de table. Dans le fonctionnement d'une application, un compteur peut servir à de multiples endroits.

Syntaxe de base

```
CREATE SEQUENCE <nom séquence>;
```

Le système va créer un objet qui prend par défaut la valeur 1 et s'incrémente de 1 en 1.

Exemple

```
CREATE SEQUENCE S_NUMERO;
```

Si maintenant nous voulons attribuer une valeur de départ, un pas précis ou une valeur maximum, il faut utiliser les options proposées.

Exemple

```
CREATE SEQUENCE S_NUMERO START WITH 5 INCREMENT BY 1  
MINVALUE 2 MAXVALUE 999999 CYCLE;
```

Dans cet exemple, la séquence va démarrer à la valeur 5, sera incrémentée de 1, et ne pourra jamais être inférieure à 2 et supérieure à 999999. Lorsque la séquence aura atteint la valeur maximum 999999, elle recommencera à 2.

Si l'on ne veut pas que la séquence boucle après avoir atteint la valeur maximum, il faut enlever l'option CYCLE. Ainsi, le SGBDR retournera une erreur lorsque l'on demandera la valeur suivante de 999999. Il est assez risqué d'autoriser une séquence à boucler sur elle-même, cela peut générer des doublons dans la base ou provoquer une erreur si la colonne est déclarée UNIQUE.

Pour connaître la valeur actuelle d'une séquence ou faire avancer une séquence, il faut utiliser les fonctions suivantes.

Pour Oracle :

- CURRVAL : récupère la valeur actuelle.
- NEXTVAL : incrémente la séquence du pas prévu (1 par défaut).

Exemple

```
SELECT S_NUMERO.NEXTVAL FROM DUAL;
```

```
SELECT S_NUMERO.CURRVAL FROM DUAL;
```

La table DUAL est utilisée pour travailler sur des fonctions sans lien avec une table réelle. C'est une table virtuelle.

SQL Server n'utilise pas DUAL mais la syntaxe suivante :

```
SELECT current_value  
FROM sys.sequences
```

```
WHERE name = 'S_NUMERO' ;  
SELECT NEXT VALUE FOR S_NUMERO;
```

Pour lier une séquence à une colonne d'une table, il faut créer pour Oracle un trigger. Nous verrons dans un autre chapitre l'utilisation des triggers, mais, pour information, voici comment créer un trigger sur la colonne idChambre de la table Chambres qui utilise la séquence S_NUMERO.

Exemple de colonne auto-incrémentée avec Oracle

```
CREATE OR REPLACE TRIGGER TR_NUMERO  
BEFORE INSERT ON Chambres  
FOR EACH ROW  
DECLARE
```

```
BEGIN
```

```
    SELECT S_NUMERO.NEXTVAL  
    INTO :NEW.NUMERO FROM DUAL;  
END;  
/
```

Ainsi, à chaque insertion dans la table Chambres, il n'est pas nécessaire d'indiquer de valeur pour la colonne idChambre.

Il existe une autre façon d'utiliser la séquence lors de l'insertion, il faut la coder dans l'ordre INSERT :

```
INSERT INTO Chambres VALUES (S_NUMERO.NEXTVAL,2,1,2,'2',NULL);
```

Attention : il peut y avoir des « trous » dans une série de chiffres issus d'une séquence. En effet, si après un INSERT l'utilisateur ne valide pas la transaction par un COMMIT ou qu'il lance un ROLLBACK, la dernière valeur de la séquence n'est pas modifiée et continue de s'incrémenter normalement.

Oracle implémente l'objet séquence, comme indiqué ci-dessus.

MySQL n'implémente pas les séquences. Il existe néanmoins la fonction AUTO_INCREMENT sur une colonne lors de la création de la table. Cette colonne doit être la clé primaire.

Exemple de colonne auto-incrémentée avec MySQL

```
CREATE TABLE Chambres (idChambre    INTEGER AUTO_INCREMENT,  
                        Hotel         INTEGER,  
                        TypeChambre  INTEGER,  
                        NumChambre   VARCHAR(6),  
                        Commentaire  VARCHAR(400),
```

```
CONSTRAINT PK_Chambres PRIMARY KEY (idChambre));
```

Cette fonction existe aussi dans SQL Server et se nomme IDENTITY. Dans PostgreSQL, pour obtenir une colonne qui s'auto-incrémente, on choisira le type SERIAL à la place du type INTEGER.

Exemple de colonne auto-incrémentée avec SQL Server

```
CREATE TABLE Chambres (idChambre    INTEGER IDENTITY(1,1) NOT NULL,  
                        Hotel         INTEGER,  
                        TypeChambre  INTEGER,  
                        NumChambre   VARCHAR(6),  
                        Commentaire  VARCHAR(400),
```

```
CONSTRAINT PK_Chambres PRIMARY KEY (idChambre));
```

Pour supprimer une séquence, il faut utiliser DROP SEQUENCE.

Exemple

```
DROP SEQUENCE S_NUMERO;
```

Pour modifier une séquence, il faut utiliser ALTER SEQUENCE.

Exemples

Changer la valeur maximum de la séquence :

```
ALTER SEQUENCE S_NUMERO MAXVALUE 888888;
```

Modifier le pas de 1 à 5 :

ALTER SEQUENCE S_NUMERO INCREMENT BY 5;

La suppression de tables

La suppression de tables est une opération simple mais qu'il faut manier avec prudence. La suppression est le plus souvent définitive. Il n'y aura aucune possibilité de récupérer les données de la table une fois l'ordre lancé.

1. L'ordre DROP

L'ordre DROP permet de supprimer définitivement une table. La table et son contenu sont supprimés. L'ordre DROP est aussi utilisé sur d'autres objets de base de données comme les vues ou une base.

On utilise souvent l'ordre DROP juste avant la création d'une table. Ainsi, on évite les erreurs sur la table existante.

La commande détruit automatiquement les index et contraintes posés sur cette table ainsi que les commentaires. En revanche, l'ordre ne détruit pas les synonymes.

S'il s'agit d'une table sensible, il est préférable de la sauvegarder au préalable par un CREATE ... AS SELECT ... par exemple.

Attention : la table ne doit pas être en cours d'utilisation par une autre personne.

Dans le cas d'une fausse manipulation, il ne sera pas possible de récupérer la table (ROLLBACK). Certaines versions de SGBDR autorisent la récupération après un DROP (Oracle à partir de la version 10g, par exemple). Dans MySQL et SQL Server, les ordres de manipulation de tables entraînent une validation automatique (COMMIT). Il est donc impossible de récupérer la table après un DROP.

Syntaxe

DROP TABLE nom_de_table;

Exemple

DROP TABLE Chambres;

Vérifier l'existence d'un objet

Un objet est un composant de la définition de données (LDD). Cela peut donc être une table, une colonne, une vue, un index...

Lorsque l'on travaille sur la définition de données, on teste couramment l'objet sur lequel on intervient. Cela fait partie des bonnes pratiques pour éviter des erreurs. Par exemple, on vérifie qu'une table existe avant de la supprimer ou qu'elle n'existe pas avant de la créer.

Il existe plusieurs manières de tester l'existence d'un objet.

IF OBJECT_ID ... IS NULL

Syntaxe (SQL Server)

IF OBJECT_ID('nom_objet') IS NULL CREATE...;

IF OBJECT_ID('nom_objet') IS NOT NULL DROP...;

Exemple

IF OBJECT_ID('Hotels') IS NULL

CREATE TABLE Hotels (idHotel INTEGER,
Libelle VARCHAR(50),
Etoile VARCHAR(5));

IF OBJECT_ID('Hotels') IS NOT NULL DROP TABLE Hotels

IF EXISTS

Syntaxe (SQL Server et PostgreSQL)

DROP TABLE IF EXISTS nom_objet;

Exemple

DROP TABLE IF EXISTS Hotels;

IF EXISTS (SELECT...)

Syntaxe (SQL Server)

IF EXISTS (SELECT 1 FROM INFORMATION_SCHEMA.TABLES

```
WHERE TABLE_NAME = 'nom_objet';
```

Exemple

```
IF EXISTS (SELECT 1 FROM INFORMATION_SCHEMA.TABLES  
WHERE TABLE_NAME = 'Hotels')  
DROP TABLE Hotels;
```

Pour Oracle, il est nécessaire de créer une fonction basée sur les tables système comme DBA_TABLES ou USER_ALL_TABLES. Dans MySQL et PostgreSQL, il existe aussi des tables système comme information_schema.tables.

La modification de table

Une fois les tables créées, elles vont évoluer dans le temps et il faudra ajouter ou supprimer une colonne ou encore intervenir sur les contraintes des colonnes.

Dans certains cas, il faudra renommer une table. C'est ce que nous allons détailler dans les sections suivantes.

1. L'ordre ALTER

L'ordre ALTER est utilisé pour réaliser plusieurs actions. Il peut être utilisé pour supprimer ou ajouter une colonne d'une table, mais également pour ajouter ou supprimer une contrainte ou ajouter une valeur par défaut à une colonne.

Attention : il est interdit de changer le nom d'une colonne ainsi que son type et ses attributs NULL ou NOT NULL.

Certains SGBDR acceptent l'ordre MODIFY et permettent ainsi de modifier le type d'une colonne.

Attention néanmoins au contenu de cette colonne. Lors du passage d'un type VARCHAR à un type INTEGER, la conversion automatique réalisée par le SGBDR va modifier le contenu des données.

Il existe un risque de perdre de l'information ou de rendre certaines données incompatibles avec leur utilisation. D'une manière générale, il est déconseillé de modifier le type d'une colonne. Pour prévenir tout problème, il faut vider la table en amont avant de changer les types.

Syntaxe

```
ALTER TABLE nom_de_table [ADD nom_de_colonne type_colonne]  
[DROP COLUMN nom_de_colonne]  
[ADD CONSTRAINT nom_contrainte]  
[DROP CONSTRAINT nom_contrainte];
```

Exemples

Ajouter une colonne :

```
ALTER TABLE Chambres ADD Vue VARCHAR(20);
```

Table Chambres

Chambres
idChambre
Hotel
TypeChambre

Chambres
NumChambre
Commentaire
Vue

La colonne ajoutée se place toujours en fin de définition de la table.

Supprimer une colonne :

```
ALTER TABLE Chambres DROP COLUMN Vue;
```

2. Renommer une table (RENAME)

L'ordre RENAME permet de renommer une table. Cette commande peut être utilisée dans le cas où la table doit être recrée tout en conservant la version actuelle en archive. Il faut donc renommer la table actuelle puis refaire le CREATE initial.

Cela est plus simple et plus rapide que de faire un CREATE ... AS SELECT ... de la table concernée.

Syntaxe

Oracle :

```
RENAME nom_de_table_ancien TO nom_de_table_nouveau;
```

MySQL :

```
RENAME TABLE Chambres TO SAV_Chambres;
```

SQL Server :

```
EXEC sp_rename 'NomTable', 'NouveauNomTable'
```

Exemple

```
EXEC sp_rename 'Chambres', 'SAV_Chambres';
```

PostgreSQL :

```
ALTER TABLE Chambres RENAME TO SAV_Chambres;
```

Attention : certains SGBDR ne connaissent pas cet ordre. Dans ce cas, il faut créer une autre table avec la même structure comme ceci :

```
CREATE TABLE SAV_Chambres AS SELECT * FROM Chambres;
```

puis supprimer la table initiale :

```
DROP TABLE Chambres;
```

Vider une table

1. L'ordre TRUNCATE

L'ordre TRUNCATE est utilisé pour supprimer toutes les occurrences d'une table, sans restriction. Cet ordre occupera une seule ligne dans le journal. En cas de retour arrière, toutes les occurrences seront récupérées. Il ne sera pas possible de reprendre une partie des enregistrements. L'intérêt de l'ordre TRUNCATE est qu'il libère l'espace disponible par la suppression des lignes, dans le fichier physique de la base. Un autre avantage de cet ordre est qu'il réinitialise l'auto-incrémentation si cette option est appliquée sur une colonne.

Les déclencheurs ne sont pas exécutés.

Cet ordre est généralement employé par les administrateurs de base de données.

Syntaxe

```
TRUNCATE TABLE <nom table>
```

Exemple

```
TRUNCATE TABLE Chambres;
```

Les vues

Dans cette section, nous allons voir comment créer ou supprimer des vues. Les vues sont des éléments très utilisés dans la programmation SQL. Elles permettent principalement de créer des tables « virtuelles » spécifiques pour un domaine ou pour une classe d'utilisateurs.

1. Pourquoi utiliser des vues ?

Dans une base de données, il y a les tables permanentes qui ont été définies après une analyse des besoins et une modélisation sous forme de tables.

Si l'on respecte le modèle relationnel, il n'y a pas de données redondantes dans les tables, à l'exception des clés utilisables pour les jointures. En revanche, les utilisateurs ou les développeurs ont des besoins d'extractions spécifiques de la base. Ces extractions se matérialisent sous forme de requêtes lancées manuellement ou incluses dans les programmes.

Si ces demandes sont répétitives ou communes à plusieurs utilisateurs, il peut être nécessaire de créer une vue. La vue est une représentation logique de la base qui résulte d'une requête pour un besoin spécifique et répétitif. Contrairement à une table, elle n'est pas stockée sur disque (sauf demande spécifique) mais en mémoire.

La vue peut également permettre de simplifier pour un utilisateur la base de données. Il n'a pas besoin de connaître l'ensemble du schéma mais simplement quelques éléments spécifiques utiles dans son métier.

Si vos tables contiennent des informations confidentielles, la vue permet de masquer les colonnes en cause. Ainsi, l'utilisateur ne voit que ce que l'on veut bien lui montrer.

La création d'une vue suit le même mécanisme que le CREATE TABLE ... AS SELECT ... expliqué dans les sections précédentes. En effet, la vue est une agrégation de colonnes issues d'une ou plusieurs tables.

L'avantage majeur d'une vue est qu'elle est en permanence mise à jour. En effet, une vue est mise à jour automatiquement lorsque les tables qu'elle utilise sont modifiées. En revanche, une vue n'étant pas un objet proprement dit mais un résultat de requête, on ne peut pas faire de mise à jour de données sur une vue. Les données appartiennent aux tables du SELECT.

Une vue représente à un instant t l'image des tables qu'elle utilise.

2. La création de vues

La création s'effectue par l'ordre `CREATE VIEW` puis un ordre `SELECT` qui récupère les colonnes que l'on veut extraire des tables concernées.

Comme pour le `CREATE TABLE ... AS SELECT ...` toutes les possibilités offertes par l'ordre `SELECT` sont utilisables. On peut avoir des requêtes assez complexes pour le remplissage d'une vue.

Attention néanmoins aux temps d'exécution de la requête, et également au nombre de lignes ramenées. Une vue n'a généralement pas d'index, donc un `SELECT` sur une vue parcourt l'ensemble de la vue. Une vue ne doit pas se substituer à une table. Si les données contenues dans les vues sont nombreuses et ont besoin d'être archivées ou utilisées de manière récurrente, peut-être faut-il revoir le schéma des données et créer une nouvelle table.

Pour un utilisateur, la vue se comporte comme une table. Il peut réaliser ses requêtes sur la vue comme sur une table. Il est ainsi possible de créer un ensemble de vues par type d'utilisateur ou métier de l'entreprise. Ainsi, chacun aura les données dont il a besoin pour une utilisation précise.

Syntaxe

```
CREATE VIEW <Nom_Vue> AS SELECT ...
```

Si on veut la liste des chambres avec le n° de l'hôtel, son numéro, le type de lit, le nombre de lits et sa description, on va faire un `SELECT` sur les trois tables et récupérer les colonnes nécessaires avec cet ordre :

Exemple

```
CREATE VIEW V_Chambre_desc AS  
  
SELECT Chambres.Hotel, Chambres.NumChambre,  
       TypesChambre.TypeLit,  
       TypesChambre.NombreLit, TypesChambre.Description  
  
FROM Chambres, TypesChambre  
  
WHERE Chambres.TypeChambre = TypesChambre.idTypeChambre;
```

Ce qui va avoir pour effet de créer la vue `V_Chambre_desc` qui aura cette structure.

Table V_Chambre_desc

V_Chambre_desc
Hotel
NumChambre
TypeLit
NombreLit
Description

À chaque fois que les données des tables `Chambres` et `TypesChambre` seront modifiées, la vue `V_Chambre_desc` sera mise à jour automatiquement.

L'accès à la vue se fera comme pour une table classique avec l'ordre `SELECT`.

Exemple

Table Chambres

idChambre	Hotel	TypeChambre	NumChambre	Commentaire
23	4	2	2	NULL
16	3	2	2	NULL
9	2	2	2	NULL
2	1	2	2	NULL
3	1	3	3	NULL
10	2	3	3	NULL
17	3	3	3	NULL
24	4	3	3	NULL
25	4	4	4	NULL

Table TypesChambre

idTypeChambre	NombreLit	TypeLit	Description
1	1	lit simple	1 lit simple avec douche
2	2	lit simple	2 lits simples avec douche
3	3	lit simple	3 lits simples avec douche et WC séparés
4	1	lit double	1 lit double avec douche
5	1	lit double	1 lit double avec douche et WC séparés
6	1	lit double	1 lit double avec bain et WC séparés
7	1	lit XL	1 lit double large avec bain et WC séparés

Avec le contenu des tables ci-dessus, la vue V_Chambres_desc contient :

SELECT * FROM V_Chambres_desc;

Hotel	NumChambre	TypeLit	NombreLit	Description
1	2	lit simple	2	2 lits simples avec douche
2	2	lit simple	2	2 lits simples avec douche
3	2	lit simple	2	2 lits simples avec douche
4	2	lit simple	2	2 lits simples avec douche
4	3	lit simple	3	3 lits simples avec douche et WC séparés
3	3	lit simple	3	3 lits simples avec douche et WC séparés
2	3	lit simple	3	3 lits simples avec douche et WC séparés

Hotel	NumChambre	TypeLit	NombreLit	Description
1	3	lit simple	3	3 lits simples avec douche et WC séparés

Stockage des vues

La vue n'est pas stockée physiquement sur disque, elle est montée en mémoire lors de sa première utilisation. La récupération des données d'une vue est donc très rapide à partir de la deuxième demande. En termes de performance, les vues sont une solution à analyser à condition d'utiliser plusieurs fois le même SELECT dans une session.

Si on accède souvent à ces données, l'accès à une vue est plus rapide que l'exécution du SELECT sur plusieurs tables.

Néanmoins, il faut se rapprocher des DBA, qui donneront les normes en vigueur dans l'entreprise et les bonnes pratiques concernant l'utilisation des vues.

Attention : certains SGBDR proposent de stocker les vues sur disque. Elles se comportent ainsi exactement comme des tables et donc nécessitent une autre analyse en termes de performance.

3. La suppression de vues

L'ordre DROP VIEW permet de supprimer définitivement une vue. La vue et son contenu sont supprimés.

L'ordre DROP est définitif, il est impossible de récupérer la vue lors d'une mauvaise manipulation.

Cet ordre n'a aucune influence sur les tables qui sont utilisées dans le SELECT de création de la vue.

Syntaxe

```
DROP VIEW nom_vue;
```

Exemple

```
DROP VIEW V_Chambres_desc;
```

Les index

Dans cette section, nous abordons une notion importante : les index. Toutes les bases de données utilisent des index. L'implémentation physique de ceux-ci diffère d'un SGBDR à un autre.

Il existe plusieurs types d'index et plusieurs méthodes pour traiter ces index. Nous verrons comment créer et supprimer ces index et pourquoi utiliser tel type d'index en fonction des besoins que l'on a.

1. Les index et la norme SQL

Tout d'abord, il faut préciser que les index ne font pas partie de la norme SQL. En effet, l'index est utilisé pour accélérer une recherche dans une table et s'appuie sur des fichiers physiques qui sont créés lors de la création d'un index.

Il s'agit donc d'une implémentation physique et, dans la norme SQL, à l'instar du stockage des tables, l'aspect physique n'est pas traité. Chaque SGBDR l'implémente à sa façon.

En revanche, les index sont quasiment indispensables dans une base de données relationnelle. Le temps d'accès aux données étant un paramètre très important pour tous les utilisateurs et les développeurs, l'utilisation ou non d'un index peut augmenter les temps de réponse de manière exponentielle.

Dans le cas de tables avec plusieurs millions de lignes, l'accès à une même donnée peut mettre plusieurs heures sans index et quelques secondes avec un index.

Sans index, l'ensemble de la table sera parcouru séquentiellement jusqu'à trouver l'enregistrement demandé.

C'est le SGBDR qui gère les fichiers d'index, l'utilisateur ne peut pas intervenir sur la technique de stockage.

Attention néanmoins à ne pas créer des index sur toutes les colonnes. La création d'un index doit découler d'une réflexion sur l'utilisation de la table par les programmes et les utilisateurs. Les index ralentissent les traitements lors des mises à jour, car le SGBDR doit recalculer les clés après chaque ajout, suppression ou modification de lignes.

Il faut cibler les colonnes, analyser les taux de mise à jour d'une table et se baser sur une analyse fonctionnelle des données, puis consulter les DBA sur les normes en place dans l'entreprise et la méthode utilisée par défaut par le SGBDR.

2. Les différentes méthodes d'organisation des index

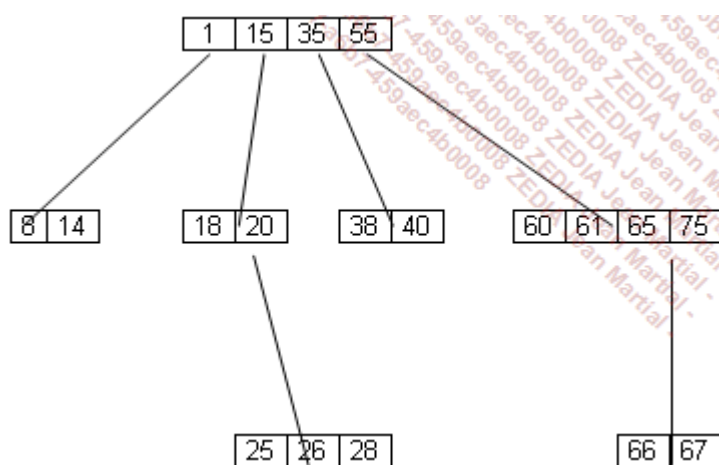
Il existe cinq méthodes principales de gestion des index : hachage, séquentiel, bitmap, arbre et cluster.

En tant que développeur, il est peu probable que le choix de la méthode d'indexation des tables soit de votre ressort. C'est un domaine souvent réservé au DBA. Mais il est bon néanmoins de connaître leur existence afin de répondre à des problématiques de performance.

Nous n'allons pas rentrer dans le détail de chacune de ces méthodes. Les plus utilisées sont les index en arbre (B*Tree) et les index bitmap.

Par défaut, Oracle va créer des index de type B*Tree si rien n'est précisé. Ce type d'index convient aux tables volumineuses avec des clés uniques ou avec très peu de doublons (moins de 5 %).

Exemple d'index B*Tree



Si l'on demande par exemple :

```
SELECT * FROM <Table> WHERE Identifiant = 18;
```

il passera directement par 15 puis accède au 18.

Les index de type bitmap sont à utiliser pour les tables volumineuses qui ont beaucoup de clés en commun avec un taux de mise à jour très faible. Le principe est de créer pour chaque valeur une chaîne de bits et ensuite de ne travailler que sur les bits.

Exemple d'index bitmap

idTypeChambre	NombreLit	TypeLit	Lit simple	Lit double	Lit XL
1	1	Lit simple	1	0	0
2	2	Lit simple	0	1	0
3	2	Lit simple	0	1	0
4	1	Lit double	0	0	1
5	1	Lit double	0	1	0
7	1	Lit XL	1	0	0

Dans la colonne Lit double, on trouve la chaîne 011010. Ensuite, le SGBDR accède aux enregistrements très rapidement.

N'oubliez pas de consulter vos DBA avant toute création d'index.

3. La création d'index

L'index est le principal élément permettant d'améliorer les performances d'accès aux données. A contrario, la mise à jour des données sera pénalisée étant donné que le SGBDR devra maintenir les fichiers d'index à chaque mise à jour de la table.

Dans le cas d'une base qui est très peu évolutive, il peut être intéressant de multiplier les index afin d'optimiser les temps de réponse. Les index doivent être posés sur des colonnes qui ont des valeurs distinctives. L'index le plus performant sera sur une clé unique, par exemple.

Il ne faut pas confondre PRIMARY KEY et CREATE INDEX UNIQUE, le premier étant une contrainte d'intégrité qui vérifie l'unicité d'une colonne (voir la section L'intégrité des données), le deuxième étant une création de fichier d'index utilisé pour les performances.

La confusion vient du fait que certains SGBDR comme Oracle, SQL Server et MySQL créent automatiquement un index lorsque l'on crée une clé primaire.

La création d'index peut se réaliser sur une ou plusieurs colonnes. Il faut bien choisir les colonnes nécessitant un index. Il faut analyser les requêtes les plus souvent utilisées, les plus coûteuses en temps, les clés primaires et étrangères posées...

Il n'y a pas de règles tout établies pour la création des index. Il est conseillé de créer des index sur les colonnes qui sont déclarées en PRIMARY KEY (si le SGBDR ne l'a pas déjà fait), sur les colonnes en FOREIGN KEY, sur les colonnes les plus accédées, les colonnes qui servent de jointure entre les tables, les colonnes les plus discriminantes, etc.

Si des problèmes de performance apparaissent après la création d'un index, il faut le supprimer et vérifier l'utilisation qui est faite de la table, notamment en mise à jour.

Syntaxe

```
CREATE [UNIQUE] INDEX <nom index> ON <nom table>
```

```
<nom colonne 1> [ASC|DESC], <nom colonne 2> [ASC|DESC], ... ..
```

Les options possibles sont :

- UNIQUE, qui indique au SGBDR que cette colonne sera unique, donc il contrôle également lors d'une insertion que cette unicité est vérifiée. Il est possible de créer autant de UNIQUE INDEX que nécessaire, contrairement à la PRIMARY KEY, qui est unique.
- ASC ou DESC, qui indique au SGBDR comment l'index doit être trié. Par défaut, il sera en ordre ascendant. Cette option peut être utile si les requêtes demandent une restitution de cette colonne (avec ORDER BY) dans un ordre précis. Si l'index est créé avec l'option DESC et que l'ordre contient un ORDER BY <colonne> ASC, l'index ne sera pas efficace.

Exemples

```
CREATE INDEX I2_TypeChambre ON Chambres (TypeChambre);
```

```
CREATE INDEX I3_TypHotel ON Chambres (TypeChambre, Hotel);
```

```
CREATE INDEX I4_NumChambre ON Chambres (NumChambre DESC);
```

Une colonne peut être utilisée dans plusieurs index. Il est intéressant de donner un numéro aux index. Ainsi, on peut connaître automatiquement le nombre d'index posés sur une table.

Lorsque plusieurs colonnes composent l'index, il est préférable de mettre la colonne la plus discriminante en premier. Ainsi, par exemple, si l'index est sur trois colonnes et qu'une requête ne teste que la première ou les deux premières colonnes, l'index sera utilisé partiellement quand même par le SGBDR.

Il n'est pas nécessaire en règle générale d'indexer les petites tables. L'analyse et la maintenance de l'index seront plus coûteuses que la lecture de la table complète.

La création d'un index peut être très longue en termes de temps de réponse si la création s'effectue sur une table importante. L'espace disque est également à surveiller car le système va créer des fichiers pour stocker ces index.

4. La suppression d'index

L'ordre DROP INDEX permet de supprimer un index.

L'ordre DROP est définitif, il sera impossible de récupérer l'index dans le cas d'une mauvaise manipulation. Un utilisateur ne peut supprimer que les index qu'il a lui-même créés et les index que l'administrateur de la base de données a autorisés.

Cet ordre ne peut pas être utilisé pour supprimer un index créé par le système après une création de clé primaire.

Syntaxe

```
DROP INDEX <nom_index>
```

Exemple

```
DROP INDEX I2_TypeChambre;
```

```
DROP INDEX I4_NumChambre;
```

L'intégrité des données

Les contraintes d'intégrité permettent de maintenir la base cohérente. On confie au SGBDR les tâches de contrôle de la validité des données qui sont insérées.

Les contraintes se substituent aux contrôles réalisés par programme.

Il existe plusieurs types de contrôles. Il est possible d'indiquer au SGBDR :

- quelle est la valeur par défaut à affecter à une colonne (DEFAULT),
- qu'une colonne ne peut pas être null (NOT NULL),
- qu'une colonne doit être unique (UNIQUE),
- ou de coder un contrôle sur une colonne (CHECK).

Il existe également deux contraintes particulières qui sont la clé primaire et la clé étrangère. Nous allons détailler leurs fonctions.

1. La clé primaire (PRIMARY KEY)

Par définition, la clé primaire est la clé principale d'une table. Le SGBDR va contrôler systématiquement à chaque insertion ou modification que la clé est unique dans la table. Dans le cas contraire, il rejette la demande de modification avec un message d'erreur de ce type : « Violation constraint ... ».

La clé primaire est toujours une clé unique. Elle est composée d'une ou de plusieurs colonnes en fonction de la méthode de création, le plus important étant qu'il ne peut y avoir deux lignes de la table avec cette même clé.

Il s'agit souvent d'un numéro que l'on incrémente de 1 à chaque création d'une ligne dans la table.

On peut utiliser aussi des données métier, comme des numéros de sécurité sociale ou des numéros de permis de conduire, mais il faut être certain que toutes les lignes de la table ont une valeur pour cette ou ces colonnes. En effet, une clé primaire ne peut pas prendre la valeur NULL.

La création d'une clé primaire génère dans la plupart des SGBDR la création automatique d'un index sur cette colonne.

Il y a deux méthodes pour déclarer une clé primaire. Si la clé correspond à une seule colonne, il faut utiliser cette syntaxe :

Exemple : déclaration d'une clé primaire sur une colonne

```
CREATE TABLE Chambres (idChambre    INTEGER PRIMARY KEY,  
                        Hotel          INTEGER,  
                        TypeChambre    INTEGER,  
                        NumChambre     VARCHAR(6),  
                        Commentaire    VARCHAR(400);
```

Automatiquement, la colonne idChambre sera NOT NULL et UNIQUE.

S'il y a plusieurs colonnes qui composent la clé, il faudra utiliser la clause CONSTRAINT qui permet de déclarer une contrainte d'intégrité.

Exemple : déclaration d'une clé primaire sur plusieurs colonnes

```
CREATE TABLE Chambres (idChambre    INTEGER,  
                        Hotel          INTEGER,
```

```
TypeChambre  INTEGER,  
NumChambre   VARCHAR(6),  
Commentaire  VARCHAR(400),
```

```
CONSTRAINT PK_Chambres PRIMARY KEY (Hotel, NumChambre));
```

Les colonnes Hotel et NumChambre composent la clé. Elles seront NOT NULL et l'association des deux est UNIQUE.

Nous pouvons remarquer également que l'on donne un nom à la contrainte, ici PK_Chambres. De cette façon, lorsque le SGBDR nous signalera une insertion incorrecte dans la table, il citera la contrainte explicitement.

Les contraintes de clé primaire sont souvent notées PK_<nom table> (PK pour *Primary Key*). Ainsi, le développeur ou l'utilisateur peut indiquer simplement en lisant le nom de la contrainte qu'il s'agit de la clé primaire et sur quelle table le problème a été rencontré.

Dans le cas où la table est déjà créée et qu'il faut ajouter une clé primaire, il faudra utiliser ALTER TABLE.

Exemple : création d'une clé primaire sur une table existante

```
ALTER TABLE Chambres ADD
```

```
CONSTRAINT PK_Chambres PRIMARY KEY (Hotel, NumChambre);
```

2. La clé étrangère (FOREIGN KEY)

La clé étrangère s'appuie sur une autre table pour indiquer comment contrôler les colonnes de notre table principale.

Il faut que la table étrangère contienne une clé primaire pour que le SGBDR puisse faire le lien.

Le principe est simple : à chaque mise à jour de la table principale, le SGBDR vérifie que la valeur de la colonne en question existe bien dans l'autre table.

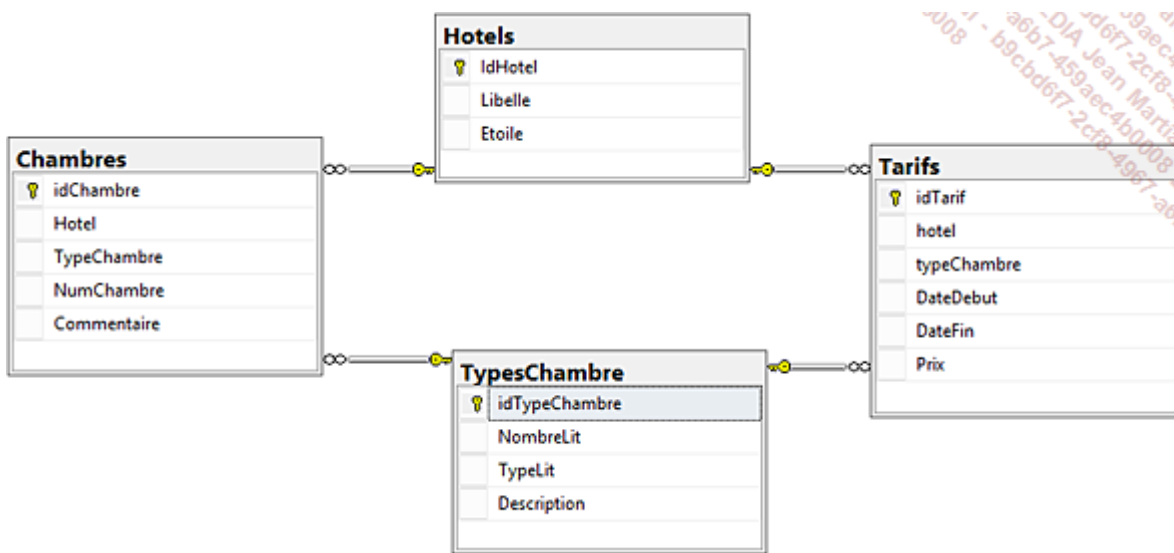
Il est préférable que le nom de la colonne soit identique entre les deux tables concernées. Dans le cas contraire, le SGBDR prendra la clé primaire de la table secondaire.

Il est possible d'avoir plusieurs contraintes de clés étrangères sur une même table.

L'utilisation ou non d'une clé étrangère dépend des normes de développement de l'entreprise. On peut préférer contrôler l'intégrité par programme et non laisser la base de données le faire.

Dans le cas d'utilisation des clés étrangères, il faut que vos programmes contrôlent les codes retour à chaque mise à jour de la base, afin de vérifier qu'aucune contrainte n'a été violée.

Exemple



Syntaxe

CONSTRAINT <nom contrainte> FOREIGN KEY (colonne 1, colonne 2 ...)

REFERENCES <table secondaire>);

Dans l'exemple, il faut tout d'abord créer une clé primaire sur la table TypesChambre et ensuite créer la clé étrangère dans la table Chambres.

Il est possible d'ajouter les instructions ON DELETE CASCADE ou ON UPDATE CASCADE pour que la suppression ou la mise à jour de la table parent impacte la table enfant. Il existe aussi les instructions ON DELETE SET NULL ou ON UPDATE SET NULL pour mettre la valeur NULL dans la table enfant lors de la suppression ou la modification de la donnée dans la table parent. Enfin, il est possible d'utiliser les instructions ON DELETE SET DEFAULT ou ON UPDATE SET DEFAULT pour que la valeur par défaut impacte également la table enfant.

Exemple de création d'une clé étrangère

```
ALTER TABLE TypeChambres ADD
```

```
CONSTRAINT PK_TypeChambres PRIMARY KEY (idTypeChambre);
```

```
CREATE TABLE Chambres (idChambre    INTEGER PRIMARY KEY,
```

```
    Hotel        INTEGER,
```

```
    TypeChambre  INTEGER,
```

```
    NumChambre   VARCHAR(6),
```

```
    Commentaire  VARCHAR(400),
```

```
CONSTRAINT FK_TypeChambre FOREIGN KEY (TypeChambre) REFERENCES
```

```
TypesChambres (idTypeChambre)) ON DELETE CASCADE ON UPDATE SET
```

```
DEFAULT;
```

```
CREATE TABLE Chambres (idChambre    INTEGER PRIMARY KEY,
```

```
    Hotel        INTEGER,
```



```
TypeChambre  INTEGER,  
NumChambre   VARCHAR(6),  
Commentaire  VARCHAR(400),
```

```
CONSTRAINT FK_TypeChambre FOREIGN KEY (TypeChambre) REFERENCES
```

```
TypesChambres (idTypeChambre)) ON DELETE SET NULL;
```

3. Les valeurs par défaut (DEFAULT)

Certaines colonnes ne sont pas forcément renseignées par l'utilisateur au moment de la saisie. Le développeur peut alors décider d'attribuer une valeur par défaut à cette colonne afin de ne pas avoir de valeur NULL dans la base.

Dans la section suivante de ce même chapitre, nous aborderons la gestion des valeurs NULL et nous verrons que celle-ci est assez particulière.

Il est préférable d'avoir des valeurs par défaut bien définies afin d'éviter d'obtenir des résultats de requêtes non conformes.

L'attribution d'une valeur par défaut s'effectue lors de la création de la table ou par un ALTER TABLE.

Il est possible d'indiquer différentes valeurs par défaut :

- une valeur numérique ou caractère,
- des résultats de fonction comme la date du jour (CURRENT_DATE ou SYSDATE) ou la date et l'heure du jour (CURRENT_TIMESTAMP), utiles notamment pour les colonnes qui tracent les mises à jour réalisées sur une table,
- pour attribuer le nom de l'utilisateur qui a réalisé la mise à jour avec la fonction USER.

Pour qu'une valeur par défaut soit prise en compte par le SGBDR, il faut qu'aucune valeur ne soit affectée à la colonne lors de l'ordre INSERT. Il faut que la colonne n'apparaisse pas dans l'ordre. Si on indique une valeur blanc ou null, elle se substitue à la valeur par défaut.

Exemple de valeurs par défaut

Lors de la création de la table :

```
CREATE TABLE Chambres (idChambre  INTEGER PRIMARY KEY,  
                        Hotel        INTEGER,  
                        TypeChambre  INTEGER DEFAULT 2,  
                        NumChambre   VARCHAR(6),  
                        Commentaire  VARCHAR(400));
```

Une fois la table créée, on peut modifier une valeur par défaut ainsi (ne fonctionne pas dans MySQL) :

```
ALTER TABLE Chambres MODIFY TypeChambre DEFAULT 3;
```

Ajouter une colonne et lui attribuer une valeur par défaut :

```
ALTER TABLE Chambres ADD (Vue VARCHAR(20) DEFAULT 'Mer');
```

4. La valeur NULL

Lors d'une insertion ou d'une modification, si l'on ne précise pas de valeur pour une colonne, celle-ci est vide et prend la valeur NULL.

NULL ne correspond ni à un espace ni à la valeur zéro. Pour utiliser une colonne contenant NULL, il faut dans le SELECT préciser IS NULL afin de tester le contenu.

Par exemple, si les lignes de la table Chambres ont ce contenu :

IdChambre	Hotel	TypeChambre	NumChambre	Commentaire	Les lignes 2, 3, 4 et 6 ont été entrées sans préciser de valeur pour le
1	1	1	1	Vue sur mer	
2	1	2	2		
3	1	3	3	NULL	
4	1	4	4	NULL	
5	1	5	5	Vue sur mer	
6	1	6	6	NULL	
7	1	7	7	Vue sur mer	

commentaire, donc la colonne est considérée comme NULL mais le commentaire de la ligne 2 n'est pas NULL.

Si on réalise ensuite une sélection sur la colonne Commentaire comme :

```
SELECT COUNT(*) FROM Chambres WHERE commentaire NOT IN  
( 'Vue sur mer' );
```

Le résultat est 1. En effet, les trois lignes qui ont des valeurs NULL ne sont pas prises en compte, même si le commentaire est bien différent des valeurs recherchées. Par contre, la ligne 2 est comptée car le commentaire n'est pas NULL même s'il paraît vide.

Il faut donc manier avec prudence les colonnes qui acceptent les NULL. Par précaution, il est préférable de mettre toutes les colonnes qui sont utilisées dans une clause WHERE en NOT NULL et leur affecter des valeurs par défaut lors des insertions et modifications.

Toutes les colonnes contenues dans une clé doivent être en NOT NULL afin d'éviter tout problème d'indexation.

5. La contrainte d'unicité UNIQUE

Comme indiqué dans la section qui traite des clés primaires, la clause UNIQUE permet de préciser au SGBDR que la valeur de cette colonne ne doit pas être en doublon dans la table. Il est conseillé d'utiliser cette propriété pour les colonnes de type Libellé.

Attention, une colonne déclarée en UNIQUE peut quand même contenir du NULL si on ne précise pas la clause NOT NULL au moment de sa création. Dans ce cas, il peut y avoir plusieurs lignes avec la colonne à NULL, et l'unicité ne s'applique alors pas.

Exemple : ajout de l'unicité lors de la création de la table

```
CREATE TABLE Hotels (idHotel INTEGER PRIMARY KEY,  
Libelle VARCHAR(50) UNIQUE,  
Etoile VARCHAR(5));
```

Modifier une colonne (ne fonctionne pas dans MySQL)

```
ALTER TABLE Hotels MODIFY (Libelle UNIQUE)
```

6. La contrainte de vérification CHECK

Cette clause permet de réaliser toutes sortes de contrôles sur les colonnes d'une table. Il faut l'utiliser avec précaution car elle peut être source de problèmes de performances significatifs. En effet, à chaque modification de la colonne, l'ordre est exécuté et peut ralentir sensiblement les mises à jour.

On peut contrôler la colonne avec des valeurs alphanumériques ou numériques, mais également par l'appel à une fonction ou encore en spécifiant un SELECT spécifique.

Attention : **la clause CHECK est implémentée dans MySQL mais n'a aucun effet**. Celui-ci ne tient absolument pas compte de ces contraintes. Il faut passer par des contrôles par programme ou utiliser un trigger. MySQL implémente la clause pour des raisons de compatibilité avec la norme uniquement. Oracle l'implémente et la gère, mais pas complètement, **on ne peut pas faire appel à d'autres tables sous forme de sous-SELECT, par exemple**. Il se contente de gérer les contrôles sur des constantes, sur des colonnes de la table et sur des fonctions simples.

Exemple de contrôles possibles selon la norme SQL92

```
CREATE TABLE Chambres (idChambre    INTEGER PRIMARY KEY,  
                        Hotel          INTEGER CHECK (VALUE BETWEEN 1 AND 999),  
                        TypeChambre    CHECK (VALUE IN (SELECT  
idTypeChambre FROM TypesChambre)),  
                        NumChambre     VARCHAR(6),  
                        Commentaire    VARCHAR(400));
```

La colonne Hotel doit prendre une valeur entre 1 et 999.

Le contenu de la colonne TypeChambre doit appartenir aux valeurs de la table TypesChambre. Généralement, il est préférable d'utiliser une clé étrangère plutôt que ce type de vérification.

Avec Oracle, on peut créer la table avec les contrôles sur Hotel mais pas sur TypeChambre.

7. La suppression d'une clé primaire

Il arrive d'avoir besoin de supprimer une clé primaire. L'index lié à la clé est également supprimé.

Syntaxe

```
ALTER TABLE <nom table> DROP CONSTRAINT <nom contrainte>
```

Exemple (SQL Server, PostgreSQL, Oracle)

```
ALTER TABLE Hotels DROP CONSTRAINT PK_Hotels;
```

Syntaxe

```
ALTER TABLE <nom table> DROP PRIMARY KEY
```

Exemple (MySQL)

```
ALTER TABLE Hotels DROP PRIMARY KEY;
```

8. Quelques conseils

Réalisez en amont une analyse détaillée du contenu de chaque table, des liens entre les tables, des clés uniques ou pas, des colonnes qui ne seront jamais vides, etc.

Il est préférable de donner des noms parlants aux tables, de donner des noms identiques ou quasi identiques aux colonnes qui représentent la même chose.

Il faut également réfléchir sur les valeurs par défaut à attribuer aux colonnes, aux séquences à mettre en place, etc.

Une réflexion un peu plus importante est à apporter sur les types numériques à utiliser. Est-ce que le gain d'espace disque est primordial ? Est-ce que l'on manipule des données financières ? Quel est le niveau d'arrondi attendu ? Quelle est la valeur maximale de la colonne ? Est-ce que j'ai besoin de décimales ? Etc.

Concernant le gain de place, il est important de se poser la question, même si les technologies évoluent pour accueillir un volume de plus en plus conséquent de données sur un disque. Parallèlement, le besoin en données est de plus en plus important, avec une incidence sur le coût de la gestion de la base.

Ensuite, il faut analyser la façon dont les programmes et/ou les utilisateurs accèdent aux données. Quels sont les champs discriminants dans les tables ? Avec quelles colonnes l'utilisateur réalise ses requêtes ? Combien de lignes sont ramenées en moyenne par interrogation ?

En fonction de l'utilisation qui sera faite de l'application, il faut opter pour créer des tables avec beaucoup de colonnes ou créer beaucoup de tables avec peu de colonnes. Cela dépend aussi de la quantité de mémoire allouée à la base de données, des accès qui sont réalisés sur les tables, de la quantité de colonnes ramenées par les sélections, etc.

Dans tous les cas, il faut se rapprocher des administrateurs afin de connaître les règles communes à respecter et les contraintes spécifiques à chaque site.

Il faut également se rapprocher de la maîtrise d'œuvre afin de connaître les évolutions futures qui sont prévues dans le schéma de données.

Exercices

Premier exercice

À partir du contenu du tableau suivant, écrire la syntaxe de création de la table FILM.

Mettre un index primaire sur la colonne IDENT_FILM puis un index non unique sur la colonne GENRE1 associée à PAYS.

Ident _ Film	Titre	Genre1	Recette	Date_ Sortie	Pay s	NB_ Entree	Date et Heure de saisie	Resume
1	SUBWA Y	POLICIE R	390 659,52	10/04/8 5	1	2 917 56 2	25/05/1 1 11:31	Conte les aventures de la population souterraine dans les couloirs du métro parisien.
2	NIKITA	DRAME	5 017 971,00	21/02/9 0	1	3 787 84 5	15/04/1 1 09:30	Nikita, condamnée à la prison à perpétuité, est contrainte à travailler secrètement pour le gouvernement en tant qu'agent hautement qualifié des services secret s.
3	STAR WARS 6 - LE RETOU R DU JEDI	ACTION	191 648 000,0 0	19/10/8 3	2	4 263 00 0	01/01/1 0 08:00	L'empire galactique est plus puissant que jamais : la construction de la nouvelle arme, l'Étoile de la Mort, menace l'univers tout entier.

Deuxième exercice

Ajouter une colonne complémentaire nommée NUM_REAL.

Cette colonne est une clé étrangère sur la table REALISATEUR.

Ajouter une valeur par défaut sur la colonne RECETTE avec la valeur 0.

Mettre les colonnes TITRE et PAYS en NOT NULL.

Supprimer la contrainte.

Troisième exercice

Créer une vue FILM2 à partir de la table FILM contenant les quatre premières colonnes de la table FILM ainsi que la colonne RESUME.

Supprimer cette vue.

Quatrième exercice

Créer une séquence sur la colonne IDENT_FILM qui commence à 12 et avec une valeur maximum de 9999.

Renommer la table FILM en FILMOLD.

Créer une nouvelle table FILM à partir de la table FILMOLD.

Supprimer la table FILM et la table FILMOLD.

Solutions des exercices

Premier exercice

Requête au format standard :

DROP TABLE FILM;

```
CREATE TABLE FILM (IDENT_FILM      INTEGER PRIMARY KEY,
                    TITRE           VARCHAR(50),
                    GENRE1          VARCHAR(20),
                    RECETTE          DECIMAL(15,2),
                    DATE_SORTIE      DATE,
                    PAYS              SMALLINT,
                    NB_ENTREE         INTEGER,
                    RESUME            VARCHAR(2000),
                    DATE_SAISIE       TIMESTAMP
                    );
```

Description Oracle de la table :

DESC FILM

Nom	NULL ?	Type
-----	--------	------

IDENT_FILM	NOT NULL NUMBER(38)
TITRE	VARCHAR2(50)
GENRE1	VARCHAR2(20)
RECETTE	NUMBER(15,2)
DATE_SORTIE	DATE
PAYS	NUMBER(38)
NB_ENTREE	NUMBER(38)
RESUME	VARCHAR2(2000)
DATE_SAISIE	TIMESTAMP(6)

On peut remarquer que la colonne IDENT_FILM est automatiquement mise en NOT NULL sans qu'on ait précisé dans le CREATE. Le fait de déclarer la zone PRIMARY KEY induit sa mise en NOT NULL.

Requête de création d'index :

```
CREATE INDEX I2_GENREPAYS ON FILM (GENRE1, PAYS);
```

Deuxième exercice

Requête d'ajout de colonne :

```
ALTER TABLE FILM ADD (NUM_REAL INTEGER);
```

Ajout d'une clé étrangère :

-- Création de la table REALISATEUR

```
DROP TABLE REALISATEUR;
```

```
CREATE TABLE REALISATEUR
```

```
(NUM_REAL    INTEGER PRIMARY KEY,
  NOM        VARCHAR(50));
```

-- Ajout de la contrainte d'intégrité

```
ALTER TABLE FILM ADD CONSTRAINT FK_REALISATEUR FOREIGN KEY
(NUM_REAL) REFERENCES REALISATEUR;
```

Ajouter une valeur par défaut sur la colonne RECETTE avec la valeur 0 :

```
ALTER TABLE FILM MODIFY RECETTE DEFAULT 0;
```

Mettre les colonnes TITRE et PAYS en NOT NULL :

```
ALTER TABLE FILM MODIFY TITRE NOT NULL;
```

```
ALTER TABLE FILM MODIFY PAYS NOT NULL;
```

Supprimer la contrainte :

```
ALTER TABLE FILM DROP CONSTRAINT FK_REALISATEUR;
```

Description de la table FILM à ce stade :

```
DESC FILM
```

Nom	NULL ?	Type

IDENT_FILM	NOT NULL	NUMBER(38)
TITRE	NOT NULL	VARCHAR2(50)
GENRE1		VARCHAR2(20)
RECETTE		NUMBER(15,2)
DATE_SORTIE		DATE
PAYS	NOT NULL	NUMBER(38)
NB_ENTREE		NUMBER(38)
RESUME		VARCHAR2(2000)
DATE_SAISIE		TIMESTAMP(6)
NUM_REAL		NUMBER(38)

Troisième exercice

Créer une vue FILM2 à partir de la table FILM contenant les quatre premières colonnes de la table FILM ainsi que la colonne RESUME.

```
CREATE VIEW FILM2 AS  
  
SELECT IDENT_FILM, TITRE, GENRE1, RECETTE, RESUME  
  
FROM FILM;
```

Supprimer cette vue :

```
DROP VIEW FILM2;
```

Quatrième exercice

Créer une séquence sur la colonne IDENT_FILM qui commence à 12 et avec une valeur maximum de 9999.

```
CREATE SEQUENCE S_FILM START WITH 12 INCREMENT BY 1  
  
MINVALUE 12 MAXVALUE 9999 CYCLE;
```

Renommer la table FILM en FILMOLD :

```
RENAME FILM TO FILMOLD;
```

Créer une nouvelle table FILM à partir de la table FILMOLD :


```
CREATE TABLE FILM AS SELECT * FROM FILMOLD;
```

Supprimer la table FILM et la table FILMOLD :

```
DROP TABLE FILM;
```

```
DROP TABLE FILMOLD;
```

La manipulation des données (LMD)

Introduction

Le langage de manipulation de données permet aux utilisateurs et aux développeurs d'accéder aux données de la base, de modifier leur contenu, d'insérer ou de supprimer des lignes.

Il s'appuie sur quatre ordres de base qui sont SELECT, INSERT, DELETE et UPDATE.

Ces quatre ordres ne sont pas toujours autorisés par l'administrateur de la base qui est le seul à pouvoir attribuer ou non les droits d'utilisation sur ces ordres.

Pour l'utilisateur lambda, il pourra indiquer que seul l'ordre SELECT est utilisable. Les ordres de modification de la base ne sont pas accessibles pour certains utilisateurs pour des raisons évidentes de sécurité.

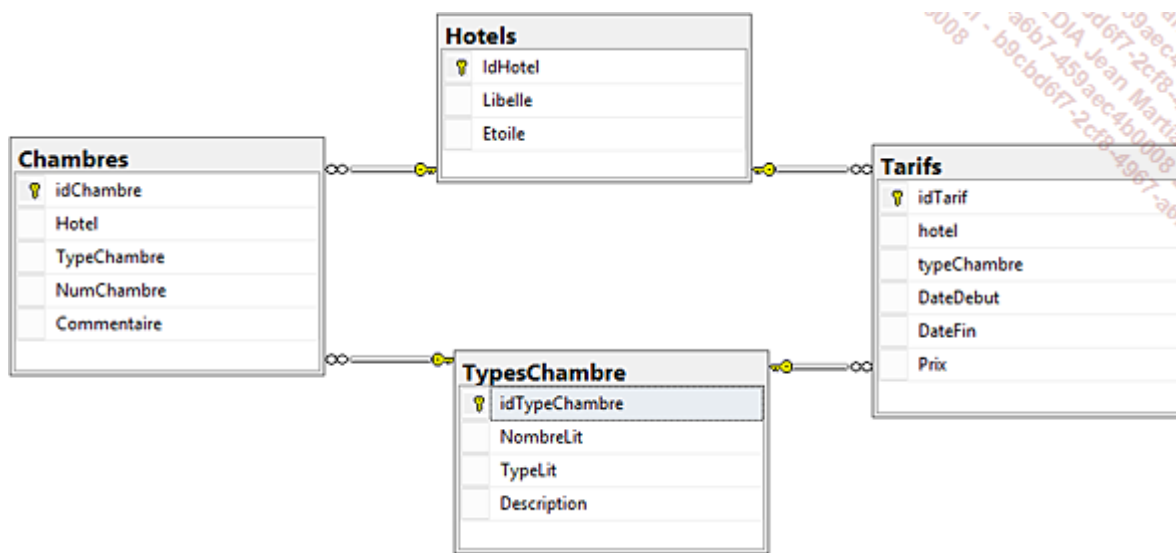
La sélection de données

L'ordre SELECT permet de réaliser des requêtes simples assez rapidement même sans connaissances approfondies en langage de programmation. C'est l'ordre de base qui permet d'indiquer au serveur que l'on désire extraire des données.

Il peut également être très puissant si l'on connaît toutes les fonctions et toutes les possibilités du langage. On peut réaliser des requêtes complexes, avec de nombreuses tables mais il faut toujours faire attention aux performances qui peuvent se dégrader très rapidement sur un ordre SQL mal construit ou n'utilisant pas les bons index dans les tables. Il faut être vigilant et utiliser les outils d'analyse de requête (cf. chapitre Approfondissement - Quelques notions de performances) avant d'exécuter une requête sur une base réelle avec des tables conséquentes.

Les principaux éléments d'une requête de sélection	
Clause	Expression
SELECT	Liste colonne(s) et ou éléments d'extraction
FROM	Table(s) source(s)
WHERE	Condition(s) ou restriction(s), optionnelle
GROUP BY	Regroupement(s), optionnelle
HAVING	Condition(s) ou restriction(s) sur le(s) regroupement(s), optionnelle
ORDER BY	Tri(s)

Les tables de base qui sont utilisées dans les sections suivantes sont celles-ci :



1. L'ordre de sélection de données SELECT

Le SELECT est l'ordre le plus important et le plus utilisé en SQL. Avec cet ordre, nous pouvons ramener des lignes d'une ou plusieurs tables mais également transformer des données par l'utilisation de fonction ou encore réaliser des calculs.

Nous allons décrire progressivement les possibilités de cet ordre dans les paragraphes suivants.

L'utilisation la plus courante consiste à sélectionner des lignes dans une table comme ceci :

```
SELECT NombreLit, Description FROM TypesChambre;
```

Dans cet exemple, nous avons sélectionné deux colonnes de la table TypesChambre.

L'ordre va donc nous ramener toutes les lignes de la table pour ces deux colonnes.

Si on avait voulu toutes les colonnes et toutes les lignes de la table, l'ordre aurait été celui-ci :

```
SELECT * FROM TypesChambre;
```

idTypeChambre	NombreLit	TypeLit	Description
1	1	lit simple	1 lit simple avec douche
2	2	lit simple	2 lits simples avec douche
3	2	lit simple	2 lits simples avec douche et WC séparés
4	1	lit double	1 lit double avec douche
5	1	lit double	1 lit double avec douche et WC séparés
6	1	lit double	1 lit double avec bain et WC séparés
7	1	lit XL	1 lit double large avec bain et WC séparés

L'étoile est pratique lorsque l'on ne connaît pas le nom des colonnes, mais le résultat est rarement lisible avec des tables contenant un nombre important de colonnes. Pour connaître le nom des colonnes, faites un DESC de la table auparavant (DESC <nom table>).

La syntaxe simple est donc :

```
SELECT <colonne 1>, <colonne 2> ... | * FROM <table1>, <table2> ...
```

Si certaines colonnes ont le même nom mais appartiennent à des tables différentes, il faudra ajouter le nom de la table devant la colonne afin que le système sache quelle colonne prendre.

Il n'est pas obligatoire de mettre le nom des tables devant chaque colonne, mais pour une question de lisibilité et de maintenance, il est préférable de les mettre sur des sélections complexes.

```
SELECT Hotels.Libelle
, Hotels.Etoile
, Chambres.NumChambre
, TypesChambre.Description
FROM Chambres, Hotels, TypesChambre;
```

Nous verrons dans la section L'utilisation des alias de ce chapitre que pour alléger la lecture il est également possible de donner un alias à chaque table. Cet alias est souvent simple et permet de retrouver facilement la table concernée, par exemple CH pour Chambres ou TYP pour TypesChambre.

2. Les options DISTINCT et ALL

Par défaut, lors de l'exécution d'un SELECT, toutes les lignes sont ramenées (l'option ALL est automatique). Si l'on veut supprimer les doublons, il faut ajouter l'ordre DISTINCT.

L'ordre DISTINCT s'applique à toutes les colonnes présentes.

Exemple

```
SELECT NombreLit, TypeLit, Description FROM TypesChambre;
```

et :

```
SELECT DISTINCT NombreLit, TypeLit, Description FROM TypesChambre;
```

Les deux SELECT ci-dessus ont le même résultat car il y a un doublon sur les trois premières lignes. Ces deux premières colonnes sont identiques mais pas la troisième.

NombreLit	TypeLit	Description
1	lit double	1 lit double avec bain et WC séparés
1	lit double	1 lit double avec douche
1	lit double	1 lit double avec douche et WC séparés
1	lit simple	1 lit simple avec douche
1	lit XL	1 lit double large avec bain et WC séparés

En revanche, si on réduit la sélection à deux colonnes comme :

```
SELECT NombreLit, TypeLit FROM TypesChambre;
```

on obtient :

NombreLit	TypeLit
1	lit simple
2	lit simple

NombreLit	TypeLit
2	lit simple
1	lit double
1	lit double

Si on ajoute un ordre DISTINCT, une des deux lignes contenant '2' et 'lit simple' sera supprimée.

```
SELECT DISTINCT NombreLit, TypeLit FROM TypesChambre;
```

La clause DISTINCT ne peut pas être utilisée avec des opérateurs de regroupement (voir le GROUP BY). En effet, les opérateurs de type COUNT ou SUM éliminent automatiquement les doublons.

NombreLit	TypeLit
1	lit double
1	lit simple
1	lit XL
2	lit simple

3. Les tris

Lorsque l'on ramène des colonnes d'une ou de plusieurs tables avec un SELECT, il est souvent intéressant d'obtenir un résultat trié sur certaines colonnes.

Pour cela, on utilisera la clause ORDER BY en fin de requête. On peut trier sur n'importe quelles colonnes d'une table, il faut pour cela que les colonnes fassent partie de la sélection, mais elles ne sont pas obligatoirement affichées.

La clause ne peut être utilisée qu'une seule fois dans une requête et doit toujours être la dernière clause de la requête.

Le tri par défaut est ascendant, noté ASC (du plus petit au plus grand). Il est possible d'indiquer que l'on désire réaliser le tri en descendant en notant DESC.

Syntaxe de l'ORDER BY

```
ORDER BY <colonne 1> [ASC | DESC], <colonne 2> [ASC | DESC]...
```

Exemple

Tri sur la DateDebut de la table Tarifs en ascendant et tri sur le prix de la table en descendant.

```
SELECT hotel
```

```
, typeChambre
```

```
, DateDebut
```

```
, prix
```

```
FROM Tarifs
```

```
ORDER BY DateDebut, prix DESC;
```

Hotel	typeChambre	DateDebut	prix
1	7	01/04/2021	103,49
4	7	01/04/2021	103,49
4	6	01/04/2021	91,99
1	6	01/04/2021	91,99
1	5	01/04/2021	80,49
4	3	01/04/2021	80,49

On constate que le tri se fait sur la colonne prix du plus grand au plus petit (DESC).

Il existe également la possibilité d'indiquer l'ordre de la colonne dans le SELECT à la place de son nom, à condition que la colonne soit présente dans la sélection.

Exemple sans le nom des colonnes

```
SELECT hotel
```

```
, typeChambre
```

```
, DateDebut
```

```
, prix
```

```
FROM Tarifs
```

```
ORDER BY 3, 4 DESC;
```

Certes, cette notation fonctionne et permet de ne pas ressaisir le nom des colonnes, mais n'aide vraiment pas à la lisibilité de la requête. Avec des sélections multiples, l'ordre devient vite illisible pour celui qui ne l'a pas écrit.

Attention : le tri peut être source de baisse de performance de la requête. En effet, si la requête ramène des millions de lignes, le système va récupérer toutes les lignes dans un premier temps, puis trier l'ensemble et ensuite seulement commencer à restituer les éléments dans l'ordre. Le système va utiliser de l'espace disque pour stocker les éléments à trier puis de la mémoire pour réaliser le tri.

Le tri est déconseillé dans le cas où il n'y a pas de clause WHERE ou que celle-ci est peu restrictive et que la table contient des millions d'enregistrements.

Autre remarque, les performances seront meilleures en réalisant les tris sur des colonnes indexées.

4. Les options TOP, LIMIT, OFFSET ou ROWNUM

L'exécution d'un SELECT ramène toutes les lignes. Si l'on veut obtenir un nombre de lignes maximum dans le résultat, comme les 10 premières lignes, il faut utiliser l'ordre TOP ou LIMIT selon le SGDBR utilisé. Si l'on souhaite les dernières lignes, il est nécessaire d'ajouter un tri décroissant. L'ordre OFFSET ajouté à l'ordre LIMIT permet d'effectuer un décalage de résultat. L'ordre ROWNUM correspond à la numérotation des lignes d'une requête et permet donc aussi d'afficher une partie du résultat selon le besoin.

Syntaxe SQL Server pour les 10 premières puis les 10 dernières lignes

```
SELECT TOP 10 idTarif, hotel, typechambre, dateDebut, DateFin, Prix
```

```
FROM Tarifs;
```

```
SELECT TOP 10 idTarif, hotel, typechambre, dateDebut, DateFin, Prix  
FROM Tarifs ORDER BY idTarif DESC;
```

Syntaxe MySQL et PostgreSQL

Les 10 premières lignes :

```
SELECT idTarif, hotel, typechambre, dateDebut, DateFin, Prix FROM  
Tarifs LIMIT 10;
```

Les 10 premières lignes à partir de la cinquième ligne :

```
SELECT idTarif, hotel, typechambre, dateDebut, DateFin, Prix FROM  
Tarifs LIMIT 10 OFFSET 5;
```

ou :

```
SELECT idTarif, hotel, typechambre, dateDebut, DateFin, Prix FROM  
Tarifs LIMIT 5, 10;
```

Les 10 dernières lignes :

```
SELECT idTarif, hotel, typechambre, dateDebut, DateFin, Prix FROM  
Tarifs ORDER BY idTarif DESC LIMIT 10;
```

Syntaxe Oracle

```
SELECT idTarif, hotel, typechambre, dateDebut, DateFin, Prix FROM  
Tarifs  
  
WHERE ROWNUM <= 10  
  
ORDER BY ROWNUM DESC;
```

5. L'utilisation des alias

Dans une requête SQL qui comporte plusieurs tables, il est souhaitable d'attribuer un diminutif à chaque nom de table, que l'on appelle un alias, toujours dans l'optique de rendre les requêtes plus lisibles pour tous les programmeurs.

Cet alias peut aussi être utilisé pour un nom de colonne, mais également pour un résultat de fonction ou de SELECT imbriqués.

Il existe trois syntaxes pour l'alias :

- Il suit directement l'élément qu'il va supplanter.
- Il peut être précédé de l'ordre AS.
- Il peut précéder la colonne avec l'opérateur =.

Exemple d'alias sur une table qui fonctionne avec toute base

```
SELECT NombreLit, Description FROM TypesChambre TYP;
```

Ou autre exemple de syntaxe SQL Server et MySQL :

```
SELECT NombreLit, Description FROM TypesChambre AS TYP
```

Lorsqu'il existe plusieurs tables, il faut donner un alias différent pour chacune des tables comme ceci :

```
SELECT CH.NumChambre  
      , TYP.NombreLit  
      , TYP.TypeLit  
FROM Chambres CH, TypesChambre TYP;
```

Ou cette autre syntaxe :

```
SELECT CH.NumChambre  
      , TYP.NombreLit  
      , TYP.TypeLit  
FROM Chambres AS CH, TypesChambre AS TYP;
```

L'alias sur une colonne remplace lors de l'affichage le nom initial de la colonne. Cela peut permettre de rendre le résultat plus explicite pour un utilisateur ou de cacher le nom réel de la colonne.

Exemple d'alias sur une colonne pour toute base :

```
SELECT hotel C1, typeChambre C2, Prix FROM Tarifs T;
```

Ou autre syntaxe pour toute base :

```
SELECT hotel AS C1, typeChambre AS C2, Prix FROM Tarifs T;
```

Ou autre encore pour SQL Server :

```
SELECT C1 = hotel, C2 = typeChambre, Prix FROM Tarifs T;
```

L'alias est aussi intéressant pour nommer un résultat de calcul. Ainsi, l'utilisateur comprendra immédiatement le contenu de cette colonne.

Exemple d'alias sur un résultat de calcul

```
SELECT hotel C1, typeChambre C2, Prix * 1.2 PUTTC FROM Tarifs T;
```

L'alias est aussi utilisé lorsque deux colonnes ont le même nom dans deux tables différentes. Il faut préciser au système l'origine de la colonne.

Par exemple, la colonne TypeChambre existe dans la table Chambres et la table Tarifs. C'est assez logique étant donné qu'il s'agit de la clé étrangère de ces tables qui pointe sur la table TypesChambre.

Donc, si l'on veut écrire une requête qui utilise ces deux tables, la jointure s'effectuera sur cette clé. Il faut alors différencier les deux colonnes.

Exemple d'alias pour différencier deux colonnes de même nom

```
SELECT CH.NumChambre  
      , TYP.NombreLit  
      , T.Prix  
FROM Chambres CH  
      , TypesChambre TYP  
      , Tarifs T
```


WHERE CH.TypeChambre = TYP.idTypeChambre

AND T.TypeChambre = TYP.idTypeChambre;

NumChambre	NombreLit	Prix
1	1	49,99
2	1	49,99
1	1	49,99
1	1	49,99
1	1	49,99
2	2	59,99
2	2	59,99

6. La clause de restriction WHERE

Nous avons vu comment récupérer toutes les lignes des tables dans les paragraphes précédents. Pour pouvoir sélectionner une partie des lignes des tables, il faut ajouter une restriction symbolisée par la clause WHERE.

Tout ce qui est écrit derrière cette clause impacte le résultat de la requête.

Elle est utilisable avec également les ordres de mise à jour DELETE et UPDATE.

La restriction peut être très simple, à l'image de cet exemple :

```
SELECT idChambre, Hotel, NumChambre FROM Chambres WHERE
```

```
TypeChambre = 3;
```

ou beaucoup plus complexe en utilisant des fonctions ou des jointures.

L'exemple suivant utilise la fonction EXISTS et la fonction IN.

Exemple

```
SELECT CH.idChambre, CH.Hotel, CH.NumChambre FROM Chambres CH
```

```
WHERE CH.TypeChambre IN (3, 6)
```

```
AND idChambre > 5
```

```
AND EXISTS
```

```
(SELECT Libelle FROM Hotels HT WHERE HT.idHotel = CH.Hotel);
```

Ici, les trois colonnes sont testées, le type doit être 3 ou 6, l'identifiant doit être supérieur à 5 et l'hôtel doit exister dans la table Hotels.

Les restrictions derrière une clause WHERE sont quasiment sans limites, le seul frein reste les performances qui pourront potentiellement se dégrader si les restrictions portent sur des colonnes non indexées dans des tables volumineuses.

Quelquefois il est préférable de faire plusieurs ordres SELECT simples et utiliser des tables temporaires ou des vues pour stocker les résultats intermédiaires plutôt que de réaliser une requête complexe. Il faut toujours garder à l'esprit que la personne qui maintiendra les programmes est rarement la personne qui est à l'origine de l'écriture.

Comme dans l'exemple précédent, il faut donc essayer d'écrire des ordres lisibles, bien indentés et commentés de préférence.

7. Les commentaires

Avec Oracle et SQL Server, pour écrire des commentaires il faut ajouter `/*` en début et `*/` en fin de commentaire, sur une ligne seule ou sur une ligne contenant du SQL.

```
/* Sélection des chambres de type 3 et 6 avec un id supérieur à 5
dont l'hôtel est référencé */
```

SQL Server accepte `--` devant chaque ligne à commenter.

Avec MySQL, il faut ajouter `--` ou `#` sur une ligne vide ou sur une ligne avec des ordres SQL.

Exemple avec MySQL ou SQL Server

```
-- Sélection des chambres de type 3 et 6 avec un id supérieur à 5
-- dont l'hôtel est référencé

# Sélection des chambres de type 3 et 6 avec un id supérieur à 5
# dont l'hôtel est référencé
```

8. Les jointures

Dans une base relationnelle, la jointure est un des éléments essentiels qui permet d'extraire les données de plusieurs tables en appliquant des conditions sur les colonnes.

Si on reprend un des exemples précédents et que l'on ne précise pas quelles colonnes permettent de réaliser la jointure, le système va réaliser le produit cartésien des tables et ramener pour chaque ligne de la première table toutes les lignes des autres tables.

Si nous avons quatre et sept lignes dans chacune des tables `Hotels` et `TypesChambre`, le système ramènera $4 \times 7 = 28$ lignes.

Exemple de jointure sans restriction qui ramène le produit cartésien du contenu des deux tables

```
SELECT Libelle, Description FROM hotels, typeschambre;
```

Libelle	Description
Art Hotel	1 lit simple avec douche
Art Hotel	2 lits simples avec douche
Art Hotel	3 lits simples avec douche et WC séparés
Art Hotel	1 lit double avec douche
Art Hotel	1 lit double avec douche et WC séparés
Art Hotel	1 lit double avec bain et WC séparés
Art Hotel	1 lit double large avec bain et WC séparés
Lions Hotel	1 lit simple avec douche
Lions Hotel	2 lits simples avec douche

Libelle	Description
Lions Hotel	3 lits simples avec douche et WC séparés
Lions Hotel	1 lit double avec douche
Lions Hotel	1 lit double avec douche et WC séparés
Lions Hotel	1 lit double avec bain et WC séparés
Lions Hotel	1 lit double large avec bain et WC séparés
Rose Hotel	1 lit simple avec douche
Rose Hotel	2 lits simples avec douche
Rose Hotel	3 lits simples avec douche et WC séparés
Rose Hotel	1 lit double avec douche
Rose Hotel	1 lit double avec douche et WC séparés
Rose Hotel	1 lit double avec bain et WC séparés
Rose Hotel	1 lit double large avec bain et WC séparés
...	...

Cet exemple peut être intéressant si on souhaite avoir tous les types de chambre pour chaque hôtel, mais le résultat est théorique.

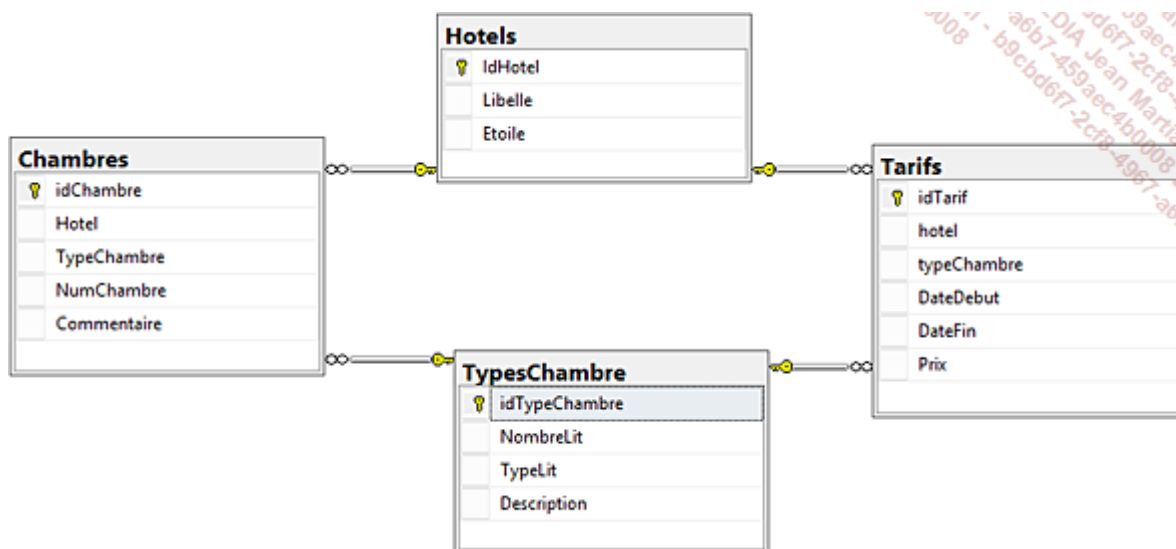
Il faut donc relier les tables entre elles afin d'obtenir un résultat conforme à nos attentes et réaliste.

La table Tarifs a deux clés étrangères vers TypesChambre et Hotels. La table Chambres a les mêmes clés étrangères que la table Tarifs.

Nous allons donc utiliser ces clés pour relier les quatre tables. L'objectif est d'obtenir pour chaque ligne de la table Tarifs la description du type de chambre, le libellé de l'hôtel et le numéro de la chambre.

a. La jointure interne

Rappel de la description de nos quatre tables de démonstration



Il faut faire la jointure sur la table TypesChambre avec la colonne idTypeChambre ainsi que la jointure sur la table Hotels avec la colonne idHotel.

Exemple d'une jointure entre quatre tables

```
SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre
FROM Tarifs AS T
, Hotels AS H
, TypesChambre AS TYP
, Chambres AS CH
WHERE H.idHotel = T.hotel
AND TYP.idTypeChambre = T.typeChambre
AND CH.Hotel = H.idHotel
AND CH.TypeChambre = TYP.idTypeChambre;
```

Résultat

Libelle	Description	Prix	NumChambre
Ski Hotel	1 lit simple avec douche	49.99	1
Art Hotel	3 lits simples avec douche et WC séparés	80.49	3
Art Hotel	1 lit double avec douche	68.99	4
Rose Hotel	1 lit simple avec douche	57.49	1

Il existe une autre notation pour obtenir le même résultat en utilisant l'ordre JOIN qui est défini dans les normes SQL. La norme préconise de travailler avec cet ordre afin de bien distinguer les conditions et les jointures. En effet, dans un WHERE, il est parfois difficile de différencier les restrictions des jointures.

Exemple d'une jointure avec quatre tables et l'ordre JOIN

```
SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre
FROM Tarifs AS T INNER JOIN Hotels AS H ON H.idHotel = T.hotel
INNER JOIN TypesChambre AS TYP ON TYP.idTypeChambre =
T.typeChambre
INNER JOIN Chambres AS CH ON CH.Hotel = H.idHotel AND
CH.TypeChambre = TYP.idTypeChambre;
```

Le mot INNER est facultatif, il indique qu'il s'agit d'une jointure interne, la jointure par défaut lorsque l'on ne précise pas autre chose.

L'opérateur ON précède le critère de jointure qui sera utilisé.

Dans ce cas, nous avons deux jointures avec la table Tarifs. Après chaque clause JOIN, on indique la table concernée et deux jointures avec la table Chambres.

Une autre difficulté pouvant être rencontrée avec l'utilisation des jointures réside dans le fait que l'on suppose que toutes lignes des tables maîtres (Tarifs et Chambres) ont une correspondance dans les deux tables filles (TypesChambre et Hotels).

Si, par exemple, il n'existe pas de chambre qui correspond à un tarif d'un hôtel, le prix de ce type de chambre (89.99 pour Ski Hotel) n'est pas restitué dans la requête.

Résultat

Libelle	Description	Prix	NumChambre
Ski Hotel	1 lit double avec douche	59.99	4
Ski Hotel	1 lit double avec douche et WC séparés	69.99	5
Ski Hotel	1 lit double avec bain et WC séparés	79.99	6
Art Hotel	1 lit simple avec douche	57.49	1

La ligne qui contient le prix 89.99 de la table Tarifs n'apparaît pas alors qu'elle est bien présente dans la table. Par défaut, si la jointure n'est pas vérifiée, le système ne ramène pas la ligne.

Il existe plusieurs types de jointure : interne, externe ou naturelle.

b. La jointure externe

Pour ramener toutes les lignes de la table maître, il faut le préciser au SGBDR même s'il n'y a pas de correspondance dans les tables filles. Dans ce cas, le système ramène les colonnes sans correspondance à NULL.

Exemple avec JOIN dans lequel il faut ajouter le mot LEFT (ou RIGHT) pour préciser que l'on veut toutes les lignes de la table à gauche du mot JOIN, ici Tarifs.

Si on voulait toutes les lignes de la table TypesChambre ou Hotels, on le préciserait avec le mot RIGHT. Les tables sont à droite du mot JOIN.

Si on veut à la fois les lignes de la table de droite et des tables de gauche, il faut utiliser le mot FULL à la place de RIGHT ou LEFT.

```
SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre
FROM Tarifs AS T LEFT OUTER JOIN Hotels AS H ON H.idHotel = T.hotel
LEFT OUTER JOIN TypesChambre AS TYP ON TYP.idTypeChambre =
T.typeChambre
LEFT OUTER JOIN Chambres AS CH ON CH.Hotel = H.idHotel AND
CH.TypeChambre = TYP.idTypeChambre;
```

Résultat

Libelle	Description	Prix	NumChambre
Ski Hotel	1 lit double avec douche et WC séparé	69.99	5
Ski Hotel	1 lit double avec bain et WC séparé	79.99	6
Ski Hotel	1 lit double large avec bain et WC séparé	89.99	NULL

Libelle	Description	Prix	NumChambre
Art Hotel	1 lit simple avec douche	57.49	1

Le mot OUTER est facultatif et indique que l'on réalise une jointure externe, par opposition au mot INNER vu plus haut.

Le tarif de 89.99 pour Ski Hotel apparaît sans numéro de chambre, ce qui veut dire qu'il n'existe pas de chambre avec un lit double large avec bain et WC séparés dans l'hôtel Ski Hotel.

La même requête avec FULL à la place de LEFT ramènera aussi les lignes des tables TypesChambre et Hotels qui ne sont pas dans Tarifs ou dans Chambres.

Exemple avec l'option FULL

```
SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre
FROM Tarifs AS T FULL OUTER JOIN Hotels AS H ON H.idHotel = T.hotel
FULL OUTER JOIN TypesChambre AS TYP ON TYP.idTypeChambre =
T.typeChambre
FULL OUTER JOIN Chambres AS CH ON CH.Hotel = H.idHotel AND
CH.TypeChambre = TYP.idTypeChambre;
```

Résultat avec l'option FULL

Libelle	Description	Prix	NumChambre
Ski Hotel	1 lit double avec douche et WC séparés	69.99	5
Ski Hotel	1 lit double avec bain et WC séparés	79.99	6
Ski Hotel	1 lit double large avec bain et WC séparés	89.99	NULL
Art Hotel	1 lit simple avec douche	57.49	1

Sur la troisième ligne, il manque le numéro de chambre car il n'y a pas de chambre pour le tarif 89.99 ou avec un lit double avec bain et WC séparés.

Autre possibilité sans utiliser le mot JOIN uniquement avec Oracle, il faut ajouter les caractères «(+)» après la colonne de la table fille concernée.

Exemple

```
SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre
FROM Tarifs T
, Hotels H
, TypesChambre TYP
, Chambres CH
WHERE H.idHotel(+) = T.hotel
AND TYP.idTypeChambre(+) = T.typeChambre
AND CH.Hotel = H.idHotel(+)
AND CH.TypeChambre(+) = TYP.idTypeChambre;
```

Libelle	Description	Prix	NumChambre
	1 lit simple avec douche	49.99	1
Art Hotel	1 lit simple avec douche	49.99	1
	1 lit simple avec douche	49.99	1
Lions Hotel	1 lit simple avec douche	57.49	1
	1 lit simple avec douche	57.49	1

Le libellé de l'hôtel est NULL, dans ce cas.

Fonctionnellement, il est important de définir lors de la construction de la base les règles, les clés et les relations entre les tables afin d'éviter ce type de problème.

Dans ce cas précis, il serait souhaitable d'interdire la création d'une ligne dans la table Tarifs si le type de chambres n'existe pas dans l'hôtel correspondant, ceci par programme ou par contrainte. La contrainte est plus sûre mais par programme, le message apparaît à l'utilisateur.

c. La jointure naturelle

La jointure naturelle laisse le système associer les colonnes qui ont le même nom entre elles. Dans cet exemple, on peut ne pas préciser que les relations se font sur TypeChambre et Hotel. La syntaxe diffère entre les différents SGBDR. Elle n'existe pas dans SQL Server.

Il n'est pas conseillé d'utiliser ce type de jointure, toujours pour les mêmes raisons de lisibilité : il y a un risque que deux colonnes portent le même nom sans être des colonnes clés, donc le résultat de la requête sera faussé.

De plus, il faut que le programmeur connaisse parfaitement la structure des tables et que ces mêmes tables n'évoluent pas à l'avenir par l'ajout de colonnes !

Exemple de jointure naturelle avec Oracle

```
SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre
FROM Tarifs T NATURAL JOIN Hotels H
NATURAL JOIN TypesChambre TYP
NATURAL JOIN Chambres CH;
```

Libelle	Description	Prix	NumChambre
Rose Hotel	1 lit simple avec douche	57.49	1
Rose Hotel	1 lit simple avec douche	49.99	1
Rose Hotel	1 lit simple avec douche	68.99	2
Rose Hotel	1 lit simple avec douche	59.99	2

Exemple de jointure naturelle avec MySQL

Il est préférable d'utiliser l'opérateur ON pour indiquer clairement les colonnes à utiliser pour la jointure afin d'éviter tout risque de jointure inadéquat.

Exemple MySQL :

```
SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre
```

FROM Tarifs T JOIN Hotels H

JOIN Chambres CH

JOIN TypesChambre TYP;

Exemple on (différent de celui de MySQL dans le commentaire ci-dessus) :

SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre

FROM Tarifs T JOIN Hotels H ON T.hotel = H.idHotel

JOIN Chambres CH ON CH.Hotel = H.idHotel

JOIN TypesChambre TYP ON TYP.idTypeChambre = CH.TypeChambre;

d. La jointure croisée

C'est la jointure la plus simple qui existe. On ne précise pas de critère de rapprochement, le système ramène donc le produit cartésien des deux tables.

On peut utiliser un FROM simple ou ajouter le mot CROSS.

SELECT Libelle, Description FROM hotels, typeschambre;

ou :

SELECT Libelle, Description FROM hotels CROSS JOIN typeschambre;

Résultats

Libelle	Description
Art Hotel	1 lit simple avec douche
Art Hotel	2 lits simples avec douche
Art Hotel	3 lits simples avec douche et WC séparés
Art Hotel	1 lit double avec douche
Art Hotel	1 lit double avec douche et WC séparés
Art Hotel	1 lit double avec bain et WC séparés
Art Hotel	1 lit double large avec bain et WC séparés
Lions Hotel	1 lit simple avec douche
Lions Hotel	2 lits simples avec douche
Lions Hotel	3 lits simples avec douche et WC séparés
Lions Hotel	1 lit double avec douche
Lions Hotel	1 lit double avec douche et WC séparés
Lions Hotel	1 lit double avec bain et WC séparés
Lions Hotel	1 lit double large avec bain et WC séparés
Rose Hotel	1 lit simple avec douche

Libelle	Description
Rose Hotel	2 lits simples avec douche
Rose Hotel	3 lits simples avec douche et WC séparés
Rose Hotel	1 lit double avec douche
Rose Hotel	1 lit double avec douche et WC séparés
Rose Hotel	1 lit double avec bain et WC séparés
Rose Hotel	1 lit double large avec bain et WC séparés
...	...

e. Syntaxes des différentes formes de jointures

Jointure interne

SELECT <colonne 1>, <colonne 2>, etc ...

FROM <table gauche> [INNER] JOIN <table droite 1> ON <critères 1>,
 [INNER] JOIN <table droite 2> ON <critères 2>,
 etc ...

ou

SELECT <colonne 1>, <colonne 2>, etc ...
 FROM <table gauche>, <table droite 1>, <table droite 2> etc ...
 WHERE <critères 1> AND <critères 2> etc ...

Jointure externe

SELECT <colonne 1>, <colonne 2>, etc ...
 FROM <table gauche> [LEFT|RIGHT|FULL] [OUTER] JOIN <table droite
 1> ON <critères>,
 [LEFT|RIGHT|FULL] [OUTER] JOIN <table droite
 2> ON <critères>,
 etc ...

Jointure naturelle

SELECT <colonne 1>, <colonne 2>, etc ...
 FROM <table gauche> NATURAL JOIN <table droite 1>,
 NATURAL JOIN <table droite 2>,
 etc ...

Jointure croisée

SELECT <colonne 1>, <colonne 2>, etc ...
 FROM <table gauche> CROSS JOIN <table droite 1>,

CROSS JOIN <table droite 2>,

etc ...

ou

SELECT <colonne 1>, <colonne 2>, etc ...

FROM <table gauche>, <table droite 1>, <table droite 2> etc ...

9. Les regroupements (GROUP BY)

Cette clause permet de supprimer les doublons dans les lignes ramenées et de réaliser des calculs sur les colonnes sélectionnées.

L'utilisation de fonctions mathématiques n'est pas obligatoire avec un GROUP BY, il est possible en revanche de réaliser un GROUP BY sur une colonne sans faire de calcul. Dans ce cas, le GROUP BY remplace le DISTINCT. En effet, une seule valeur par colonne sera ramenée.

Exemple

SELECT DISTINCT Prix FROM Tarifs;

est équivalent à :

SELECT Prix FROM Tarifs GROUP BY Prix;

Dans l'ordre SELECT, il y a aura d'une part, les colonnes sur lesquelles on désire réaliser un regroupement puis les colonnes qui feront l'objet d'un calcul.

La clause GROUP BY permet de regrouper les lignes par valeur et de réaliser des calculs sur les colonnes.

Cette clause peut être utilisée avec les fonctions SUM, COUNT, AVG, MAX ou MIN qui sont les plus connues. Il existe également VARIANCE et STDDEV pour l'écart type mais qui sont moins utilisées.

Par exemple, pour connaître le nombre de chambres présentes dans la table par type, la colonne de regroupement sera donc TypeChambre et le système comptera le nombre de lignes présentes dans la table pour chaque type différent.

Autre exemple : pour connaître le chiffre d'affaires d'une journée par type de chambre si toutes les chambres des hôtels sont réservées, le système va cumuler le prix des chambres par type.

Les exemples ci-dessous sont basés sur une table Chambres contenant les éléments suivants :

idChambre	Hotel	TypeChambre	NumChambre
1	1	1	1
2	1	2	2
3	1	3	3
4	1	4	4
5	1	5	5
6	1	6	6
8	2	1	1
9	2	2	2

idChambre	Hotel	TypeChambre	NumChambre
10	2	3	3
11	2	4	4
12	2	5	5
13	2	6	6
14	2	7	7
15	3	1	1
1	1	1	1

Exemple 1 : nombre de chambres par type

```
SELECT TypeChambre, Description, COUNT(idChambre) AS NumChambre
FROM Chambres INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =
Chambres.TypeChambre
GROUP BY TypeChambre, Description;
```

TypeChambre	Description	NbChambres
1	1 lit simple avec douche	4
2	2 lits simples avec douche	4
3	3 lits simples avec douche et WC séparés	4
4	1 lit double avec douche	4
5	1 lit double avec douche et WC séparés	4
6	1 lit double avec bain et WC séparés	4
7	1 lit double large avec bain et WC séparés	3

Dans cet exemple il y a 4 chambres pour chaque type sauf le dernier type qui comprend seulement 3 chambres.

À noter que le résultat du COUNT est associé au nom de colonne NbChambres à l'aide de la clause AS.

Exemple 2 : prix cumulés par type et période de réservation

```
SELECT Chambres.TypeChambre, Description, DateDebut, SUM(prix) as CA
FROM Chambres INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =
Chambres.TypeChambre
INNER JOIN Tarifs ON Chambres.Hotel = Tarifs.hotel AND Chambres.TypeChambre
= Tarifs.typeChambre
GROUP BY DateDebut, Chambres.TypeChambre, Description;
```

TypeChambre	Description	DateDebut	CA
1	1 lit simple avec douche	2021-04-01	114.98
1	1 lit simple avec douche	2021-04-16	99.98
1	1 lit simple avec douche	2021-10-01	99.98
1	1 lit simple avec douche	2021-12-15	114.98
2	2 lits simples avec douche	2021-04-01	137.98
2	2 lits simples avec douche	2021-04-16	119.98
2	2 lits simples avec douche	2021-10-01	119.98

Dans cet exemple, nous avons deux chambres de type 1 à 57.49 pour la période de réservation qui commence le 1^{er} janvier 2021, d'où le CA de $2 \times 57.49 = 114.98$.

10. Les fonctions utilisées lors d'un regroupement

Nous venons de voir l'ordre GROUP BY qui s'utilise avec des fonctions spécifiques. Nous détaillons ci-dessous les différentes fonctions liées à l'ordre GROUP BY.

Il est à noter que ces fonctions peuvent être utilisées sans le GROUP BY, dans ce cas elles s'appliquent à toutes les lignes ramenées par le SELECT. On ne peut pas avoir des colonnes simples et des fonctions de calcul dans une requête sans avoir une fonction GROUP BY.

Exemple de fonction de calcul sans regroupement

```
SELECT COUNT(idTarif) FROM Tarifs;
```

Le résultat est 56, correspondant au nombre de lignes de la table Tarifs.

```
SELECT SUM(Prix) FROM Tarifs;
```

Le résultat est 4 127.44, correspondant au cumul de tous les prix de la table Tarifs.

a. Compter des lignes (COUNT)

Cette fonction permet de comptabiliser un nombre de lignes par rapport aux critères demandés.

Elle se note COUNT(*) ou COUNT(<nom de colonne>). Le résultat est une valeur numérique. La notation « * » indique que l'on prend en compte toutes les lignes indifféremment de leur contenu. En revanche, si l'on note un nom de colonne dans le COUNT, seules seront comptabilisées les lignes sans valeur NULL dans la colonne.

Par exemple, pour connaître le nombre de lignes de la table Chambres qui ont un même type :

```
SELECT TypeChambre, Description, COUNT(idChambre) AS NbChambres
```

```
FROM Chambres INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =
```

```
Chambres.TypeChambre
```

```
GROUP BY TypeChambre, Description;
```

TypeChambre	Description	NbChambres
1	1 lit simple avec douche	4
2	2 lits simples avec douche	4

TypeChambre	Description	NbChambres
3	3 lits simples avec douche et WC séparés	4
4	1 lit double avec douche	4
5	1 lit double avec douche et WC séparés	4
6	1 lit double avec bain et WC séparés	4
7	1 lit double large avec bain et WC séparés	3

On constate qu'il y a 4 chambres de chaque type sauf le type 7 qui compte 3 chambres.

On peut également ajouter des restrictions, par exemple :

```
SELECT TypeChambre, Description, COUNT(idChambre) AS NbChambres
```

```
FROM Chambres INNER JOIN TypesChambre
```

```
ON TypesChambre.idTypeChambre = Chambres.TypeChambre
```

```
WHERE Hotel <> 2
```

```
GROUP BY TypeChambre, Description;
```

TypeChambre	Description	NbChambres
1	1 lit simple avec douche	3
2	2 lits simples avec douche	3
3	3 lits simples avec douche et WC séparés	3
4	1 lit double avec douche	3
5	1 lit double avec douche et WC séparés	3
6	1 lit double avec bain et WC séparés	3
7	1 lit double large avec bain et WC séparés	2

On constate que le nombre de chambres a diminué.

Cas particulier du COUNT DISTINCT : il permet de compter le nombre de valeurs différentes dans la base.

Si l'on veut connaître le nombre de commentaires différents dans la table, on écrira :

```
SELECT COUNT(DISTINCT Commentaire) AS NbCommentaires FROM Chambres;
```

Le résultat est 2. Les lignes qui n'ont pas de valeur dans la colonne Commentaire ne sont pas prises en compte.

La valeur NULL est ignorée.

Si on enlève la clause DISTINCT, le résultat est de 3. Il y a bien trois lignes qui ont une valeur dans la colonne Commentaire.

```
SELECT COUNT(Commentaire) AS NbCommentaires FROM Chambres;
```

Si on remplace Commentaire par * ou 1, le résultat est de 27. En effet, dans ce cas on compte le nombre de lignes de la table indépendamment de la colonne Commentaire. L'utilisation du chiffre 1 donne le résultat plus rapidement que l'étoile. Le chiffre 1 correspond à la valeur logique d'un espace.

```
SELECT COUNT(1) AS NbChambres FROM Chambres;
```

b. Additionner des valeurs (SUM)

Cette fonction permet de cumuler les valeurs d'une colonne. Elle s'applique uniquement à des colonnes de type numérique.

Pour calculer la somme des prix par type, on écrit :

```
SELECT typeChambre, SUM(Prix) AS PRIX_CUMULE  
FROM Tarifs  
GROUP BY typeChambre;
```

typeChambre	PRIX_CUMULE
1	429,92
2	515,92
3	601,92
4	515,92

Pour connaître la somme totale, il faut écrire :

```
SELECT SUM(Prix) AS PRIX_CUMULE  
FROM Tarifs;
```

c. Valeurs maximum et minimum (MAX et MIN)

Ces deux fonctions permettent de ramener soit la plus grande valeur, soit la plus petite valeur d'une colonne.

Par exemple, trouver le tarif le plus cher :

```
SELECT MAX(Prix) AS PRIX_MAX  
FROM Tarifs;
```

ou trouver le tarif le plus cher et le moins cher par type :

```
SELECT TypeChambre, Description, MIN(Prix) AS PRIX_MIN, MAX(Prix)  
AS PRIX_MAX, COUNT(1) AS NbTarifs  
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =  
Tarifs.typeChambre  
GROUP BY TypeChambre, Description;
```

TypeChambre	Description	PRIX_MIN	PRIX_MAX	NbTarifs
1	1 lit simple avec douche	49,99	57,49	8
2	2 lits simples avec douche	59,99	68,99	8
3	3 lits simples avec douche et WC séparés	69,99	80,49	8
4	1 lit double avec douche	59,99	68,99	8

Dans ce résultat, nous voyons que les types de chambres ont un prix min différent du prix max.

Chaque type n'a qu'une seule ligne dans la table et donc qu'un prix min et max.

En combinant les fonctions de calcul, les possibilités sont très importantes.

On peut également envisager de créer des vues qui contiennent des résultats de calcul à des fins statistiques, par exemple. Ainsi, l'information est disponible immédiatement.

Calcul de la moyenne du prix d'une chambre par type

Dans cet exemple, on ajoute tous les prix puis on divise par le nombre de lignes par type.

Ainsi, on additionne tous les prix (SUM) puis on divise par le nombre de lignes (COUNT).

```
SELECT TypeChambre, Description, MIN(Prix) AS PRIX_MIN, MAX(Prix)
AS PRIX_MAX, SUM(prix) / COUNT(1) AS PrixMoyen
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre
= Tarifs.typeChambre
GROUP BY TypeChambre, Description;
```

TypeChambre	Description	PRIX_MIN	PRIX_MAX	PRIX_MOYEN
1	1 lit simple avec douche	49,99	57,49	53,74
2	2 lits simples avec douche	59,99	68,99	64,49
3	3 lits simples avec douche et WC séparés	69,99	80,49	75,24
4	1 lit double avec douche	59,99	68,99	64,49

Il existe une fonction qui calcule directement la moyenne : il s'agit de AVG.

d. Moyenne de valeurs (AVG)

Cette fonction permet de calculer la moyenne d'une série de chiffres.

Elle s'applique à une colonne de type numérique, bien sûr.

Par exemple, pour connaître le prix moyen des chambres par type comme dans l'exemple précédent, il suffit de remplacer le SUM/COUNT par un AVG pour obtenir le même résultat.

```
SELECT TypeChambre, Description, MIN(Prix) AS PRIX_MIN, MAX(Prix)
AS PRIX_MAX, AVG(prix) AS PrixMoyen
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre
= Tarifs.typeChambre
GROUP BY TypeChambre, Description;
```

e. La clause de restriction sur un regroupement (HAVING)

On ne peut pas considérer la clause HAVING comme une fonction mais comme un filtre supplémentaire qui travaille sur le résultat d'un GROUP BY.

À l'instar du WHERE qui filtre les lignes ramenées par le SELECT, le HAVING va filtrer le résultat des fonctions utilisées.

Sélection des chambres dans la base qui sont présentes moins de quatre fois :

```
SELECT typechambre, description, COUNT(IdChambre) AS NbChambres
```

FROM Chambres CH

INNER JOIN TypesChambre TYP

HAVING COUNT(IdChambre) < 4;

TypeChambre	Description	NbChambres
7	1 lit double large avec bain et WC séparés	3

Autre exemple, si l'on veut calculer la moyenne des prix par type uniquement sur les prix de chambres de plus de 70 € :

SELECT TypeChambre, Description, MIN(Prix) AS PRIX_MIN, MAX(Prix)

AS PRIX_MAX, AVG(prix) AS PrixMoyen

FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =

Tarifs.typeChambre

GROUP BY TypeChambre, Description

HAVING MIN(Prix) > 70;

TypeChambre	Description	PRIX_MIN	PRIX_MAX	PRIX_MOYEN
6	1 lit double avec bain et WC séparés	79,99	91,99	85,99
7	1 lit double large avec bain et WC séparés	89,99	103,49	96,74

11. Les instructions de condition CASE et IIF

L'instruction CASE peut être utilisée dans la sélection de données afin de formater un résultat selon le contenu d'une colonne.

Il est possible de tester les valeurs d'une colonne et d'agir en fonction du contenu de celle-ci.

Il existe deux façons de rédiger le CASE, soit en testant des constantes par rapport à une colonne soit en ajoutant des conditions (=, <, >, ...) sur cette colonne.

Syntaxe avec test d'une constante

SELECT <colonne 1>, <colonne 2> ,

CASE <colonne 3>

WHEN <constante 1> THEN <valeur affichée>

WHEN <constante 2> THEN <valeur affichée>

WHEN <constante 3> THEN <valeur affichée>

... ..

ELSE <valeur par défaut>

END AS <entête de colonne>

FROM <table1>, <table2>

WHERE

Syntaxe avec test à l'aide d'une condition

```
SELECT <colonne 1>, <colonne 2>  
, IIF(<colonne 3> <condition> <constante ou valeur>, <valeur si vrai>,  
<valeur si faux>) AS <entête de colonne>  
FROM <table1>, <table2> ... ...  
WHERE ... ... ;
```

Exemple SQL Server avec le contrôle de la date de début de saison sur une valeur fixe

```
SELECT TypeChambre, Description, DateDebut  
, CASE DateDebut  
    WHEN '2021-04-01' THEN 'été 2021'  
    WHEN '2021-04-16' THEN 'été 2021'  
    WHEN '2021-10-01' THEN 'Hiver 2022'  
    ELSE 'Hiver 2022'  
END AS Saison  
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre  
= Tarifs.typeChambre  
ORDER BY typeChambre, DateDebut;
```

Maintenant si on veut mettre des bornes sur ce même contrôle de la date de début de saison, on écrira :

```
SELECT TypeChambre, Description, DateDebut  
, CASE  
    WHEN DateDebut >= '2021-04-01' AND DateDebut < '2021-10-01'  
THEN 'été 2021'  
    WHEN DateDebut >= '2021-10-01' AND DateDebut < '2022-04-01'  
THEN 'hiver 2022'  
END AS Saison  
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre  
= Tarifs.typeChambre  
ORDER BY typeChambre, DateDebut;
```

Il est possible aussi d'utiliser l'instruction IIF avec SQL Server :

```
SELECT TypeChambre, Description, DateDebut  
, IIF (DateDebut < '2021-10-01', 'été 2021', 'Hiver 2022') as Saison  
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =  
Tarifs.typeChambre
```

ORDER BY typeChambre, DateDebut;

Résultat pour les trois requêtes

TypeChambre	description	DateDebut	Saison
1	1 lit simple avec douche	2021-04-01	été 2021
1	1 lit simple avec douche	2021-04-16	été 2021
1	1 lit simple avec douche	2021-04-16	été 2021
1	1 lit simple avec douche	2021-10-01	hiver 2022
1	1 lit simple avec douche	2021-10-01	hiver 2022
1	1 lit simple avec douche	2021-12-15	hiver 2022

Les syntaxes Oracle et MySQL sont équivalentes.

Le CASE est à utiliser avec parcimonie étant donné que les performances auront tendance à se dégrader. Cet ordre est assez gourmand en termes de ressources, les conditions étant exécutées ligne par ligne.

12. La concaténation

La concaténation permet de mettre plusieurs valeurs de plusieurs colonnes dans une seule colonne. Les valeurs doivent être de type chaîne de caractères. Nous verrons plus loin comment transformer un type numérique ou date en chaîne de caractères.

Il existe deux syntaxes possibles pour la concaténation.

Syntaxe avec le caractère de concaténation

```
SELECT <colonne 1>, <colonne 2>  
, <colonne 1> + <colonne 2> AS <entête de colonne>  
FROM <table1>, <table2> ... ...  
WHERE ... ... ;
```

Pour Oracle et PostgreSQL, le caractère de concaténation est un double pipe ||.

Exemple avec le caractère de concaténation

```
SELECT Libelle, Etoile  
, Libelle + ' ' + Etoile AS Hotel  
FROM Hotels ;
```

```
SELECT Libelle, Etoile  
, Libelle || ' ' || Etoile AS Hotel  
FROM Hotels;
```

Syntaxe avec la fonction CONCAT

```
SELECT <colonne 1>, <colonne 2>
```

, CONCAT(<colonne 1>, <colonne 2>,) AS <entête de colonne>

FROM <table1>, <table2>

WHERE ;

Oracle n'admet que deux arguments.

Exemple avec la fonction CONCAT

SELECT Libelle, Etoile

, CONCAT(Libelle, ' ', Etoile) AS Hotel

FROM Hotels

Résultat pour les deux requêtes

Libelle	Etoile	Hotel
Ski Hotel	*	Ski Hotel *
Art Hotel	**	Art Hotel **
Rose Hotel	***	Rose Hotel ***
Lions Hotel	****	Lions Hotel ****

On ajoute un espace en troisième élément de concaténation pour ne pas que les chaînes de caractères soient collées.

MySQL utilise la fonction de concaténation ainsi qu'une autre fonction où l'on place le séparateur en premier dans la liste.

SELECT Libelle, Etoile

, CONCAT_WS(' ', Libelle, Etoile, idHotel) AS Hotel

FROM Hotels;

MySQL n'a pas de caractère de concaténation comme le double pipe dans Oracle. Il est possible de mettre des valeurs de type chaîne de caractères les unes à côté des autres, mais cela ne fonctionne pas avec des colonnes.

SELECT 'conc' 'até' 'nation';

13. L'instruction de choix (CHOOSE)

Cette instruction est un peu spéciale car elle porte sur des constantes et non sur des valeurs présentes dans une table. Elle est utilisée essentiellement en programmation dans un déclencheur, une procédure stockée ou une fonction. Elle permet de choisir un élément, par son numéro, d'une liste de constantes. Elle n'est pas implémentée dans tous les SGDBR comme Oracle.

Syntaxe SQL Server avec la fonction CHOOSE

SELECT CHOOSE (<index>, <constante 1>, <constante 2>,) AS

<entête de colonne>;

Exemple

SELECT CHOOSE(3, 'petit déjeuner inclus', 'demi-pension',

'pension complète') AS Choix;

Choix
pension complète

14. Résumé des syntaxes possibles du SELECT

Voici les principales syntaxes rencontrées dans cette section. Ce ne sont pas toutes les possibilités du SELECT mais l'essentiel.

```
SELECT <nom colonne1>, <nom colonne 2> ... ... ,  
      [SUM/COUNT/AVG/MIN/MAX](<nom colonne 3>),  
      [SUM/COUNT/AVG/MIN/MAX](<nom colonne 4>),
```

```
[CASE <nom colonne 5>  
  WHEN ...  
  END AS <entête de colonne>],  
  <nom colonne 5>,  
  ... ...
```

```
FROM <table 1>  
  
  JOIN <table 2> ON <table1.colonne1> = <table2.colonne1>  
  JOIN <table 3> ON <table1.colonne2> = <table3.colonne2>  
  
WHERE ... ... ...  
  
  GROUP BY <nom colonne1>, <nom colonne 2> ... ...  
  HAVING [SUM/COUNT/AVG/MIN/MAX]<nom colonne 4>  
        <opérateur arithmétique> <valeur> ... ...  
  
ORDER BY <nom colonne1>, <nom colonne 2> ... ...
```

15. Les opérateurs ensemblistes

Les opérateurs ensemblistes travaillent, comme leur nom l'indique, sur des ensembles.

On distingue trois opérateurs principaux qui sont UNION qui va fusionner deux tables ou plus, INTERSECT qui va chercher les lignes communes à deux tables ou plus, et enfin EXCEPT qui va donner les lignes d'une table non présentes dans une autre table.

a. L'opérateur UNION

UNION permet de fusionner les données de plusieurs tables.

Syntaxe de l'opérateur UNION

```
SELECT <colonne 1>, <colonne 2>, ... FROM <table1>  
  
WHERE ...
```

UNION

SELECT <colonne 1>, <colonne 2>, ... FROM <table2>

WHERE ...

ORDER BY <colonne 1> ...

Par exemple, nous avons une table Hotels_FR et une table Hotels_GB qui contiennent les données suivantes :

Table Hotels_FR

idHotel	Libelle	Etoile
1	Ski Hotel	*
2	Art Hotel	**
3	Rose Hotel	***
4	Lions Hotel	****

Table Hotels_GB

idHotel	Libelle	Etoile
1	Lochness Hotel	*
2	Art Hotel	**
3	Rose Hotel	***
4	Lions Hotel	*****
5	Trafalgar Hotel	*****

Si on veut récupérer tous les types d'étoiles contenus dans ces deux tables, on écrira :

SELECT Etoile FROM Hotels_FR

UNION

SELECT Etoile FROM Hotels_GB;

Etoile
*
**

Automatiquement, le système va supprimer les doublons et afficher les valeurs unitairement.

Pour obtenir toutes les valeurs, il faut ajouter la clause ALL comme ceci :

SELECT Etoile FROM Hotels_FR

UNION ALL

SELECT Etoile FROM Hotels_GB;

Etoile
*
**

*
**

Ce qu'il faut retenir sur l'opérateur UNION :

- Il faut que les colonnes sélectionnées dans les différentes tables soient du même type et de la même taille.
- On peut joindre autant de tables que l'on veut.
- Il y a une suppression automatique des doublons par le système.

Autre exemple : si on sélectionne deux colonnes avec la requête ci-dessous, le système supprimera les lignes en double sur les deux colonnes (couples Art Hotel/** et Rose Hotel/**).

L'union peut s'apparenter à un SELECT DISTINCT sur les deux tables.

SELECT Libelle, Etoile FROM Hotels_FR

UNION

SELECT Libelle, Etoile FROM Hotels_GB;

Libelle	Etoile
Art Hotel	**
Lions Hotel	****
Lions Hotel	*****
Lochness Hotel	*
Rose Hotel	***
Ski Hotel	*
Trafalgar Hotel	*****

La clause WHERE peut être utilisée afin de restreindre le champ des lignes sélectionnables. Dans ce cas, les deux SELECT se comportent indépendamment, il faudra ajouter un WHERE soit au premier SELECT soit au deuxième soit aux deux en fonction du résultat recherché.

Si par exemple on ne veut pas de trois étoiles dans la première table Hotels_FR, on écrira :

```
SELECT Libelle, Etoile FROM Hotels_FR
```

```
WHERE Etoile <> '***'
```

```
UNION
```

```
SELECT Libelle, Etoile FROM Hotels_GB;
```

Rose Hotel est sélectionné étant donné qu'il est présent dans l'autre table.

Libelle	Etoile
Art Hotel	**
Lions Hotel	****
Lions Hotel	*****
Lochness Hotel	*
Rose Hotel	***
Ski Hotel	*
Trafalgar Hotel	*****

Si maintenant on veut tous les hôtels qui ne sont pas trois étoiles dans les deux tables, il faudra noter :

```
SELECT Libelle, Etoile FROM Hotels_FR
```

```
WHERE Etoile <> '***'
```

```
UNION
```

```
SELECT Libelle, Etoile FROM Hotels_GB
```

```
WHERE Etoile <> '***';
```

Libelle	Etoile
Art Hotel	**
Lions Hotel	****
Lions Hotel	*****
Lochness Hotel	*
Ski Hotel	*
Trafalgar Hotel	*****

La clause ORDER BY peut également être indiquée, elle sera mise en dernière instruction et s'applique à l'ensemble du résultat obtenu.

Exemple

```
SELECT Libelle, Etoile FROM Hotels_FR
```

```
WHERE Etoile <> '***'
```

```
UNION
```

```
SELECT Libelle, Etoile FROM Hotels_GB
```

```
WHERE Etoile <> '***'
```

```
ORDER BY Etoile;
```

b. L'opérateur INTERSECT

La différence avec l'UNION est simple. UNION sélectionne les données d'une table puis les données de l'autre table, toutes les lignes des deux tables sont récupérées puis les doublons sont retirés.

L'intersection va sélectionner les données qui sont à la fois dans la table 1 et dans la table 2 puis supprimer également les doublons.

Il est également possible d'ajouter les clauses WHERE et ORDER BY comme pour l'UNION.

Syntaxe de l'opérateur INTERSECT

```
SELECT <colonne 1>, <colonne 2>, ... FROM <table1>
```

```
WHERE ...
```

```
INTERSECT
```

```
SELECT <colonne 1>, <colonne 2>, ... FROM <table2>
```

```
WHERE ...
```

```
ORDER BY <colonne 1> ...
```

Exemple

```
SELECT Libelle, Etoile FROM Hotels_FR
```

```
INTERSECT
```

```
SELECT Libelle, Etoile FROM Hotels_GB;
```

Les lignes présentes à la fois dans les deux tables sont celles-ci :

Libelle	Etoile
Art Hotel	**
Rose Hotel	***

Si on reprend le dernier exemple de l'UNION, il ne reste qu'une seule ligne en commun dans les deux tables.

```
SELECT Libelle, Etoile FROM Hotels_FR
```

```
WHERE Etoile <> '***'
```

```
INTERSECT
```

```
SELECT Libelle, Etoile FROM Hotels_GB
```

```
WHERE Etoile <> '***';
```

Libelle	Etoile
Art Hotel	**

À noter que l'opérateur INTERSECT n'est pas implémenté dans tous les SGBDR. Dans ce cas il peut être remplacé par ce type d'ordre :

```
SELECT Libelle, Etoile FROM Hotels_FR AS H1
```


WHERE EXISTS (SELECT Libelle, Etoile FROM Hotels_GB AS H2

WHERE H1.Libelle = H2.Libelle

AND H1.Etoile = H2.Etoile);

c. L'opérateur EXCEPT

L'EXCEPT est le contraire de l'INTERSECT, seules les lignes de la première table qui ne sont pas dans la deuxième table seront sélectionnées.

Il est également possible d'ajouter les clauses WHERE et ORDER BY comme pour l'UNION.

Syntaxe de l'opérateur EXCEPT

SELECT <colonne 1>, <colonne 2>, ... FROM <table1>

WHERE ...

EXCEPT

SELECT <colonne 1>, <colonne 2>, ... FROM <table2>

WHERE ...

ORDER BY <colonne 1> ...

Dans l'exemple suivant, seules les lignes Lions Hotel 4 étoiles et Ski Hotel sont présentes dans la table Hotels_FR et absentes de la table Hotels_GB.

Exemple

SELECT Libelle, Etoile FROM Hotels_FR

EXCEPT

SELECT Libelle, Etoile FROM Hotels_GB;

Libelle	Etoile
Lions Hotel	****
Ski Hotel	*

Attention, l'ordre des instructions avec EXCEPT est important, contrairement aux ordres UNION et INTERSECT, puisqu'il se base sur la première table.

Exemple en inversant les instructions

SELECT Libelle, Etoile FROM Hotels_GB

EXCEPT

SELECT Libelle, Etoile FROM Hotels_FR;

Libelle	Etoile
Lions Hotel	*****
Lochness Hotel	*
Trafalgar Hotel	*****

À noter que l'opérateur EXCEPT n'est pas implémenté dans tous les SGBDR. Dans ce cas, il peut être remplacé par ce type d'ordre :

```
SELECT Libelle, Etoile FROM Hotels_FR AS H1
WHERE NOT EXISTS (SELECT Libelle, Etoile FROM Hotels_GB AS H2
                  WHERE H1.Libelle = H2.Libelle
                  AND H1.Etoile = H2.Etoile);
```

16. Les opérateurs arithmétiques

Les opérateurs arithmétiques permettent d'effectuer des opérations mathématiques sur deux ou plusieurs valeurs dont le type de données est numérique ou temporel.

Liste des opérateurs :

- + : addition.
- - : soustraction.
- * : multiplication.
- / : division.
- % : modulo.

L'opérateur Modulo retourne le reste entier d'une division.

Exemple de multiplication du prix

```
SELECT TOP 3 Prix, Prix * 3 AS Prix3Nuits
FROM Tarifs;
```

Prix	Prix3Nuits
49,99	149,97
59,99	179,97
69,99	209,97

17. Les opérateurs de comparaison

Les opérateurs de comparaison sont utilisés dans les instructions comme IIF ou dans la clause WHERE, par exemple. Ils retournent un type de données Booléen ayant la valeur TRUE pour vrai, FALSE pour faux ou UNKNOWN pour inconnu.

Liste des opérateurs de comparaison :

- = : égal.
- > : strictement supérieur.
- < : strictement inférieur.
- >= : supérieur ou égal.
- <= : inférieur ou égal.

- <>: différent.
- != : différent.
- !> : non supérieur.
- !< : non inférieur.

Exemple de comparateur dans la clause WHERE

```
SELECT TOP 3 TypeChambre, Description, Prix
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =
Tarifs.TypeChambre
WHERE prix > 70;
```

TypeChambre	Description	Prix
6	1 lit double avec bain et WC séparés	79,99
7	1 lit double large avec bain et WC séparés	89,99
3	3 lits simples avec douche et WC séparés	80,49

18. Les opérateurs logiques

Les opérateurs logiques sont utilisés dans les instructions comme dans la clause WHERE, par exemple. Ils retournent un type de données booléen ayant la valeur TRUE pour vrai, FALSE pour faux ou UNKNOWN pour inconnu.

Liste des opérateurs de logiques :

- ALL : tous les éléments d'un jeu de comparaison.
- AND : et.
- ANY : n'importe quel élément d'un jeu de comparaison.
- BETWEEN : entre deux éléments.
- EXISTS : les éléments d'une sous-requête.
- IN : égal à une liste d'éléments.
- LIKE : correspond à un modèle.
- NOT : inverse la valeur de tout autre opérateur (NOT IN, NOT EXISTS).
- OR : ou.
- SOME : certains éléments d'un jeu de comparaison.

Exemple de comparateurs BETWEEN, NOT, IN et AND dans la clause WHERE

```
SELECT TOP 4 TypeChambre, Description, DateDebut
FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre
= Tarifs.TypeChambre
WHERE DateDebut BETWEEN '2021-04-01' AND '2021-10-01'
```

AND hotel NOT IN (1, 2)

ORDER BY typeChambre, DateDebut;

TypeChambre	Description	DateDebut
1	1 lit simple avec douche	2021-04-01
1	1 lit simple avec douche	2021-04-16
1	1 lit simple avec douche	2021-10-01

Dans cet exemple, on remarque que BETWEEN est inclusif sur les deux bornes.

Si on souhaite que la requête soit exclusive sur l'une ou les deux bornes, il est nécessaire d'utiliser les opérateurs > et <.

Exemple avec comparaison exclusive

SELECT TOP 4 TypeChambre, Description, DateDebut

FROM Tarifs INNER JOIN TypesChambre ON TypesChambre.idTypeChambre

= Tarifs.TypeChambre

WHERE DateDebut > '2021-04-01' AND DateDebut < '2021-10-01'

AND hotel NOT IN (1, 2)

ORDER BY typeChambre, DateDebut;

TypeChambre	Description	DateDebut
1	1 lit simple avec douche	2021-04-16
2	2 lits simples avec douche	2021-04-16
3	3 lits simples avec douche et WC séparés	2021-04-16

Le comparateur LIKE peut signifier :

- 'comme' si la chaîne de caractères est précédée et se termine par %.
- 'commence par' si la chaîne de caractères se termine par %.
- 'se termine par' si la chaîne de caractères est précédée par %.

Il est possible d'ajouter un caractère de soulignement _ pour préciser l'emplacement du caractère.

Exemples de comparateur LIKE dans la clause WHERE

SELECT * FROM Hotels

WHERE Libelle like '%s%';

idHotel	Libelle	Etoile
1	Ski Hotel	*
3	Rose Hotel	***
4	Lions Hotel	****

```
SELECT * FROM Hotels
```

```
WHERE Libelle like 's%';
```

idHotel	Libelle	Etoile
1	Ski Hotel	*

```
SELECT * FROM Hotels
```

```
WHERE Libelle like '%s';
```

Cette requête ne retourne aucun résultat car il n'existe pas d'hôtel dans la table dont le libellé se termine par la lettre s.

```
SELECT * FROM Hotels
```

```
WHERE Libelle like '__s%';
```

idHotel	Libelle	Etoile
3	Rose Hotel	***

Pour certains SGDBR comme PostgreSQL, la clause LIKE est sensible à la casse.

19. Comment construire une requête : quelques conseils

Avant de construire une requête, il est souhaitable de disposer du modèle de données de la base, de préférence sous format graphique. Il faut tout d'abord identifier la table principale que l'on souhaite interroger.

Si possible, il faut également se procurer la description des index qui sont posés sur chaque table.

Rechercher dans la table les colonnes dont on a besoin puis noter celles qui manquent.

Pour connaître le nombre de lignes concernées par la requête, nous pouvons faire un COUNT(*) dans un premier temps puis en fonction du résultat, affiner la clause WHERE. Une fois que le nombre de lignes sélectionnées est correct, nous pouvons remplacer le COUNT(*) avec les colonnes de la table principale.

Ensuite, il faut rechercher dans quelle(s) table(s) sont les colonnes manquantes, puis trouver le lien qui permet de réaliser une jointure entre ces deux tables. Attention, le lien peut être constitué de plusieurs colonnes. Pour chaque table, il faut donc ajouter dans la requête les colonnes que l'on souhaite récupérer et les égalités entre les colonnes de clés. De la même façon, le COUNT(*) peut être utilisé pour tester la requête.

Attention également au nombre de lignes de chaque table qui peut influencer énormément les performances.

Il est nécessaire d'utiliser le plus possible les colonnes déclarées dans les index. Une jointure entre trois ou quatre tables importantes peut être très rapide si on utilise les index de chacune des tables. Il suffit d'une table sans index pour écrouler les performances de la requête.

Il existe des outils qui nous indiquent comment le système réalise sa navigation entre les tables pour accéder à une information (voir chapitre Approfondissement - Quelques notions de performances).

Parfois le nombre de lignes ramenées après l'ajout d'une ou de plusieurs jointures est inférieur au nombre de lignes sélectionnables dans la table principale.

C'est normal si toutes les occurrences de la table principale ne sont pas présentes dans les tables annexes. La jointure interne ne ramène que les lignes en commun. Pour vérifier la requête, il est possible d'utiliser dans ce cas une jointure externe qui sélectionnera toutes les lignes de la table principale.

Avec Oracle, on peut utiliser EXPLAIN PLAN pour analyser l'efficacité d'une requête ou EXPLAIN avec MySQL. Nous aborderons ce sujet dans le chapitre Approfondissement - Quelques notions de performances.

Comment être certain que le résultat de la requête est correct ? Pas facile de répondre à cette question, surtout qu'une requête peut souvent s'écrire de plusieurs manières.

Le plus simple est de réaliser l'exercice à la main en regardant table par table leur contenu, en comptant le nombre de lignes de la table principale, puis en cherchant dans les tables annexes les éléments correspondants par des requêtes unitaires sur ces tables. En croisant les résultats, cela permet de valider ou pas une requête.

Lors de l'utilisation de fonctions (count, substr, to_char...) dans les clauses WHERE notamment, on augmente le risque de ne pas ramener le nombre de lignes souhaité si l'on ne maîtrise pas parfaitement la syntaxe. Pour tester une fonction, le plus simple est de la mettre dans les colonnes à sélectionner afin de visualiser son contenu.

Exemple

```
SELECT ACTEUR.NOM || ' ' || ACTEUR.PRENOM ACTEUR,  
ACTEUR.DATE_NAISSANCE "NE LE",  
CAST.ROLE, PAYS.LIBELLE, FILM.TITRE, FILM.DATE_SORTIE  
FROM FILM FILM, CASTING CAST, ACTEUR ACTEUR, PAYS PAYS  
WHERE  
FILM.IDENT_FILM = CAST.IDENT_FILM AND  
CAST.IDENT_ACTEUR = ACTEUR.IDENT_ACTEUR AND  
PAYS.IDENT_PAYS = ACTEUR.NATIONALITE AND  
SUBSTR(FILM.TITRE,1,1) = 'S' AND SUBSTR(FILM.TITRE,6,3) = 'WAR'  
ORDER BY ACTEUR.NOM || ' ' || ACTEUR.PRENOM, FILM.TITRE DESC;
```

En remontant les SUBSTR dans le SELECT et en simplifiant la clause WHERE, on peut visualiser la valeur des SUBSTR ainsi.

```
SELECT FILM.TITRE, SUBSTR(FILM.TITRE,1,1) SUB1,  
SUBSTR(FILM.TITRE,6,3) SUB2  
FROM FILM ORDER BY FILM.TITRE;
```

TITRE	SUB1	SUB2
AVATAR	A	R
BIENVENUE CHEZ LES CH'TIS	B	ENU
NIKITA	N	A
STAR WARS 6 : LE RETOUR DU JEDI	S	WAR
SUBWAY	S	Y

Encore plus simple, en utilisant la pseudo-table DUAL.

```
SELECT SUBSTR('STAR WARS 6 : LE RETOUR DU JEDI',1,1) SUB1,  
SUBSTR('STAR WARS 6 : LE RETOUR DU JEDI',6,3) SUB2 FROM  
DUAL;
```

SUB1	SUB2
S	WAR

Le principe de base lors de la construction d'une requête est de la découper en morceaux simples à tester et de lancer chaque morceau individuellement avant de tester la requête complète.

Ne pas hésiter à tester plusieurs syntaxes pour croiser les résultats.

20. Exercices sur la sélection de données

a. Questions générales

1. Quels sont les quatre types de jointure décrits dans cet ouvrage ?
2. Quelle est la fonction qui permet de compter le nombre d'occurrences ?
3. Quelle est la commande de tri et par défaut le tri est-il descendant ou ascendant ?
4. Quelles sont les fonctions de regroupement utilisables avec un GROUP BY ?
5. Quels sont les trois opérateurs ensemblistes ?

b. Exercices d'application

Description et contenu des tables utilisées dans les exercices suivants.

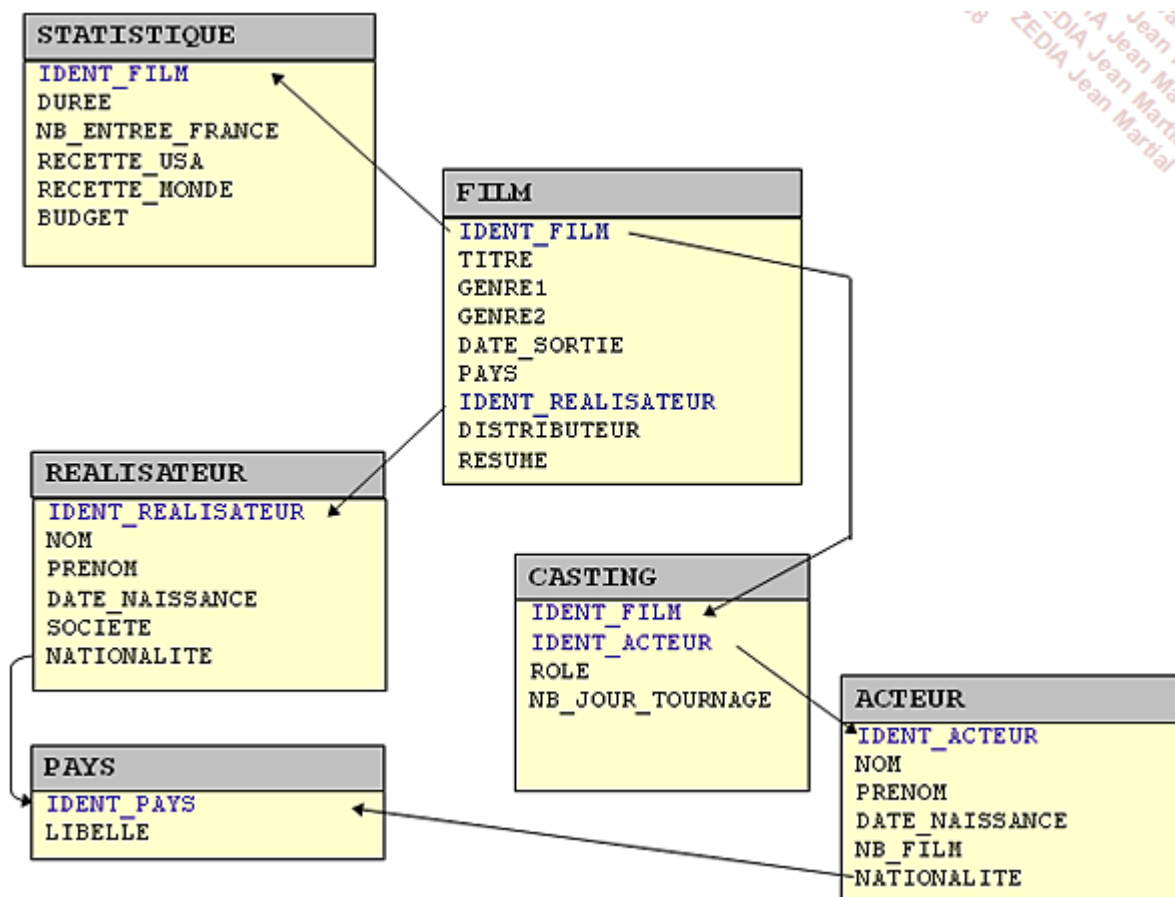


TABLE FILM

Requête de création de la table (syntaxe standard) :

```
CREATE TABLE FILM (IDENT_FILM    INTEGER,
                    TITRE        VARCHAR(50),
```

```

GENRE1      VARCHAR(20),
GENRE2      VARCHAR(20),
DATE_SORTIE  DATE,
PAYS        SMALLINT,
IDENT_REALISATEUR INTEGER,
DISTRIBUTEUR VARCHAR(50),
RESUME       VARCHAR(2000));

```

Requête d'insertion de lignes (syntaxe Oracle) :

```

INSERT INTO FILM VALUES
(1,'SUBWAY','POLICIER','DRAME',TO_DATE('10/04/1985','DD/MM/YYYY'),
1,1,'GAUMONT','Conte les aventures de la population souterraine
dans les couloirs du métro parisien');

```

```

INSERT INTO FILM VALUES
(2,'NIKITA','DRAME','ROMANTIQUE',TO_DATE('21/02/1990','DD/MM/YYYY'
),1,1,'GAUMONT','Nikita condamnée à la prison à perpétuité est
contrainte à travailler secrètement pour le gouvernement en tant
que agent hautement qualifié des services secrets.');
```

```

INSERT INTO FILM VALUES (3,'STAR WARS 6 : LE RETOUR DU JEDI',
'ACTION','SF',TO_DATE('19/10/1983','DD/MM/YYYY'),2,2,'20th
Century Fox ','L'"Empire galactique est plus puissant que jamais :
la construction de la nouvelle arme, l'"Etoile de la Mort, menace
l'univers tout entier.');
```

```

INSERT INTO FILM VALUES
(4,'AVATAR','ACTION','SF',TO_DATE('16/10/2009','DD/MM/YYYY'),2,3,
'20th Century Fox ','Malgré sa paralysie, Jake Sully, un ancien
marine immobilisé dans un fauteuil roulant, est resté un
combattant au plus profond');
```

```

INSERT INTO FILM VALUES (5,'BIENVENUE CHEZ LES CH"'TIS','COMEDIE',
',TO_DATE('27/02/2008','DD/MM/YYYY'),1,4,'PATHE','Philippe Abrams

```


est directeur de la poste de Salon-de-Provence est muté dans le Nord.');

Pour la syntaxe MySQL, il suffit de changer le TO_DATE('10/04/1985','DD/MM/YYYY') par STR_TO_DATE('10/04/1985','%d/%m/%Y').

Contenu de la table FILM :

Ident_Film	Titre	Genre1	Genre2	Date_Sortie	Pay s	Ident_Realisateur	Distributeur	Resume
1	SUBWAY	POLICIER	DRAME	10/04/85	1	1	GAUMONT	Conte les aventures de la population souterraine dans les couloirs du métro parisien
2	NIKITA	DRAME	ROMANTIQUE	21/02/90	1	1	GAUMONT	Nikita condamnée à la prison à perpétuité est contrainte de travailler secrètement pour le gouvernement en tant que agent hautement qualifié des services secrets.
3	STAR WARS 6 - LE RETOUR DU JEDI	ACTION	SF	19/10/83	2	2	20th Century Fox	L'Empire galactique est plus puissant que jamais : la construction de la nouvelle arme, l'Etoile de la Mort, menace l'univers tout entier.

Ident_Film	Titre	Genre1	Genre2	Date_Sortie	Pays	Ident_Realisateur	Distributeur	Resume
4	AVATAR	ACTION	SF	16/10/09	2	3	20th Century Fox	Malgré sa paralysie, Jake Sully, un ancien marine immobilisé dans un fauteuil roulant, est resté un combattant au plus profond
5	BIENVENUE CHEZ LES CH'TIS	COMEDIE		27/02/08	1	4	PATHE	Philippe Abrams est directeur de la poste de Salon-de-Provence est muté dans le Nord.

TABLE ACTEUR

Requête de création de la table (syntaxe standard) :

```
CREATE TABLE ACTEUR (IDENT_ACTEUR INTEGER,
                        NOM          VARCHAR(50),
                        PRENOM       VARCHAR(20),
                        DATE_NAISSANCE DATE,
                        NB_FILM       INTEGER,
                        NATIONALITE   SMALLINT);
```

Pour la syntaxe MySQL il suffit de changer les 'DD/MM/YYYY' par '%d/%m/%Y' et les TO_DATE par STR_TO_DATE.

Requête d'insertion de lignes (syntaxe Oracle) :

```
INSERT INTO ACTEUR VALUES
(1,'ADJANI','ISABELLE',TO_DATE('27/06/1955','DD/MM/YYYY'),42,1);
INSERT INTO ACTEUR VALUES
(2,'LAMBERT','CHRISTOPHE',TO_DATE('29/03/1957','DD/MM/YYYY'),64,1);
INSERT INTO ACTEUR VALUES
```

(3,'BOHRINGER','RICHARD',TO_DATE('16/06/1942','DD/MM/YYYY'),132,1);

INSERT INTO ACTEUR VALUES

(4,'GALABRU','MICHEL',TO_DATE('27/10/1922','DD/MM/YYYY'),277,1);

INSERT INTO ACTEUR VALUES

(5,'PARILLAUD','ANNE',TO_DATE('06/05/1960','DD/MM/YYYY'),35,1);

INSERT INTO ACTEUR VALUES

(6,'FORD','HARRISON',TO_DATE('13/06/1942','DD/MM/YYYY'),64,2);

INSERT INTO ACTEUR VALUES

(7,'FISHER','CARRIE',TO_DATE('21/10/1956','DD/MM/YYYY'),74,2);

INSERT INTO ACTEUR VALUES

(8,'SALDANA','ZOE',TO_DATE('19/06/1978','DD/MM/YYYY'),31,2);

INSERT INTO ACTEUR VALUES

(9,'WEAVER','SIGOURNEY',TO_DATE('08/10/1949','DD/MM/YYYY'),66,2);

INSERT INTO ACTEUR VALUES

(10,'RENO','JEAN',TO_DATE('30/06/1948','DD/MM/YYYY'),75,1);

INSERT INTO ACTEUR VALUES

(11,'BOON','DANY',TO_DATE('26/06/1966','DD/MM/YYYY'),23,1);

INSERT INTO ACTEUR VALUES

(12,'MERAD','KAD',TO_DATE('27/03/1964','DD/MM/YYYY'),55,3);

Contenu de la table ACTEUR :

IDENT_ ACTEUR	NOM	PRENOM	DATE_ NAISSANCE	NB_ FILM	NATIONALITE
1	ADJANI	ISABELLE	27/06/55	42	1
2	LAMBERT	CHRISTOPHE	29/03/57	64	1
3	BOHRINGER	RICHARD	16/06/42	132	1
4	GALABRU	MICHEL	27/10/22	277	1
5	PARILLAUD	ANNE	06/05/60	35	1
6	FORD	HARRISON	13/06/42	64	2
7	FISHER	CARRIE	21/10/56	74	2
8	SALDANA	ZOE	19/06/78	31	2
9	WEAVER	SIGOURNEY	08/10/49	66	2
10	RENO	JEAN	30/06/48	75	1
11	BOON	DANY	26/06/66	23	1

IDENT_ ACTEUR	NOM	PRENOM	DATE_ NAISSANCE	NB_ FILM	NATIONALITE
12	MERAD	KAD	27/03/64	55	3

TABLE STATISTIQUE

Requête de création de la table (syntaxe standard) :

```
CREATE TABLE STATISTIQUE
(
  IDENT_FILM      INTEGER,
  DUREE           INTEGER,
  NB_ENTREE_FRANCE DECIMAL(15,0),
  RECETTE_USA     DECIMAL(15,2),
  RECETTE_MONDE   DECIMAL(15,2),
  BUDGET          DECIMAL(12,2));
```

Requête d'insertion de lignes (syntaxe standard) :

```
INSERT INTO STATISTIQUE VALUES
(1,104,2917562,390659,1272637.45,2.6);

INSERT INTO STATISTIQUE VALUES (2,118,3787845,5017971,0,7.6);

INSERT INTO STATISTIQUE VALUES
(3,133,4263000,191648000,472000000,32);

INSERT INTO STATISTIQUE VALUES
(4,170,12018251,760505847,2946271769,237);

INSERT INTO STATISTIQUE VALUES (5,100,21000000,0,245000000,11);
```

Contenu de la table STATISTIQUE :

IDENT_ FILM	DUREE	NB_ENTREE_ FRANCE	RECETTE_ USA	RECETTE_ MONDE	BUDGET
1	104	2 917 562	390 659	1 272 637	2,6
2	118	3 787 845	5 017 971	0	7,6
3	133	4 263 000	191 648 000	472 000 000	32
4	170	12 018 251	760 505 847	2 946 271 769	237
5	100	21 000 000	0	245 000 000	11

TABLE REALISATEUR

Requête de création de la table (syntaxe standard) :

```
CREATE TABLE REALISATEUR
(
  IDENT_REALISATEUR INTEGER,
```

```

NOM          VARCHAR(50),
PRENOM       VARCHAR(20),
DATE_NAISSANCE  DATE,
NB_FILM_ECRIT  INTEGER,
NB_FILM_PRODUI  INTEGER,
NATIONALITE    SMALLINT);

```

Requête d'insertion de lignes (syntaxe Oracle) :

```

INSERT INTO REALISATEUR VALUES
('1','BESSON','LUC',TO_DATE('18/03/1959','DD/MM/YYYY'),40,99,1);
INSERT INTO REALISATEUR VALUES
('2','LUCAS','GEORGES',TO_DATE('14/05/1944','DD/MM/YYYY'),79,64,2);
INSERT INTO REALISATEUR VALUES
('3','CAMERON','JAMES',TO_DATE('16/08/1954','DD/MM/YYYY'),22,23,2);
INSERT INTO REALISATEUR VALUES
('4','BOON','DANY',TO_DATE('26/06/1966','DD/MM/YYYY'),5,1,1);

```

Contenu de la table REALISATEUR :

IDENT_ REALISATEU R	NOM	PRENO M	DATE_ NAISSANC E	NB_ FILM - ECRI T	NB_ FILM_ PRODUI T	NATIONALIT E
1	BESSON	LUC	18/03/59	40	99	1
2	LUCAS	GEORGE S	14/05/44	79	64	2
3	CAMERO N	JAMES	16/08/54	22	23	2
4	BOON	DANY	26/06/66	5	1	1

TABLE CASTING

Requête de création de la table (syntaxe standard) :

```

CREATE TABLE CASTING
(
  IDENT_FILM    INTEGER,
  IDENT_ACTEUR  INTEGER,
  ROLE          VARCHAR(100),
  NB_JOUR_TOURNAGE INTEGER);

```

Requête d'insertion de lignes (syntaxe standard) :

INSERT INTO CASTING VALUES (1,1,'HELENA',100);
 INSERT INTO CASTING VALUES (1,2,'FRED',100);
 INSERT INTO CASTING VALUES (1,3,'INSPECTEUR GESBERG',NULL);
 INSERT INTO CASTING VALUES (1,4,'LE FLEURISTE',35);
 INSERT INTO CASTING VALUES (1,10,'LE BATTEUR',20);
 INSERT INTO CASTING VALUES (2,5,'NIKITA',68);
 INSERT INTO CASTING VALUES (2,10,'VICTOR LE NETTOYEUR',9);
 INSERT INTO CASTING VALUES (3,6,'HAN SOLO',201);
 INSERT INTO CASTING VALUES (3,7,'PRINCESSE LEIA',203);
 INSERT INTO CASTING VALUES (4,8,'NEYTIRI',50);
 INSERT INTO CASTING VALUES (4,9,'Dr. Grace Augustine',45);
 INSERT INTO CASTING VALUES (5,11,'ANTOINE BAILLEUL',125);
 INSERT INTO CASTING VALUES (5,12,'PHILIPPE ABRAMS',126);

Contenu de la table CASTING :

IDENT_FILM	IDENT_ACTEUR	Rôle	NB_JOUR_TOURNAGE
1	1	HELENA	100
1	2	FRED	100
1	3	INSPECTEUR GESBERG	NULL
1	4	LE FLEURISTE	35
1	10	LE BATTEUR	20
2	5	NIKITA	68
2	10	VICTOR LE NETTOYEUR	9
3	6	HAN SOLO	201
3	7	PRINCESSE LEIA	203
4	8	NEYTIRI	50
4	9	Dr. Grace Augustine	45
5	11	ANTOINE BAILLEUL	125
5	12	PHILIPPE ABRAMS	126

TABLE PAYS

Requête de création de la table (syntaxe standard) :

```
CREATE TABLE PAYS
```

```
(IDENT_PAYS SMALLINT,  
  LIBELLE  VARCHAR(100));
```

Requête d'insertion de lignes (syntaxe standard) :

```
INSERT INTO PAYS VALUES (1,'FRANCE');
```

```
INSERT INTO PAYS VALUES (2,'USA');
```

```
INSERT INTO PAYS VALUES (3,'ALGERIE');
```

Contenu de la table PAYS :

IDENT_PAYS	LIBELLE
1	FRANCE
2	USA
3	ALGERIE

Premier exercice

Sélectionner toutes les informations sur les films réalisés par un réalisateur français triés par le nom du film.

Deuxième exercice

Sélectionner le nom du film, la date de sortie, le nom du réalisateur, le nom des acteurs, leur date de naissance, ainsi que le budget du film. Le tout trié en ordre descendant par titre du film et nom des acteurs.

Troisième exercice

Trouver le nombre d'acteurs par film dans la base de données. Afficher le titre, la date de sortie, le nom du réalisateur et le distributeur.

Quatrième exercice

Sélectionner le titre du film, la date de sortie, le nom et prénom du réalisateur, le nom et prénom de l'acteur, sa date de naissance, le budget du film et le nombre d'entrées en France des films qui ont un acteur algérien.

Cinquième exercice

Sélectionner le film qui a réalisé la recette la plus élevée dans le monde.

Sixième exercice

Sélectionner l'acteur qui a joué dans deux films différents.

Septième exercice

Sélectionner la personne qui est à la fois réalisateur et acteur.

Huitième exercice

Sélectionner les acteurs qui ont joué dans des films dont le nom du film commence par la lettre S. Indiquer leur rôle et leur nationalité.

Neuvième exercice

Sélectionner les acteurs qui sont nés entre 1948 et mai 1978 ainsi que le nombre de jours total de tournage qu'ils ont réalisés.

L'insertion de données

Après avoir créé la structure des tables et avant de pouvoir sélectionner des données dans la base, il faut insérer des valeurs.

Nous avons vu dans le chapitre sur le LDD, qu'il est possible de remplir une table à partir d'une autre lors de la création de la table par CREATE TABLE, dans la majorité des cas la table est créée à vide puis ensuite on insère des données.

1. L'ordre INSERT

L'ordre INSERT va permettre d'insérer des valeurs dans une table.

La méthode la plus simple consiste à insérer une ligne à la fois et toutes les colonnes de la table.

Dans ce cas, il suffit d'indiquer au système toutes les valeurs que l'on veut insérer dans l'ordre des colonnes de la table.

Exemple d'insertion dans la table Tarifs

```
INSERT INTO Tarifs
```

```
VALUES
```

```
(57, 1, 1, '2022-04-15', '2022-09-30', 58.49);
```

Il faut absolument que les valeurs soient dans l'ordre des colonnes dans ce cas précis. C'est assez risqué de travailler ainsi, si la table évolue avec des nouvelles colonnes cet ordre ne fonctionnera plus. Le système indiquera qu'il manque des valeurs.

Il est préférable d'indiquer les colonnes concernées ainsi :

```
INSERT INTO Tarifs
```

```
(idTarif, Hotel, TypeChambre, DateDebut, DateFin, Prix)
```

```
VALUES
```

```
(58, 1, 2, '2022-04-15', '2022-09-30', 69.99);
```

Pour la syntaxe MySQL, il suffit de changer le TO_DATE('10/04/1985','DD/MM/YYYY') par STR_TO_DATE('10/04/1985','%d/%m/%Y').

Il est possible d'insérer plusieurs lignes à la fois en séparant les lignes de valeurs par des virgules, ainsi :

```
INSERT INTO Tarifs
```

```
(idTarif, Hotel, TypeChambre, DateDebut, DateFin, Prix)
```

```
VALUES
```

```
(59, 1, 3, '2022-04-15', '2022-09-30', 81.49),
```

```
(60, 1, 4, '2022-04-15', '2022-09-30', 69.99),
```

```
(61, 1, 5, '2022-04-15', '2022-09-30', 81.49);
```


Comme il n'y a pas d'option d'auto-incrémentation sur la colonne idTarif, il est nécessaire de préciser cette colonne avec un nombre différent à chaque ligne puisqu'il s'agit de la clé primaire.

La plupart des colonnes de la table n'étant pas déclarées NOT NULL, il est possible de ne remplir que certaines colonnes. Attention néanmoins aux contraintes d'intégrité éventuelles.

L'ordre ci-dessous ne renseigne que trois colonnes de la table :

```
INSERT INTO Tarifs
```

```
(idTarif, Hotel, TypeChambre)
```

```
VALUES
```

```
(62, 1, 6);
```

Suite à l'ajout de six lignes dans la table Tarifs, celle-ci contient maintenant les données suivantes :

idTarif	Hotel	TypeChambre	DateDebut	DateFin	Prix
56	3	7	2021-04-16	2021-12-14	89,99
57	1	1	2022-04-15	2022-09-30	58,49
59	1	3	2022-04-15	2022-09-30	81,49
60	1	4	2022-04-15	2022-09-30	69,99
61	1	5	2022-04-15	2022-09-30	81,49
62	1	6	NULL	NULL	NULL

On peut remarquer que la dernière ligne n'a pas de date de début ni de date de fin ni de prix, étant donné que l'on n'a rien précisé lors de l'INSERT.

Syntaxe générale

```
INSERT INTO <nom table>
```

```
(<colonne 1>, <colonne 2>, ... ... )
```

```
VALUES
```

```
(<val1>, <val2>, ... ... );
```

2. L'insertion à partir d'une autre table

Dans les exemples précédents, on insère des constantes dans la table et les lignes une par une.

Il peut être intéressant de récupérer les données dans d'autres tables ou à partir de la même table afin de gagner en productivité.

S'il existe une autre table qui contient des données que l'on désire insérer dans la table Chambres, il faut dans ce cas utiliser un SELECT à l'intérieur de l'INSERT et indiquer les colonnes que l'on désire récupérer.

Exemple avec la table Chambres SAV que nous avons créée lors du chapitre précédent et qui contient 27 chambres comme la table Chambres

Table Chambres_SAV

idChambre	Hotel	TypeChambre	NumChambre	Commentaire
1	1	1	1	Vue sur mer
2	1	2	2	
3	1	3	3	NULL
4	1	4	4	NULL
5	1	5	5	

Il faut que l'on récupère les colonnes Hotel et TypeChambre. idChambre et NumChambre vont être calculés car les valeurs ne peuvent pas être identiques à des enregistrements déjà existants dans la table Chambres.

Requêtes de création de la table Chambres_SAV :

--SQL Server

```
SELECT * INTO Chambres_SAV
```

```
FROM Chambres;
```

--Oracle et PostgreSQL

```
CREATE TABLE SAV_Chambres AS SELECT * FROM Chambres;
```

--MySQL

```
CREATE TABLE SAV_Chambres SELECT * FROM Chambres;
```

Exemple d'insertion à partir d'une autre table avec la syntaxe SQL Server pour l'hôtel n° 1 :

```
INSERT INTO Chambres
```

```
SELECT idChambre + 28, Hotel, TypeChambre, NumChambre + 6, NULL
```

```
FROM Chambres_SAV
```

```
WHERE hotel = 1;
```

Exemple d'insertion à partir d'une autre table avec la syntaxe Oracle

```
INSERT INTO chambres
```

```
SELECT idChambre.NEXTVAL, Hotel, TypeChambre, NumChambre.NEXTVAL, NULL
```

```
FROM Chambres_SAV
```

```
WHERE hotel = 1;
```

Cette requête insère les sept lignes de la table Chambres dans la table Chambres_SAV.

Résultat dans la table Chambres, les six lignes sont en gras :

idChambre	Hotel	TypeChambre	NumChambre	Commentaire
27	4	6	6	NULL
28	4	7	7	NULL
29	1	1	7	NULL
30	1	2	8	NULL
31	1	3	9	NULL
32	1	4	10	NULL
33	1	5	11	NULL
34	1	6	12	NULL

3. Résumé des syntaxes de l'INSERT

INSERT INTO <nom table> (<nom colonne1>, <nom colonne 2>)

VALUE (<valeur 1>, <valeur 2>)

ou :

INSERT INTO <nom table> (<nom colonne1>, <nom colonne 2>)

SELECT <nom colonne1>, <nom colonne 2>

FROM <table 2>

[JOIN <table 3> ON <table2.colonne1> = <table3.colonne1>]

[JOIN <table 4> ON <table2.colonne2> = <table4.colonne2>]

[WHERE]

[GROUP BY <nom colonne1>, <nom colonne 2>]

[HAVING [SUM/COUNT/AVG/MIN/MAX]<nom colonne 4> <opérateur
arithmétique> <valeur>]

[ORDER BY <nom colonne1>, <nom colonne 2> ...]

4. Exercices sur l'insertion de données

Ces exercices sont basés sur les tables précisées dans la section Exercices sur la sélection de données de ce chapitre.

Premier exercice

Ajout d'un nouveau film dans la table FILM :

Numéro 6, Die Hard 4, film d'action sorti le 4 juillet 2007. Le réalisateur est Len Wiseman et distribué par la Twentieth Century Fox Film Corporation.

Résumé : « Pour sa quatrième aventure, l'inspecteur John McClane se trouve confronté à un nouveau genre de terrorisme. Le réseau informatique national qui contrôle absolument toutes les communications, les transports et l'énergie des États-Unis est détruit de façon systématique, plongeant le pays dans le chaos. »

Deuxième exercice

Création d'une nouvelle table ACTEUR_FRANCE à partir de la table ACTEUR en ne sélectionnant que les acteurs français.

Troisième exercice

Insertion de « Bruce Willis » né le 19 mars 1955 avec 89 films à son actif et 152 jours de tournage dans la table des acteurs et insertion d'un enregistrement dans la table CASTING le reliant au film numéro 6.

La suppression de données

La suppression n'est pas l'ordre le plus utilisé mais il est important de connaître la syntaxe. Il faut faire très attention lors de l'utilisation du DELETE car il arrive souvent que l'on supprime plus de lignes que prévu.

Pour éviter cela, deux conseils : tester au préalable avec un SELECT la clause WHERE afin de vérifier le nombre de lignes concernées, et utiliser à bon escient les ordres COMMIT et ROLLBACK que nous détaillerons dans le chapitre Le contrôle de transactions (TCL).

1. L'ordre DELETE

L'ordre DELETE va permettre de supprimer une ou plusieurs lignes dans une table.

La syntaxe de la clause DELETE est simple si l'on ne restreint pas les lignes. Si l'on ne précise pas de clause WHERE, toutes les lignes de la table sont supprimées.

Si ce sont des tables importantes, il est déconseillé d'utiliser un DELETE sans restriction, il existe un risque important de faire planter le système qui ne pourra pas stocker toutes les lignes détruites afin de pouvoir les restituer si un ROLLBACK est demandé (voir chapitre Approfondissement).

Une requête DELETE peut ne pas aboutir si une contrainte d'intégrité n'est plus vérifiée à cause de la suppression d'une ligne.

Exemple de suppression de toutes les lignes d'une table

```
DELETE FROM Chambres;
```

Maintenant, pour ne supprimer que certaines lignes, il faut ajouter la clause WHERE.

Ensuite, toutes les possibilités de la syntaxe utilisée par le SELECT sont possibles.

Par exemple, pour supprimer les lignes de la table Chambres insérées précédemment qui ont un numéro d'id supérieur à 28 :

```
DELETE FROM Chambres
```

```
WHERE idChambre > 28;
```

Il est également possible de conditionner la suppression avec un ordre SELECT. Par exemple, pour supprimer les lignes de la table Chambres dont l'hôtel n'existe pas dans la table Hotels :

```
DELETE FROM Chambres
```

```
WHERE Hotel NOT IN (SELECT idHotel FROM Hotels);
```

2. Exercices sur la suppression de données

Ces exercices sont basés sur les tables précisées dans la section Exercices sur la sélection de données de ce chapitre.

Premier exercice

Suppression de la table ACTEUR de 'CARRIE FISHER'.

Deuxième exercice

Suppression de la table STATISTIQUE dont la recette mondiale est égale à 0.

La modification de données

Nous avons vu l'insertion et la suppression de données, il reste maintenant la modification d'une ou de plusieurs lignes d'une table avec l'ordre UPDATE.

1. L'ordre UPDATE

Cet ordre est syntaxiquement simple à mettre en œuvre. Il permet de mettre à jour 1 à x colonnes d'une table et de 1 à x lignes avec le même ordre.

Comme le DELETE il est possible d'ajouter des restrictions avec la clause WHERE.

Exemple de modification dans la table Tarifs

UPDATE Tarifs SET Prix = 99;

Si on ne met pas de restriction, **tous les prix** de la table Tarifs vont passer à 99.

idTarif	hotel	typeChambre	DateDebut	DateFin	Prix
1	1	1	2021-10-01	2022-04-14	99
2	1	2	2021-10-01	2022-04-14	99
3	1	3	2021-10-01	2022-04-14	99
4	1	4	2021-10-01	2022-04-14	99
5	1	5	2021-10-01	2022-04-14	99
6	1	6	2021-10-01	2022-04-14	99
7	1	7	2021-10-01	2022-04-14	99
8	2	1	2021-12-15	2022-04-15	99

Il est donc important avant d'exécuter un ordre UPDATE de bien vérifier le nombre de lignes impactées par cette modification. Il est conseillé de faire un SELECT en amont pour visualiser les lignes sélectionnées.

Dans Oracle, en cas d'erreur, il est toujours possible d'exécuter un ROLLBACK juste après l'UPDATE afin de rétablir les données d'origine.

Exemple de modification dans la table Tarifs sur une seule ligne

UPDATE Tarifs SET Prix = 68.99

WHERE idTarif = 3;

Seule la ligne avec le numéro 3 est mise à jour.

idTarif	hotel	typeChambre	DateDebut	DateFin	Prix
1	1	1	2021-10-01	2022-04-14	49,99
2	1	2	2021-10-01	2022-04-14	59,99
3	1	3	2021-10-01	2022-04-14	68,99
4	1	4	2021-10-01	2022-04-14	59,99
5	1	5	2021-10-01	2022-04-14	69,99

```
UPDATE Tarifs SET Prix = 58.99
```

```
WHERE hotel = 1 AND Prix BETWEEN 50 AND 60;
```

Dans ce cas, deux lignes sont mises à jour : la 2 et 4. À noter que le système met à jour même si la valeur actuelle est déjà de 58,99.

Toutes les possibilités du WHERE sont donc utilisables avec le UPDATE.

Il est également possible de réaliser un calcul lors de la mise à jour. Si par exemple nous voulons augmenter tous les tarifs de 20 %, il faut écrire :

```
UPDATE Tarifs SET Prix = Prix * 1.20;
```

Pour pouvoir modifier plusieurs colonnes, il faut mettre les colonnes à suivre.

Par exemple, pour modifier le prix et la date de fin d'achat, la syntaxe est la suivante :

```
UPDATE Tarifs SET Prix = Prix * 1.20
```

```
, DateFin = '2022-04-01'
```

```
WHERE hotel = 1
```

```
AND Prix BETWEEN 50 AND 60 AND DateDebut = '2021-10-01';
```

Il est possible d'attribuer une valeur NULL à une colonne en saisissant :

```
UPDATE Tarifs SET Prix = NULL
```

```
WHERE Prix > 99;
```

Syntaxe de l'ordre UPDATE

```
UPDATE <nom table> SET <nom colonne 1> = <valeur 1>,
```

```
<nom colonne 2> = <valeur 2>,
```

```
<nom colonne 3> = <valeur 3>,
```

```
... ..
```

```
WHERE ... .. ;
```

2. Exercices sur la modification de données

Ces exercices sont basés sur les tables précisées dans la section Exercices sur la sélection de données de ce chapitre.

Premier exercice

Modifier le GENRE2 de la table FILM afin de compléter la zone lorsque celle-ci n'est pas renseignée, puis remplacer 'SF' par 'SCIENCE FICTION'.

Deuxième exercice

Division de RECETTE_USA et RECETTE_MONDE par 1 000 000 afin d'exprimer le chiffre en millions dans la table STATISTIQUE.

Agir sur les données à partir d'une autre table

1. L'ordre MERGE

Cette fonction permet d'insérer, de modifier ou de supprimer des enregistrements sur une table à partir de données d'une autre table ou vue.

Syntaxe

MERGE INTO <nom table1>

USING <nom table2>

ON <conditions>

WHEN MATCHED THEN

UPDATE SET <table1.colonne1> = valeur1, <table1.colonne2> =
valeur2,

DELETE WHERE <conditions2>

WHEN NOT MATCHED THEN

INSERT <colonne1>, <colonne3>,

VALUES (valeur1, valeur3,)

- MERGE INTO : table à modifier.
- USING : les données source.
- ON : conditions.
- WHEN MATCHED THEN : les modifications ou suppressions effectuées lorsque la ou les conditions sont vérifiées.
- WHEN NOT MATCHED THEN : les ajouts effectués lorsque la condition n'est pas vérifiée.

Exemple

Nous souhaitons augmenter le prix de 15% de la table Tarifs lorsque la colonne Commentaire de la chambre contient une donnée.

Voici la requête qui sélectionne les tarifs à augmenter de 15%. La jointure porte sur les deux clés étrangères de la table Tarifs pour qu'il n'y ait pas de doublon. L'idTarif est affiché pour s'en assurer. La restriction permet ne pas sélectionner les commentaires dont la valeur est NULL ou contient un espace.

SELECT idTarif, Prix, Commentaire FROM Tarifs

INNER JOIN Hotels ON Hotels.idHotel = Tarifs.hotel

INNER JOIN Chambres ON Hotels.idHotel = Chambres.Hotel AND

Chambres.TypeChambre = Tarifs.typeChambre

WHERE Commentaire IS NOT NULL AND Commentaire <> '';

Résultat

idTarif	Prix	Commentaire
1	49,99	Belle vue
5	69,99	Belle vue
11	68,99	Vue sur mer
12	80,49	Vue sur mer
16	68,99	Vue sur mer
18	68,99	Vue sur mer
29	57,49	Belle vue
33	80,49	Belle vue
46	59,99	Vue sur mer
47	69,99	Vue sur mer
51	59,99	Vue sur mer
53	59,99	Vue sur mer
57	58,49	Belle vue
61	81,49	Belle vue

Et une requête pour vérifier quelques lignes qui ne vont pas être modifiées :

SELECT idTarif, Prix FROM Tarifs

WHERE idTarif BETWEEN 2 AND 4;

idTarif	Prix
2	59,99
3	68,99
4	59,99

Requête avec la clause MERGE

MERGE INTO Tarifs

USING (SELECT idHotel, TypeChambre, Commentaire FROM Hotels

INNER JOIN Chambres ON Hotels.idHotel = Chambres.Hotel

) Source

ON (Source.idHotel = Tarifs.hotel

AND Source.TypeChambre = Tarifs.typeChambre

AND Commentaire IS NOT NULL AND Commentaire <> ")

WHEN MATCHED THEN

UPDATE SET Prix = Prix * 1.15;

Résultat

idTarif	Prix	Commentaire
1	57,4885	Belle vue
5	80,4885	Belle vue
11	79,3385	Vue sur mer
12	92,5635	Vue sur mer
16	79,3385	Vue sur mer
18	79,3385	Vue sur mer
29	66,1135	Belle vue
33	92,5635	Belle vue
46	68,9885	Vue sur mer
47	80,4885	Vue sur mer
51	68,9885	Vue sur mer
53	68,9885	Vue sur mer
57	67,2635	Belle vue
61	93,7135	Belle vue

Le prix a été augmenté pour les chambres commentées et n'a pas été modifié pour les autres tarifs.

idTarif	Prix
2	59,99
3	68,99
4	59,99

Solutions des exercices

1. Solutions des exercices sur la sélection de données

a. Questions générales

1. *Quels sont les quatre types de jointure décrits dans cet ouvrage ?*

Interne, externe, naturelle et croisée

2. *Quelle est la fonction qui permet de compter le nombre d'occurrences ?*

COUNT

3. *Quelle est la commande de tri et par défaut le tri est-il descendant ou ascendant ?*

ORDER BY, par défaut ascendant (ASC)

4. *Quelles sont les fonctions de regroupement utilisables avec un GROUP BY ?*

Les plus utilisées sont COUNT, SUM, AVG, MIN, MAX. Il existe également VARIANCE et STDDEV

5. *Quels sont les trois opérateurs ensemblistes ?*

UNION, INTERSECT et EXCEPT

b. Exercices d'application

Premier exercice

Sélectionner toutes les informations sur les films réalisés par un réalisateur français trié par le nom du film.

Première possibilité, on connaît la valeur de la colonne NATIONALITE qui correspond à FRANCE : 1.

```
SELECT * FROM FILM T1, REALISATEUR T2 WHERE  
    T1.IDENT_REALISATEUR = T2.IDENT_REALISATEUR AND  
    T2.NATIONALITE = 1
```

```
ORDER BY T1.TITRE;
```

Deuxième possibilité, on ajoute une jointure avec la table PAYS et l'on teste sur le libellé « FRANCE ».

```
SELECT * FROM FILM T1, REALISATEUR T2, PAYS T3 WHERE  
    T1.IDENT_REALISATEUR = T2.IDENT_REALISATEUR AND  
    T2.NATIONALITE = T3.IDENT_PAYS AND  
    T3.LIBELLE = 'FRANCE'
```

```
ORDER BY T1.TITRE;
```

ou autre syntaxe possible :

```
SELECT * FROM FILM T1  
    INNER JOIN REALISATEUR T2 ON T1.IDENT_REALISATEUR =  
        T2.IDENT_REALISATEUR  
    INNER JOIN PAYS T3    ON T2.NATIONALITE = T3.IDENT_PAYS
```

WHERE T3.LIBELLE = 'FRANCE'

ORDER BY T1.TITRE;

Résultats

On constate que TOUTES les colonnes des trois tables sont prises en compte. En effet, nous avons mis * dans la requête donc le système prend les colonnes de la première table, puis la deuxième et ainsi de suite.

IDENT_FILM				TITRE	GENRE1			
GENRE2				DATE_SOR		PAYS		
IDENT_REALISATEUR				DISTBIUTEUR	RESUME			
IDENT_REALISATEUR				NOM	PRENOM			
DATE_NAI		NB_FILM_ECRIT		NB_FILM_PRODUIT				
NATIONALITE		IDENT_PAYS		LIBELLE				
5 BIENVENUE CHEZ LES CH'TIS					COMEDIE			
27/02/08		1	4	PATHE	Philippe Abrams est directeur			
de la poste de Salon-de-Provence est muté dans le Nord.								
4 BOON				DANY		26/06/66		
5	1	1	1 FRANCE					
2 NIKITA					DRAME			
ROMANTIQUE		21/02/90	1	1 GAUMONT				Nikita
condamnée à la prison à perpétuité est contrainte de travailler								
secrètement pour le gouvernement en tant que agent hautement qualifié des services secrets.								
1 BESSON				LUC		18/03/59		
40	99	1	1 FRANCE					
1 SUBWAY					POLICIER			
DRAME		10/04/85	1	1 GAUMONT				
Conte les aventures de la population souterraine dans les couloirs								
du métro parisien								
1 BESSON				LUC		18/03/59		
40	99	1	1 FRANCE					

Pour avoir un résultat plus lisible, il faut sélectionner certaines colonnes comme ceci :

```
SELECT T1.TITRE, T1.GENRE1, T1.DATE_SORTIE, T2.NOM, T2.PRENOM
```

```
FROM FILM T1, REALISATEUR T2, PAYS T3 WHERE
```

```
T1.IDENT_REALISATEUR = T2.IDENT_REALISATEUR AND
```

```
T2.NATIONALITE = T3.IDENT_PAYS AND
```

```
T3.LIBELLE = 'FRANCE'
```

ORDER BY T1.TITRE;

Résultats

TITRE	GENRE1	DATE_SORTIE	NOM	PRENOM
BIENVENUE CHEZ LES CH'TIS	COMEDIE	27/02/08	BOON	DANY
NIKITA	DRAME	21/02/90	BESSON	LUC
SUBWAY	POLICIER	10/04/85	BESSON	LUC

Deuxième exercice

Sélectionner le nom du film, la date de sortie, le nom du réalisateur, le nom des acteurs, leur date de naissance, ainsi que le budget du film. Le tout trié en descendant par titre du film et nom des acteurs.

```
SELECT T1.TITRE, T1.DATE_SORTIE, T2.NOM, T4.NOM,  
T4.DATE_NAISSANCE, T5.BUDGET  
FROM FILM T1, REALISATEUR T2, CASTING T3, ACTEUR T4, STATISTIQUE  
T5 WHERE  
T1.IDENT_REALISATEUR = T2.IDENT_REALISATEUR AND  
T1.IDENT_FILM = T3.IDENT_FILM AND  
T1.IDENT_FILM = T5.IDENT_FILM AND  
T3.IDENT_ACTEUR = T4.IDENT_ACTEUR  
ORDER BY T1.TITRE, T4.NOM DESC;
```

ou

```
SELECT FILM.TITRE, FILM.DATE_SORTIE, REAL.NOM, ACTEUR.NOM,  
ACTEUR.DATE_NAISSANCE, STAT.BUDGET  
FROM FILM FILM, REALISATEUR REAL, CASTING CAST, ACTEUR ACTEUR,  
STATISTIQUE STAT WHERE  
FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND  
FILM.IDENT_FILM = CAST.IDENT_FILM AND  
FILM.IDENT_FILM = STAT.IDENT_FILM AND  
CAST.IDENT_ACTEUR = ACTEUR.IDENT_ACTEUR  
ORDER BY FILM.TITRE, ACTEUR.NOM DESC;
```

Dans cet exemple, nous avons pris CAST comme alias à la table CASTING. Il faut faire attention aux mots réservés par le SGBDR. Dans ce cas sur Oracle, la syntaxe passe, alors que sur MySQL, elle ne passe pas car CAST est un mot réservé.

Résultats

TITRE	DATE_SORTIE	NOM	NOM	DATE_NAISSANCE	BUDGET
AVATAR	16/10/09	CAMERON	WEAVER	08/10/49	237
AVATAR	16/10/09	CAMERON	SALDANA	19/10/78	237
BIENVENUE CHEZ LES CH'TIS	27/02/08	BOON	MERAD	27/03/64	11
BIENVENUE CHEZ LES CH'TIS	27/02/08	BOON	BOON	26/06/66	11
NIKITA	21/02/90	BESSON	RENO	30/04/48	7,6
NIKITA	21/02/90	BESSON	PARILLAUD	06/05/60	7,6
STAR WARS 6 : LE RETOUR DU JEDI	19/10/83	LUCAS	FORD	13/06/42	32
STAR WARS 6 : LE RETOUR DU JEDI	19/10/83	LUCAS	FISHER	21/10/56	32
SUBWAY	10/04/85	BESSON	RENO	30/06/48	2,6
SUBWAY	10/04/85	BESSON	LAMBERT	29/03/57	2,6
SUBWAY	10/04/85	BESSON	GALABRU	27/10/22	2,6
SUBWAY	10/04/85	BESSON	BOHRINGER	16/06/42	2,6
SUBWAY	10/04/85	BESSON	ADJANI	27/06/55	2,6

On constate que deux colonnes ont le même nom car on n'a pas mis d'alias sur les noms de colonne. On aurait pu écrire :

```
SELECT FILM.TITRE, FILM.DATE_SORTIE, REAL.NOM REALISATEUR,  
ACTEUR.NOM ACTEUR, ACTEUR.DATE_NAISSANCE, STAT.BUDGET  
FROM FILM FILM, REALISATEUR REAL, CASTING CAST, ACTEUR ACTEUR,  
STATISTIQUE STAT WHERE  
  
FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND  
FILM.IDENT_FILM      = CAST.IDENT_FILM AND  
FILM.IDENT_FILM      = STAT.IDENT_FILM AND  
CAST.IDENT_ACTEUR    = ACTEUR.IDENT_ACTEUR  
ORDER BY FILM.TITRE, ACTEUR.NOM DESC;
```

Troisième exercice

Trouver le nombre d'acteurs par film dans la base de données. Afficher le titre, la date de sortie, le nom du réalisateur et le distributeur.

```
SELECT T1.TITRE, T1.DATE_SORTIE, T2.NOM,  
T1.DISTRIBUTEUR, COUNT(T3.IDENT_FILM) AS NB_ACTEUR  
FROM FILM T1, REALISATEUR T2, CASTING T3, ACTEUR T4, STATISTIQUE  
T5 WHERE  
  
T1.IDENT_REALISATEUR = T2.IDENT_REALISATEUR AND  
  
T1.IDENT_FILM      = T3.IDENT_FILM AND  
  
T1.IDENT_FILM      = T5.IDENT_FILM AND  
  
T3.IDENT_ACTEUR     = T4.IDENT_ACTEUR  
  
GROUP BY T1.TITRE, T1.DATE_SORTIE, T2.NOM, T1.DISTRIBUTEUR  
  
ORDER BY T1.TITRE;
```

Toutes les colonnes qui ne sont pas mathématiques doivent apparaître dans le GROUP BY. On voit ainsi que chaque film n'est indiqué qu'une seule fois.

Résultats

TITRE	DATE_SORTIE	NOM	DISTRIBUTEUR	NB_ACTEUR
AVATAR	16/10/09	CAMERON	20th Century Fox	2
BIENVENUE CHEZ LES CH'TIS	27/02/08	BOON	PATHE	2
NIKITA	21/02/90	BESSON	GAUMONT	2
STAR WARS 6 : LE RETOUR DU JEDI	19/10/83	LUCAS	20th Century Fox	2
SUBWAY	10/04/85	BESSON	GAUMONT	5

Quatrième exercice

Sélectionner le titre du film, la date de sortie, le nom et prénom du réalisateur, le nom et prénom de l'acteur, sa date de naissance, le budget du film et le nombre d'entrées en France des films qui ont un acteur algérien.

```
SELECT FILM.TITRE, FILM.DATE_SORTIE,  
  
REAL.NOM || ' ' || REAL.PRENOM REALISATEUR, ACTEUR.NOM NOM,  
  
ACTEUR.PRENOM PRENOM, ACTEUR.DATE_NAISSANCE,  
  
ACTEUR.NB_FILM, STAT.BUDGET, STAT.NB_ENTREE_FRANCE ENTREES  
FROM FILM FILM, REALISATEUR REAL, CASTING CAST, ACTEUR ACTEUR,  
  
STATISTIQUE STAT, PAYS PAYS WHERE  
  
FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND
```

```

FILM.IDENT_FILM      = CAST.IDENT_FILM AND
FILM.IDENT_FILM      = STAT.IDENT_FILM AND
CAST.IDENT_ACTEUR     = ACTEUR.IDENT_ACTEUR AND
PAYS.IDENT_PAYS       = ACTEUR.NATIONALITE AND
PAYS.LIBELLE          = 'ALGERIE'

ORDER BY FILM.TITRE;

```

Résultats

TITRE	DATE – SORTIE	REALISATEUR	NOM	PRENOM	DATE_NAISSANCE	NB_FILM	BUDGET	ENTREES
BIENVENUE CHEZ LES CH'TIS	27/02/08	BOONDARY	MERAD	KAD	27/03/64	55	11	21000000

On peut remarquer dans la requête une autre façon de récupérer le nom et le prénom sur une seule colonne en utilisant les doubles pipes qui servent à la concaténation de deux colonnes. Dans cet exemple on associe le nom et le prénom et leur attribue l'alias 'REALISATEUR'.

On peut également rédiger la requête ainsi :

```

SELECT FILM.TITRE, FILM.DATE_SORTIE,
       REAL.NOM || ' ' || REAL.PRENOM REALISATEUR, ACTEUR.NOM NOM,
       ACTEUR.PRENOM PRENOM, ACTEUR.DATE_NAISSANCE,
       ACTEUR.NB_FILM, STAT.BUDGET, STAT.NB_ENTREE_FRANCE ENTREES
FROM FILM FILM, REALISATEUR REAL, CASTING CAST, ACTEUR ACTEUR,
     STATISTIQUE STAT WHERE
FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND
FILM.IDENT_FILM      = CAST.IDENT_FILM AND
FILM.IDENT_FILM      = STAT.IDENT_FILM AND
CAST.IDENT_ACTEUR     = ACTEUR.IDENT_ACTEUR AND
ACTEUR.NATIONALITE IN (SELECT PAYS.IDENT_PAYS FROM PAYS WHERE
      PAYS.LIBELLE = 'ALGERIE')

ORDER BY FILM.TITRE;

```

ou encore :

```

SELECT FILM.TITRE, FILM.DATE_SORTIE,
       REAL.NOM || ' ' || REAL.PRENOM REALISATEUR, ACTEUR.NOM NOM,
       ACTEUR.PRENOM PRENOM, ACTEUR.DATE_NAISSANCE,

```

```

    ACTEUR.NB_FILM,STAT.BUDGET, STAT.NB_ENTREE_FRANCE ENTREES
FROM FILM FILM, REALISATEUR REAL, CASTING CAST, ACTEUR ACTEUR,
    STATISTIQUE STAT WHERE
    FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND
    FILM.IDENT_FILM      = CAST.IDENT_FILM AND
    FILM.IDENT_FILM      = STAT.IDENT_FILM AND
    CAST.IDENT_ACTEUR    = ACTEUR.IDENT_ACTEUR AND
    EXISTS (SELECT PAYS.IDENT_PAYS FROM PAYS WHERE PAYS.IDENT_PAYS
        = ACTEUR.NATIONALITE AND PAYS.LIBELLE = 'ALGERIE')
ORDER BY FILM.TITRE;

```

Les trois syntaxes renvoient les mêmes résultats. En termes de performance, la première syntaxe sera sans doute la plus performante si la table pays n'a pas trop de lignes. Les deux autres syntaxes sont équivalentes en termes de performance.

Cinquième exercice

Sélectionner le film qui a réalisé la recette la plus élevée dans le monde.

```

SELECT FILM.TITRE, FILM.DATE_SORTIE,
    REAL.NOM || ' ' || REAL.PRENOM REALISATEUR, STAT.BUDGET,
    STAT.NB_ENTREE_FRANCE ENTREES, STAT.RECETTE_USA CA_USA,
    STAT.RECETTE_MONDE CA_MONDE
FROM FILM FILM, REALISATEUR REAL, STATISTIQUE STAT
WHERE
    FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND
    FILM.IDENT_FILM      = STAT.IDENT_FILM AND
    STAT.RECETTE_MONDE = (SELECT MAX(RECETTE_MONDE) FROM
    STATISTIQUE)
ORDER BY FILM.TITRE;

```

Résultats

TITRE	DATE_SORTIE	REALISATEUR	BUDGET	ENTREES	CA_USA	CA_MONDE
AVATAR	16/10/09	CAMERON JAMES	237	12018251	760505847	2946271769

Sixième exercice

Sélectionner l'acteur qui a joué dans deux films différents.

```
SELECT ACTEUR.NOM || ' ' || ACTEUR.PRENOM ACTEUR,  
ACTEUR.DATE_NAISSANCE "NE LE", FILM.TITRE TITRE, FILM.DATE_SORTIE  
"SORTIE LE", REAL.NOM || ' ' || REAL.PRENOM REALISATEUR  
FROM FILM FILM, REALISATEUR REAL, CASTING CAST, ACTEUR ACTEUR  
WHERE  
  
FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND  
FILM.IDENT_FILM = CAST.IDENT_FILM AND  
CAST.IDENT_ACTEUR = ACTEUR.IDENT_ACTEUR AND  
ACTEUR.IDENT_ACTEUR IN  
(SELECT IDENT_ACTEUR FROM CASTING GROUP BY IDENT_ACTEUR  
HAVING COUNT(*) = 2)  
  
ORDER BY FILM.TITRE, ACTEUR.NOM DESC;
```

Notez l'utilisation des doubles quotes dans les alias afin d'utiliser des en-têtes de colonnes significatifs.

Résultats

ACTEUR	NE LE	TITRE	SORTIE LE	REALISATEUR
RENO JEAN	30/06/48	NIKITA	21/02/90	BESSON LUC
RENO JEAN	30/06/48	SUBWAY	10/04/85	BESSON LUC

Dans cet exemple, on récupère en même temps les titres de films et les réalisateurs. On utilise une sous-requête pour calculer le nombre de films par acteur, et on teste sur l'identifiant de l'acteur par une clause 'IN' la présence ou pas de l'acteur.

Autre possibilité, si on veut uniquement récupérer le nom de l'acteur mais pas le nom des films.

```
SELECT ACTEUR.NOM || ' ' || ACTEUR.PRENOM ACTEUR,  
ACTEUR.DATE_NAISSANCE "NE LE", COUNT(*) NB_FILMS  
FROM CASTING CAST, ACTEUR ACTEUR  
WHERE  
  
CAST.IDENT_ACTEUR = ACTEUR.IDENT_ACTEUR  
  
GROUP BY ACTEUR.NOM || ' ' || ACTEUR.PRENOM, ACTEUR.DATE_NAISSANCE  
HAVING COUNT(*) = 2  
  
ORDER BY ACTEUR.NOM || ' ' || ACTEUR.PRENOM;
```

Résultats

ACTEUR	NE LE	NB_FILMS
RENO JEAN	30/06/48	2

Ici, on ne prend pas le film associé sinon le HAVING COUNT ne ramène pas plus de 1. Le titre du film étant inclus dans le SELECT et dans le GROUP BY, il trouve une ligne avec Jean Reno pour Subway et une ligne avec Jean Reno pour Nikita.

Septième exercice

Sélectionner la personne qui est à la fois réalisateur et acteur.

```
SELECT ACTEUR.NOM || ' ' || ACTEUR.PRENOM ACTEUR,  
ACTEUR.DATE_NAISSANCE "NE LE"  
FROM FILM FILM, REALISATEUR REAL, CASTING CAST, ACTEUR ACTEUR  
WHERE  
  
FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND  
FILM.IDENT_FILM = CAST.IDENT_FILM AND  
CAST.IDENT_ACTEUR = ACTEUR.IDENT_ACTEUR AND  
ACTEUR.NOM = REAL.NOM AND  
ACTEUR.PRENOM = REAL.PRENOM AND  
ACTEUR.DATE_NAISSANCE = REAL.DATE_NAISSANCE  
ORDER BY ACTEUR.NOM || ' ' || ACTEUR.PRENOM;
```

Ne sachant pas si pour une personne l'IDENT_ACTEUR est identique à l'IDENT_REALISATEUR, nous devons tester sur les NOM, PRENOM et DATE_NAISSANCE afin de comparer les personnes. Il y a un risque (minime) que des homonymes aient les mêmes noms et prénoms et dates de naissance.

Il faudrait faire évoluer le modèle de donnée afin de donner un identifiant unique aux personnes et ensuite leur affecter un ou plusieurs métiers.

Résultats

ACTEUR	NE LE
BOON DANY	20/06/66

Autre possibilité mais qui ne teste pas d'occurrence dans la table FILM. On se contente de tester les noms, prénoms et dates de naissance identiques sans vérifier leur présence dans la table FILM.

```
SELECT ACTEUR.NOM || ' ' || ACTEUR.PRENOM ACTEUR,  
ACTEUR.DATE_NAISSANCE "NE LE"  
FROM REALISATEUR REAL, ACTEUR ACTEUR  
WHERE  
  
ACTEUR.NOM = REAL.NOM AND  
ACTEUR.PRENOM = REAL.PRENOM AND
```

```

    ACTEUR.DATE_NAISSANCE = REAL.DATE_NAISSANCE

ORDER BY ACTEUR.NOM|'| |ACTEUR.PRENOM;

Huitième exercice

Sélectionner les acteurs qui ont joué dans des films dont le nom du film commence par la lettre 'S'. Indiquer leur rôle et leur nationalité.

SELECT ACTEUR.NOM|'| |ACTEUR.PRENOM ACTEUR,
ACTEUR.DATE_NAISSANCE "NE LE",
    CAST.ROLE, PAYS.LIBELLE, FILM.TITRE, FILM.DATE_SORTIE
FROM FILM FILM, CASTING CAST, ACTEUR ACTEUR, PAYS PAYS
WHERE

    FILM.IDENT_FILM      = CAST.IDENT_FILM AND
    CAST.IDENT_ACTEUR    = ACTEUR.IDENT_ACTEUR AND
    PAYS.IDENT_PAYS      = ACTEUR.NATIONALITE AND
    SUBSTR(FILM.TITRE,1,1) = 'S'
ORDER BY ACTEUR.NOM|'| |ACTEUR.PRENOM, FILM.TITRE DESC;

```

ou :

```

SELECT ACTEUR.NOM|'| |ACTEUR.PRENOM ACTEUR,
ACTEUR.DATE_NAISSANCE "NE LE",
    CAST.ROLE, PAYS.LIBELLE, FILM.TITRE, FILM.DATE_SORTIE
FROM FILM FILM, CASTING CAST, ACTEUR ACTEUR, PAYS PAYS
WHERE

    FILM.IDENT_FILM      = CAST.IDENT_FILM AND
    CAST.IDENT_ACTEUR    = ACTEUR.IDENT_ACTEUR AND
    PAYS.IDENT_PAYS      = ACTEUR.NATIONALITE AND
    FILM.TITRE    LIKE 'S%'
ORDER BY ACTEUR.NOM|'| |ACTEUR.PRENOM, FILM.TITRE DESC;

```

Dans le premier script, on cherche tous les films ayant un 'S' en première position et dans ce deuxième script on cherche tous les films qui commencent par 'S' avec le 'LIKE S%'.

Résultats

ACTEUR	NE LE	ROLE	PAYS	TITRE	SORTIE LE
ADJANI ISABELLE	27/06/55	HELENA	FRANCE	SUBWAY	10/04/85
BOHRINGER RICHARD	16/06/42	INSPECTEUR GESBERG	FRANCE	SUBWAY	10/04/85

ACTEUR	NE LE	ROLE	PAYS	TITRE	SORTIE LE
FISHER CARRIE	21/10/56	PRINCESSE LEIA	USA	STAR WARS 6 : LE RETOUR DU JEDI	19/10/83
FORD HARRISON	13/06/42	HAN SOLO	USA	STAR WARS 6 : LE RETOUR DU JEDI	19/10/83
GALABRU MICHEL	27/10/22	LE FLEURISTE	FRANCE	SUBWAY	10/04/85
LAMBERT CHRISTOPHE	29/03/57	FRED	FRANCE	SUBWAY	10/04/85
RENO JEAN	30/06/48	LE BATTEUR	FRANCE	SUBWAY	10/04/85

Neuvième exercice

Sélectionner les acteurs qui sont nés entre janvier 1948 et mai 1978 ainsi que le nombre de jours de tournage total qu'ils ont réalisés.

```
SELECT ACTEUR.NOM|'|ACTEUR.PRENOM ACTEUR,
ACTEUR.DATE_NAISSANCE "NE LE",
SUM(CAST.NB_JOUR_TOURNAGE) NB_JOURS
FROM CASTING CAST, ACTEUR ACTEUR
WHERE
CAST.IDENT_ACTEUR = ACTEUR.IDENT_ACTEUR AND
ACTEUR.DATE_NAISSANCE >= TO_DATE('01/01/1948','DD/MM/YYYY') AND
ACTEUR.DATE_NAISSANCE <= TO_DATE('31/05/1978','DD/MM/YYYY')
GROUP BY ACTEUR.NOM|'|ACTEUR.PRENOM,ACTEUR.DATE_NAISSANCE
ORDER BY ACTEUR.NOM|'|ACTEUR.PRENOM ASC;
```

ou

```
SELECT ACTEUR.NOM|'|ACTEUR.PRENOM ACTEUR,
ACTEUR.DATE_NAISSANCE "NE LE",
SUM(CAST.NB_JOUR_TOURNAGE) NB_JOURS
FROM CASTING CAST, ACTEUR ACTEUR
WHERE
CAST.IDENT_ACTEUR = ACTEUR.IDENT_ACTEUR AND
ACTEUR.DATE_NAISSANCE BETWEEN
TO_DATE('01/01/1948','DD/MM/YYYY') AND
TO_DATE('31/05/1978','DD/MM/YYYY')
```

GROUP BY ACTEUR.NOM || ' ' || ACTEUR.PRENOM,ACTEUR.DATE_NAISSANCE

ORDER BY ACTEUR.NOM || ' ' || ACTEUR.PRENOM ASC;

Résultats

ACTEUR	NE LE	NB_JOURS
ADJANI ISABELLE	27/06/55	100
BOON DANY	26/06/66	125
FISHER CARRIE	21/10/56	203
LAMBERT CHRISTOPHE	29/03/57	100
MERAD KAD	27/03/64	126
PARILLAUD ANNE	06/05/60	68
RENO JEAN	30/06/48	29
WEAVER SIGOURNEY	08/10/49	45

2. Solutions des exercices sur l'insertion de données

Premier exercice

Ajout d'un nouveau film dans la table FILM :

Numéro 6, Die Hard 4, film d'action sorti le 4 juillet 2007. Le réalisateur est Len Wiseman et distribué par la Twentieth Century Fox Film Corporation.

Résumé : « Pour sa quatrième aventure, l'inspecteur John McClane se trouve confronté à un nouveau genre de terrorisme. Le réseau informatique national qui contrôle absolument toutes les communications, les transports et l'énergie des États-Unis est détruit de façon systématique, plongeant le pays dans le chaos. »

Récupérer le numéro de film le plus élevé.

```
SELECT MAX(IDENT_FILM) FROM FILM;
```

```
MAX(IDENT_FILM)
```

5

Créer la nouvelle ligne :

- En Oracle :

```
INSERT INTO FILM VALUES (6,'DIE HARD 4','ACTION',",
```

```
TO_DATE('04/07/2007','DD/MM/YYYY'),2,5,'Twentieth
```

```
Century Fox Film Corporation','Pour sa quatrième aventure,
```

```
l'inspecteur John McClane se trouve confronté à un nouveau genre
```

```
de terrorisme. Le réseau informatique national qui contrôle
```

absolument toutes les communications, les transports et l'énergie des Etats-Unis, est détruit de façon systématique, plongeant le pays dans le chaos.');

- En MySQL :

```
INSERT INTO FILM VALUES (6,'DIE HARD 4','ACTION',",  
STR_TO_DATE('04/07/2007','%d/%m/%Y'),2,5,'Twentieth  
Century Fox Film Corporation','Pour sa quatrième aventure,  
l'inspecteur John McClane se trouve confronté à un nouveau genre  
de terrorisme. Le réseau informatique national qui contrôle  
absolument toutes les communications, les transports et l'énergie  
des Etats-Unis, est détruit de façon systématique, plongeant le  
pays dans le chaos.');
```

Deuxième exercice

Création d'une nouvelle table ACTEUR_FRANCE à partir de la table ACTEUR en ne sélectionnant que les acteurs français.

```
CREATE TABLE ACTEUR_FRANCE AS SELECT * FROM ACTEUR WHERE  
NATIONALITE = (SELECT IDENT_PAYS FROM PAYS WHERE LIBELLE =  
'FRANCE');
```

Contenu de la table ACTEUR_FRANCE

IDENT_ACTEUR	NOM	PRENOM	DATE_NAISSANCE	NB_FILM	NATIONALITE
1	ADJANI	ISABELLE	27/06/55	42	1
2	LAMBERT	CHRISTOPHE	29/03/57	64	1
3	BOHRINGER	RICHARD	16/06/42	132	1
4	GALABRU	MICHEL	27/10/22	277	1
5	PARILLAUD	ANNE	06/05/60	35	1
10	RENO	JEAN	30/06/48	75	1
11	BOON	DANY	26/06/66	23	1

Troisième exercice

Insertion de « Bruce Willis » né le 19 mars 1955 avec 89 films à son actif et 152 jours de tournage dans la table des acteurs et insertion d'un enregistrement dans la table CASTING le reliant au film numéro 6.

Récupérer le numéro d'acteur le plus élevé :

```
SELECT MAX(IDENT_ACTEUR) FROM ACTEUR;
```

ou utilisation d'une sous-requête pour récupérer le numéro le plus élevé +1 :

```
INSERT INTO ACTEUR VALUES ((SELECT MAX(IDENT_ACTEUR)+1 FROM  
ACTEUR),'WILLIS','BRUCE',TO_DATE('19/03/1955','DD/MM/YYYY'),152,2);
```

Pour insérer dans la table CASTING :

```
INSERT INTO CASTING VALUES (6,12,'John McClane',152);
```

3. Solutions des exercices sur la suppression de données

Premier exercice

Suppression de la table ACTEUR de 'CARRIE FISHER'.

Vérification avant la suppression que la requête ne supprime bien qu'une seule ligne :

```
SELECT COUNT(*) FROM ACTEUR WHERE NOM = 'FISHER' AND PRENOM =  
'CARRIE';
```

```
COUNT(*)
```

1

Puis lancement de la suppression.

```
DELETE FROM ACTEUR WHERE NOM = 'FISHER' AND PRENOM = 'CARRIE';
```

Deuxième exercice

Suppression de la table STATISTIQUE des lignes dont la recette mondiale est égale à 0.

Vérification du nombre de lignes impactées.

```
SELECT COUNT(*) FROM STATISTIQUE WHERE RECETTE_MONDE = 0;
```

```
COUNT(*)
```

1

Puis lancement de la suppression :

```
DELETE FROM STATISTIQUE WHERE RECETTE_MONDE = 0;
```

4. Solutions des exercices sur la modification de données

Premier exercice

Modifier le GENRE2 de la table FILM afin de compléter la zone lorsque celle-ci n'est pas renseignée, puis remplacer 'SF' par 'SCIENCE FICTION'.

Vérification des lignes impactées :

```
SELECT IDENT_FILM,TITRE,GENRE1,GENRE2 FROM FILM WHERE GENRE2 IS  
NULL OR GENRE2 = '';
```

IDENT_FILM	TITRE	GENRE1	GENRE2
5	BIENVENUE CHEZ LES CH'TIS	COMEDIE	
6	DIE HARD 4	ACTION	

```
UPDATE FILM SET GENRE2 = 'AVENTURE' WHERE GENRE2 IS NULL OR GENRE2  
= '';
```

Vérification :

```
SELECT IDENT_FILM,TITRE,GENRE1,GENRE2 FROM FILM WHERE GENRE2 =  
'AVENTURE';
```

IDENT_FILM	TITRE	GENRE1	GENRE2
5	BIENVENUE CHEZ LES CH'TIS	COMEDIE	AVENTURE
6	DIE HARD 4	ACTION	AVENTURE

Deuxième exercice

Division de RECETTE_USA et RECETTE_MONDE par 1 000 000 afin d'exprimer le chiffre en millions dans la table STATISTIQUE.

```
UPDATE STATISTIQUE SET RECETTE_MONDE = RECETTE_MONDE / 1000000;
```

Contenu de la table STATISTIQUE

IDENT_FILM	DUREE	NB_ENTREE_FRANCE	RECETTE_USA	RECETTE_MONDE	BUDGET
1	104	2 917 562	390 659	1,27	2,6
2	118	3 787 845	5 017 971	0,00	7,6
3	133	4 263 000	191 648 000	472,00	32
4	170	12 018 251	760 505 847	2946,27	237
5	100	21 000 000	0	245,00	11

Les fonctions

Introduction

Les fonctions sont multiples et souvent implémentées différemment dans chaque SGBDR. Nous n'allons pas dérouler ici toutes les fonctions possibles. Nous allons aborder les plus couramment utilisées.

Les fonctions numériques

Tous les opérateurs sont utilisables : +, -, *, / mais également des fonctions comme la valeur absolue, le cosinus, les logarithmes, le modulo, l'arrondi, etc.

Consulter la documentation du SGBDR pour connaître les fonctions qui sont implémentées dans la version de base de données utilisée.

Nous ne serons pas exhaustifs sur toutes les fonctions existantes, mais nous allons décrire quelques fonctions généralement implémentées dans les SGBDR.

1. ABS : valeur absolue

Exemple

```
SELECT idTarif, DateDebut, Prix, ABS(Prix) AS ValeurAbsolue FROM  
Tarifs;
```

idTarif	DateDebut	Prix	ValeurAbsolue
1	2021-10-01	49,99	49,99
2	2021-10-01	59,99	59,99
3	2021-10-01	68,99	68,99
4	2021-10-01	59,99	59,99
5	2021-10-01	69,99	69,99
6	2021-10-01	79,99	79,99
7	2021-10-01	89,99	89,99
8	2021-12-15	57,49	57,49

2. Valeur ASCII d'un caractère

Dans cet exemple, on va afficher le code ASCII du premier caractère de la colonne Libelle des hôtels :

SQL Server, MySQL et PostgreSQL

```
SELECT idHotel, Libelle, ASCII(SUBSTRING(Libelle, 1, 1)) AS Code  
FROM Hotels;
```

Oracle, MySQL et PostgreSQL

```
SELECT idHotel, Libelle, ASCII(SUBSTR(Libelle, 1, 1)) AS Code  
FROM Hotels;
```

idHotel	Libelle	Code
1	Ski Hotel	83
2	Art Hotel	65
3	Rose Hotel	82
4	Lions Hotel	76

3. COS : cosinus - SIN : sinus

Dans cet exemple, on va afficher le cosinus du prix de la table Tarifs.

Exemple

```
SELECT idTarif, DateDebut, Prix, COS(Prix) AS Cosinus FROM Tarifs;
```

idTarif	DateDebut	Prix	Cosinus
1	2021-10-01	49,99	0,962294075784641
2	2021-10-01	59,99	-0,955413415572478
3	2021-10-01	68,99	0,992192881610071

Pour obtenir le sinus, c'est la même syntaxe :

```
SELECT idTarif, DateDebut, Prix, SIN(Prix) AS Sinus FROM Tarifs;
```

4. LOG (<numéro base>,<colonne>) : logarithme de la colonne sélectionnée dans la base indiquée

Dans cet exemple, on va afficher le logarithme en base 2 des tarifs.

Exemple

```
SELECT idTarif, DateDebut, Prix, LOG(2, Prix) AS Log FROM Tarifs;
```

idTarif	DateDebut	Prix	Log
1	2021-10-01	49,99	0,177192879928986
2	2021-10-01	59,99	0,169300699821586

5. MOD(<colonne>,<valeur>) : modulo

Le modulo donne le reste de la division d'une colonne par une valeur. Dans cet exemple, on va afficher le modulo du prix divisé par 4.

Oracle, MySQL et PostgreSQL

```
SELECT idTarif, DateDebut, Prix, MOD(Prix, 4) AS Modulo FROM Tarifs;
```

idTarif	DateDebut	Prix	Modulo
1	2021-10-01	49,99	1,99
2	2021-10-01	59,99	3,99

Cette fonction n'existe pas dans SQL Server. L'opérateur % est utilisé (cf. chapitre précédent).

6. ROUND(<colonne>,[<précision>]) : arrondi

L'arrondi retourne l'entier le plus proche d'une valeur décimale. Dans cet exemple, on va afficher l'arrondi de la colonne Prix pour les tarifs.

À noter que l'arrondi de 2,5 sera 2 et l'arrondi de -2,5 sera -3.

```
SELECT idTarif, DateDebut, Prix, ROUND(Prix) AS Arrondi FROM Tarifs;
```

idTarif	DateDebut	Prix	Arrondi
1	2021-10-01	49,99	50
2	2021-10-01	59,99	60

Il est également possible de préciser le nombre de chiffres après la virgule en indiquant une valeur dans le champ <précision>.

```
SELECT idTarif, DateDebut, Prix, ROUND(Prix, 1) AS Arrondi FROM Tarifs;
```

idTarif	DateDebut	Prix	Arrondi
3	01-OCT-21	68.99	69
4	01-OCT-21	67.839	67.8

7. SQRT : racine carrée

Cet exemple indique la racine carrée des tarifs.

```
SELECT idTarif, DateDebut, Prix, SQRT(Prix) AS Racine FROM Tarifs;
```

idTarif	DateDebut	Prix	Racine
1	2021-10-01	49,99	7,07036066972541
2	2021-10-01	59,99	7,74532116829251

Les fonctions de gestion des dates et heures

Il existe une multitude de formats d'affichage, d'utilisation et de fonctions dans chaque base de données, nous ne pouvons pas tous les décrire ici. Nous verrons dans la section Les différents formats d'affichage des dates les formats les plus couramment utilisés et la méthode pour les manipuler.

1. Date du jour : CURRENT_DATE

Pour Oracle, il existe CURRENT_DATE et SYSDATE. Pour MySQL, on peut utiliser CURRENT_DATE, CURDATE, NOW. Pour SQL Server, c'est GETDATE() et SYSDATETIME().

Ces fonctions permettent de retourner la date du jour, avec des précisions différentes (millisecondes, nanosecondes...). En fonction du SGBDR, il existe différentes fonctions qui réalisent la même chose et qui sont synonymes.

Petite particularité pour Oracle et SQL Server, SYSDATE et SYSDATETIME() donnent la date du serveur sur lequel est installée la base de données et CURRENT_DATE ou GETDATE() donnent la date de la session de l'utilisateur.

Par exemple, si la base est installée sur un serveur américain et que l'utilisateur est en France, il y aura une différence de 6 heures entre les deux dates.

Oracle

```
SELECT CURRENT_DATE, SYSDATE FROM DUAL;
```

MySQL

```
SELECT CURRENT_DATE, CURRENT_DATE(), CURDATE(), NOW() FROM  
DUAL;
```

SQL Server

```
SELECT GETDATE() AS dateSession, GETUTCDATE() AS dateUTCSession,  
CURRENT_TIMESTAMP AS dateSession, SYSDATETIME() AS dateSystème,  
SYSUTCDATETIME() AS dateUTCSystème, SYSDATETIMEOFFSET() AS  
dateOffsetSystème;
```

PostgreSQL

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP, LOCALTIMESTAMP, now(),  
transaction_timestamp(), statement_timestamp(), clock_timestamp(),  
timeofday();
```

Dans MySQL, SQL Server et PostgreSQL, le () signifie qu'il s'agit d'une fonction. S'il n'y a pas de parenthèses, c'est une variable.

Il existe quelques différences d'affichage entre ces fonctions et variables. NOW par exemple affiche la date et l'heure courante et les autres uniquement la date pour MySQL. L'affichage écran de ces fonctions de type date est aussi dépendant des options et paramètres utilisés pour la base de données (NLS_DATE_FORMAT pour Oracle et dateformat pour SQL Server).

Nous avons déjà vu qu'il est préférable de fournir dans la requête le format de date que l'on désire obtenir en sortie avec la fonction TO_CHAR pour Oracle ou la fonction DATE_FORMAT pour MySQL.

Exemple pour Oracle

```
SELECT TO_CHAR(CURRENT_DATE,'DD/MM/YYYY') FROM DUAL;
```

Exemple pour MySQL

```
SELECT DATE_FORMAT(NOW(), '%d/%m/%Y')
```

2. Heure actuelle

Sous Oracle et SQL Server, il n'existe pas de fonction qui ramène l'heure du jour. Il faut utiliser SYSDATE pour Oracle et CAST pour SQL Server en appliquant un masque afin de n'afficher que l'heure.

En revanche, MySQL et PostgreSQL utilisent les fonctions CURRENT_TIME ou CURTIME() qui permettent de ramener l'heure au moment de l'interrogation.

Exemple pour Oracle

```
SELECT TO_CHAR(SYSDATE,'HH24:MI:SS') FROM DUAL;
```

Exemple pour SQL Server

```
SELECT CAST(GETDATE() AS time), CAST(GETDATE() AS time(0)),  
CAST(GETDATE() AS time(7));
```

Exemple pour MySQL

```
SELECT CURRENT_TIME;
```

Exemple pour PostgreSQL

```
SELECT CURRENT_TIME, LOCALTIME;
```

Il existe plusieurs formats d'affichage qui sont spécifiques à chaque base de données.

Pour Oracle, les principaux sont :

- HH, HH12, HH24 : format d'affichage de l'heure sur 12 ou 24
- MI : minutes
- SS : secondes

Pour MySQL, les principaux sont :

- %h, %H : format d'affichage de l'heure sur 12 ou 24
- %i : minutes
- %s : secondes

Autre exemple, pour transformer un nombre de secondes en format HH:MM:SS il faut appliquer une fonction de ce type :

```
SELECT HEURE_EN_SECONDE,  
       TRIM(TO_CHAR(TRUNC(TRUNC(HEURE_EN_SECONDE/60)/60), '999900')  
|| ':' || MOD(TRUNC(HEURE_EN_SECONDE/60), 60)  
|| ':' || MOD(HEURE_EN_SECONDE, 60)) AS HMS  
FROM « TABLE CONCERNEE »;
```

3. Date et heure du jour : CURRENT_TIMESTAMP

Le timestamp est en fait une concaténation de la date et de l'heure. On peut appliquer des formats d'affichage différents également. Dans MySQL, c'est équivalent à NOW().

Exemple pour Oracle avec l'affichage standard

```
SELECT CURRENT_TIMESTAMP FROM DUAL;
```

```
CURRENT_TIMESTAMP
```

```
21-SEP-20 10.04.16.777397000 AM AMERICA/NEW_YORK
```

Exemple pour Oracle en appliquant un format

```
SELECT TO_CHAR(CURRENT_TIMESTAMP, 'DD/MM/YYYY HH24:MI:SS') AS Date_et_heure  
FROM DUAL;
```

```
+-----+
| Date_et_heure      |
+-----+
| 21/09/2020 10:42:19 |
+-----+
```

Exemple pour MySQL avec l’affichage standard

```
SELECT CURRENT_TIMESTAMP;
```

```
+-----+
| current_timestamp   |
+-----+
| 2020-09-21 12:24:31 |
+-----+
```

Exemple pour MySQL en appliquant un format

```
SELECT DATE_FORMAT(current_timestamp, '%d/%m/%Y à %H:%i:%S') AS
Date_et_heure;
```

```
+-----+
| Date_et_heure      |
+-----+
| 21/09/2020 à 10:42:19 |
+-----+
```

4. Les différents formats d’affichage des dates

Selon la base de données utilisée, les formats d’affichage et de manipulation des dates divergent fortement. Nous allons décrire ici quelques formats et méthodes simples afin de pouvoir réaliser les opérations les plus courantes sur les dates et heures. Nous ne pouvons pas décrire toutes les possibilités, néanmoins pour aller encore plus loin consultez la documentation associée afin de découvrir toutes les possibilités offertes pour gérer les dates et les heures.

Nous allons décrire dans les exemples ci-dessous comment extraire le jour, le mois ou l’année.

Nous avons déjà vu la syntaxe pour afficher une date au format standard.

Exemple pour Oracle

```
SELECT DATEDEBUT, TO_CHAR(DATEDEBUT,'DD/MM/YY') MaDate FROM Tarifs;
```

DATEDEBUT	MADATE
01-OCT-21	01/10/21
15-DEC-21	15/12/21

Exemple pour MySQL

SELECT DATE_FORMAT(dateDebut,'%d/%c/%Y') maDate FROM tarifs;

Exemple pour SQL Server

SELECT DateDebut, CONVERT(CHAR(8), DateDebut, 3) as MaDate FROM Tarifs;

Si on veut maintenant récupérer, en plus de la date, le mois dans la même requête, il faudra modifier le format attendu ainsi :

Exemple pour Oracle et PostgreSQL

SELECT TO_CHAR(DATEDEBUT,'DD/MM/YY') MaDate, TO_CHAR(DATEDEBUT,'MM') MOIS
FROM Tarifs;

MADATE	MOIS
01/10/21	10
15/12/21	12

Exemple pour MySQL

SELECT DATE_FORMAT(DateDebut,'%d/%c/%Y') DateDebut,
DATE_FORMAT(DateDebut,'%c') MOIS FROM tarifs;

Exemple pour SQL Server

SELECT DateDebut, CONVERT(CHAR(8), DateDebut, 3) as MaDate,
DATEPART(Month, DateDebut) as Mois FROM Tarifs;

ou :

SELECT DateDebut, CONVERT(CHAR(8), DateDebut, 3) as MaDate,
Month(DateDebut) as Mois FROM Tarifs;

Si l'on veut obtenir uniquement l'année sur quatre chiffres, on écrira :

Exemples pour Oracle avec la date système

SELECT TO_CHAR(SYSDATE,'YYYY') ANNEE FROM DUAL ;

ANNEE
2020

Exemples pour Oracle et PostgreSQL avec la date de la table Tarifs

SELECT TO_CHAR(DATEDEBUT,'YYYY') Année FROM Tarifs;

ANNEE
2021
2021
2022
2022
2022

Exemple pour MySQL avec la fonction DATE_FORMAT

```
SELECT DATE_FORMAT(DateDebut,'%Y') ANNEE FROM tarifs;
```

Exemple pour SQL Server avec les fonctions DATEPART et YEAR

```
SELECT Year(DateDebut) as Année FROM Tarifs;
```

ou

```
SELECT DATEPART(YEAR, DateDebut) as Année FROM Tarifs;
```

Si on souhaite afficher la date sous la forme "Numéro_du_jour Nom_du_mois Année_sur_quatre_chiffres" (par exemple, 21 SEPTEMBRE 2020), il faudra saisir :

Exemples pour Oracle

```
SELECT TO_CHAR(SYSDATE,'DD MONTH YYYY') DATE_EN_CLAIR FROM DUAL;
```

DATE_EN_CLAIR
21 SEPTEMBRE 2020

```
SELECT TO_CHAR(DATEDEBUT,'DD MONTH YYYY') DATE_EN_CLAIR FROM Tarifs;
```

DATE_EN_CLAIR
01 OCTOBER 2021
15 DECEMBER 2021

Cela fonctionne aussi avec PostgreSQL.

Exemples pour MySQL

```
SELECT DATE_FORMAT(DateDebut,'%d %b %Y') DATE_EN_CLAIR FROM tarifs;
```

Exemple pour SQL Server

```
SELECT CONVERT(CHAR(2), Day(DateDebut)) + ' ' + FORMAT(DateDebut,
'MMMM') + ' ' + CONVERT(CHAR(4),YEAR(DateDebut)) FROM Tarifs;
```

Voici une liste des formats d'affichage des dates utilisés par Oracle :

Format	Signification	Exemple
D	Numéro du jour dans la semaine (1 à 7)	7
DAY	Nom du jour en lettres	SAMEDI

Format	Signification	Exemple
DD	Numéro du jour	13
DDD	Numéro du jour dans l'année (1 à 366)	13
DY	Abréviation du nom du jour en lettres	SAM.
HH HH12	ou Heures en chiffres (1 à 12)	11
HH24		18
J	Nombre de jours depuis le 1er janvier 4712 avant JC (calendrier julien)	2458132
MI	Minutes en chiffres	12
MM	Numéro du mois	01
MON	Abréviation du nom du mois	JANV
MONTH	Nom du mois en lettres	JANVIER
Q	Numéro du trimestre	1
RM	Numéro du mois en chiffres romains	I
SS	Secondes en chiffres	11
SSSSS	Millisecondes en chiffres	98451
W	Numéro du week-end dans le mois (1 à 5)	2
WW	Numéro du week-end dans l'année (1 à 53)	2
IW	Numéro du week-end dans l'année (1 à 53) norme ISO	2
Y	Dernier chiffre de l'année	8
YEAR	Année en lettres	DEUX MILLE DIX-HUIT
YY	Deux derniers chiffres de l'année	18
YYY	Trois derniers chiffres de l'année	018
YYYY	Quatre chiffres de l'année	2022

Voici une liste des formats d'affichage des dates pour MySQL :

Format	Signification	Exemple
%a	Nom du jour en lettres en anglais et tronqué	Sat
%b	Nom du mois en lettres en anglais et tronqué	Jan
%c	Numéro du mois (1 à 12)	13
%d	Numéro du jour	13
%D	Jour du mois, avec un suffixe anglais (1st, 2nd, 3rd, etc.)	17th

Format	Signification	Exemple
%e	Numéro du jour	11
%f	Millisecondes en chiffres	98451
%H	Heures en chiffres (00 à 23)	23
%h	Heures en chiffres (01 à 12)	12
%I	Heures en chiffres (01 à 12)	01
%i	Minutes en chiffres	38
%j	Numéro du jour dans l'année (1 à 366)	356
%k	Heures en chiffres (1 à 23)	22
%l	Heures en chiffres (1 à 12)	04
%m	Numéro du mois (01 à 12)	11
%M	Nom du mois en lettres en anglais	January
%p	AM ou PM	PM
%r	Heure, au format 12 heures (hh:mm:ss [AP/PM])	10:23:26 AM
%s	Secondes en chiffres	4
%S	Secondes en chiffres	5
%T	Heure, au format 24 heures (hh:mm:ss)	10:23:26
%U	Numéro de la semaine (00..53) (dimanche étant le premier jour)	02
%u	Numéro de la semaine (00..53) (lundi étant le premier jour)	03
%V	Numéro de la semaine (01..53) (dimanche étant le premier jour)	02
%v	Numéro de la semaine (01..53) (lundi étant le premier jour)	03
%W	Nom du jour en lettres (en anglais)	Friday
%w	Numéro du jour de la semaine (0=dimanche)	5
%X	Année sur quatre chiffres (semaine commence le dimanche)	2021
%x	Année sur quatre chiffres (semaine commence le lundi)	2021
%y	Deux derniers chiffres de l'année	14
%Y	Quatre chiffres de l'année	2021

Pour afficher le numéro de semaine, il est préférable en France d'utiliser ISO_WEEK plutôt que WEEK. Le calcul du numéro de la semaine est différent selon les pays. La France applique la norme ISO-8601 avec la semaine 1 qui est la première série de 4 jours dans l'année, par déduction le 1^{er} jeudi de l'année. La semaine commence un lundi. Il peut y avoir 53 semaines dans une année.

Pour les Américains, la semaine commence le dimanche. La semaine 1 est celle du 1^{er} janvier et par déduction le 1^{er} samedi de l'année.

Exemple pour SQL Server

```
SELECT DATEPART(WEEK, '20210101') AS NumSemaine, DATEPART(ISO_WEEK,  
'20210101') AS NumSemaineIso;
```

Exemple pour Oracle

```
SELECT TO_CHAR(TO_DATE('01/01/2021', 'DD/MM/YYYY'), 'WW') AS  
NumSemaine, TO_CHAR(TO_DATE('01/01/2021', 'DD/MM/YYYY'), 'IW') AS  
NumSemaineIso FROM DUAL;
```

NumSemaine	NumSemaineIso
01	53

5. La manipulation des dates et des heures

Il est souvent nécessaire de manipuler les dates et/ou les heures, dans la majorité des développements on est amené à calculer une différence entre deux dates ou à calculer une date de fin par ajout d'un nombre de jours à une date.

La manipulation des dates doit respecter les formats utilisés par les bases de données. Nous avons déjà vu comment afficher une date en indiquant un format spécifique d'affichage.

Si on souhaite par exemple calculer le nombre de jours écoulés entre deux dates, on écrira :

Exemple pour Oracle

```
SELECT '28/03/2022' AS MaDateDebut, '15/08/2023' AS MaDateFin,  
TO_DATE('15/08/2023', 'DD/MM/YYYY') - TO_DATE('28/03/2022',  
'DD/MM/YYYY') AS NB_JOURS FROM DUAL;
```

MADATEDEBUT	MADATEFIN	NB_JOURS
28/03/2022	15/08/2023	505

Pour afficher le nombre de mois entre deux dates, on utilisera la fonction MONTHS_BETWEEN pour Oracle.

Exemple pour Oracle

```
SELECT '28/03/2022' AS MaDateDebut, '15/08/2023' AS MaDateFin,  
TO_CHAR(MONTHS_BETWEEN(TO_DATE('15/08/2023', 'DD/MM/YYYY'),  
TO_DATE('28/03/2022', 'DD/MM/YYYY'))) AS NB_MOIS FROM DUAL;
```

MADATEDEBUT	MADATEFIN	NB_MOIS
28/03/2022	15/08/2023	16.58

Exemple pour PostgreSQL

```
SELECT '28/03/2022' AS MaDateDebut, '15/08/2023' AS MaDateFin, date  
'15/08/2023' - date '28/03/2022' AS NB_JOURS, age('15/08/2023',  
'28/03/2022') AS AGE;
```

MADATEDEBUT	MADATEFIN	NB_JOURS	AGE
28/03/2022	15/08/2023	505	1 year 4 mons 18 days

Pour afficher le nombre d'années, de mois et de jours entre deux dates, on utilisera la fonction `TIMESTAMPDIFF` ou `DATEDIFF` (ne tient pas compte de l'heure) pour MySQL ou SQL Server.

Exemple pour MySQL en utilisant la fonction `TIMESTAMPDIFF`

SELECT

`TIMESTAMPDIFF(YEAR , '20220328', '20230815') NB_ANNEES,`

`TIMESTAMPDIFF(MONTH , '20220328', '20230815') NB_MOIS,`

`TIMESTAMPDIFF(DAY , '20220328', '20230815') NB_JOURS;`

Exemple pour SQL Server en utilisant la fonction `DATEDIFF`

SELECT `DATEDIFF(DAY , '20220328', '20230815')` as nbJours

, `DATEDIFF(MONTH, '20220328', '20230815')` as nbMois

, `DATEDIFF(YEAR, '20220328', '20230815')` as nbAnnée

nbJours	nbMois	nbAnnée
505	17	1

Il existe plusieurs autres fonctions qui peuvent être utilisées pour manipuler des dates. En voici quelques-unes.

Avec Oracle

Récupération du dernier jour du mois : `LAST_DAY`.

SELECT `LAST_DAY(SYSDATE)` FROM DUAL;

LAST_DAY

30/09/20

Récupération de la prochaine date du jour demandé : `NEXT_DAY`.

SELECT `SYSDATE,NEXT_DAY(SYSDATE,'MERCREDI')` FROM DUAL;

SYSDATE NEXT_DAY

21/09/21 23/09/21

Ajouter un nombre de mois à une date : `ADD_MONTHS`.

SELECT `SYSDATE,ADD_MONTHS(SYSDATE,4)` FROM DUAL;

SYSDATE ADD_MONTHS

21/09/21 21/01/22

Arrondir une date au jour suivant si l'on a passé 12H00 : `ROUND`.

```
SELECT TO_CHAR(SYSDATE,'DD/MM/YYYY HH24:MI'),ROUND(SYSDATE) FROM DUAL;
```

```
TO_CHAR(SYSDATE,' ROUND(SY
```

21/09/2021 **15:43** 22/09/21

```
TO_CHAR(SYSDATE,' ROUND(SY
```

21/09/2021 **09:43** 21/09/11

Arrondir une date au matin du jour : TRUNC.

```
SELECT TO_CHAR(SYSDATE,'DD/MM/YYYY HH24:MI'),TRUNC(SYSDATE) FROM DUAL;
```

```
TO_CHAR(SYSDATE,' TRUNC(SY
```

21/09/2021 **15:43** 21/09/21

Avec MySQL

Réaliser des calculs sur une date : ajouter (DATE_ADD), soustraire : (DATE_SUB).

```
SELECT CURRENT_DATE DATEJOUR,DATE_ADD(CURRENT_DATE,INTERVAL 3 DAY)
```

```
AJOUT3JOURS, DATE_SUB(CURRENT_DATE,INTERVAL 3 DAY) SOUSTRAIT3JOURS;
```

```
+-----+-----+-----+
```

```
| DATEJOUR | AJOUT3JOURS | SOUSTRAIT3JOURS |
```

```
+-----+-----+-----+
```

```
| 2021-09-21 | 2021-09-24 | 2021-09-03 |
```

```
+-----+-----+-----+
```

L'intervalle doit être exprimé en chiffres. Ensuite, on trouve le format utilisé. Dans l'exemple, on demande un décalage en jours. Il existe une multitude de possibilités d'exprimer l'intervalle.

Valeurs possibles pour le type d'intervalle
DAY
DAY_HOUR
DAY_MICROSECOND
DAY_MINUTE
DAY_SECOND
HOUR
HOUR_MICROSECOND
HOUR_MINUTE

Valeurs possibles pour le type d'intervalle

HOUR_SECOND

MICROSECOND

MINUTE

MINUTE_MICROSECOND

MINUTE_SECOND

MONTH

QUARTER

SECOND

SECOND_MICROSECOND

WEEK

YEAR

YEAR_MONTH

Obtenir le nom du jour : DAYNAME.

```
SELECT CURRENT_DATE DATEJOUR, DAYNAME(CURRENT_DATE) NOMJOUR;
```

+-----+-----+

| DATEJOUR | NOMJOUR |

+-----+-----+

| 2021-09-21 | Monday |

+-----+-----+

Obtenir le numéro du jour dans la semaine (0=dimanche) : DAYOFWEEK.

```
SELECT CURRENT_DATE DATEJOUR, DAYOFWEEK(CURRENT_DATE) NUMJOUR;
```

+-----+-----+

| DATEJOUR | NUMJOUR |

+-----+-----+

| 2021-09-21 | 1 |

+-----+-----+

Obtenir le numéro du jour dans l'année : DAYOFYEAR.

```
SELECT CURRENT_DATE DATEJOUR, DAYOFYEAR(CURRENT_DATE) NUMJOUR;
```

+-----+-----+

| DATEJOUR | NUMJOUR |

+-----+-----+

| 2021-09-21 | 21 |

+-----+-----+

Récupération du dernier jour du mois : LAST_DAY.

```
SELECT LAST_DAY(CURRENT_DATE) DERNJOUR;

DERNJOUR
```

2021-09-30

Avec SQL Server

Récupération du dernier jour du mois : EOMONTH.

```
SELECT EOMONTH('20210215', -1) as dernierJourDuMoisPrécédent,
EOMONTH('20210215') as dernierJourDuMois;
```

```
DernierJourDuMoisPrécédent    dernierJourDuMois
```

2021-01-31

2021-02-28

Ajouter un nombre d'intervalle à une date : DATEADD.

```
SELECT DATEADD(DAY, -2, '20220328') AS DeuxJoursPrécédent,
DATEADD(MONTH, 3, '20220328') AS ajoutMois, DATEADD(YEAR, 10,
'20220328') AS ajoutAnnées;
```

```
DeuxJoursPrécédent    ajoutMois            ajoutAnnées
```

```
2022-03-26 00:00:00.000  2022-06-28 00:00:00.000  2032-03-28
00:00:00.000
```

Récupérer une partie de la date : DATEPART.

```
SELECT DATEPART(DW, '20220328') AS NumJourSemaine,
DATEPART(DAYOFYEAR, '20220328') AS NumJourAnnée;
```

```
NumJourSemaine    NumJourAnnée
```

3

87

Transformer des chiffres en date : ...FROMPARTS.

```
SELECT DATEFROMPARTS(2022, 8, 15) AS MaDate, DATETIME2FROMPARTS(2022,
8, 15, 8, 30, 0, 0, 0) AS MaDateHeure,
TIMEFROMPARTS(8, 30, 0, 0, 0) AS heure;
```

```
MaDate    MaDateHeure    heure
```

2022-08-15 2022-08-15 08:30:00 08:30:00

Transformer une date en texte au format date : PARSE.

Select PARSE('09 october 2022' AS date USING 'en-US') as US,

PARSE('09 october 2022' AS date USING 'en-GB') as GB,

PARSE('09 octobre 2022' AS date USING 'fr-FR') as FR;

US GB FR

2022-10-09 2022-10-09 2022-10-09

Les fonctions sur les chaînes de caractères

1. Changement de casse LOWER / UPPER / UCASE / LCASE (minuscules et majuscules)

Il existe des fonctions pour mettre les chaînes de caractères en minuscules ou en majuscules. Les noms diffèrent là encore entre chaque SGBD.

En Oracle et SQL Server, on utilisera LOWER et UPPER. En MySQL, LOWER, UPPER, UCASE et LCASE sont autorisés.

LCASE et LOWER sont utilisés pour mettre en minuscules et UCASE et UPPER sont utilisés pour mettre en majuscules.

Exemple pour Oracle

SELECT LOWER('Ceci est un TEST') AS MINUSCULE FROM DUAL;

MINUSCULE

ceci est un test

SELECT UPPER('Ceci est un TEST') AS MAJUSCULE FROM DUAL;

MAJUSCULE

CECI EST UN TEST

Exemple pour MySQL

SELECT LCASE('Ceci est un TEST') AS MINUSCULE

+-----+

| MINUSCULE |


```
+-----+
| ceci est un test |
+-----+
```

```
SELECT UCASE('Ceci est un TEST') AS MAJUSCULE
```

```
+-----+
| MAJUSCULE      |
+-----+
| CECI EST UN TEST |
+-----+
```

Exemple pour SQL Server et PostgreSQL

```
SELECT LOWER('Ceci est un TEST') AS MINUSCULE;
```

```
MINUSCULE
```

```
-----
ceci est un test
```

```
SELECT UPPER('Ceci est un TEST') AS MAJUSCULE;
```

```
MAJUSCULE
```

```
-----
CECI EST UN TEST
```

La syntaxe est donc :

```
SELECT LOWER ou UPPER (<colonne ou variable>) ... FROM <table1>,
<table2> ...
```

avec la possibilité d'utiliser LCASE et UCASE en MySQL.

2. Supprimer les espaces à droite ou à gauche d'une chaîne de caractères : TRIM / LTRIM / RTRIM

Pour supprimer des espaces dans une chaîne de caractères, il faut utiliser soit LTRIM pour supprimer les espaces à gauche (Left) ou RTRIM pour supprimer les espaces à droite (Right). Pour supprimer les espaces à droite et à gauche, il existe TRIM.

Exemple pour Oracle et PostgreSQL

```
SELECT '*' || LTRIM(' SUPPRESSION DES ESPACES A GAUCHE ') || '*' AS
SUPG FROM DUAL;
```

SUPG

*SUPPRESSION DES ESPACES A GAUCHE *

SELECT '*' || **RTRIM**(' SUPPRESSION DES ESPACES A DROITE ') || '*' AS
SUPD FROM DUAL;

SUPD

* SUPPRESSION DES ESPACES A DROITE*

SELECT '*' || **TRIM**(' SUPPRESSION DES ESPACES A DROITE ET A GAUCHE
) || '*' AS SUPDG FROM DUAL;

SUPDG

SUPPRESSION DES ESPACES A DROITE ET A GAUCHE

Exemple pour MySQL

SELECT CONCAT('*,**LTRIM**(' SUPPRESSION DES ESPACES A GAUCHE
,*) AS SUPG FROM DUAL;

+-----+
| SUPG |
+-----+
| *SUPPRESSION DES ESPACES A GAUCHE * |
+-----+

SELECT CONCAT('*,**RTRIM**(' SUPPRESSION DES ESPACES A DROITE
,*) AS SUPD FROM DUAL;

```

+-----+
| SUPD      |
+-----+
| *  SUPPRESSION DES ESPACES A DROITE* |
+-----+

```

```

SELECT CONCAT('*,TRIM('  SUPPRESSION DES ESPACES A DROITE ET A
GAUCHE  '),*') AS SUPDG FROM DUAL;

```

```

+-----+
| SUPDG      |
+-----+
| *SUPPRESSION DES ESPACES A DROITE ET A GAUCHE* |
+-----+

```

Exemple pour SQL Server

```

SELECT '*' + LTRIM('  SUPPRESSION DES ESPACES A GAUCHE  ') + '*' AS SUPG;

```

SUPG

```

-----

```

```

*SUPPRESSION DES ESPACES A GAUCHE  *

```

```

SELECT '*' + RTRIM('  SUPPRESSION DES ESPACES A DROITE  ') + '*' AS SUPD;

```

SUPD

```

-----

```

```

*  SUPPRESSION DES ESPACES A DROITE*

```

La syntaxe est donc :

```

SELECT RTRIM ou LTRIM ou TRIM (<colonne ou variable>) ... FROM

```

```

<table1>, <table2> ...

```

3. Trouver la position d'une chaîne de caractères dans une chaîne : INSTR, CHARINDEX et POSITION

À l'intérieur d'une chaîne, il peut être intéressant de trouver la position d'un caractère particulier ou de plusieurs caractères. Pour ceci, il faut utiliser la fonction INSTR (chaîne, chaîne cherchée) ou CHARINDEX pour SQL Server.

Exemple pour Oracle et MySQL

```
SELECT INSTR('CECI EST UN EXEMPLE POUR TROUVER LA POSITION D'UNE  
CHAINE','EXEMPLE') AS POSITION FROM DUAL;
```

Exemple pour PostgreSQL

```
SELECT POSITION('EXEMPLE' IN 'CECI EST UN EXEMPLE POUR TROUVER LA  
POSITION D'UNE CHAINE') AS POSITION;
```

Exemple pour SQL Server

```
SELECT CHARINDEX('EXEMPLE', 'CECI EST UN EXEMPLE POUR TROUVER LA  
POSITION D'UNE CHAINE') AS POSITION;
```

POSITION

13

La chaîne « EXEMPLE » commence à la position 13 dans la chaîne.

Pour insérer une apostrophe dans un texte, il est nécessaire de la doubler pour ne pas la confondre avec une simple quote qui délimite une chaîne de caractères.

Cette instruction peut être très utile pour extraire une partie de chaîne dans un libellé sans connaître sa position initiale.

Par exemple, si l'on veut ramener les deux mots qui suivent le mot « large » dans la description du type de chambre 7 « 1 lit double large avec bain et WC séparé » :

Exemple pour Oracle et MySQL

Ici on récupère les 15 caractères à partir du mot recherché.

```
SELECT SUBSTR(DESCRIPTION,INSTR(DESCRIPTION,'large'),15) AS
```

```
EXTRAIT FROM
```

```
TypesChambre WHERE idTypeChambre = 7;
```

EXTRAIT

large avec bain

La syntaxe est donc :

SELECT INSTR(<colonne ou variable>,<chaîne recherchée>) ... FROM

<table1>, <table2> ...

4. Ajouter des caractères avant ou après une chaîne : LPAD / RPAD

Si on veut compléter une chaîne par des espaces ou par tout autre caractère, il faut utiliser les fonctions LPAD pour ajouter à gauche et RPAD pour ajouter à droite.

Il faut indiquer la chaîne, le caractère que l'on veut ajouter et le nombre de fois que l'on veut répéter le caractère.

La fonction correspondante dans SQL Server est REPLICATE.

Exemple pour Oracle et MySQL

Ici on complète la chaîne par des points jusqu'à ce que la chaîne complète fasse 50 caractères.

SELECT **LPAD**('JE RAJOUTE DES POINTS A GAUCHE',50, '.') AS AGAUCHE

FROM DUAL;

AGAUCHE

.....JE RAJOUTE DES POINTS A GAUCHE

SELECT **RPAD**('JE RAJOUTE DES POINTS A DROITE',50, '.') AS ADROITE

FROM DUAL;

ADROITE

JE RAJOUTE DES POINTS A DROITE.....

La syntaxe est donc :

SELECT RPAD(<colonne ou variable>,<taille maximum au

final>,<caractère(s) servant au padding>) ... FROM <table1>,

<table2> ...

SELECT LPAD(<colonne ou variable>,<taille maximum au

final>,<caractère(s) servant au padding>) ... FROM <table1>,

<table2> ...

Exemple pour SQL Server

SELECT REPLICATE('.', 50) + 'JE RAJOUTE DES POINTS A GAUCHE' AS AGAUCHE;

AGAUCHE

.....JE RAJOUTE DES POINTS A GAUCHE

SELECT 'JE RAJOUTE DES POINTS A DROITE' + REPLICATE('.', 50) AS ADROITE;

ADROITE

JE RAJOUTE DES POINTS A DROITE.....

5. Extraire une partie d'une chaîne de caractères : SUBSTR

Il est possible d'extraire 1 ou n caractères d'une chaîne en indiquant la position de début et la longueur souhaitée. Dans l'exemple ci-dessous, on récupère les quatre premiers caractères du libellé de la marque. La fonction utilisée dans SQL Server est SUBSTRING.

Exemple pour Oracle, PostgreSQL et MySQL

SELECT SUBSTR(Libelle, 1, 4) AS SUBS FROM Hotels;

La syntaxe est donc :

SELECT SUBSTR(<colonne ou variable>,<position départ>,<Longueur>) ...

FROM <table1>, <table2> ...

Exemple pour SQL Server, PostgreSQL et MySQL

SELECT SUBSTRING(Libelle, 1, 4) AS SUBS FROM Hotels;

SUBS

Ski

Art

Rose

Lions

Les principales fonctions de conversion

1. Transformer un numérique ou une date en texte : TO_CHAR

Ce sont les fonctions TO_CHAR sous Oracle et CAST sous MySQL qui permettent de convertir un numérique ou une date en caractères.

Exemple pour Oracle

```
SELECT TO_CHAR(PRIX) PRIX, TO_CHAR(DATEDEBUT,'DD/MM/YY') DateDebut  
FROM Tarifs;
```

PRIX	DATEDEBUT
69.99	16/04/21
59.99	16/04/21
69.99	16/04/21
79.99	16/04/21
89.99	16/04/21

Exemple pour PostgreSQL, MySQL et SQL Server

```
SELECT CAST(Prix as char(6)) AS Prix, CAST(DateDebut as char(10))  
AS DateDebut FROM Tarifs;
```

Autre exemple pour SQL Server

```
SELECT CONVERT(char(6), Prix) AS Prix, CONVERT(char(10),  
DateDebut) AS DateDebut FROM Tarifs;
```

La syntaxe est donc pour Oracle :

```
SELECT TO_CHAR(<colonne ou variable>,[<FORMAT>],<chaîne  
recherchée> ... FROM <table1>,<table2> ...
```

La syntaxe est donc pour MySQL et SQL Server :

```
SELECT CAST(<colonne ou variable> as <TYPE>) ... FROM <table1>,  
<table2> ...
```

La syntaxe est donc pour SQL Server :

```
SELECT CONVERT(<TYPE>,<colonne ou variable>) ... FROM <table1>,  
<table2> ...
```

2. Changer le type d'une colonne : CAST ou CONVERT

Ces fonctions permettent de changer le type d'une colonne le temps de l'ordre SQL, par exemple changer une colonne qui est initialement en VARCHAR en INTEGER afin de faire un calcul ou tester la valeur numérique. Attention, il faut que le nouveau type soit compatible avec le contenu réel de la colonne.

Syntaxe

```
CAST(<colonne> AS <type>)
```

La colonne NumChambre est en VARCHAR2 ou VARCHAR, on la passe en INTEGER pour la tester :

```
SELECT Hotel, TypeChambre, CAST(NumChambre AS int) AS Entier
```

```
FROM Chambres;
```

ou :

```
SELECT Hotel, TypeChambre, CONVERT(int, NumChambre) AS Entier
```

```
FROM Chambres;
```

Résultat

Hotel	TypeChambre	Entier
1	1	1
1	2	2
1	3	3

3. Changer le classement d'une colonne de type alphanumérique (COLLATE)

Ces fonctions permettent de changer le classement d'une colonne de type varchar ou nvarchar. Dans SQL Server, par défaut le classement (*collation* en américain) est FRENCH_CI_AS. CI signifie *Case Insensitive*, donc insensible à la casse, et AS *Accent Sensitive*, c'est-à-dire sensible aux accents.

Dans certaines circonstances, il est nécessaire de modifier le classement d'une colonne dans la requête pour qu'elle fonctionne dans une application.

Syntaxe

```
COLLATE <classement>
```

Exemple avec une restriction sur une colonne

```
SELECT description FROM typeschambre
```

```
WHERE description like '%wc%';
```

Résultat

Description
3 lits simples avec douche et WC séparés
1 lit double avec douche et WC séparés
1 lit double avec bain et WC séparés
1 lit double large avec bain et WC séparés

```
SELECT description FROM typeschambre
```

```
WHERE description COLLATE Latin1_General_CS_AI like '%wc%'
```

Résultat : aucune ligne car la colonne de la restriction est sensible à la casse.

```
SELECT description FROM typeschambre
```

```
WHERE description COLLATE Latin1_General_CS_AI like '%WC%'
```

Le résultat est identique à la première requête.

Les fonctions de fenêtrage

Les fonctions de fenêtrage permettent de ranger des lignes ou d'effectuer des calculs sur des sous-groupes après avoir partitionné des enregistrements.

1. Numérotation séquentielle et rangement de lignes

Ce sont les fonctions qui permettent de ranger des lignes. RANK permet de numéroter des lignes dans un ordre. DENSE_RANK permet de numéroter les lignes dans un ordre avec des numéros à suivre. ROW_NUMBER permet de numéroter des lignes avec des numéros à suivre, telle une suite. La numérotation peut être effectuée sur des sous-groupes d'enregistrements.

Voici des exemples de numérotation de la table tarifs par prix, sans et avec partitionnement par type de chambre.

Exemples pour SQL Server, Oracle et PostgreSQL

```
SELECT Hotel, typeChambre, Prix, RANK() OVER(ORDER BY Prix DESC)
AS Numero, DENSE_RANK() OVER(ORDER BY Prix DESC) AS NumeroASuivre,
ROW_NUMBER() OVER(ORDER BY Prix DESC) AS Suite FROM Tarifs;
```

Hotel	typeChambre	Prix	Numero	NumeroASuivre	Suite
2	7	103,49	1	1	1
3	7	103,49	1	1	2
1	7	103,49	1	1	3
4	7	103,49	1	1	4
4	6	91,99	5	2	5
1	6	91,99	5	2	6

```
SELECT Hotel, typeChambre, Prix
, RANK() OVER(PARTITION BY TypeChambre ORDER BY Prix DESC) AS Numero
, DENSE_RANK() OVER(PARTITION BY TypeChambre ORDER BY Prix DESC) AS
NumeroASuivre
, ROW_NUMBER() OVER(PARTITION BY TypeChambre ORDER BY Prix DESC) AS Suite
FROM Tarifs;
```

Hotel	typeChambre	Prix	Numero	NumeroASuivre	Suite
1	1	58,49	1	1	1
2	1	57,49	2	2	2
3	1	57,49	2	2	3
1	1	57,49	2	2	4
4	1	57,49	2	2	5
2	1	49,99	6	3	6

Hotel	typeChambre	Prix	Numero	NumeroASuivre	Suite
3	1	49,99	6	3	7
4	1	49,99	6	3	8
1	1	49,99	6	3	9
1	2	69,99	1	1	1
2	2	68,99	2	2	2
3	2	68,99	2	2	3
4	2	68,99	2	2	4

Dans cet exemple, il y a deux types de chambres que l'on classe par prix. Le premier type a trois prix différents. On voit la différence entre le Numero et NumeroASuivre sur le troisième prix où le numéro (fonction RANK) reprend le positionnement du tarif c'est-à-dire 6, alors que NumeroASuivre (fonction DENSE_RANK) continue la numérotation. La suite est basée sur un pas de 1. La numérotation reprend à 1 sur le second type de chambre, car le partitionnement est basé sur le champ TypeChambre

2. Distribution de lignes en groupes numérotés

La fonction NTILE permet de distribuer et numéroter des lignes dans un nombre de groupes définis. Chaque groupe est aussi numéroté. Si le nombre de lignes total n'est pas divisible par le nombre de groupes demandé, certains groupes auront une ligne de moins que les autres.

Voici un exemple de numérotation de la table Tarifs par prix, divisée en 21 groupes.

Exemples pour SQL Server

```
SELECT Hotel, typeChambre, Prix, NTILE(21) OVER(ORDER BY Prix DESC)
```

```
AS Numero FROM Tarifs;
```

Hotel	typeChambre	Prix	Numero
2	7	103,49	1
3	7	103,49	1
1	7	103,49	1
4	7	103,49	2
4	6	91,99	2
1	6	91,99	2
3	6	91,99	3
2	6	91,99	3
1	7	89,99	3
4	7	89,99	4
2	7	89,99	4

3. Décalage de valeurs d'une ligne à une autre

Les fonctions LAG et LEAD renvoient la valeur de la ligne précédente ou suivante, pratique pour effectuer des calculs.

Voici un exemple avec prix, prix précédent et calcul de la différence entre le prix et le prix précédent trié par type de chambre et année de début de saison :

Exemple pour SQL Server

```
SELECT Hotel, typeChambre, Prix, YEAR(dateDebut) AS Année,  
LAG(Prix, 1, 0) OVER(ORDER BY typeChambre, YEAR(dateDebut), Prix  
DESC) AS PrixPrécédent, Prix - LAG(Prix, 1, 0) OVER(ORDER BY  
typeChambre, YEAR(dateDebut), Prix DESC) AS Différence FROM Tarifs;
```

Hotel	typeChambre	Prix	Année	PrixPrécédent	Différence
2	1	57,49	2021	0,00	57,49
3	1	57,49	2021	57,49	0,00
1	1	57,49	2021	57,49	0,00
4	1	57,49	2021	57,49	0,00
2	1	49,99	2021	57,49	-7,50
3	1	49,99	2021	49,99	0,00
4	1	49,99	2021	49,99	0,00
1	1	49,99	2021	49,99	0,00
1	1	58,49	2022	49,99	8,50
2	2	68,99	2021	58,49	10,50
3	2	68,99	2021	68,99	0,00

Voici un exemple avec prix, prix suivant et calcul de la différence entre le prix et le prix suivant trié par type de chambre et année de début de saison :

Exemple pour SQL Server

```
SELECT Hotel, typeChambre, Prix, YEAR(dateDebut) AS Année, LEAD(Prix,  
1, 0) OVER(ORDER BY typeChambre, YEAR(dateDebut), Prix DESC) AS  
PrixSuivant, Prix - LEAD(Prix, 1, 0) OVER(ORDER BY typeChambre,  
YEAR(dateDebut), Prix DESC) AS Différence FROM Tarifs;
```

Hotel	typeChambre	Prix	Année	PrixPrécédent	Différence
2	1	57,49	2021	57,49	0,00
3	1	57,49	2021	57,49	0,00
1	1	57,49	2021	57,49	0,00

Hotel	typeChambre	Prix	Année	PrixPrécédent	Différence
4	1	57,49	2021	49,99	7,50
2	1	49,99	2021	49,99	0,00
3	1	49,99	2021	49,99	0,00
4	1	49,99	2021	49,99	0,00
1	1	49,99	2021	58,49	-8,50
1	1	58,49	2022	68,99	-10,50
2	2	68,99	2021	68,99	0,00
3	2	68,99	2021	68,99	0,00

Les autres fonctions

1. NVL : tester une colonne à null

Le NVL pour « Null Value » permet de savoir si une colonne est renseignée ou pas et de lui attribuer une valeur dans le cas où elle est à null. La fonction correspondante dans SQL Server est ISNULL.

Syntaxe

SELECT NVL(<nom colonne>,<valeur attribuée>), ...

La valeur attribuée doit être de même type que la colonne testée.

Exemple Oracle

```
SELECT NumChambre, Commentaire, NVL(Commentaire, 'Vue sur le jardin') AS
Commentaires FROM Chambres;
```

Exemple SQL Server

```
SELECT NumChambre, Commentaire, ISNULL(Commentaire, 'Vue sur le jardin')
AS Commentaires FROM Chambres;
```

Résultat

NumChambre	Commentaire	Commentaires
1	Belle vue	Belle vue
2		
3	NULL	Vue sur le jardin
4	NULL	Vue sur le jardin

On constate que les chambres qui ont la colonne Commentaire à NULL sont remplacées par la valeur « Vue sur le jardin ». La seconde ligne contient un espace, qui n'est pas considéré comme NULL.

2. Tester plusieurs valeurs : COALESCE

Cette fonction permet de tester plusieurs valeurs NULL de colonnes sur une même fonction évitant ainsi de faire des tests avec des « IF » « THEN », etc.

Elle teste chaque colonne puis attribue le résultat de gauche à droite. La première colonne non null est attribuée. Si toutes les colonnes sont NULL, la fonction prendra la valeur par défaut qui doit être le dernier argument.

Syntaxe

COALESCE(<colonne1>, <colonne 2>,... <valeur par défaut>);

Exemple

Si on a dans la table Tarifs les données suivantes :

idTarif	Hotel	typeChambre	DateDebut	DateFin	Prix	idTarif
60	1	4	2022-04-15	2022-09-30	69,99	60
61	1	5	NULL	2022-09-30	81,49	61
62	1	6	NULL	NULL	NULL	62

On veut typer les tarifs en fonction du contenu de ces deux colonnes. Si DateDebut est NULL, on prend la valeur de la colonne DateFin et si DateDebut et DateFin sont NULL, on affiche '2045-12-31' :

```
SELECT *, COALESCE(DateDebut, Datefin, '2045-12-31') AS Result
```

```
FROM Tarifs where idTarif > 59;
```

Résultat

idTarif	Hotel	typeChambre	DateDebut	DateFin	Prix	Result
60	1	4	2022-04-15	2022-09-30	69,99	2022-04-15
61	1	5	NULL	2022-09-30	81,49	2022-09-30
62	1	6	NULL	NULL	NULL	2045-12-31

3. Comparer deux colonnes : NULLIF

Cette fonction compare deux colonnes. Si les deux sont identiques, elle ramène la valeur NULL. Sinon, elle ramène la première colonne.

Syntaxe

NULLIF(<colonne1>, <colonne 2>)

Exemple

```
SELECT Hotel, TypeChambre, NULLIF(Hotel, TypeChambre) AS Comparaison
```

```
FROM Chambres;
```

Résultats

Hotel	TypeChambre	Comparaison
2	7	2
3	1	3
3	2	3
3	3	NULL
3	4	3

Exercices

Ces exercices sont basés sur les tables précisées dans la section La sélection de données - Exercices sur la sélection de données du chapitre précédent.

Premier exercice

Récupérer la date du jour au format DD-MM-YYYY.

Deuxième exercice

Mettre la chaîne suivante en majuscules puis rechercher la position de la chaîne 'BUT'.

'Lors du match Lille - Brest le premier but a été marqué à la 69ème minute'

Troisième exercice

Enlever les espaces à gauche et ajouter des '-' à la chaîne ci-dessous jusqu'à obtenir une chaîne de 50 caractères.

' Le temps ne permet pas de réaliser les travaux'

Quatrième exercice

Sélectionner les films dont le réalisateur a comme prénom 'LUC' et qui sont sortis entre le '01/01/85' et le '30/05/1995'.

Cinquième exercice

Afficher la date et l'heure du jour sous la forme :

Nous sommes le Vendredi 25 septembre 2021 il est 16 heures et 26 minutes

Sixième exercice

Calculer le nombre de jours passés depuis la sortie de tous les films de la table FILM.

Puis afficher ces nombres exprimés en mois.

Solutions des exercices

Premier exercice

Récupérer la date du jour au format DD-MM-YYYY.

```
SELECT TO_CHAR(CURRENT_DATE,'DD/MM/YYYY') FROM DUAL;
```

Deuxième exercice

Mettre la chaîne suivante en majuscules puis rechercher la position de la chaîne 'BUT'.

'Lors du match Lille - Brest le premier but a été marqué à la 69ème minute'

```
SELECT INSTR(UPPER('Lors du match Lille / Brest le premier but a  
été marqué à la 69ème minute'),'BUT') AS POSITION FROM DUAL;
```

POSITION

40

Troisième exercice

Enlever les espaces à gauche et ajouter des '-' à la chaîne ci-dessous jusqu'à obtenir une chaîne de 50 caractères.

' Le temps ne permet pas de réaliser les travaux'

```
SELECT RPAD(LTRIM(' Le temps ne permet pas de réaliser  
les travaux'),50,'-') AS MEF FROM DUAL;
```

MEF

Le temps ne permet pas de réaliser les travaux---

Quatrième exercice

Sélectionner les films dont le réalisateur a comme prénom 'LUC' et qui sont sortis entre le '01/01/85' et le '30/05/1995'.

```
SELECT * FROM FILM T1 WHERE
```

```
EXISTS (SELECT IDENT_REALISATEUR FROM REALISATEUR WHERE PRENOM  
= 'LUC' AND IDENT_REALISATEUR = T1.IDENT_REALISATEUR )
```

```
AND DATE_SORTIE BETWEEN ('01/01/85') AND ('30/05/1995')
```

```
ORDER BY TTTRE;
```

ou

```
SELECT * FROM FILM T1, REALISATEUR T2 WHERE
```

```

T1.IDENT_REALISATEUR = T2.IDENT_REALISATEUR AND
T2.PRENOM = 'LUC' AND
DATE_SORTIE BETWEEN ('01/01/85') AND ('30/05/1995')
ORDER BY TTRE;

ou

SELECT * FROM FILM T1 WHERE
T1.IDENT_REALISATEUR IN (SELECT IDENT_REALISATEUR FROM
REALISATEUR WHERE PRENOM = 'LUC')
AND DATE_SORTIE BETWEEN ('01/01/85') AND ('30/05/1995')
ORDER BY TTRE;

```

Cet exercice montre bien qu'il existe plusieurs possibilités de coder une requête pour obtenir le même résultat.

S'il y a des index posés sur les colonnes IDENT_REALISATEUR dans les deux tables, la requête numéro 2 sera la plus performante.

Résultat du SELECT

Ident_Film	Titre	Genre1	Genre2	Date_Sortie	Pays	Ident_Realisateur	Distributeur	Resume
2	NIKITA	DRAME	ROMANTIQUE	21/02/90	1	1	GAUMONT	Nikita condamnée à la prison à perpétuité est contrainte à travailler secrètement pour le gouvernement en tant que agent hautement qualifié des services secrets.
1	SUBWAY	POLICIER	DRAME	10/04/85	1	1	GAUMONT	Conte les aventures de la population souterraine dans les couloirs du métro parisien

Cinquième exercice

Afficher la date et l'heure du jour ainsi :

Nous sommes le Vendredi 25 septembre 2021 est 16 heures et 26 minutes

Avec Oracle la syntaxe est la suivante

```
SELECT 'NOUS SOMMES LE ' || TO_CHAR(CURRENT_TIMESTAMP, 'DAY DD MONTH  
YYYY') || ' IL EST ' || TO_CHAR(CURRENT_TIMESTAMP, 'HH24') || ' HEURES ET  
' || TO_CHAR(CURRENT_TIMESTAMP, 'MI') || ' MINUTES' AS EXERCICE_5 FROM DUAL;
```

EXERCICE_5

NOUS SOMMES LE VENDREDI 25 SEPTEMBRE 2021 IL EST 16 HEURES ET 26 MINUTES

Au lieu de CURRENT_TIMESTAMP on peut aussi utiliser CURRENT_DATE, Oracle stocke également l'heure dans la variable CURRENT_DATE.

```
SELECT 'NOUS SOMMES LE ' || TO_CHAR(CURRENT_DATE, 'DAY DD MONTH YYYY') || '  
IL EST ' || TO_CHAR(CURRENT_DATE, 'HH24') || ' HEURES ET  
' || TO_CHAR(CURRENT_DATE, 'MI') || ' MINUTES' AS EXERCICE_5 FROM DUAL;
```

Avec MySQL la syntaxe est la suivante

```
SELECT DATE_FORMAT(CURRENT_TIMESTAMP, 'NOUS SOMMES LE %A %D %M %Y A %H  
HEURES ET %I MINUTES') AS DATE_ET_HEURE;
```

```
+-----+  
| DATE_ET_HEURE |  
+-----+  
| NOUS SOMMES LE FRI 21 SEPTEMBER 2021 A 16 HEURES ET 26 MINUTES |  
+-----+
```

Sixième exercice

Calculer le nombre de jours passés depuis la sortie de tous les films de la table FILM.

Avec Oracle la syntaxe est la suivante

```
SELECT TITRE, DATE_SORTIE, CURRENT_DATE - DATE_SORTIE AS  
NB_JOURS_DEPUIS_SORTIE FROM  
FILM;
```

TITRE	DATE_SOR	NB_JOURS_DEPUIS_SORTIE
-------	----------	------------------------

SUBWAY	10/04/85	10516,7427
--------	----------	------------

NIKITA	21/02/90	8738,74267
STAR WARS 6 : LE RETOUR DU JEDI	19/10/83	11055,7427
AVATAR	16/10/09	1561,74267
BIENVENUE CHEZ LES CH'TIS	27/02/08	2158,74267

Avec MySQL la syntaxe est la suivante

```
SELECT TITRE,DATE_SORTIE,TIMESTAMPDIFF(DAY , DATE_SORTIE, CURRENT_DATE )
NB_JOURS_DEPUIS_SORTIE FROM FILM;
```

+	-----	+	-----	+	-----	+
	TITRE		DATE_SORTIE		NB_JOURS_DEPUIS_SORTIE	
	SUBWAY		1985-04-10		10516	
+	-----	+	-----	+	-----	+
	NIKITA		1990-02-21		8738	
	STAR WARS 6 : LE RETOUR DU JEDI		1983-10-19		11055	
	AVATAR		2009-10-16		1561	
	BIENVENUE CHEZ LES CH'TIS		2008-02-27		2158	
+	-----	+	-----	+	-----	+

Puis afficher ces nombres exprimés en mois.

Avec Oracle la syntaxe est la suivante

```
SELECT
TITRE,DATE_SORTIE,TO_CHAR(MONTHS_BETWEEN(CURRENT_DATE,DATE_SORTIE),'9999'
) AS
NB_MOIS_DEPUIS_SORTIE FROM FILM;
```

TITRE	DATE_SOR	NB_MOI
-----	-----	-----
SUBWAY	10/04/85	345
NIKITA	21/02/90	287
STAR WARS 6 : LE RETOUR DU JEDI	19/10/83	363
AVATAR	16/10/09	51
BIENVENUE CHEZ LES CH'TIS	27/02/08	71

Avec MySQL la syntaxe est la suivante

```
SELECT TITRE,DATE_SORTIE,TIMESTAMPDIFF(MONTH , DATE_SORTIE, CURRENT_DATE )
NB_MOIS_DEPUIS_SORTIE FROM FILM;
```

+-----+-----+-----+		
TITRE	DATE_SORTIE	NB_MOIS_DEPUIS_SORTIE
+-----+-----+-----+		
SUBWAY	1985-04-10	345
NIKITA	1990-02-21	287
STAR WARS 6 : LE RETOUR DU JEDI	1983-10-19	363
AVATAR	2009-10-16	51
BIENVENUE CHEZ LES CH'TIS	2008-02-27	70
+-----+-----+-----+		

La sécurité des données

Introduction

Le langage SQL permet d'attribuer des droits aux utilisateurs à travers les commandes du DCL pour *Data Control Language* ou en français langage de contrôle des données (LCD).

En général, les commandes du DCL sont utilisées principalement par le DBA. La sécurité des données est très importante dans une entreprise, ces fonctions sont donc à manier avec précaution.

Pourquoi définir des droits ?

Au sein d'une entreprise, il existe des typologies d'utilisateurs très différentes. Certains vont simplement consulter quelques tables, d'autres vont être amenés à insérer et à modifier des données, les personnes du service informatique vont avoir des besoins plus étendus leur permettant quelquefois de créer ou de supprimer des tables.

Il faut donc pouvoir contrôler tous les accès à la base de données et permettre à chacun de réaliser ses fonctions sans empiéter sur les fonctions d'une autre personne.

Avant d'attribuer des droits sur telle ou telle table, il faut classer tous les utilisateurs de la base de données par fonction et par métier et déterminer pour chaque fonction quelles sont les tables utilisées et pour chacune des tables indiquer si l'accès doit se faire en mise à jour ou en lecture uniquement. En général, ces informations font partie d'un document d'architecture générale ou technique.

Ce travail doit être fait par les personnes de l'informatique et par les personnes en charge des applications métier (souvent appelés la MOA).

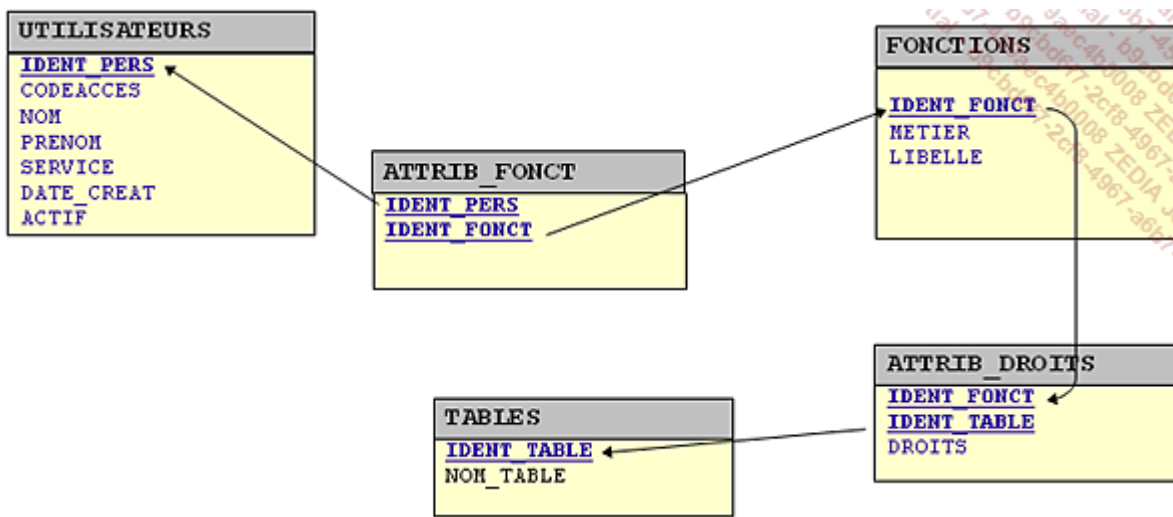
Une fois l'ensemble des informations récoltées, on peut imaginer les classer dans une table afin de rendre la gestion des droits évolutive et facilement maintenable par une personne qui peut être l'administrateur de la base ou une autre personne du service informatique dédiée à cette tâche.

Il est préférable d'utiliser un contrôleur de domaine dans lequel on définit des comptes utilisateurs, membres de groupes. On donne accès aux bases, tables ou enregistrements à ces groupes. Les utilisateurs SQL sont généralement créés pour être dédiés à une application et non à une personne physique.

Le fait d'utiliser des tables dans ce chapitre va permettre de comprendre la mise en place de la sécurité dans une base.

Par exemple

Description d'un modèle de données permettant de gérer les droits des utilisateurs.



Modèle de données utilisé pour illustrer le DCL

TABLE : UTILISATEURS

Requête de création de la table :

```
CREATE TABLE UTILISATEURS (IDENT_PERS INTEGER, CODEACCES CHAR(8),
NOM VARCHAR(50), PRENOM VARCHAR(50), SERVICE VARCHAR(40),
DATE_CREAT DATE, ACTIF CHAR(1));
```

Requêtes d'insertion des données :

```
INSERT INTO UTILISATEURS VALUES
```

```
(1,'ASMITH','SMITH','ALAN','COURRIER',TO_DATE('03/04/2012','DD/MM/
YYYY'),'O');
```

```
INSERT INTO UTILISATEURS VALUES
```

```
(2,'GPEROC','PEROC','GEORGES','COMMERCIAL',TO_DATE('04/05/2016',
'DD/MM/YYYY'),'O');
```

```
INSERT INTO UTILISATEURS VALUES
```

```
(3,'MBALTOP','BALTOP','MICHELE','DIRECTION',TO_DATE('01/02/2008',
'DD/MM/YYYY'),'O');
```

```
INSERT INTO UTILISATEURS VALUES
```

```
(4,'SVERSACE','VERSACE','SANDRINE','INFORMATIQUE
DEV',TO_DATE('05/02/2019','DD/MM/YYYY'),'O');
```

```
INSERT INTO UTILISATEURS VALUES
```

```
(5,'XMARTIN','MARTIN','XAVIER','CONTROLE DE
GESTION',TO_DATE('06/11/1997','DD/MM/YYYY'),'O');
```

```
INSERT INTO UTILISATEURS VALUES
```

```
(6,'RSANCHEZ','SANCHEZ','ROBERTO','FACTURATION',TO_DATE('08/09/2011',
```

'DD/MM/YYYY'),'O');

INSERT INTO UTILISATEURS VALUES

(7,'MSEGAFRE','SEGAFREDI','MANUELLA','ADMINISTRATEUR',TO_DATE('15/05/2015','DD/MM/YYYY'),'O');

Contenu de la table UTILISATEURS :

IDENT_PERS	CODEACCES	NOM	PRENOM	SERVICE	DATE_CREAT	ACTIF
1	ASMITH	SMITH	ALAN	COURRIER	03/04/12	O
2	GPEROC	PEROC	GEORGES	COMMERCIAL	04/05/16	O
3	MBALTOP	BALTOP	MICHELE	DIRECTION	01/02/08	O
4	SVERSACE	VERSACE	SANDRINE	INFORMATIQUE DEV	05/02/19	O
5	XMARTIN	MARTIN	XAVIER	CONTROLE DE GESTION	06/11/97	O
6	RSANCHEZ	SANCHEZ	ROBERTO	FACTURATION	08/09/11	O
7	MSEGAFRE	SEGAFREDI	MANUELLA	ADMINISTRATEUR	15/05/15	O

TABLE : FONCTIONS

Requête de création de la table :

CREATE TABLE FONCTIONS (IDENT_FONCT INTEGER, METIER VARCHAR(50),
LIBELLE VARCHAR(80));

Requêtes d'insertion des données :

INSERT INTO FONCTIONS VALUES (1,'GESTION COMMERCIALE','SAISIE
COMMANDE');

INSERT INTO FONCTIONS VALUES (2,'GESTION COMMERCIALE','SAISIE
CLIENT');

INSERT INTO FONCTIONS VALUES (3,'FACTURATION','GESTION FACTURE');

INSERT INTO FONCTIONS VALUES (4,'FACTURATION','CONTROLE FACTURE');

INSERT INTO FONCTIONS VALUES (5,'STOCK','CONSULTATION');

INSERT INTO FONCTIONS VALUES (6,'STOCK','AFFECTATION
EMPLACEMENT');

INSERT INTO FONCTIONS VALUES (7,'GESTION
COMMERCIALE','CONSULTATION CLIENT');

Contenu de la table FONCTIONS :

IDENT_FONCT	METIER	LIBELLE
1	GESTION COMMERCIALE	SAISIE COMMANDE
2	GESTION COMMERCIALE	SAISIE CLIENT
3	FACTURATION	GESTION FACTURE
4	FACTURATION	CONTROLE FACTURE
5	STOCK	CONSULTATION
6	STOCK	AFFECTATION EMPLACEMENT
7	GESTION COMMERCIALE	CONSULTATION CLIENT

TABLE : ATTRIB_FONC

Requête de création de la table :

CREATE TABLE ATTRIB_FONC (IDENT_PERS INTEGER, IDENT_FONCT
INTEGER);

Requêtes d'insertion des données :

INSERT INTO ATTRIB_FONC VALUES (1,1);
INSERT INTO ATTRIB_FONC VALUES (1,4);
INSERT INTO ATTRIB_FONC VALUES (1,7);
INSERT INTO ATTRIB_FONC VALUES (2,2);
INSERT INTO ATTRIB_FONC VALUES (2,3);
INSERT INTO ATTRIB_FONC VALUES (3,4);
INSERT INTO ATTRIB_FONC VALUES (4,1);
INSERT INTO ATTRIB_FONC VALUES (4,7);
INSERT INTO ATTRIB_FONC VALUES (5,2);
INSERT INTO ATTRIB_FONC VALUES (6,3);
INSERT INTO ATTRIB_FONC VALUES (7,1);
INSERT INTO ATTRIB_FONC VALUES (7,2);
INSERT INTO ATTRIB_FONC VALUES (7,3);
INSERT INTO ATTRIB_FONC VALUES (7,4);
INSERT INTO ATTRIB_FONC VALUES (7,5);
INSERT INTO ATTRIB_FONC VALUES (7,6);

Contenu de la table ATTRIB_FONC :

IDENT_PERS	IDENT_FONCT
1	1
1	4
1	7
2	2
2	3
3	4
4	1
4	7
5	2
6	3
7	1
7	2
7	3
7	4
7	5
7	6

TABLE : TABLES

Requête de création de la table :

```
CREATE TABLE TABLES (IDENT_TABLE INTEGER, NOM_TABLE VARCHAR(40));
```

Requêtes d'insertion des données :

```
INSERT INTO TABLES VALUES (1,'CLIENT');
```

```
INSERT INTO TABLES VALUES (2,'FOURNISSEUR');
```

```
INSERT INTO TABLES VALUES (3,'FACTURE');
```

```
INSERT INTO TABLES VALUES (4,'COMMANDE');
```

```
INSERT INTO TABLES VALUES (5,'STOCK');
```

```
INSERT INTO TABLES VALUES (6,'COMM_FACT');
```

```
INSERT INTO TABLES VALUES (7,'ACHETEUR');
```

```
INSERT INTO TABLES VALUES (8,'ARTICLE');
```

```
INSERT INTO TABLES VALUES (999,'ALL');
```

Contenu de la table TABLES :

IDENT_TABLE	NOM_TABLE
1	CLIENT
2	FOURNISSEUR
3	FACTURE
4	COMMANDE
5	STOCK
6	COMM_FACT
7	ACHETEUR
8	ARTICLE
999	ALL

TABLE : ATTRIB_DROITS

Requête de création de la table :

```
CREATE TABLE ATTRIB_DROITS (IDENT_FONCT INTEGER, IDENT_TABLE
INTEGER, DROITS VARCHAR(30));
```

Requêtes d'insertion des données :

```
INSERT INTO ATTRIB_DROITS VALUES (1,1,'SELECTION');
INSERT INTO ATTRIB_DROITS VALUES (1,1,'SUPPRESSION');
INSERT INTO ATTRIB_DROITS VALUES (1,1,'MODIFICATION');
INSERT INTO ATTRIB_DROITS VALUES (1,2,'ALL');
INSERT INTO ATTRIB_DROITS VALUES (1,3,'ALL');
INSERT INTO ATTRIB_DROITS VALUES (2,4,'SELECTION');
INSERT INTO ATTRIB_DROITS VALUES (2,5,'SELECTION');
INSERT INTO ATTRIB_DROITS VALUES (2,4,'INSERTION');
INSERT INTO ATTRIB_DROITS VALUES (3,1,'ALL');
INSERT INTO ATTRIB_DROITS VALUES (3,2,'ALL');
INSERT INTO ATTRIB_DROITS VALUES (3,3,'ALL');
INSERT INTO ATTRIB_DROITS VALUES (4,999,'ALL');
INSERT INTO ATTRIB_DROITS VALUES (5,999,'SELECTION');
INSERT INTO ATTRIB_DROITS VALUES (6,999,'ALL');
INSERT INTO ATTRIB_DROITS VALUES (7,1,'SELECTION');
```

Contenu de la table ATTRIB_DROITS :

IDENT_FONCT	IDENT_TABLE	DROITS
1	1	SELECTION
1	1	SUPPRESSION
1	1	MODIFICATION
1	2	ALL
1	3	ALL
2	4	SELECTION
2	5	SELECTION
2	4	INSERTION
3	1	ALL
3	2	ALL
3	3	ALL
4	999	ALL
5	999	SELECTION
6	999	ALL
7	1	SELECTION

La valeur 999 en colonne 2 signifie que l'on n'indique pas de numéro de table mais que l'on prend toutes les tables par défaut. On attribue le droit indiqué dans la colonne 3 à toutes les tables.

Le « ALL » en colonne 3 signifie que l'on accorde tous les droits sur la table indiquée en colonne 2.

Créer une connexion

Les SGDBR gèrent différemment leurs droits d'accès même si les syntaxes sont similaires. Avec SQL Server, il est nécessaire de créer une connexion au serveur SQL (qui peut être créée à partir de l'annuaire du contrôleur de domaine). SQL Server est un moteur de bases de données, indépendant du système d'exploitation.

Syntaxe SQL Server

```
CREATE LOGIN <login> WITH PASSWORD = '<Mot de passe>';
```

On création de la connexion à partir d'un compte Windows :

```
CREATE LOGIN <domainName>\<login> WITH PASSWORD = '<Mot de passe>';
```

Exemple avec SQL Server

Création de la connexion SQL au serveur à partir de la base 'master' qui est une base système :

```
USE master;
```

```
GO
```

```
CREATE LOGIN ASMITH
```

```
WITH PASSWORD = '@Smith.72';
```

Créer un utilisateur

Les utilisateurs sont souvent créés par l'administrateur de la base selon des règles de sécurité propres à chaque entreprise. La commande de création d'un user est assez simple. Avec SQL Server, l'utilisateur est créé à partir de la connexion.

Syntaxe Oracle

```
CREATE <user> IDENTIFIED BY <Mot de passe>;
```

Syntaxe MySQL

```
CREATE '<user>' IDENTIFIED BY '<Mot de passe>';
```

Syntaxe SQL Server

```
CREATE '<user>' FROM LOGIN <login>
```

Syntaxe PostgreSQL

```
CREATE USER '<user>'
```

Exemple avec Oracle

```
CREATE USER ASMITH IDENTIFIED BY ASMITH;
```

Exemple avec MySQL

```
CREATE USER 'ASMITH' IDENTIFIED BY 'ASMITH';
```

Autre méthode pour déclarer un utilisateur avec MySQL

```
GRANT ALL PRIVILEGES ON *.* TO ASMITH@localhost  
IDENTIFIED BY 'ASMITH' WITH GRANT OPTION;
```

Exemple avec SQL Server

```
USE RESAHOTEL;  
  
GO  
  
CREATE USER ASMITH FROM LOGIN ASMITH;
```

Exemple avec PostgreSQL

```
CREATE USER ASMITH;
```

M. Smith pourra donc se connecter à la base de données en utilisant ASMITH/ASMITH@<Nom de la base>.

La création d'un utilisateur ne lui permet pas d'accéder aux tables et de faire des sélections. Il faut ensuite lui attribuer des droits spécifiques en fonction de son profil.

Changer le mot de passe d'un utilisateur

Pour des raisons de sécurité, il peut être nécessaire de modifier le mot de passe d'un utilisateur. Dans la plupart des systèmes, c'est le DBA qui peut utiliser cette commande.

Syntaxe Oracle

```
ALTER USER <user> IDENTIFIED BY <Nouveau Mot de passe>;
```

Syntaxe MySQL

```
SET PASSWORD FOR '<user>'@<host>' = PASSWORD('mypass');
```

Syntaxe SQL Server

```
ALTER LOGIN <user> WITH PASSWORD='mypass';
```

Syntaxe PostgreSQL

```
ALTER USER <user> PASSWORD 'mypass';
```

Exemple Oracle

```
ALTER USER ASMITH IDENTIFIED BY ABCD12E;
```

Exemple MySQL

```
SET PASSWORD FOR ASMITH@localhost=PASSWORD('ABCD12E');
```

Exemple SQL Server

```
USE [master];
```

```
GO
```

```
ALTER LOGIN ASMITH WITH PASSWORD='@Smith.50';
```

Exemple PostgreSQL

```
USE [master];
```

```
GO
```

```
ALTER USER ASMITH PASSWORD '@Smith.50';
```

Attribuer des droits (GRANT)

1. Attribuer des droits sur la manipulation d'une table

À partir de ces quelques tables d'exemples, nous allons pouvoir attribuer les droits aux utilisateurs avec les ordres GRANT et REVOKE.

L'attribution des droits est réservée aux créateurs de la table, néanmoins l'administrateur de la base peut donner l'autorisation à un autre user la possibilité de gérer les droits sur les tables.

La commande GRANT permet d'attribuer par utilisateur de la base de données des accès sur une ou plusieurs tables.

Les droits les plus couramment utilisés sont :

- SELECT : autorise la sélection de données.
- UPDATE : autorise la modification de données.
- DELETE : autorise la suppression de données.
- INSERT : autorise l'insertion de données.

La syntaxe est la suivante

GRANT <droit1>, <droit2>, ...

ON TABLE <nom table>

TO <user1>, <user2> ...

[WITH GRANT OPTION]

Pour attribuer tous les droits, il faut utiliser la syntaxe suivante :

Syntaxe Oracle et PostgreSQL

GRANT ALL PRIVILEGES

ON <table1>, <table2>, ...

[WITH GRANT OPTION] TO <user1>, <user2> ...

Syntaxe MySQL

GRANT ALL PRIVILEGES

ON TABLE <table1>, <table2>, ...

[WITH GRANT OPTION] TO <user1>, <user2> ...

Exemple SQL Server

ALTER AUTHORIZATION ON SCHEMA::[db_owner] TO [ASMITH]

Le ALL PRIVILEGES ou db_owner donne vraiment tous les droits, donc il doit être utilisé avec précaution et de préférence attribué à des personnes de profil administrateur.

La clause WITH GRANT OPTION donne l'autorisation aux users désignés d'attribuer des droits sur ces tables à d'autres utilisateurs.

Avec SQL Server, on affecte le rôle db_owner pour attribuer tous les droits à une base de données.

L'analyse des tables d'exemples permet de sortir les éléments qui vont nous indiquer comment attribuer ou pas des droits aux utilisateurs avec l'ordre GRANT.

Sélection des droits par utilisateur :

SELECT CODEACCES, ATFONC.IDENT_FONCT, NOM_TABLE, DROITS

FROM UTILISATEURS UTIL

INNER JOIN ATTRIB_FONC ATFONC ON UTIL.IDENT_PERS = ATFONC.IDENT_PERS

INNER JOIN FONCTIONS FONC ON ATFONC.IDENT_FONCT = FONC.IDENT_FONCT

INNER JOIN ATTRIB_DROITS ATDROIT ON FONC.IDENT_FONCT =

ATDROIT.IDENT_FONCT

INNER JOIN TABLES TAB ON ATDROIT.IDENT_TABLE = TAB.IDENT_TABLE

ORDER BY CODEACCES,ATFONC.IDENT_FONCT, NOM_TABLE, DROITS;

CODEACCES	IDENT_FONCT	NOM_TABLE	DROITS
ASMITH	1	CLIENT	MODIFICATION
ASMITH	1	CLIENT	SELECTION
ASMITH	1	CLIENT	SUPPRESSION
ASMITH	1	FACTURE	ALL
ASMITH	1	FOURNISSEUR	ALL
ASMITH	4	ALL	ALL
ASMITH	7	CLIENT	SELECTION
GPEROC	2	COMMANDE	INSERTION
GPEROC	2	COMMANDE	SELECTION
GPEROC	2	STOCK	SELECTION
GPEROC	3	CLIENT	ALL
GPEROC	3	FACTURE	ALL
GPEROC	3	FOURNISSEUR	ALL
MBALTOP	4	ALL	ALL
MSEGAFRE	1	CLIENT	MODIFICATION
MSEGAFRE	1	CLIENT	SELECTION
MSEGAFRE	1	CLIENT	SUPPRESSION
MSEGAFRE	1	FACTURE	ALL
MSEGAFRE	1	FOURNISSEUR	ALL
MSEGAFRE	2	COMMANDE	INSERTION
MSEGAFRE	2	COMMANDE	SELECTION
MSEGAFRE	2	STOCK	SELECTION
MSEGAFRE	3	CLIENT	ALL
MSEGAFRE	3	FACTURE	ALL
MSEGAFRE	3	FOURNISSEUR	ALL
MSEGAFRE	4	ALL	ALL
MSEGAFRE	5	ALL	SELECTION
MSEGAFRE	6	ALL	ALL

CODEACCES	IDENT_FONCT	NOM_TABLE	DROITS
RSANCHEZ	3	CLIENT	ALL
RSANCHEZ	3	FACTURE	ALL
RSANCHEZ	3	FOURNISSEUR	ALL
SVERSACE	1	CLIENT	MODIFICATION
SVERSACE	1	CLIENT	SELECTION
SVERSACE	1	CLIENT	SUPPRESSION
SVERSACE	1	FACTURE	ALL
SVERSACE	1	FOURNISSEUR	ALL
SVERSACE	7	CLIENT	SELECTION
XMARTIN	2	COMMANDE	INSERTION
XMARTIN	2	COMMANDE	SELECTION
XMARTIN	2	STOCK	SELECTION

Si nous prenons la deuxième ligne :

ASMITH	CLIENT	SELECTION
--------	--------	-----------

Elle peut se traduire ainsi :

GRANT SELECT ON TABLE CLIENT TO ASMITH;

GRANT SELECT ON CLIENT TO ASMITH;

Maintenant, pour gérer toutes les autorisations de l'utilisateur ASMITH, il faut écrire plusieurs ordres. On peut remarquer également que M. Smith ayant plusieurs fonctions (1, 4 et 7), les accès aux tables se cumulent.

CODEACCES	IDENT_FONCT	NOM_TABLE	DROITS
ASMITH	1	FACTURE	ALL
ASMITH	1	FOURNISSEUR	ALL
ASMITH	1	CLIENT	SELECTION
ASMITH	1	CLIENT	SUPPRESSION
ASMITH	1	CLIENT	MODIFICATION
ASMITH	4	ALL	ALL
ASMITH	7	CLIENT	SELECTION

M. Smith a sur une ligne NOM_TABLE à « ALL » et DROITS à « ALL », ce qui signifie qu'il a droit à tous les accès sur toutes les tables.

Ce qui se traduit par :

GRANT ALL PRIVILEGES ON CLIENT TO ASMITH;

GRANT ALL PRIVILEGES ON FOURNISSEUR TO ASMITH;

GRANT ALL PRIVILEGES ON FACTURE TO ASMITH;
 GRANT ALL PRIVILEGES ON COMMANDE TO ASMITH;
 GRANT ALL PRIVILEGES ON STOCK TO ASMITH;
 GRANT ALL PRIVILEGES ON COMM_FACT TO ASMITH;
 GRANT ALL PRIVILEGES ON ACHETEUR TO ASMITH;
 GRANT ALL PRIVILEGES ON ARTICLE TO ASMITH;

Attribution de tous les droits sur la table CLIENT à M. Peroc avec SQL Server :

GRANT VIEW DEFINITION ON CLIENT TO GPEROC
 GRANT VIEW CHANGE TRACKING ON CLIENT TO GPEROC
 GRANT CONTROL ON CLIENT TO GPEROC
 GRANT INSERT ON CLIENT TO GPEROC
 GRANT UPDATE ON CLIENT TO GPEROC
 GRANT ALTER ON CLIENT TO GPEROC
 GRANT TAKE OWNERSHIP ON CLIENT TO GPEROC
 GRANT REFERENCES ON CLIENT TO GPEROC
 GRANT SELECT ON CLIENT TO GPEROC
 GRANT DELETE ON CLIENT TO GPEROC

Pour attribuer des droits à M. VERSACE sur la table CLIENT :

SVERSACE	1	CLIENT	MODIFICATION
SVERSACE	1	CLIENT	SUPPRESSION
SVERSACE	1	CLIENT	SELECTION

il faut écrire :

GRANT SELECT, DELETE, UPDATE ON CLIENT TO AVERSACE;

Il est également possible d'aller encore plus loin dans le contrôle des accès en spécifiant les colonnes qui sont autorisées ou non pour chaque utilisateur.

Par exemple pour autoriser M. Smith à modifier uniquement les colonnes IDENT_PERS, NOM et PRENOM de la table UTILISATEURS, il faudra écrire :

GRANT UPDATE (IDENT_PERS,NOM,PRENOM) ON UTILISATEURS TO ASMITH;

Pour attribuer un privilège à tous les utilisateurs de la base de données, il est possible d'utiliser le mot PUBLIC ainsi :

GRANT UPDATE (IDENT_PERS,NOM,PRENOM) ON UTILISATEURS TO **PUBLIC**;

2. Attribuer des droits sur les objets de la base

Nous avons vu dans la section précédente comment attribuer des droits sur les manipulations de tables. L'ordre GRANT permet également de donner l'autorisation de créer des tables, des vues ou encore des sessions.

Communément on appelle ceci des privilèges système. Ils sont donc à manipuler avec précaution car il s'agit ici de sécurité de la base. La plupart du temps, ce sont les DBA qui accordent ces droits.

Les ordres GRANT s'utilisent pour donner ou pas les droits sur des commandes du LDD comme CREATE, ALTER et DROP.

Ils s'appliquent sur tous les objets de la base comme :

- TABLES
- INDEX
- SEQUENCES
- SESSIONS
- VUES
- PROCEDURES
- SYNONYMES
- TRIGGERS
- Etc.

Par exemple, si l'on souhaite autoriser la création de table pour l'utilisateur ASMITH, on écrira :

Syntaxe Oracle et SQL Server

```
GRANT CREATE TABLE TO ASMITH;
```

Syntaxe MySQL

```
GRANT CREATE ON * TO 'ASMITH';
```

De la même façon, pour donner les droits de modifier une table :

```
GRANT ALTER ANY TABLE TO ASMITH;
```

Les ordres DROP et ALTER sont suivis du mot ANY, contrairement au CREATE.

Le mot ANY signifie que l'autorisation est donnée uniquement sur le schéma de l'utilisateur. Si l'on ne met pas le mot ANY, les droits s'appliquent pour tous les schémas.

Pour donner plusieurs privilèges avec une seule commande, la syntaxe est la suivante :

```
GRANT CREATE TABLE,  
DROP ANY TABLE,  
ALTER ANY TABLE  
TO ASMITH;
```

Avec cet ordre, l'utilisateur ASMITH aura les droits de créer, modifier et supprimer une table.

La syntaxe générale Oracle est

```
GRANT <privilège> [, <privilège 2>, <privilège 3>] <objet> TO  
<nom utilisateur>;
```

La syntaxe générale MySQL est

GRANT <privilège> [,<privilège 2>,<privilège 3>] ON <objet>,[*] TO

'<nom utilisateur>';

En fonction de chaque objet, les droits peuvent être différents. Voici une liste des droits par objet les plus fréquents.

Objet	Actions possibles
TABLE	CREATE TABLE CREATE ANY TABLE ALTER ANY TABLE DROP ANY TABLE COMMENT ANY TABLE SELECT ANY TABLE INSERT ANY TABLE UPDATE ANY TABLE DELETE ANY TABLE LOCK ANY TABLE FLASHBACK ANY TABLE
INDEX	CREATE ANY INDEX ALTER ANY INDEX DROP ANY INDEX
VIEW (VUE)	CREATE VIEW CREATE ANY VIEW DROP ANY VIEW COMMENT ANY TABLE FLASHBACK ANY TABLE
PROCEDURE	CREATE PROCEDURE CREATE ANY PROCEDURE ALTER ANY PROCEDURE DROP ANY PROCEDURE EXECUTE ANY PROCEDURE
SEQUENCE	CREATE SEQUENCE CREATE ANY SEQUENCE DROP ANY SEQUENCE SELECT ANY SEQUENCE

Objet	Actions possibles
SESSION	CREATE SESSION ALTER SESSION ALTER RESOURCE COST RESTRICT SESSION
SYNONYM	CREATE SYNONYM CREATE ANY SYNONYM CREATE PUBLIC SYNONYM DROP ANY SYNONYM DROP PUBLIC SYNONYM
TRIGGER	CREATE TRIGGER CREATE ANY TRIGGER ALTER ANY TRIGGER DROP ANY TRIGGER ADMINISTER DATABASE TRIGGER

3. Les autres droits possibles

Dans cette section, nous allons voir quelques ordres GRANT qui s'appliquent sur des objets moins utilisés ou moins connus.

Il est possible d'attribuer des droits sur les objets physiques de la base comme les tablespaces, mais également sur des ordres de type arrêt de la base. Les tablespaces n'existent pas dans SQL Server. Il est possible d'attribuer des droits à un schéma qui regroupe un ensemble d'objets SQL comme des tables, vues, fonctions ou procédures stockées. Généralement, on crée un schéma par métier et on donne les droits à ce schéma, via un rôle, de préférence.

Par exemple, si on souhaite autoriser l'utilisateur ASMITH à créer des tablespaces, on écrira :

Exemple Oracle

```
GRANT CREATE TABLESPACE TO ASMITH;
```

De la même façon, pour que ASMITH puisse créer d'autres utilisateurs, on lui attribuera les droits CREATE USER.

Exemple Oracle

```
GRANT CREATE USER TO ASMITH;
```

Autre exemple, pour autoriser l'utilisateur ASMITH à créer et à supprimer des rôles, on écrira :

Exemple Oracle

```
GRANT CREATE ROLE, DROP ANY ROLE TO ASMITH;
```

En fonction de chaque objet, les droits peuvent être différents. Voici une liste des droits par objet avec Oracle.

Objet	Actions possibles
DATABASE	ALTER SYSTEM AUDIT SYSTEM AUDIT ANY
PROFILE	CREATE PROFILE ALTER PROFILE DROP PROFILE
ROLE	CREATE ROLE DROP ANY ROLE ALTER ANY ROLE
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE MANAGE TABLESPACE UNLIMITED TABLESPACE
USER	CREATE USER ALTER USER DROP USER

On peut aussi attribuer des droits sur des actions particulières comme celles indiquées ci-dessous pour Oracle :

Action particulière	Syntaxe	Explications
ANALYZE ANY	GRANT ANALYZE ANY TO ASMITH;	Autorise ASMITH à lancer des analyses des objets de la base.
GRANT ANY	GRANT GRANT ANY OBJECT PRIVILEGE TO ASMITH;	Autorise ASMITH à donner des autorisations aux autres utilisateurs sur les objets non système.
	GRANT GRANT ANY PRIVILEGE TO ASMITH;	Autorise ASMITH à donner des autorisations aux autres utilisateurs sur tous les objets système.
SYSDBA SYSOPER	GRANT SYSDBA TO ASMITH;	Autorise ASMITH à lancer des opérations système sur la base comme STARTUP, SHUTDOWN ou CREATE DATABASE par exemple.
	ou GRANT SYSOPER TO ASMITH;	À noter que ce GRANT doit être réalisé par un utilisateur qui soit lui-même SYSDBA ou SYSOPER.

Action particulière	Syntaxe	Explications
SELECT ANY DICTIONARY	GRANT SELECT ANY DICTIONARY TO ASMITH;	Autorise ASMITH à interroger toutes les tables appartenant au compte SYS.

Vous avez la possibilité de visualiser les droits attribués en interrogeant la table :

USER_TAB_PRIVS (SELECT * FROM USER_TAB_PRIVS;)

Interdire l'accès (DENY)

1. Interdire l'accès à certains objets de la base

Avec SQL Server, il est possible d'interdire l'accès à certains objets de la base avec l'ordre DENY.

Exemple SQL Server

Interdit à M. Peroc de modifier la table ATTRIB_DROITS :

DENY ALTER ON ATTRIB_DROITS TO GPEROC

Interdit tout à M. Peroc sur la table ATTRIB_DROITS :

DENY VIEW DEFINITION ON ATTRIB_DROITS TO GPEROC

DENY VIEW CHANGE TRACKING ON ATTRIB_DROITS TO GPEROC

DENY CONTROL ON ATTRIB_DROITS TO GPEROC

DENY INSERT ON ATTRIB_DROITS TO GPEROC

DENY UPDATE ON ATTRIB_DROITS TO GPEROC

DENY ALTER ON ATTRIB_DROITS TO GPEROC

DENY TAKE OWNERSHIP ON ATTRIB_DROITS TO GPEROC

DENY REFERENCES ON ATTRIB_DROITS TO GPEROC

DENY SELECT ON ATTRIB_DROITS TO GPEROC

DENY DELETE ON ATTRIB_DROITS TO GPEROC

Supprimer des droits (REVOKE)

1. Supprimer des droits sur la manipulation d'une table

Une fois les droits attribués, il faut pouvoir les enlever si l'utilisateur quitte l'entreprise par exemple ou change de service. Pour ceci, nous utiliserons l'ordre REVOKE.

La syntaxe Oracle est la suivante

REVOKE <droit1>, <droit2>, ...

ON TABLE <nom table>

FROM <user1>, <user2> ...;

La syntaxe MySQL est la suivante

REVOKE <droit1>, <droit2>, ...

ON [TABLE <nom table>],[*]

FROM <user1>, <user2> ...;

Pour enlever les droits de lecture de la table UTILISATEURS à Mr SMITH par exemple :

REVOKE SELECT ON UTILISATEURS FROM ASMITH;

Pour enlever tous les droits à Mr Smith :

REVOKE ALL PRIVILEGES ON UTILISATEURS FROM ASMITH;

De la même façon que le GRANT, si on ajoute le mot PUBLIC, on supprime les droits sur cette table pour tous les utilisateurs de la base de données.

REVOKE ALL PRIVILEGES ON UTILISATEURS FROM PUBLIC;

2. Supprimer des droits sur les objets de la base

Comme pour la manipulation d'une table, on peut supprimer des droits que l'on a attribués sur la création de table ou la création d'index par exemple.

Exemple Oracle

REVOKE CREATE TABLE FROM ASMITH;

Exemple MySQL

REVOKE CREATE ON * ON 'ASMITH';

Pour supprimer les droits de modification, de suppression et de création de tables en une seule commande, la syntaxe est :

Exemple Oracle

REVOKE CREATE TABLE, ALTER ANY TABLE, DROP ANY TABLE FROM ASMITH;

Exemple MySQL

REVOKE CREATE, ALTER, DROP ON * TO 'ASMITH';

Utilisation des rôles

Un rôle est un groupe auquel on va attribuer les mêmes droits. Au lieu d'attribuer les droits individuellement à chaque utilisateur, on peut créer des groupes qui auront des droits et ensuite affecter les utilisateurs dans un groupe.

Cette notion de rôle n'existe pas à l'heure actuelle dans MySQL.

Syntaxe Oracle

CREATE ROLE <nom rôle>;

Exemple Oracle

CREATE ROLE CONTROLE_GESTION;

Puis, il faut attribuer des droits au rôle que l'on vient de créer de la même façon que pour un utilisateur lambda.

```
GRANT ALL PRIVILEGES ON CLIENT TO CONTROLE_GESTION;  
GRANT ALL PRIVILEGES ON FOURNISSEUR TO CONTROLE_GESTION;  
GRANT ALL PRIVILEGES ON FACTURE TO CONTROLE_GESTION;
```

Pour SQL Server

```
GRANT UPDATE ON FOURNISSEUR TO CONTROLE_GESTION
```

Il faut ensuite attribuer ce rôle aux utilisateurs :

```
GRANT CONTROLE_GESTION TO ASMITH;  
GRANT CONTROLE_GESTION TO BMARTIN;  
etc ...
```

```
exec sp_addrolemember [CONTROLE_GESTION], ASMITH;
```

La notion de rôle est assez intéressante dans les sites avec beaucoup d'utilisateurs, elle permet de simplifier la gestion des différents profils.

Il faut bien définir en amont les différents rôles dans l'utilisation de la base de données et dans les applications existantes et ensuite classer les utilisateurs par rôle. À chaque nouvel arrivant, il suffit de créer son user et de lui attribuer un rôle.

Il existe également un certain nombre de rôles définis dans Oracle qui sont CONNECT, DBA ou RESOURCE mais ils sont plutôt réservés aux administrateurs de la base. Il est préférable de créer ses propres rôles afin de maîtriser sa sécurité.

Supprimer un rôle

Pour SQL Server, il est nécessaire de supprimer les membres du rôle avant de supprimer ce dernier.

Syntaxe SQL Server

```
ALTER ROLE <nom rôle> DROP MEMBER <user>;
```

Exemple SQL Server

```
ALTER ROLE CONTROLE_GESTION DROP MEMBER ASMITH;
```

Syntaxe

```
DROP ROLE <nom rôle>;
```

Exemple

```
DROP ROLE CONTROLE_GESTION;
```

Exercices

Premier exercice

Créer un utilisateur ALFRED et lui attribuer les droits de créer une session et de sélectionner des données dans la table CASTING.

Deuxième exercice

Attribuer à tous les utilisateurs le droit de sélectionner des données dans la table FILM.

Troisième exercice

Attribuer des droits à l'utilisateur ALFRED afin qu'il puisse modifier dans la table FILM uniquement les colonnes TITRE et RESUME.

Quatrième exercice

Supprimer pour l'utilisateur ALFRED les droits sur la modification de la colonne RESUME.

Solutions des exercices

Premier exercice

Créer l'utilisateur :

```
CREATE USER ALFRED IDENTIFIED BY ALFRED;
```

L'autoriser à se connecter :

```
GRANT CREATE SESSION TO ALFRED;
```

Lui permettre de sélectionner :

```
GRANT SELECT ON CASTING TO ALFRED;
```

Deuxième exercice

Utilisation du mot PUBLIC :

```
GRANT SELECT ON FILM TO PUBLIC;
```

Troisième exercice

Attribution des droits :

```
GRANT UPDATE (TITRE,RESUME) ON FILM TO ALFRED;
```

Si maintenant l'utilisateur ALFRED essaye de modifier une autre colonne, il aura un message d'erreur lui indiquant qu'il n'a pas les droits :

```
UPDATE FILM SET GENRE1='Toto' WHERE IDENT FILM =1
```

```
*
```

ERREUR à la ligne 1 :

ORA-01031: privilèges insuffisants

Sur la colonne TITRE en revanche, il est autorisé :

```
UPDATE TITRE SET TITRE='SUBWAY-' WHERE IDENT =1 ;
```


1 ligne(s) mise(s) à jour.

Quatrième exercice

Suppression des droits UPDATE sur la colonne TITRE.

On ne peut pas supprimer les droits uniquement sur une colonne, le REVOKE s'applique sur toutes les colonnes.

REVOKE UPDATE ON FILM FROM ALFRED;

Il faudra ensuite réattribuer les droits sur la colonne ARTICLE uniquement.

GRANT UPDATE (RESUME) ON FILM TO ALFRED;

Le contrôle de transactions (TCL)

Problématique des accès concurrents

Dans la majorité des développements informatiques se pose la question des accès simultanés à une donnée par plusieurs utilisateurs différents.

En effet, un développeur d'application doit prévoir la gestion des accès concurrents en utilisant les outils fournis par la base de données.

La majorité des SGBDR autorisent la réservation de données avant mise à jour afin d'empêcher un autre utilisateur de modifier cette même donnée avant que le premier n'ait validé sa modification.

1. Illustration des accès concurrents

a. Exemple 1 : mises à jour simultanées

Reprenons la table Tarifs :

```
SELECT * FROM Tarifs;
```

idTarif	Hotel	typeChambre	DateDebut	DateFin	Prix
1	1	1	2021-10-01	2022-04-14	49,99
2	1	2	2021-10-01	2022-04-14	59,99
3	1	3	2021-10-01	2022-04-14	68,99
4	1	4	2021-10-01	2022-04-14	59,99
5	1	5	2021-10-01	2022-04-14	69,99

Maintenant un utilisateur lit l'enregistrement numéro 2 et réalise une modification de la table en ajoutant 10 € au tarif du type de chambre n° 2 de l'hôtel n°1.

Au même moment, un autre utilisateur ajoute également 15 € à cet enregistrement numéro 2.

UTILISATEUR 1	Valeur PRIX	UTILISATEUR 2	Valeur PRIX
LECTURE Tarifs NUMERO 2	59,99		
MISE A JOUR PRIX = PRIX + 10	69,99	LECTURE Tarifs NUMERO 2	59,99
		MISE A JOUR PRIX= PRIX + 15	74,99
SAUVEGARDE ENREGISTREMENT NUMERO 2	69,99		
		SAUVEGARDE ENREGISTREMENT NUMERO 2	74,99

À l'issue de ce petit scénario, la valeur finale du prix de la chambre sera de 74,99 €. Il aurait dû être de 84,99 €. En effet, si les deux transactions s'étaient déroulées l'une après l'autre, cela aurait donné :

$59,99 + 10 = 69,99$ puis $69,99 + 15 = 84,99$ €

En conséquence, l'utilisateur 1 pense avoir ajouté 10 € au prix et le deuxième a effectivement ajouté les 15 € à son prix.

L'utilisateur 2 n'a pas pris en compte la modification du premier utilisateur qui passe le prix à 69,99. L'ajout des 15 € se serait appliqué à 69,99 et non à 59,99.

b. Exemple 2 : incohérence des données suite à une modification d'un autre utilisateur

Un utilisateur lit la table Tarifs au début de son programme, puis réalise des traitements par rapport à ce qu'il a récupéré.

Il relit cette même table en fin de traitement pour récupérer les mêmes informations. Entre-temps, un autre utilisateur a réalisé une modification ou une insertion dans la table.

UTILISATEUR 1	Valeur Prix	UTILISATEUR 2	Valeur Prix
LECTURE Tarifs NUMERO 3	68,99		
SI Prix = 68,99 APPEL PROG 1		LECTURE Tarifs NUMERO 3	68,99
SI COULEUR = NOIR MAJ TABLE Y		MISE A JOUR Prix a 78,99	78,99
		SAUVEGARDE ENREGISTREMENT NUMERO 3	78,99
LECTURE Tarifs NUMERO 3	78,99		
SI Prix = 68,99 APPEL PROG 2	!ERREUR!		

L'utilisateur 1 doit verrouiller l'enregistrement afin que personne ne puisse le modifier pendant le passage de son programme.

Ce phénomène est appelé : **lecture non répétitive**.

Autre exemple avec l'apparition de nouvelles lignes entre deux interrogations.

UTILISATEUR 1	Nombre trouvé	UTILISATEUR 2	Nombre dans la base
LECTURE DU NOMBRE DE Tarifs DE l'hôtel N° 1	20		
PREPARATION TABLEAU POUR STOCKER LES Tarifs AVEC 20 LIGNES		INSERT DE DEUX NOUVELLES LIGNES DANS LA TABLE Tarifs pour l'hôtel N° 1	22

UTILISATEUR 1	Nombre trouvé	UTILISATEUR 2	Nombre dans la base
		SAUVEGARDE DES NOUVEAUX ENREGISTREMENTS	22
LECTURE DU NOMBRE DE Tarifs DE l'hôtel N° 1	22		
INSERTION DES Tarifs DE l'hôtel N° 1 DANS LE TABLEAU	!ERREUR SUR NOMBRE DE LIGNES > TAILLE DU TABLEAU		

Ce phénomène est appelé : **lignes fantômes**.

2. Le mécanisme de verrouillage

Les mécanismes de verrouillage existent dans les SGBDR pour permettre aux développeurs de sécuriser leurs mises à jour et garantir une cohérence de la base.

Néanmoins, les verrouillages peuvent entraîner des blocages complets des utilisateurs. Il s'agit d'un problème d'interblocage illustré par l'exemple ci-dessous.

UTILISATEUR 1	UTILISATEUR 2
VERROUILLAGE LIGNE DE TELEPHONE POUR LE NUMERO 3	
	VERROUILLAGE LIGNE DE TELEPHONE POUR LE NUMERO 4
VERROUILLAGE LIGNE DE TELEPHONE POUR LE NUMERO 4	
ATTENTE !	VERROUILLAGE LIGNE DE TELEPHONE POUR LE NUMERO 3
	ATTENTE !

Les deux utilisateurs sont bloqués en attente de libération de la ligne désirée.

Il faut « tuer » un des deux utilisateurs pour débloquer les deux.

Ceci pour montrer que le verrouillage doit être réfléchi et intégré dans les normes de développement de l'entreprise. L'exemple montre qu'il faut libérer les ressources réservées le plus rapidement possible.

Si le besoin est vraiment de modifier beaucoup de lignes avant de valider les modifications, il est préférable de réserver la table entière pour éviter les inter-blocages ou deadlock en anglais.

Notion de transaction

Afin de limiter les problématiques indiquées dans les paragraphes précédents, il faut que le développeur se pose la question : quels sont les objets (ligne, colonne, table...) que je manipule dans mon traitement et comment éviter qu'un autre utilisateur ne puisse les atteindre avant que j'aie terminé mes mises à jour ?

À ce moment, on commence à parler de « transaction ».

1. Définition d'une transaction

La transaction permet de borner le début et la fin d'une action dans la base, sur une ou plusieurs tables, qui doivent rester cohérentes.

La transaction est une notion qui vient des applications dites « transactionnelles ». À chaque fois qu'il existe un dialogue écran entre l'application et l'utilisateur, on parle d'application transactionnelle.

Toutes les applications utilisent des mécanismes de verrouillage.

2. Comment éviter les incohérences de données

Plusieurs méthodes existent pour garantir une cohérence dans la base :

- Lancer les transactions les unes derrière les autres (en série). L'inconvénient principal est le temps d'attente très important pour les utilisateurs. Cela revient à avoir une application mono-utilisateur !
- Possibilité de bloquer en début de programme principal tous les objets implicitement puis les libérer à la fin. Cette méthode risque de bloquer les autres utilisateurs trop longtemps si l'utilisateur ne valide pas rapidement son écran et part en réunion quelque temps.
- Pas de verrouillage des enregistrements en début mais lecture des données puis sauvegarde en mémoire de leur contenu puis relecture juste avant la mise à jour pour s'assurer que les données n'ont pas été modifiées entre temps. Si les valeurs sauvegardées sont différentes des valeurs relues, alors abandon de la transaction et affichage d'un message à l'utilisateur pour signaler le problème et lui indiquer qu'il doit ressaisir les éléments.

3. Mise en œuvre d'un verrouillage

Selon la norme SQL-92, on peut indiquer à la base de données quel type de méthode on veut utiliser.

Pour activer un mode de verrouillage, il faut utiliser la commande : SET TRANSACTION ISOLATION LEVEL.

Il existe quatre modes de verrouillage.

Le mode par défaut dans MySQL est le mode REPEATABLE-READ. Pour Oracle, PostgreSQL et SQL Server, il s'agit du mode READ COMMITTED.

Généralement, l'action s'applique à une session, celle de l'utilisateur qui l'active, certains SGBDR permettent d'appliquer le niveau de verrouillage à toutes les sessions en cours en utilisant le mot GLOBAL.

Nous allons décrire les différents modes.

a. READ UNCOMMITTED

Quand ce mode est activé, l'utilisateur a la possibilité de visualiser les lignes qui sont en cours de modification par une autre personne, même si elles ne sont pas validées. Les données ne sont donc pas garanties du point de vue fonctionnel, en effet tant que l'autre utilisateur n'a pas validé sa modification les données sont volatiles.

b. READ COMMITTED

L'utilisateur ne visualise que les données validées. En aucun cas il ne peut voir des lignes en cours de modification non validées. C'est le mode de fonctionnement par défaut utilisé par Oracle. Ce mode garantit une cohérence des données entre les utilisateurs.

c. REPEATABLE-READ

Toutes les données utilisées par une requête sont verrouillées, ceci empêche toute modification de données par les autres utilisateurs. En revanche, les autres utilisateurs ont encore la possibilité d'insérer de nouvelles lignes dans la table. Ce niveau est assez restrictif et peut provoquer des attentes importantes auprès des utilisateurs.

d. SERIALIZABLE

C'est le mode le plus restrictif et le plus contraignant pour les utilisateurs. En contrepartie, c'est le mode le plus sécurisant pour le développeur mais il est déconseillé de l'utiliser sauf sur des applications avec très peu d'utilisateurs. Ce mode bloque l'ensemble de la table à chaque demande de l'utilisateur. Aucune action n'est possible par les autres utilisateurs tant que le premier n'a pas validé sa transaction.

e. Syntaxes

Syntaxe MySQL

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
```

```
{ READ UNCOMMITTED
```

```
| READ COMMITTED
```

```
| REPEATABLE READ
```

```
| SERIALIZABLE
```

```
};
```

Syntaxe Oracle

```
SET TRANSACTION ISOLATION LEVEL
```

```
{ READ COMMITTED
```

```
| SERIALIZABLE
```

```
};
```

Syntaxe SQL Server et PostgreSQL

```
SET TRANSACTION ISOLATION LEVEL
```

```
{ READ UNCOMMITTED
```

```
| READ COMMITTED
```

```
| REPEATABLE READ
```

```
| SERIALIZABLE
```

```
};
```

Oracle propose d'autres possibilités, notamment avec le READ ONLY qui permet d'indiquer que l'on ne fait que de la lecture ou le READ WRITE pour indiquer que l'on fait de la mise à jour.

4. Mise en œuvre d'un verrouillage applicatif

Les règles de verrouillage qui s'appliquent pour les transactions sont mises en œuvre la plupart du temps par les administrateurs de la base et s'appliquent à l'ensemble de développeurs.

Indépendamment des règles gérées au niveau d'une transaction comme indiqué dans la section précédente, il est possible de bloquer des lignes d'une table pour éviter qu'un autre utilisateur les modifie avant la fin de notre traitement.

Il existe deux méthodes principales pour gérer ces cas. Le verrou exclusif (FOR UPDATE) qui bloque toutes les lignes ramenées par le SELECT, empêchant toute modification d'un autre utilisateur. Le verrou partagé (SHARE MODE) qui laisse la possibilité aux autres utilisateurs de visualiser les lignes bloquées dans leur état avant la pose du verrou.

Syntaxe Oracle et PostgreSQL

```
SELECT ... FROM .... WHERE ....
```

```
FOR UPDATE;
```

ou :

```
LOCK TABLE <nom table> IN SHARE MODE;
```

Syntaxe MySQL

```
SELECT ... FROM .... WHERE ....
```

```
FOR UPDATE;
```

ou :

```
SELECT ... FROM .... WHERE ....
```

```
LOCK IN SHARE MODE;
```

Syntaxe SQL Server

```
SELECT ... FROM .... WHERE ....
```

WITH (NOWAIT) ; --> affiche un message à la place du résultat

```
SELECT ... FROM .... WHERE ....
```

WITH (READPAST); --> n'affiche pas les lignes verrouillées

```
SELECT ... FROM .... WHERE ....
```

WITH (NOLOCK) ; --> affiche les lignes verrouillées dirtyread

Exemple Oracle et PostgreSQL

```
SELECT idTarif, hotel, typeChambre, DateDebut, DateFin, Prix FROM
```

```
Tarifs WHERE hotel = 1
```

```
FOR UPDATE;
```

ou :

```
LOCK TABLE Tarifs IN SHARE MODE;
```

Exemple MySQL

```
SELECT idTarif, hotel, typeChambre, DateDebut, DateFin, Prix FROM
```

```
Tarifs WHERE hotel = 1
```

LOCK IN SHARE MODE;

Exemple SQL Server

```
select * from Tarifs with (NOWAIT);
```

```
select * from Tarifs with (READPAST);
```

```
select * from Tarifs with (NOLOCK);
```

Les verrous partagés ne peuvent être posés avec Oracle dans un ordre SELECT. Il faut pour cela utiliser le verbe LOCK TABLE qui permet de poser plusieurs types de verrou sur une table (ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVES). Le descriptif de chaque possibilité est dans la documentation officielle.

a. Comment connaître les verrous posés sur une table ?

Lorsque l'on accède à une table et que le SGBDR nous indique que cette donnée n'est pas disponible en renvoyant par exemple le code ORA-00054 ou ORA-00060, il est possible de connaître les verrous posés sur une table.

Pour ce faire, on doit accéder à des tables dites « système » qui contiennent toutes les actions réalisées à un instant t par les utilisateurs de la base de données.

Si un utilisateur a réservé la table Tarifs, il aura exécuté une requête de ce type :

```
SELECT * FROM Tarifs FOR UPDATE;
```

Si une autre personne essaye de mettre à jour cette table, une erreur lui sera renvoyée indiquant que la table est « lockée ».

Pour connaître la liste des tables lockées, il faut interroger la table V\$LOCKED_OBJECT qui contient par objet la session qui l'utilise. Si on fait une jointure avec la table ALL_OBJECTS, on peut lister les tables concernées.

Exemple Oracle

```
SELECT T1.OBJECT_ID, T2.OBJECT_NAME, T1.SESSION_ID FROM V$LOCKED_OBJECT  
T1, ALL_OBJECTS T2 WHERE T1.OBJECT_ID = T2.OBJECT_ID;
```

Le résultat est :

OBJECT_ID	OBJECT_NAME	SESSION_ID
13637	TARIFS	28

Pour savoir qui est sur la session 28 dans notre exemple, on doit interroger la table V\$SESSION qui contient d'autres informations sur l'utilisateur.

Exemple Oracle

```
SELECT T3.SERIAL# ,T3.SID,T1.OBJECT_ID,T2.OBJECT_NAME,  
T1.SESSION_ID,T3.USERNAME,T3.STATUS,T3.OSUSER,T3.PROCESS  
FROM V$LOCKED_OBJECT T1, ALL_OBJECTS T2, V$SESSION T3
```



```
WHERE T1.OBJECT_ID = T2.OBJECT_ID
```

```
AND T3.SID = T1.SESSION_ID;
```

Le résultat est :

```
SERIAL#      SID OBJECT_ID OBJECT_NAME
-----
      997      28   13637  TARIFS
```

```
SESSION_ID USERNAME          STATUS
```

```
-----
      28 SYSTEM              ACTIVE
```

```
OSUSER          PROCESS
```

```
-----
xxxxxxxxxxxxxx  7328:9188
```

Dans le résultat, on notera le OSUSER qui est le nom en clair de l'utilisateur, le nom de connexion dans USERNAME et la table concernée dans OBJECT_NAME.

Dans SQL Server, on utilisera :

```
select object_name(p.object_id) as TableName, resource_type,
resource_description from sys.dm_tran_locks l join sys.partitions p on
l.resource_associated_entity_id = p.hobt_id
```

Exemple MySQL

```
SELECT USER, STATE FROM information_schema.processlist
WHERE ID = 28;
```

Exemple PostgreSQL

```
SELECT username, state FROM pg_stat_activity
WHERE pid = 28;
```

b. Comment supprimer des verrous posés sur une table ?

Maintenant que l'on a vu comment retrouver les verrous posés sur une table, on peut, dans la mesure où on a les droits accordés par le DBA, supprimer ce verrou.

Il existe deux méthodes pour supprimer un verrou : soit on supprime la session Oracle de l'utilisateur, soit on supprime la session système (UNIX/Windows).

La première méthode consiste à déconnecter l'utilisateur de la base, ce qui a pour effet de libérer la session et la ou les tables concernées.

Si l'on reprend l'exemple précédent, on avait :

SERIAL#	SID	OBJECT_ID	OBJECT_NAME
997	28	13637	TARIFS

SESSION_ID	USERNAME	STATUS
28	SYSTEM	ACTIVE

OSUSER	PROCESS
XXXXXXXXXXXXXX	7328:9188

Les deux éléments les plus importants sont le SID et le SERIAL#. Avec ces deux numéros, vous pouvez tuer la session de l'utilisateur avec la commande ALTER SYSTEM KILL SESSION 'SID,SERIAL#';.

Exemple Oracle

```
ALTER SYSTEM KILL SESSION '28,997';
```

La session de l'utilisateur reste ouverte (mais inactive), mais à la prochaine commande il aura le message suivant, qui signifie qu'il n'est plus connecté à la base :

ORA-03113: fin de fichier sur canal de communication

La deuxième méthode consiste à tuer la session système de l'utilisateur, la session UNIX ou Windows.

Reprenons l'exemple de la section précédente.

Exemple Oracle

```
SELECT T3.SERIAL# ,T3.SID,T1.OBJECT_ID,T2.OBJECT_NAME,
T1.SESSION_ID,T3.USERNAME,T3.STATUS,T3.OSUSER,T3.PROCESS
FROM V$LOCKED_OBJECT T1, ALL_OBJECTS T2, V$SESSION T3
WHERE T1.OBJECT_ID = T2.OBJECT_ID
AND T3.SID = T1.SESSION_ID;
```

Le résultat est :

SERIAL#	SID	OBJECT_ID	OBJECT_NAME
997	28	13637	TARIFS

SESSION_ID	USERNAME	STATUS
28	SYSTEM	ACTIVE

XXXXXXXXXXXXXX

7328:9188

Le contenu de la colonne PROCESS correspond sous UNIX au PID permettant de tuer la session en cas de blocage. Pour cela, il faudra faire un kill -9 7328, comme dans SQL Server, MySQL et PostgreSQL.

Sous Windows, pour retrouver le numéro de session, il faut se rendre dans le Gestionnaire des tâches, sélectionner l'onglet **Processus**, cliquer sur la ligne dont le numéro de la colonne **PID** correspond à celui recherché, puis cliquer sur le bouton **Arrêter le processus** ou clic droit **Arrêter le processus**.

5. Validation des modifications (COMMIT)

Le principe de base dans tous les SGBDR est de laisser à l'utilisateur le soin de valider ses modifications à la fin de sa transaction et ainsi de rendre visible les changements réalisés dans les données aux autres utilisateurs.

Celui-ci peut être programmé par le développeur au moment souhaité afin de garantir l'intégrité référentielle de la base de données. C'est donc lié au fonctionnement même de chaque programme.

Le COMMIT valide toutes les modifications, insertions ou suppressions qui ont été réalisées depuis le dernier COMMIT ou depuis le début de la transaction.

Dans un programme, on peut mettre plusieurs COMMIT permettant ainsi de découper le traitement en séquences cohérentes fonctionnellement.

Le COMMIT permet également de libérer tous les verrous qui auraient été posés auparavant.

Syntaxe

COMMIT;

Il existe également la notion d'AUTO COMMIT notamment dans MySQL et Oracle qui indique que chaque commande SQL devient une transaction et donc chaque commande est validée automatiquement après son exécution.

Cette option est à manier avec précaution car elle interdit tout retour arrière en cas de mauvaise manipulation.

Syntaxe

SET AUTOCOMMIT ON;

SET AUTOCOMMIT OFF;

6. Abandon des modifications (ROLLBACK)

Le ROLLBACK est le contraire du COMMIT. Celui-ci invalide toutes les modifications, insertions et suppressions qui auraient été réalisées depuis le dernier COMMIT ou le début de la transaction ou depuis le dernier point de synchronisation (SAVEPOINT).

C'est très utile lorsque l'on réalise des essais d'ordres SQL sur une base et que l'on fait de fausses manipulations. Dans ce cas avant de quitter, il faut lancer le ROLLBACK et la base revient dans son état initial.

Syntaxe Oracle, MySQL et PostgreSQL

ROLLBACK [TO SAVEPOINT <nom du point de contrôle>];

Syntaxe SQL Server

ROLLBACK [TRANSACTION <nom du point de contrôle>];

7. Les points de synchronisation (SAVEPOINT)

Lorsque l'on déroule un ensemble d'ordre SQL visant à valider un traitement fonctionnel, il existe la possibilité de poser des points de contrôle à plusieurs stades, permettant ainsi en cas de problème de revenir à un point cohérent de la base.

À noter qu'un COMMIT annule tous les points de synchronisation. Un ROLLBACK sans point de synchronisation annule également tous les points de synchronisation et remet la base dans l'état initial.

Syntaxe Oracle, MySQL et PostgreSQL

SAVEPOINT <nom du point de contrôle>;

Exemple

SAVEPOINT AVANT_INSERT;

Syntaxe SQL Server

SAVE TRANSACTION <nom du point de contrôle>;

Exemple

SAVE TRANSACTION AVANT_INSERT;

8. Exemple d'utilisation des points de synchronisation

Vidage d'une table puis création de lignes avec activation de points de synchronisation avant chaque insertion :

Supprimer les clés étrangères des tables TYPESCHAMBRE et TARIFS avant d'effectuer ce script

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

/ Vidage de la table typeschambre */*

DELETE FROM typeschambre;

SELECT COUNT(*) FROM chambres WHERE hotel = 1;

Première insertion dans la table TYPESCHAMBRE : 1 lit simple

INSERT INTO typeschambre VALUES (1,1,'lit simple','1 lit simple avec
douche et WC');

Première synchronisation

/ Positionnement du premier point */***SAVEPOINT ETAPE_NUMERO1;**

Deuxième insertion dans la table TYPESCHAMBRE : 1 lit double

INSERT INTO typeschambre VALUES (2,1,'lit double', '1 lit double avec douche et WC');

DELETE FROM CHAMBRES WHERE hotel = 1;

SELECT * FROM typeschambre WHERE nombrelit = 1;

Deuxième synchronisation

/ Positionnement du deuxième point */*

SAVEPOINT ETAPE_NUMERO2;

INSERT INTO typeschambre VALUES (3,1, 'lit double', '1 lit double avec douche et WC');

/ Retour au point numéro 2 afin de ne pas insérer 2 fois*

*1 lit double */*

ROLLBACK TO ETAPE_NUMERO2;

COMMIT;

Ce qui donne le résultat suivant :

Succès de l'élément transaction ISOLATION.

7 lignes supprimé.

COUNT(1)

6

1 ligne inséré.

Savepoint créé(e).

1 ligne inséré.

6 ligne(s) supprimée(s).

IDTYPECHAMBRE NOMBRELIT TYPELIT DESCRIPTION

1 1 lit simple 1 lit simple avec douche et WC

2 1 lit double 1 lit double avec douche et WC

Savepoint créé.

1 ligne insérée.

Annulation (rollback) terminée.

Validation (commit) effectuée

Et si on regarde dans la table TYPECHAMBRE, il n'y a qu'une seule ligne avec 1 lit double, la deuxième insertion n'a pas été prise en compte étant donné que l'on a annulé l'insertion avec le ROLLBACK TO ETAPE_NUMERO2 tout en conservant les mises à jour antérieures au point numéro 2.

Contenu de la table TYPECHAMBRE :

IDTYPECHAMBRE	NOMBRELIT	TYPELIT	DESCRIPTION
---------------	-----------	---------	-------------

1	1 lit simple		1 lit simple avec douche et WC
2	1 lit double		1 lit double avec douche et WC

Exercice

- Créer une transaction qui vide la table FILM.
- Puis compter le nombre d'actrices Carrie Fisher.
- Insérer dans la table FILM le film Subway, genres POLICIER et DRAME, sorti le 10 avril 1985 en France, du réalisateur Luc Besson, distribué par GAUMONT avec ce résumé : « Conte les aventures de la population souterraine dans les couloirs du métro parisien ».
- Créer un point de synchronisation.
- Insérer dans la table FILM le film Nikita, genres DRAME et ROMANTIQUE, sorti le 21 février 1990 en France, du réalisateur Luc Besson, distribué par GAUMONT avec ce résumé : « Nikita condamnée à la prison à perpétuité est contrainte à travailler secrètement pour le gouvernement en tant qu'agent hautement qualifié des services secrets ».
- Supprimer l'actrice Carrie Fisher.
- Sélectionner les films du réalisateur Luc Besson sortis entre le 1er janvier 1985 et le 30 mai 1985, trier par titre de film.
- Positionner un second point de synchronisation.
- Insérer dans la table FILM le film Subway, genres POLICIER et DRAME, sorti le 10 avril 1985 en France, du réalisateur Luc Besson, distribué par GAUMONT avec ce résumé : « Conte les aventures de la population souterraine dans les couloirs du métro parisien ».
- Annuler les actions jusqu'au deuxième point de synchronisation.
- Valider le script puis l'exécuter.

Solution de l'exercice

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
/* Vidage de la table FILM */
```

```
DELETE FROM FILM;
```

```
SELECT COUNT(*) FROM ACTEUR WHERE NOM = 'FISHER' AND PRENOM =  
'CARRIE';
```

```
INSERT INTO FILM VALUES (1,'SUBWAY','POLICIER','DRAME',  
TO_DATE('10/04/1985','DD/MM/YYYY'),1,'GAUMONT',  
'Conte les aventures de la population souterraine dans les  
couloirs du métro parisien');
```

```
/* Positionnement du premier point */
```

```
SAVEPOINT ETAPE_NUMERO1;
```

```
INSERT INTO FILM VALUES (2,'NIKITA','DRAME','ROMANTIQUE',  
TO_DATE('21/02/1990','DD/MM/YYYY'),1,1,'GAUMONT',  
'Nikita condamnée à la prison à perpétuité est contrainte à  
travailler secrètement pour le gouvernement en tant que agent  
hautement qualifié des services secrets.');
```

```
DELETE FROM ACTEUR WHERE NOM = 'FISHER' AND PRENOM = 'CARRIE';
```

```
SELECT * FROM FILM T1 WHERE  
EXISTS (SELECT IDENT_REALISATEUR FROM REALISATEUR  
WHERE PRENOM = 'LUC' AND IDENT_REALISATEUR =  
T1.IDENT_REALISATEUR ) AND DATE_SORTIE  
BETWEEN ('01/01/85') AND ('30/05/1995')  
ORDER BY TTTRE;
```

/* Positionnement du deuxième point */

SAVEPOINT ETAPE_NUMERO2;

INSERT INTO FILM VALUES (3,'SUBWAY','POLICIER','DRAME',
TO_DATE('10/04/1985','DD/MM/YYYY'),1,1,'GAUMONT',
'Conte les aventures de la population souterraine dans les
couloirs du métro parisien');

/* Retour au point numéro 2 afin de ne pas insérer 2 fois le film
'SUBWAY' */

ROLLBACK TO ETAPE_NUMERO2;

COMMIT;

Ident _ Film	Titre	Genre1	Genre2	Date_ Sortie	Pay s	Ident_ Realisate ur	Distribute ur	Resume
1	SUBWA Y	POLICI ER	DRAME	10/04/ 85	1	1	GAUMO NT	Conte les aventures de la population souterraine dans les couloirs du métro parisien
2	NIKIT A	DRAME	ROMANTIQ UE	21/02/ 90	1	1	GAUMO NT	Nikita condamnée à la prison à perpétuité est contrainte à travailler secrètement pour le gouverneme nt en tant que agent hautement qualifié des services secrets.

La programmation

Introduction

La programmation permet de créer des procédures stockées, des fonctions et des déclencheurs ou même de réaliser des applications plus ou moins complexes.

Oracle a créé son propre langage structuré : le PL/SQL. Il permet d'associer des ordres SQL avec des commandes d'un langage procédural.

Les éléments créés en PL/SQL doivent être compilés avant d'être exécutés.

Toutes les instructions SQL sont utilisables dans un bloc PL/SQL. Un « bloc » est un morceau de code PL/SQL, équivalent à une fonction ou une procédure dans un autre langage.

PostgreSQL propose le PL/pgSQL.

Syntaxe générale

Un programme peut se décomposer en trois parties :

- une partie déclarative,
- une partie traitement,
- une partie gestion des erreurs.

La partie déclarative permet de déclarer et d'initialiser toutes les variables utilisées dans la partie traitement. Dans un programme PL/SQL, on peut utiliser les types Oracle pour les variables mais également créer ses propres types.

La partie gestion des erreurs permet d'indiquer les instructions à appliquer lorsqu'une erreur est rencontrée dans la partie traitement.

Ces deux sections (déclarative et erreur) sont facultatives.

La syntaxe d'un programme est la suivante :

```
[DECLARE
```

```
...]
```

```
BEGIN
```

```
...
```

```
...
```

```
[EXCEPTION
```

```
...]
```

```
END;
```

Dans MySQL et PostgreSQL, ces blocs ne peuvent être utilisés seuls. Ils doivent être inclus dans une fonction ou un déclencheur, et pour MySQL une procédure.

Exemple de script SQL Server à exécuter

```
DECLARE @Hotel int
```

BEGIN

SET @Hotel = 2

SELECT NumChambre, Description

FROM Chambres INNER JOIN TypesChambre ON TypesChambre.idTypeChambre =

Chambres.TypeChambre

WHERE Hotel = @Hotel;

END;

Résultat

NumChambre	Description
1	1 lit simple avec douche
2	2 lits simples avec douche
3	3 lits simples avec douche et WC séparés
4	1 lit double avec douche
5	1 lit double avec douche et WC séparés
6	1 lit double avec bain et WC séparés
7	1 lit double large avec bain et WC séparés

Par exemple, sélectionner le libellé, le nombre d'étoiles, le numéro de chambre, le nombre de lits, le type de lit et la description de la chambre dont le type de lit est un lit XL et l'hôtel un 2 étoiles.

On peut coder en PL/SQL ainsi :

DECLARE

Etoile_recherche varchar2(5) := '**';

Libelle_hotel varchar2(50);

Num_Chambre varchar2(6);

NbLit number(38,0);

TypLit varchar2(20) := 'lit XL';

Descript varchar2(255);

BEGIN

SELECT Hotels.Libelle, Chambres.NumChambre,

TypesChambre.NombreLit, TypesChambre.Description

INTO Libelle_hotel, Num_Chambre, NbLit, descript

FROM Chambres INNER JOIN

Hotels ON Chambres.Hotel = Hotels.idHotel INNER JOIN

```

TypesChambre ON Chambres.TypeChambre = TypesChambre.idTypeChambre
WHERE TypeLit = typlit and Etoile = etoile_recherche;
DBMS_OUTPUT.put_line('SQLCODE : ' || TO_CHAR(SQLCODE));
IF SQLCODE = 0 THEN
DBMS_OUTPUT.PUT_LINE('Nom de l'hôtel : ' || Libelle_hotel);
DBMS_OUTPUT.PUT_LINE('Numéro de chambre : ' || num_chambre);
.PUT_LINE('Nombre de lit : ' || nblit);
DBMS_OUTPUT.PUT_LINE('Type de lit : ' || typlit);
DBMS_OUTPUT.PUT_LINE('Description : ' || descript);
END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Aucune ligne trouvée avec ' || etoile_re
cherche || ' étoiles. ');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Le numéro de l'erreur est :
' || TO_CHAR(SQLCODE));
        DBMS_OUTPUT.PUT_LINE('correspondant à : ' || TO_CHAR(SQLERRM));
END;

```

La déclaration des variables peut se faire en indiquant le type que l'on désire mais on peut également indiquer le nom d'une colonne d'une table et ainsi la nouvelle variable prend le même type que la colonne. Il faut pour cela ajouter « %type » après le nom de la variable.

Exemples

```
Etoile_recherche varchar2(5) := '**';
```

```
Libelle_hotel hotels.etoile%TYPE;
```

```
Num_Chambre chambres.numchambre%TYPE;
```

```
NbLit typeschambre.nombrelit%TYPE;
```

```
TypLit varchar2(20) := 'lit XL';
```

```
Descript typeschambre.description%TYPE;
```

DBMS_OUTPUT.PUT_LINE est une fonction Oracle qui permet d'afficher des informations en ligne de commande.

Pour voir les résultats de la procédure, il faut activer sous SQL*Plus l'affichage avec la commande SET SERVEROUTPUT ON.

Résultat de l'exécution sous SQL Developer

```
SQLCODE : 0
```

Nom de l'hôtel : Art Hotel

Numéro de chambre : 7

Nombre de lit : 1

Type de lit : lit XL

Description : 1 lit double large avec bain et WC séparés

Procédure PL/SQL terminée.

Si par exemple on remplace '**' par '*'.

Aucune ligne trouvée avec le pays * étoiles.

Procédure PL/SQL terminée

La procédure nous indique qu'aucune ligne ne correspond pour une étoile. L'exception NO_DATA_FOUND a intercepté l'erreur et affiche le message prévu à cet effet. Le SQLCODE correspondant à NO_DATA_FOUND est +100.

SQLCODE contient toujours le numéro de l'erreur de la dernière instruction SQL exécutée. Il est à zéro si aucune erreur n'est rencontrée.

SQLERRM contient le libellé correspondant au dernier numéro d'erreur SQL rencontré.

Si en revanche on recherche les chambres avec un lit simple, voici le résultat :

Le numéro de l'erreur est : -1422

correspondant à : ORA-01422: l'extraction exacte ramène plus que

le nombre de lignes demandé

Procédure PL/SQL terminée.

Il y a plusieurs lignes qui correspondent à la sélection, et comme on ne peut en réceptionner qu'une seule dans le INTO, l'exception nous affiche le code et le libellé d'erreur correspondant.

Lorsque l'on désire ramener plusieurs lignes, il faut utiliser des curseurs.

Les curseurs

Un curseur est un élément qui permet de stocker le résultat d'une requête contenant plusieurs lignes.

Il faut le déclarer dans la section déclarative.

Il faut l'ouvrir par OPEN, l'exécuter par FETCH et le fermer par CLOSE.

Dans l'exemple, le type de lit recherché est passé en paramètre au curseur : CURSOR C_chambres_par_type_lit (TypLit IN VARCHAR2) IS.

TypLit est renseigné lors de l'OPEN CURSOR avec la variable qui contient le libellé du type de lit : OPEN C_chambres_par_type_lit(TypLit_recherche).

Exemple avec la même requête que précédemment :

```
DECLARE

-- Déclaration du curseur C_chambres_par_type_lit
CURSOR C_chambres_par_type_lit (TypLit in varchar2) IS
SELECT Hotels.Libelle, Chambres.NumChambre, TypesChambre.NombreLit,
TypesChambre.Description
FROM Chambres INNER JOIN
Hotels ON Chambres.Hotel = Hotels.idHotel INNER JOIN
TypesChambre ON Chambres.TypeChambre = TypesChambre.idTypeChambre
WHERE TypeLit = typlit and Etoile = '**';

-- Déclaration des variables réceptrices
Libelle_hotel varchar2(50);
Num_Chambre varchar2(6);
NbLit number(38,0);
Descript varchar2(255);

-- Déclaration des autres variables
TypLit_recherche varchar2(20) := 'lit simple';

BEGIN

-- Ouverture
OPEN C_chambres_par_type_lit(typlit_recherche);

-- Boucle de lecture
LOOP

-- Récupération des éléments ligne par ligne
FETCH C_chambres_par_type_lit
INTO Libelle_hotel, Num_Chambre, NbLit, descript;
EXIT WHEN C_chambres_par_type_lit%NOTFOUND;

-- Affichage des éléments récupérés
DBMS_OUTPUT.PUT_LINE('Nom de l'hôtel : ' || Libelle_hotel);
DBMS_OUTPUT.PUT_LINE('Numéro de chambre : ' || num_chambre);
DBMS_OUTPUT.PUT_LINE('Nombre de lit : ' || nblit);
```

```

DBMS_OUTPUT.PUT_LINE('Type de lit : ' || typlit_recherche);
DBMS_OUTPUT.PUT_LINE('Description : ' || descript);
END LOOP;
-- Fermeture du curseur (libération mémoire)
CLOSE C_chambres_par_type_lit;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Aucune ligne trouvée avec
' || etoile_recherche || ' étoiles. ');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Le numéro de l'erreur est :
' || TO_CHAR(SQLCODE));
        DBMS_OUTPUT.PUT_LINE('correspondant à : ' || TO_CHAR(SQLERRM));
END;

```

Résultats de la requête

Nom de l'hôtel : Art Hotel

Numéro de chambre : 1

Nombre de lit : 1

Type de lit : lit simple

Description : 1 lit simple avec douche

Nom de l'hôtel : Art Hotel

Numéro de chambre : 2

Nombre de lit : 2

Type de lit : lit simple

Description : 2 lits simples avec douche

Nom de l'hôtel : Art Hotel

Numéro de chambre : 3

Nombre de lit : 3

Type de lit : lit simple

Description : 3 lits simples avec douche et WC séparés

Procédure PL/SQL terminée.

Exemple SQL Server

BEGIN

-- Liste des chambres

DECLARE @libelle varchar(50)

DECLARE @etoile varchar(5)

DECLARE @numChambre int

DECLARE @typeLit varchar(20)

DECLARE @nombreLit int

DECLARE @description varchar(255)

DECLARE C1 CURSOR

FOR

SELECT TOP 1 Hotels.Libelle, Hotels.Etoile, Chambres.NumChambre,
TypesChambre.TypeLit, TypesChambre.NombreLit, TypesChambre.Description
FROM Hotels INNER JOIN Chambres ON Hotels.idHotel = Chambres.Hotel
INNER JOIN TypesChambre ON Chambres.TypeChambre =
TypesChambre.idTypeChambre
ORDER BY Hotels.idHotel, Chambres.NumChambre;

OPEN C1;

FETCH C1 into @libelle, @etoile, @numChambre, @typeLit, @nombreLit,
@description;

WHILE @@FETCH_STATUS=0

BEGIN

print 'Nom de l'hôtel : ' + @libelle

print 'Etoiles : ' + @etoile

print 'Numéro de chambre : ' + CONVERT(varchar(3),

@numChambre)

print 'Type de lit : ' + @typeLit

print 'Nombre de lits : ' + CONVERT(char(1), @nombreLit)

print 'Description de la chambre : ' + @description

FETCH C1 into @libelle, @etoile, @numChambre, @typeLit,

```
@nombreLit, @description; --ligne suivante
```

```
END;
```

```
CLOSE C1;
```

```
DEALLOCATE C1;
```

```
END;
```

Résultat

Nom de l'hôtel : Ski Hotel

Etoiles : *

Numéro de chambre : 1

Type de lit : lit simple

Nombre de lits : 1

Description de la chambre : 1 lit simple avec douche

Le contrôle des flux

1. La boucle WHILE

Le WHILE permet de répéter un bout de code tant que la condition testée au début est vraie. Si la condition est fausse, on sort directement de la boucle sans exécuter le code.

Exemple

```
DECLARE
```

```
-- Déclaration du curseur C_chambres_par_type_lit
```

```
CURSOR C_chambres_par_type_lit (TypLit in varchar2) IS
```

```
SELECT  Chambres.idChambre,  Hotels.Libelle,  Chambres.NumChambre,  TypesChambre.NombreLit,  
TypesChambre.Description
```

```
FROM Chambres INNER JOIN
```

```
Hotels ON Chambres.Hotel = Hotels.idHotel INNER JOIN
```

```
TypesChambre ON Chambres.TypeChambre = TypesChambre.idTypeChambre
```

```
WHERE TypeLit = typelit and Etoile = '**';
```

```
-- Déclaration des variables réceptrices
```

```
id_chambre number :=0;
```

```
Libelle_hotel varchar2(50);
```

```
Num_Chambre varchar2(6);
```

```
NbLit number(38,0);
```

```
Descript varchar2(255);
```



```

-- Déclaration des autres variables
TypLit_recherche varchar2(20) := 'lit simple';

BEGIN

-- Ouverture
OPEN C_chambres_par_type_lit(typlit_recherche);

-- Lecture du premier élément
FETCH C_chambres_par_type_lit
INTO id_chambre, Libelle_hotel, Num_Chambre, NbLit, descript;

-- Boucle de lecture tant que l'identifiant de la chambre est < 10
WHILE id_chambre < 10
LOOP

-- Affichage des éléments récupérés
DBMS_OUTPUT.PUT_LINE('Id Chambre : ' || id_chambre);
DBMS_OUTPUT.PUT_LINE('Nom de l'hôtel : ' || Libelle_hotel);
DBMS_OUTPUT.PUT_LINE('Numéro de chambre : ' || num_chambre);
DBMS_OUTPUT.PUT_LINE('Nombre de lit : ' || nblit);
DBMS_OUTPUT.PUT_LINE('Type de lit : ' || typlit_recherche);
DBMS_OUTPUT.PUT_LINE('Description : ' || descript);

-- Lecture de l'élément suivant
FETCH C_chambres_par_type_lit
INTO id_chambre, Libelle_hotel, Num_Chambre, NbLit, descript;

EXIT WHEN C_chambres_par_type_lit%NOTFOUND;

END LOOP;

-- Fermeture du curseur (libération mémoire)
CLOSE C_chambres_par_type_lit;

END;

```

Résultat

```

Id Chambre : 8
Nom de l'hôtel : Art Hotel
Numéro de chambre : 1
Nombre de lit : 1

```

Type de lit : lit simple

Description : 1 lit simple avec douche

Id Chambre : 9

Nom de l'hôtel : Art Hotel

Numéro de chambre : 2

Nombre de lit : 2

Type de lit : lit simple

Description : 2 lits simples avec douche

Procédure PL/SQL terminée.

Exemple SQL Server

```
BEGIN
```

```
DECLARE @i int;
```

```
SET @i = 0;
```

```
WHILE @i < 8
```

```
    BEGIN
```

```
        SET @i = @i + 1
```

```
    END
```

```
print @i
```

```
END;
```

Résultat

8

Si on souhaite sortir de la boucle, il est possible d'utiliser BREAK et CONTINUE.

```
BEGIN
```

```
DECLARE @i int;
```

```
SET @i = 0;
```

```
WHILE @i < 8
```

```
    BEGIN
```

```
        SET @i = @i + 1
```

```
        if @i = 5
```

```
            BREAK
```

```
ELSE
    CONTINUE
END
print @i
END;
```

Résultat

5

2. La boucle FOR

Comme le WHILE, le FOR permet de réaliser des boucles en précisant dès le début le nombre de fois où l'on passe dans le code. Le FOR n'existe pas en Transact SQL, mais il peut être remplacé par l'instruction WHILE.

Il faut indiquer la variable testée, la valeur de début et la valeur de fin.

Exemple

```
DECLARE
-- Déclaration du curseur C_chambres_par_type_lit
CURSOR C_chambres_par_type_lit (TypLit in varchar2) IS
SELECT Chambres.idChambre, Hotels.Libelle, Chambres.NumChambre,
TypesChambre.NombreLit, TypesChambre.Description
FROM Chambres INNER JOIN
Hotels ON Chambres.Hotel = Hotels.idHotel INNER JOIN
TypesChambre ON Chambres.TypeChambre = TypesChambre.idTypeChambre
WHERE TypeLit = typelit and Etoile = '**';

-- Déclaration des variables réceptrices
id_chambre number :=0;
Libelle_hotel varchar2(50);
Num_Chambre varchar2(6);
NbLit number(38,0);
Descript varchar2(255);

-- Déclaration des autres variables
TypLit_recherche varchar2(20) := 'lit simple';
nb_lecture number := 0;
```

BEGIN

-- Ouverture

OPEN C_chambres_par_type_lit(typlit_recherche);

-- Lecture des deux premiers éléments

FOR nb_lecture in 1..2

LOOP

FETCH C_chambres_par_type_lit

INTO id_chambre, Libelle_hotel, Num_Chambre, NbLit, descript;

EXIT WHEN C_chambres_par_type_lit%NOTFOUND;

-- Affichage des éléments récupérés

DBMS_OUTPUT.PUT_LINE('Id Chambre : ' || id_chambre);

DBMS_OUTPUT.PUT_LINE('Nom de l'hôtel : ' || Libelle_hotel);

DBMS_OUTPUT.PUT_LINE('Numéro de chambre : ' || num_chambre);

DBMS_OUTPUT.PUT_LINE('Nombre de lit : ' || nbLit);

DBMS_OUTPUT.PUT_LINE('Type de lit : ' || typlit_recherche);

DBMS_OUTPUT.PUT_LINE('Description : ' || descript);

END LOOP;

-- Fermeture du curseur (libération mémoire)

CLOSE C_chambres_par_type_lit;

END;

À noter que l'on garde quand même le test sur les lignes trouvées ou pas (Exit When C_Chambres_par_type_lit%NOTFOUND;) afin de sortir de la boucle si on arrive à la fin de la lecture avant d'avoir atteint les quatre lectures demandées.

Résultat obtenu, seuls les deux premiers éléments s'affichent :

Id Chambre : 8

Nom de l'hôtel : Art Hotel

Numéro de chambre : 1

Nombre de lit : 1

Type de lit : lit simple

Description : 1 lit simple avec douche

Id Chambre : 9

Nom de l'hôtel : Art Hotel

Numéro de chambre : 2

Nombre de lit : 2

Type de lit : lit simple

Description : 2 lits simples avec douche

Procédure PL/SQL terminée.

3. La boucle LOOP

La boucle LOOP n'a pas de condition de sortie indiquée au début, il faut à l'intérieur de la boucle mettre le mot « exit » pour sortir de la boucle. Le LOOP n'existe pas en Transact SQL, mais il peut être remplacé par l'instruction WHILE.

Nous l'avons vu précédemment avec la condition de sortie du FETCH dans une lecture de curseur :

... ..

BEGIN

-- Ouverture

OPEN C_chambres_par_type_lit(typlit_recherche);

-- boucle de lecture

LOOP

-- Récupération des éléments ligne par ligne

FETCH C_chambres_par_type_lit

INTO Libelle_hotel, Num_Chambre, NbLit, descript;

EXIT WHEN C_chambres_par_type_lit%NOTFOUND;

-- Affichage des éléments récupérés

DBMS_OUTPUT.PUT_LINE('Nom de l'hôtel : ' || Libelle_hotel);

DBMS_OUTPUT.PUT_LINE('Numéro de chambre : ' || num_chambre);

DBMS_OUTPUT.PUT_LINE('Nombre de lit : ' || nblit);

DBMS_OUTPUT.PUT_LINE('Type de lit : ' || typlit_recherche);

DBMS_OUTPUT.PUT_LINE('Description : ' || descript);

END LOOP;

...

Si on désire sortir une fois que l'identifiant de la chambre est supérieur à 9, on notera :

exit when id_chambre > 9;

4. Les structures conditionnelles CASE et IF

CASE permet de tester une variable et, en fonction des valeurs possibles, de réaliser l'action adéquate. Une seule action est vérifiée à la fois, chaque condition WHEN est exclusive.

On peut par exemple tester la valeur du type de lit à chaque ligne au lieu d'ajouter un WHERE dans le curseur. Ainsi, on lit toutes les lignes de la table mais on n'affiche que celles qui correspondent au type de lit recherché.

Exemple

```
DECLARE

-- Déclaration du curseur C_chambres_par_type_lit

CURSOR C_chambres_par_type_lit IS

SELECT Hotels.Libelle, typeschambre.typelit

FROM Chambres INNER JOIN

Hotels ON Chambres.Hotel = Hotels.idHotel INNER JOIN

TypesChambre ON Chambres.TypeChambre = TypesChambre.idTypeChambre

WHERE Etoile = '**';


-- Déclaration des variables réceptrices

Libelle_hotel varchar2(50);

TypLit typeschambre.typelit%TYPE;


-- Déclaration des autres variables

nb_ligne_litXL number :=0;

nb_ligne_litsimple number :=0;

nb_ligne_autre number :=0;


BEGIN

--O uverture

OPEN C_chambres_par_type_lit;

-- Boucle de lecture

LOOP

FETCH C_chambres_par_type_lit

INTO Libelle_hotel, typelit;

EXIT WHEN C_chambres_par_type_lit%NOTFOUND;

CASE typelit

  WHEN 'Lit XL' THEN nb_ligne_litxl := nb_ligne_litxl + 1;
```

```

WHEN 'Lit simple' THEN nb_ligne_litsimple := nb_ligne_litsimple + 1;
ELSE nb_ligne_autre := nb_ligne_autre + 1;
END CASE;
END LOOP;
-- Fermeture du curseur (libération mémoire)
CLOSE C_chambres_par_type_lit;
-- Affichage des valeurs
DBMS_OUTPUT.PUT_LINE('Nombre de chambres avec lit XL : ' || nb_ligne_litxl);
DBMS_OUTPUT.PUT_LINE('Nombre de chambres avec lit simple :
' || nb_ligne_litsimple);
DBMS_OUTPUT.PUT_LINE('Nombre de chambres avec autre lit :
' || nb_ligne_autre);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Aucune ligne trouvée dans la sélection.');
```

```

    WHEN OTHERS THEN
```

```

        DBMS_OUTPUT.PUT_LINE('Le numéro de l'erreur est : ' ||
TO_CHAR(SQLCODE));
        DBMS_OUTPUT.PUT_LINE('correspondant à : ' || TO_CHAR(SQLERRM));
END;
```

Résultats

Nombre de chambres avec lit XL : 0

Nombre de chambres avec lit simple : 0

Nombre de chambres avec autre lit : 7

Procédure PL/SQL terminée.

Dans SQL Server, il est possible d'utiliser IF.

Exemple SQL Server CASE puis IF

```

BEGIN
DECLARE @i int;
SET @i = 0;

WHILE @i < 8
```

```

BEGIN
    SET @i = @i + 1
    SELECT CASE @i
        WHEN 5 THEN 'OK'
        ELSE 'NON OK'
    END
END
END
print @i
END;

```

```

BEGIN
DECLARE @i int;
SET @i = 0;

WHILE @i < 8
BEGIN
    SET @i = @i + 1
    if @i = 5
        print 'OK'
    ELSE
        print @i
    END
END;

```

Les exceptions les plus utilisées

En dehors de l'exception « NOT_DATA_FOUND » que nous avons vue dans les exemples précédents, il existe une multitude d'autres exceptions. Nous n'allons pas les citer toutes dans ce livre mais en voici quelques-unes qui peuvent être utiles.

CURSOR_ALREADY_OPEN : le curseur est déjà ouvert. Il faut le fermer avant de le réouvrir (sqlcode --> 06511)

INVALID_NUMBER : la variable utilisée ne contient pas un numéro valide (sqlcode --> 01722)

NOT_LOGGED_ON : l'utilisateur n'est pas connecté à la base de données (sqlcode --> 01012)

TOO_MANY_ROWS : la sélection ramène plusieurs lignes alors que le select ne prévoit qu'une seule occurrence, faire un curseur (sqlcode --> 01422)

ZERO_DIVIDE : division par zéro (sqlcode --> 01476)

Pour traiter tout type d'erreur, il est préférable d'ajouter systématiquement un test de ce type afin d'afficher l'erreur au moindre problème.

```
WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.PUT_LINE( 'Le numéro de l erreur est : ' ||  
TO_CHAR( SQLCODE )) ;
```

```
    DBMS_OUTPUT.PUT_LINE( 'correspondant à : ' ||  
TO_CHAR( SQLERRM )) ;
```

La gestion des erreurs en Transact SQL

La gestion des erreurs permet d'anticiper les problèmes qui peuvent se dérouler au cours de l'exécution d'un programme.

Le principe est de tester le code dans un premier bloc avec BEGIN TRY ... END TRY puis d'intercepter l'exception avec BEGIN CATCH ... END CATCH.

Syntaxe

```
BEGIN
```

```
... ..
```

```
[BEGIN TRY
```

```
... ..
```

```
END TRY]
```

```
[BEGIN CATCH
```

```
... ..
```

```
END CATCH]
```

```
END;
```

Exemple

```
DECLARE @i int
```

```
BEGIN
```

```
BEGIN TRY
```

```
SET @i = 2
```

```
SET @i = @i / 0
```

```
END TRY
```

```
BEGIN CATCH
```

```
SELECT ERROR_NUMBER() AS ErrorNumber
```

```
, ERROR_MESSAGE() AS ErrorMessage
```

```
, ERROR_LINE() AS ErrorLine;END CATCH
```

```
END;
```

Résultat

ErrorNumber	ErrorMessage	ErrorLine
8134	Division par zéro.	5

Il est aussi possible de traiter l'exception.

```
DECLARE @i int
```

```
BEGIN
```

```
BEGIN TRY
```

```
SET @i = 2
```

```
SET @i = @i / 0
```

```
END TRY
```

```
BEGIN CATCH
```

```
IF @@ERROR = 8134
```

```
SET @i = @i / 1
```

```
print @i
```

```
END CATCH
```

```
END;
```

Résultat

2

Il est possible de lever une erreur grâce à la fonction RAISERROR(). Cette fonction accepte trois arguments (constante ou message ou variable, numéro de la gravité d'erreur, état de l'erreur).

Exemple

```
DECLARE @i int
```

```
DECLARE @f float
```

```
BEGIN
```

```
IF @i IS NULL OR @i = 0
```

```
RAISERROR('valeur non divisible à traiter', 16, 1)
```

```
SET @f = 18 / @i
```

```
END;
```

Résultat

Msg 50000, Niveau 16, État 1, Ligne 5

valeur non divisible à traiter

Création d'une procédure stockée

Lorsque l'on veut partager un morceau de code réalisé en PL/SQL, on peut l'enregistrer dans la base et ainsi le rendre accessible aux autres développeurs. Une procédure stockée est un bloc de code compilé et stocké par la base de données. Il suffit de l'appeler par son nom pour l'exécuter.

Le principal avantage de la procédure stockée, c'est qu'elle est enregistrée dans un format « exécutable », le serveur de base de données ne va pas interpréter les commandes lors de l'appel mais l'exécuter directement, d'où un gain de temps non négligeable par rapport au lancement multiple de la même requête dans un programme.

Un autre avantage de la procédure stockée est que l'on peut lui passer des paramètres.

Syntaxe Oracle

```
CREATE OR REPLACE PROCEDURE <nom procédure>
```

```
[(  
  <variable entrée 1> IN <format>,  
  <variable entrée 2> IN <format>,  
  ... ..  
  <variable sortie> OUT <format>)]
```

```
IS
```

```
BEGIN
```

```
... ..
```

```
[EXCEPTION
```

```
... ..
```

```
]
```

```
END;
```

Syntaxe SQL Server

```
CREATE OR ALTER PROCEDURE <nom procédure>
```

```
[(  
  @@<variable 1> <format>,  
  @<variable 2> <format>,  
  ... .. )]
```

```
AS
```

```
BEGIN
```

```
... ..
```

```
[BEGIN TRY
```

```
... ..
```

```
END TRY]
```

```
[BEGIN CATCH
```

```
... ..
```

```
END CATCH]
```

```
END;
```

Par exemple, la procédure suivante affiche la liste des chambres à partir de l'identifiant de l'hôtel.

```
CREATE OR REPLACE PROCEDURE LISTE_CHAMBRE_HOTEL
```

```
(nidhotel IN NUMBER)
```

```
IS
```

```
CURSOR cChambres IS
```

```
SELECT numchambre, description
```

```
FROM chambres INNER JOIN typeschambre
```

```
ON chambres.typechambre = typeschambre.idtypechambre
```

```
WHERE hotel = nidhotel;
```

```
-- Déclaration des variables internes
```

```
vnumchambre chambres.numchambre%TYPE;
```

```
vdsc typeschambre.description%TYPE;
```

```
BEGIN
```

```
DBMS_OUTPUT.PUT_LINE('** Liste des chambres **');
```

```
OPEN cChambres;
```

```
LOOP
```

```
FETCH cChambres INTO vnumchambre, vdsc;
```

```
EXIT WHEN cChambres%NOTFOUND;
```

```
DBMS_OUTPUT.PUT_LINE('Chambre N° ' || vnumchambre || ' - ' || vdsc);
```

```
END LOOP;
```

```
EXCEPTION
```

```
WHEN OTHERS THEN
```

```
DBMS_OUTPUT.PUT_LINE('Le numéro de l'erreur est :
```

```
' || TO_CHAR(SQLCODE));
```

```
DBMS_OUTPUT.PUT_LINE('correspondant à : ' || TO_CHAR(SQLERRM));
```

```
END;
```

En cas de problème de compilation signalé par le SGBDR, il est possible de visualiser, dans SQL*Plus, la dernière erreur rencontrée avec la commande SHOW ERRORS.

Appel de cette procédure :

```
EXECUTE liste_chambre_hotel(2);
```

Résultats

**** Liste des chambres ****

Chambre N° 1 - 1 lit simple avec douche

Chambre N° 2 - 2 lits simples avec douche

Chambre N° 3 - 3 lits simples avec douche et WC séparés

Chambre N° 4 - 1 lit double avec douche

Chambre N° 5 - 1 lit double avec douche et WC séparés

Chambre N° 6 - 1 lit double avec bain et WC séparés

Chambre N° 7 - 1 lit double large avec bain et WC séparés

Procédure PL/SQL terminée.

La commande pour appeler une procédure stockée est la commande EXECUTE pour Oracle et SQL Server et la commande CALL pour MySQL.

Syntaxe Oracle

EXECUTE <Nom procédure stockée> (<param 1>, <param 2>, etc ...)

Syntaxe MySQL

CALL <Nom procédure stockée> (<param 1>, <param 2>, etc ...)

Exemple SQL Server

CREATE PROCEDURE Liste_chambre_hotel (@hotel int)

AS

BEGIN

BEGIN TRY

SELECT NumChambre, Description

FROM Chambres INNER JOIN TypesChambre

ON TypesChambre.idTypeChambre = Chambres.TypeChambre

WHERE Hotel = @Hotel;

END TRY

BEGIN CATCH

SELECT ERROR_NUMBER() AS ErrorNumber,

ERROR_MESSAGE() AS ErrorMessage;

END CATCH

END;

Résultat SQL Server

execute Liste_chambre_hotel 2;

NumChambre	Description
1	1 lit simple avec douche
2	2 lits simples avec douche
3	3 lits simples avec douche et WC séparés
4	1 lit double avec douche
5	1 lit double avec douche et WC séparés
6	1 lit double avec bain et WC séparés
7	1 lit double large avec bain et WC séparés

Exemple MySQL

```

CREATE PROCEDURE Liste_chambre_hotel(IN pHotel INT)
BEGIN
    DECLARE v_numChambre VARCHAR(6);
    DECLARE v_description VARCHAR(255);
    -- Variable utilisée pour stopper la boucle
    DECLARE fin TINYINT DEFAULT 0;

    DECLARE C1 CURSOR
    FOR SELECT NumChambre, Description
    FROM Chambres INNER JOIN TypesChambre
    ON TypesChambre.idTypeChambre = Chambres.TypeChambre
    WHERE Hotel = pHotel;

    -- Gestionnaire d'erreur pour la condition NOT FOUND
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1;

    OPEN C1;
    loop_curseur: LOOP
        FETCH C1 INTO v_numChambre, v_description;
        -- Structure IF pour quitter la boucle à la fin des résultats
        IF fin = 1 THEN
            LEAVE loop_curseur;
        END IF;
        SELECT CONCAT('Chambre N° : ', v_numChambre, ' - ',

```

```

v_description) as 'Chambre';

END LOOP;

CLOSE C1;

END

call Liste_chambre_hotel(2)

```

Création d'une fonction stockée

Dans le même exemple, il est également possible de créer une fonction à la place de la procédure. La différence entre une fonction et une procédure ? La première renvoie une valeur.

Syntaxe Oracle

```

CREATE OR ALTER FUNCTION <nom fonction>

[(<variable entrée 1> IN <format>,
  <variable entrée 2> IN <format>,
  ... ... ]
  RETURN <format>

IS
<variable sortie> <format>)]

BEGIN

... ...

[EXCEPTION
... ...

]

END;

```

Syntaxe SQL Server

```

CREATE OR ALTER FUNCTION <nom fonction>

[(@<variable 1> <format>,
 @<variable 2> <format>,
 ... ... )]

RETURNS <format>

```

```
AS
```

```
BEGIN
```

```
... ..
```

```
END;
```

Par exemple, la fonction suivante ramène le prix de la chambre à partir d'une date, du nom de l'hôtel, du type et du nombre de lits.

```
CREATE OR REPLACE FUNCTION PRIX_CHAMBRE
```

```
(vhotel IN VARCHAR2, vtypelit IN VARCHAR2, inblit IN INT, ddate IN DATE)
```

```
RETURN NUMBER
```

```
IS
```

```
DPrix NUMBER;
```

```
BEGIN
```

```
SELECT Prix INTO dPrix FROM Tarifs t
```

```
INNER JOIN Hotels h ON t.hotel = h.idHotel
```

```
INNER JOIN TypesChambre tc ON tc.idTypeChambre = t.typeChambre
```

```
WHERE h.Libelle = vhotel AND tc.TypeLit = vTypeLit AND tc.NombreLit =
```

```
iNbLit
```

```
and DateDebut <= dDate and DateFin >= dDate;
```

```
RETURN (dPrix);
```

```
EXCEPTION
```

```
WHEN OTHERS THEN
```

```
DBMS_OUTPUT.PUT_LINE('Le numéro de l erreur est : ' ||
```

```
TO_CHAR( SQLCODE )) ;
```

```
DBMS_OUTPUT.PUT_LINE('correspondant à : ' ||
```

```
TO_CHAR( SQLERRM )) ;
```

```
END;
```

Cette fonction peut ensuite être utilisée directement dans un programme.

On peut l'utiliser directement dans un ordre SELECT, où les valeurs peuvent être remplacées par des champs :

```
DECLARE mPrix NUMBER;
```



```
BEGIN
```

```
SELECT PRIX_Chambre('Ski Hotel', 'lit simple', 2, '24/03/2022')
```

```
INTO mPrix FROM DUAL;
```

```
DBMS_OUTPUT.PUT_LINE('Prix de la nuit : ' || mprix);
```

```
END;
```

Résultat

Prix de la nuit : 59.99

Procédure PL/SQL terminée.

Exemple SQL Server

```
CREATE FUNCTION PRIX_Chambre (@vhotel varchar(50), @vTypeLit varchar(20),
```

```
@iNbLit int, @dDate date)
```

```
RETURNS money
```

```
AS
```

```
BEGIN
```

```
DECLARE @prix money
```

```
SELECT @prix = Prix FROM Tarifs t
```

```
INNER JOIN Hotels h ON t.hotel = h.idHotel
```

```
INNER JOIN TypesChambre tc ON tc.idTypeChambre = t.typeChambre
```

```
WHERE h.Libelle = @vhotel AND tc.TypeLit = @vTypeLit AND tc.NombreLit =
```

```
@iNbLit and DateDebut <= @dDate and DateFin >= @dDate
```

```
RETURN @prix
```

```
END;
```

Appel de la fonction et résultat SQL Server

```
BEGIN
```

```
DECLARE @mPrix money;
```

```
EXEC @mPrix = PRIX_Chambre 'Ski Hotel', 'lit simple', 2, '24/03/2022'
```

```
Print @mPrix
```

```
Print 'Prix de la nuit : ' + CONVERT(varchar(6), @mPrix)
```

```
END;
```

59.99

Prix de la nuit : 59.99

Exemple MySQL

```
CREATE FUNCTION LibelleHotel(idHotel INT) RETURNS VARCHAR(50)
```

```
DETERMINISTIC
```

```
BEGIN
```

```
DECLARE vLibelle VARCHAR(50);
```

```
SELECT libelle INTO vLibelle FROM Hotels
```

```
WHERE idhotel = idHotel;
```

```
RETURN vLibelle;
```

```
END
```

```
SELECT LibelleHotel(1);
```

Exemple PostgreSQL

```
CREATE OR REPLACE FUNCTION PRIX_Chambre (vhotel VARCHAR, vTypeLit
```

```
VARCHAR, iNbLit INT, dDate DATE)
```

```
RETURNS money AS
```

```
$$
```

```
DECLARE mPrix money;
```

```
BEGIN
```

```
    SELECT Prix INTO mPrix FROM Tarifs t
```

```
INNER JOIN Hotels h ON t.hotel = h.idHotel
```

```
INNER JOIN TypesChambre tc ON tc.idTypeChambre = t.typeChambre
```

```
WHERE h.Libelle = $1 AND tc.TypeLit = $2 AND tc.NombreLit = $3 and
```

```
DateDebut <= $4 and DateFin >= $4;
```

```
    RETURN mPrix;
```

```
END;
```

```
$$
```

```
LANGUAGE plpgsql;
```

```
SELECT PRIX_Chambre ('Ski Hotel', 'lit simple', 2, '24/03/2022');
```

Les packages

L'appellation « package » signifie que l'on regroupe sous un même nom toutes les procédures et fonctions sur le même thème, on peut ainsi créer de véritables applications.

Dans un package, on peut avoir des déclarations de variables publiques ou privées. Des fonctions et procédures privées non visibles de l'extérieur.

Dans un package, il faut créer une partie déclaration et une partie contenant les fonctions et procédures.

Dans la partie déclaration, on liste les procédures et fonctions qui sont décrites dans l'autre partie. Toutes les fonctions ou procédures qui sont déclarées à ce niveau sont dites « publiques ». Pour les variables, c'est le même fonctionnement, si elles sont dans la partie déclaration, elles sont « publiques ».

Les packages n'existent pas pour SQL Server.

Syntaxe

```
CREATE OR REPLACE PACKAGE <nom package> IS
```

```
    PROCEDURE <nom procédure 1>;
```

```
    FUNCTION <nom fonction 1> (<variabl 1> IN <format>) RETURN
```

```
<format>; END;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY <nom package> IS
```

```
    FUNCTION <fonction 1>
```

```
    ... ..
```

```
    END;
```

```
    PROCEDURE <procédure 1> IS
```

```
    ... ..
```

```
    END;
```

```
END;
```

```
/
```

On peut regrouper la fonction FN_PRIX et la procédure d'affichage des chambres d'hôtel (que l'on peut nommer LST_CHAMBRES) et créer un package nommé AFFICHAGE_HOTEL.

```
CREATE OR REPLACE PACKAGE AFFICHAGE_HOTEL AS
```

```
/* DETAILS FONCTION */
```

```
FUNCTION FN_PRIX (nhotel IN NUMBER, ntypeChambre IN NUMBER, ddate  
IN DATE) RETURN NUMBER;
```

```
/* DETAILS PROCEDURE */
```

```
PROCEDURE LST_CHAMBRES (nidhotel IN NUMBER, ddate IN DATE);
```

END AFFICHAGE_HOTEL;

/

CREATE OR REPLACE PACKAGE BODY AFFICHAGE_HOTEL AS

FUNCTION FN_PRIX (nhotel IN NUMBER, ntypeChambre IN NUMBER, ddate
IN DATE) RETURN NUMBER

IS

/* DECLARATION VARIABLES */

dPrix NUMBER;

BEGIN

/* CORPS FONCTION*/

SELECT Prix INTO dPrix FROM Tarifs

WHERE hotel = nhotel AND typeChambre = ntypeChambre
and DateDebut <= dDate and DateFin >= dDate;

RETURN (dPrix);

EXCEPTION

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('Le numéro de l erreur est : ' ||
TO_CHAR(SQLCODE)) ;

DBMS_OUTPUT.PUT_LINE('correspondant à : ' ||
TO_CHAR(SQLERRM)) ;

END FN_PRIX;

PROCEDURE LST_CHAMBRES (nidhotel IN NUMBER, ddate IN DATE)

IS

/* DECLARATION VARIABLES */

CURSOR cChambres IS

SELECT h.libelle, c.numchambre, TEST(h.idhotel, tc.idtypechambre,
ddate) as prix, tc.description

FROM hotels h INNER JOIN chambres c ON c.hotel = h.idhotel

```
INNER JOIN typeschambre tc ON c.typechambre = tc.idtypechambre
WHERE h.idhotel = nidhotel;

vlibelle hotels.libelle%TYPE;
vnumchambre chambres.numchambre%TYPE;
nprix NUMBER;
vdesc typeschambre.description%TYPE;
```

```
BEGIN
/* CORPS PROCEDURE*/

DBMS_OUTPUT.PUT_LINE('** Liste des chambres **');
OPEN cChambres;
LOOP
FETCH cChambres INTO vlibelle, vnumchambre, nprix, vdesc;
EXIT WHEN cChambres%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Chambre N° ' || vnumchambre || ' - ' || vdesc);
DBMS_OUTPUT.PUT_LINE('Prix : ' || nprix);
END LOOP;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Le numéro de l'erreur est :
' || TO_CHAR(SQLCODE));
DBMS_OUTPUT.PUT_LINE('correspondant à : ' || TO_CHAR(SQLERRM));
END LST_CHAMBRES;

END AFFICHAGE_HOTEL;
/
```

Pour pouvoir ensuite utiliser les fonctions du package, il faut utiliser la syntaxe suivante :
<nomdupackage.nomfonction>.

```
BEGIN
AFFICHAGE_HOTEL.LST_CHAMBRES(1, '24/03/2022');
END;
```

Compilation d'une procédure, d'une fonction ou d'un package

Syntaxe

ALTER <'PROCEDURE' ou 'FUNCTION' ou 'PACKAGE'> <Nom procédure ou fonction ou package> COMPILE;

Exemple

ALTER FUNCTION PRIX_CHAMBRE COMPILE;

ALTER PROCEDURE LISTE_CHAMBRE_HOTEL COMPILE;

ALTER PACKAGE AFFICHAGE_HOTEL COMPILE PACKAGE;

ALTER PACKAGE AFFICHAGE_HOTEL COMPILE BODY;

- compile body et package

ALTER PACKAGE AFFICHAGE_HOTEL COMPILE;

Suppression d'une procédure, d'une fonction ou d'un package

Syntaxe

DROP <'PROCEDURE' ou 'FUNCTION' ou 'PACKAGE'> <Nom procédure ou fonction ou package>;

Exemple

DROP FUNCTION PRIX_CHAMBRE;

DROP PROCEDURE LISTE_CHAMBRE;

- suppression de tout le package (corps et déclaration)

DROP PACKAGE AFFICHAGE_HOTEL;

- suppression corps de package

DROP PACKAGE BODY AFFICHAGE_HOTEL;

Les déclencheurs

Un déclencheur ou trigger en anglais permet de lancer des commandes qui vont s'exécuter à chaque fois qu'un événement se produit sur une table.

Le contenu du code lancé par un trigger est souvent du PL/SQL ou du C ou du Java.

Les déclencheurs sont souvent utilisés pour gérer l'intégrité fonctionnelle d'une application. Ils permettent de réaliser des contrôles sur le contenu des tables en automatique. Les déclencheurs peuvent également servir à récupérer des informations tout au long d'une journée sur les activités de la base de données, ces données étant traitées ensuite par une autre application.

En général, on code les contrôles dans les programmes applicatifs exécutés côté client. Les déclencheurs permettent d'ajouter d'autres contrôles qui seront exécutés côté serveur.

L'avantage premier du déclencheur est qu'il est lié à une action sur la base (INSERT, UPDATE, DELETE), donc on ne risque pas d'oublier de modifier un programme. En effet, il est souvent compliqué de modifier tous les programmes d'un applicatif pour ajouter un contrôle sur un INSERT par exemple. Il faudra retrouver tous les programmes concernés, les modifier, et tester chacun des programmes modifiés.

Le déclencheur se déclenchant systématiquement, on ne peut pas oublier une mise à jour, et la modification se fait indépendamment des programmes applicatifs.

Un déclencheur peut se déclencher avant ou après l'ordre SQL demandé. On l'indique par AFTER ou BEFORE. Dans SQL Server, il s'exécute après ou à la place (INSTEAD OFF).

Dans un trigger BEFORE, on peut contrôler avant toute modification de la base certains éléments et empêcher ainsi les mises à jour.

Dans un trigger AFTER, la mise à jour a eu lieu et on déclenche les actions qui en découlent.

Syntaxe générale d'un déclencheur PL/SQL

```
CREATE OR REPLACE TRIGGER <nom du trigger>
[BEFORE ou AFTER] [INSERT ou DELETE ou UPDATE]
ON <nom de table>
[FOR EACH ROW]
[WHEN]

DECLARE

... ..

BEGIN

... ..

END;
```

La clause FOR EACH ROW signifie que le trigger agit sur toutes les lignes affectées par l'ordre SQL.

La clause WHEN permet d'ajouter un critère supplémentaire qui s'applique aux lignes sélectionnées, par exemple si on veut interdire les suppressions uniquement sur les enregistrements qui ont un identifiant supérieur à une certaine valeur.

Syntaxe générale d'un déclencheur Transact SQL

```
CREATE ou ALTER TRIGGER <nom du trigger>
ON TABLE ou VIEW
[BEFORE ou AFTER] [INSERT ou DELETE ou UPDATE]
ON <nom de table>
(FOR | AFTER | INSTEAD OF )
{ [ DELETE ] [, ] [ INSERT ] [, ] [ UPDATE ] }
AS

DECLARE

... ..

BEGIN

... ..

END;
```

On retrouve la même syntaxe que pour les procédures stockées à partir du DECLARE (cf. chapitre La programmation - Création d'une procédure stockée).

Prenons pour exemple un DELETE détruisant plusieurs lignes. Dans ce cas, le déclencheur s'appliquera à toutes les lignes détruites. Dans le cas contraire, le déclencheur s'active avant ou après la destruction des lignes.

Si on veut empêcher les utilisateurs de supprimer une ligne dans la table HOTELS, on écrit :

```
CREATE OR REPLACE TRIGGER T_SUP_HOTEL
BEFORE DELETE
ON HOTELS
FOR EACH ROW
DECLARE

BEGIN

    DBMS_OUTPUT.PUT_LINE( 'Il vous est interdit de supprimer un
hôtel' ) ;

    DBMS_OUTPUT.PUT_LINE( 'Veuillez contacter votre
administrateur' );

    RAISE_APPLICATION_ERROR(-20502, 'Commande non autorisée');

END;
```


Si maintenant on essaye de supprimer une ligne dans la table HOTELS :

Déclencheur créé.

```
SQL> DELETE from hotels where idhotel = 1;
```

Il vous est interdit de supprimer un hôtel

Veuillez contacter votre administrateur

Erreur commençant à la ligne: 1 de la commande -

```
DELETE FROM HOTELS
```

```
WHERE idhotel = 1
```

Rapport d'erreur -

ORA-20502: **Commande non autorisée**

ORA-06512: à "xxxxx.T_SUP_HOTEL", ligne 9

ORA-04088: erreur lors d'exécution du déclencheur 'xxxxx.T_SUP_HOTEL'

Les messages prévus s'affichent ("Il vous est interdit...") puis l'appel au module d'erreur (RAISE_APPLICATION_ERROR) arrête le traitement en indiquant le nom du déclencheur qui a provoqué l'erreur (xxxx.T_SUP_HOTEL).

Cet exemple ne sera pas géré par un déclencheur dans SQL Server mais via la gestion des droits en interdisant la suppression.

RAISE_APPLICATION_ERROR est une procédure stockée fournie par Oracle et qui permet de créer ses propres messages d'erreur. Il faut utiliser les codes erreurs compris entre -20000 et -20999 qui ne sont pas utilisés par Oracle.

1. Création d'un déclencheur de contrôle et mise à jour dans une table

Si on pousse un peu plus loin, il est possible par exemple à l'insertion dans une table de vérifier une valeur dans une autre table et, en fonction du résultat, d'accepter ou de refuser l'insertion.

Par exemple, lors de la création d'une ligne dans la table CHAMBRES, on déclenche une vérification du type de chambre : il faut que celui-ci soit présent dans la table TYPESCHAMBRE.

Si la nationalité est inconnue, il faut la créer dans la table PAYS, avec comme libellé : « A COMPLETER ».

Exemple de script PL/SQL

```
CREATE OR REPLACE TRIGGER T_INS_CHAMBRE
```

```
BEFORE INSERT
```

```
ON CHAMBRES
```

```
FOR EACH ROW
```

```
DECLARE
```

```
    nbvaleur NUMBER(5);
```

```
BEGIN
```

```
    nbvaleur := 0;    SELECT COUNT(1) INTO nbvaleur FROM typeschambre
```

```

WHERE idtypechambre = :new.typechambre;

IF (nbvaleur = 0) THEN
    DBMS_OUTPUT.PUT_LINE( 'Le type de chambre est inconnu :
'| | :new.typechambre) ;
    DBMS_OUTPUT.PUT_LINE( 'Une insertion dans la table TYPESCHAMBRE
va être réalisée');
    DBMS_OUTPUT.PUT_LINE( 'Merci de compléter les informations de ce
nouveau type de chambre ultérieurement');
    INSERT INTO TYPESCHAMBRE VALUES (:new.typechambre,0, 'A
COMPLETER', 'A COMPLETER');
END IF;
END;

```

Exemple de script Transact SQL

```

CREATE TRIGGER T_INS_Chambres
ON Chambres
AFTER INSERT
AS
BEGIN
    DECLARE @nbvaleur int
    DECLARE @idtypechambre int
    SET @nbvaleur = 1;

    SELECT @idtypechambre = TypeChambre FROM inserted

    SELECT @nbvaleur = COUNT(1) FROM TypesChambre WHERE idTypeChambre =
@idtypechambre

    if @nbvaleur = 0
        BEGIN
            print 'Le type de chambre est inconnu : ' + CONVERT(varchar(3),
@idtypechambre)
            print 'Une insertion dans la table TypesChambre va être réalisée'

```

```
print 'Merci de compléter les informations de ce nouveau type de
chambre ultérieurement'
```

```
INSERT INTO TypesChambre VALUES (@idtypechambre,0, 'A COMPLETER',
'A COMPLETER')
END
END;
```

Exemple de script PostgreSQL

```
CREATE FUNCTION T_INS_Chambres() RETURNS trigger AS $T_INS_Chambres$
DECLARE nbvaleur int = 1;
BEGIN
    SELECT COUNT(*) INTO nbvaleur FROM TypesChambre WHERE idTypeChambre
= NEW.TypeChambre;
    IF (nbvaleur = 0) THEN
        INSERT INTO TypesChambre VALUES (NEW.TypeChambre,0,
'A COMPLETER', 'A COMPLETER');
    END IF;

    RETURN NULL;
END;
$T_INS_Chambres$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER T_INS_Chambres AFTER INSERT OR UPDATE ON Chambres
FOR EACH ROW EXECUTE PROCEDURE T_INS_Chambres();
```

Exemple de script MySQL

```
CREATE OR REPLACE TRIGGER T_INS_CHAMBRE
AFTER INSERT
ON CHAMBRES
FOR EACH ROW
BEGIN
    DECLARE nbvaleur integer;

    SELECT COUNT(*) INTO nbvaleur FROM typeschambre WHERE idtypechambre =
new.typechambre;
```

```

IF nbvaleur = 0 THEN
    SELECT 'Le type de chambre est inconnu';
    SELECT 'Une insertion dans la table TypesChambre va être
réalisée';
    SELECT 'Merci de compléter les informations de ce nouveau type
de chambre ultérieurement';
    INSERT INTO TYPESCHAMBRE VALUES (new.typechambre,0, 'A COMPLETER',
'A COMPLETER');
END IF;
END;

```

Avant de tester l'insertion, vérifier si une clé étrangère existe pour la supprimer ainsi :

```
ALTER TABLE Chambres DROP CONSTRAINT FK_Chambres_TypesChambre
```

Voici le résultat lorsqu'on essaye d'insérer une nouvelle chambre (PL/SQL ou TSQL) :

```
INSERT INTO Chambres VALUES (7, 1, 8, 7, 'chambre neuve');
```

Le type de chambre est inconnu : 8

Une insertion dans la table TypesChambre va être réalisée

Merci de compléter les informations de ce nouveau type de chambre

ultérieurement

Les messages prévus dans le trigger s'affichent. Vérifions dans la table TYPESCHAMBRE si une ligne a été ajoutée.

```
SELECT * FROM typeschambre;
```

```
6  1 lit double      1 lit double avec bain et WC séparés
```

```
7  1 lit XL          1 lit double large avec bain et WC séparés
```

8 0 A COMPLETER A COMPLETER

Dernière vérification, pour contrôler que la chambre a bien été créée également :

```
SELECT * FROM Chambres WHERE hotel = 1;
```

```
6      1      6 6
```

```
7      1      8 7  chambre neuve
```

Si on veut utiliser les valeurs de la table CHAMBRES qui vont être modifiées, il faut utiliser les variables spéciales appelées **new** et **old** pour Oracle et PostgreSQL.

new.<colonne> contient les nouvelles valeurs qui vont être prises en compte et old.<colonne> contient les anciennes valeurs de la table CHAMBRES avant modification.

Pour SQL Server, on utilise **inserted** et **deleted** comme une table.

Si maintenant on ne veut appliquer le trigger que pour l'hôtel n° 1, il faut ajouter une clause WHEN ainsi :

```
TRIGGER T_INS_CHAMBRE
BEFORE INSERT
ON CHAMBRES
FOR EACH ROW
WHEN (old.hotel = 1)
DECLARE
    nbvaleur NUMBER(5);
BEGIN
    nbvaleur := 0;
    SELECT COUNT(1) INTO nbvaleur FROM typeschambre WHERE idtypechambre =
:new.typechambre;

    IF (nbvaleur = 0) THEN
        DBMS_OUTPUT.PUT_LINE( 'Le type de chambre est inconnu :
'| | :new.typechambre) ;
        DBMS_OUTPUT.PUT_LINE( 'Une insertion dans la table TYPESCHAMBRE va être
réalisée');
        DBMS_OUTPUT.PUT_LINE( 'Merci de compléter les informations de ce nouveau
type de chambre ultérieurement');
        INSERT INTO TYPESCHAMBRE VALUES (:new.typechambre,0, 'A COMPLETER',
'A COMPLETER');
    END IF;
END;
```

À noter ici que dans le cas où on ajoute un contrôle sur une colonne de la table dans la clause WHEN, il ne faut pas la préfixer par '?':

2. Création d'un déclencheur de suivi des mises à jour

Prenons l'exemple suivant : on souhaite enregistrer dans une table annexe le nombre de mises à jour réalisées dans la journée sur les tables CHAMBRES, TARIFS et TYPESCHAMBRE.

Descriptif de la table de suivi des mises à jour

Table SUIV_MAJ

SUIV_MAJ

DATE_JOUR

NB_CHAMBRES_AJOUTES

NB_CHAMBRES_MODIFIES

NB_CHAMBRES_SUPPRIMES

NB_TARIFS_AJOUTES

NB_TARIFS_MODIFIES

NB_TARIFS_SUPPRIMES

NB_TYPESCHAMBRE_AJOUTES

NB_TYPESCHAMBRE_MODIFIES

NB_TYPESCHAMBRE_SUPPRIMES

Script de création

```
CREATE TABLE SUIV_MAJ(  
DATE_JOUR DATE  
, NB_CHAMBRES_AJOUTES INT  
, NB_CHAMBRES_MODIFIES INT  
, NB_CHAMBRES_SUPPRIMES INT  
, NB_TARIFS_AJOUTES INT  
, NB_TARIFS_MODIFIES INT  
, NB_TARIFS_SUPPRIMES INT  
, NB_TYPESCHAMBRE_AJOUTES INT  
, NB_TYPESCHAMBRE_MODIFIES INT  
, NB_TYPESCHAMBRE_SUPPRIMES INT  
);
```

Initialisation de la table

Oracle

```
INSERT INTO SUIV_MAJ VALUES (SYSDATE,0,0,0,0,0,0,0,0,0);
```

SQL Server

```
INSERT INTO SUIV_MAJ VALUES (GETDATE(),0,0,0,0,0,0,0,0,0);
```

Création du trigger sur la table CHAMBRES

```
CREATE OR REPLACE TRIGGER MAJ_CHAMBRES  
AFTER INSERT OR DELETE OR UPDATE
```

```

ON CHAMBRES
FOR EACH ROW
DECLARE
BEGIN
    IF INSERTING THEN
        UPDATE SUIV_MAJ SET
            NB_CHAMBRES_AJOUTES = NB_CHAMBRES_AJOUTES + 1
        WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
            TO_CHAR(SYSDATE,'DD/MM/YYYY');
    END IF;
    IF UPDATING THEN
        UPDATE SUIV_MAJ SET
            NB_CHAMBRES_MODIFIES = NB_CHAMBRES_MODIFIES + 1
        WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
            TO_CHAR(SYSDATE,'DD/MM/YYYY');
    END IF;
    IF DELETING THEN
        UPDATE SUIV_MAJ SET
            NB_CHAMBRES_SUPPRIMES = NB_CHAMBRES_SUPPRIMES + 1
        WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
            TO_CHAR(SYSDATE,'DD/MM/YYYY');
    END IF;
END;

```

Création du trigger sur la table TARIFS

```

CREATE OR REPLACE TRIGGER MAJ_TARIFS
AFTER INSERT OR DELETE OR UPDATE
ON TARIFS
FOR EACH ROW
DECLARE
BEGIN
    IF INSERTING THEN
        UPDATE SUIV_MAJ SET
            NB_TARIFS_AJOUTES = NB_TARIFS_AJOUTES + 1

```

```

WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
    TO_CHAR(SYSDATE,'DD/MM/YYYY');
END IF;
IF UPDATING THEN
    UPDATE SUIV_MAJ SET
        NB_TARIFS_MODIFIES = NB_TARIFS_MODIFIES + 1
    WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
        TO_CHAR(SYSDATE,'DD/MM/YYYY');
END IF;
IF DELETING THEN
    UPDATE SUIV_MAJ SET
        NB_TARIFS_SUPPRIMES = NB_TARIFS_SUPPRIMES + 1
    WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
        TO_CHAR(SYSDATE,'DD/MM/YYYY');
END IF;
END;

```

Création du trigger sur la table TYPESCHAMBRE

```

CREATE OR REPLACE TRIGGER MAJ_TYPESCHAMBRE
AFTER INSERT OR DELETE OR UPDATE
ON TYPESCHAMBRE
FOR EACH ROW
DECLARE
BEGIN
    IF INSERTING THEN
        UPDATE SUIV_MAJ SET
            NB_TYPESCHAMBRE_AJOUTES = NB_TYPESCHAMBRE_AJOUTES + 1
        WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
            TO_CHAR(SYSDATE,'DD/MM/YYYY');
    END IF;
    IF UPDATING THEN
        UPDATE SUIV_MAJ SET
            NB_TYPESCHAMBRE_MODIFIES = NB_TYPESCHAMBRE_MODIFIES + 1
        WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =

```



```

        TO_CHAR(SYSDATE,'DD/MM/YYYY');
END IF;
IF DELETING THEN
    UPDATE SUIV_MAJ SET
        NB_TYPESCHAMBRE_SUPPRIMES = NB_TYPESCHAMBRE_SUPPRIMES + 1
    WHERE TO_CHAR(DATE_JOUR,'DD/MM/YYYY') =
        TO_CHAR(SYSDATE,'DD/MM/YYYY');
END IF;
END;

```

Pour chaque trigger indiqué ci-dessus, les actions s'appliquent sur tous les ordres grâce à l'ordre **AFTER INSERT OR DELETE OR UPDATE**.

Les triggers se déclenchent après les mises à jour afin de ne prendre en compte que les mises à jour effectives.

Maintenant, on exécute quelques commandes sur ces tables :

```

DELETE FROM TARIFS;
DELETE FROM CHAMBRES WHERE IDCHAMBRE = 7;
INSERT INTO TYPESCHAMBRE VALUES (9,2,'lit double','2 lits double avec
douche');
INSERT INTO TYPESCHAMBRE VALUES (10,2,'lit double','2 lits double avec
douche et WC séparés');
INSERT INTO TYPESCHAMBRE VALUES (11,2,'lit double','2 lits double avec bain
et WC séparés');
INSERT INTO TYPESCHAMBRE VALUES (12,2,'lit double','1 lit double et 1 lit
simple avec douche');
UPDATE TYPESCHAMBRE SET NOMBRELIT = 2, TYPELIT = 'lit XL', DESCRIPTION =
'1 lit XL et 1 lit simple avec bain' WHERE IDTYPECHAMBRE = 8;
Ce qui a pour effet d'alimenter automatiquement la table SUIV_MAJ :
SELECT * FROM SUIV_MAJ;

```

DATE_JOUR	05/09/20
NB_CHAMBRES_AJOUTES	0
NB_CHAMBRES_MODIFIES	0
NB_CHAMBRES_SUPPRIMES	1
NB_TARIFS_AJOUTES	0
NB_TARIFS_MODIFIES	0

DATE_JOUR	05/09/20
NB_TARIFS_SUPPRIMES	56
NB_TYPERCHAMBRE_AJOUTES	4
NB_TYPERCHAMBRE_MODIFIES	1
NB_TYPERCHAMBRE_SUPPRIMES	0

À noter que le TRUNCATE n'est pas considéré comme un DELETE donc n'a aucun effet sur un trigger ON DELETE.

Pour SQL Server, il est tout à fait possible de créer les déclencheurs correspondants, mais à partir de la version 2012, il est possible de gérer l'historisation avec les outils décisionnels (SSAS).

À partir de la version 2016, il existe une fonctionnalité (table temporelle) à activer par les administrateurs de la base de données qui crée un historique sans avoir à créer des triggers.

Voici les scripts pour SQL Server :

```
CREATE TRIGGER MAJ_Chambres
ON Chambres
AFTER INSERT, UPDATE, DELETE
AS
BEGIN

DECLARE @i int
DECLARE @d int

SET @i = 0;
SET @d = 0;

SELECT @i = COUNT(1) FROM inserted;
SELECT @d = COUNT(1) FROM deleted;
-- S'il existe des lignes à la fois dans la table inserted et deleted
alors les lignes sont modifiées
-- Utiliser les valeurs des variables pour un résultat fiable
if @i > 0 AND @d > 0
BEGIN
    UPDATE SUIV_MAJ SET NB_CHAMBRES_MODIFIES = NB_CHAMBRES_MODIFIES + @i
    WHERE date_jour = CAST(getdate() AS date);
END;
```

/* S'il existe des lignes dans la table inserted mais pas dans la table
deleted alors il s'agit d'un ajout.

Il est important de tester les 2 critères pour ne pas comptabiliser
les lignes modifiées*/

if @i > 0 AND @d = 0

BEGIN

UPDATE SUIV_MAJ SET NB_CHAMBRES_AJOUTES = NB_CHAMBRES_AJOUTES + @i

WHERE date_jour = CAST(getdate() AS date);

END;

/* S'il n'existe pas de ligne dans la table inserted mais dans la table
deleted alors il s'agit d'une suppression.

Il est important de tester les 2 critères pour ne pas comptabiliser

les lignes modifiées*/

if @i = 0 AND @d > 0

BEGIN

UPDATE SUIV_MAJ SET NB_CHAMBRES_SUPPRIMES = NB_CHAMBRES_SUPPRIMES + @d

WHERE date_jour = CAST(getdate() AS date);

END;

END;

CREATE TRIGGER MAJ_TARIFS

ON TARIFS

AFTER INSERT, UPDATE, DELETE

AS

BEGIN

DECLARE @i int

DECLARE @d int

SET @i = 0;

SET @d = 0;

```

SELECT @i = COUNT(1) FROM inserted;
SELECT @d = COUNT(1) FROM deleted;

-- S'il existe des lignes à la fois dans la table inserted et deleted
alors les lignes sont modifiées

-- Utiliser les valeurs des variables pour un résultat fiable
if @i > 0 AND @d > 0
BEGIN
    UPDATE SUIV_MAJ SET NB_TARIFS_MODIFIES = NB_TARIFS_MODIFIES + @i
    WHERE date_jour = CAST(getdate() AS date);
END;

/* S'il existe des lignes dans la table inserted mais pas dans la table
deleted alors il s'agit d'un ajout.

Il est important de tester les 2 critères pour ne pas comptabiliser
les lignes modifiées*/
if @i > 0 AND @d = 0
BEGIN
    UPDATE SUIV_MAJ SET NB_TARIFS_AJOUTES = NB_TARIFS_AJOUTES + @i
    WHERE date_jour = CAST(getdate() AS date);
END;

/* S'il n'existe pas de ligne dans la table inserted mais dans la table
deleted alors il s'agit d'une suppression.

Il est important de tester les 2 critères pour ne pas comptabiliser
les lignes modifiées*/
if @i = 0 AND @d > 0
BEGIN
    UPDATE SUIV_MAJ SET NB_TARIFS_SUPPRIMES = NB_TARIFS_SUPPRIMES + @d
    WHERE date_jour = CAST(getdate() AS date);
END;

END;

```

```
CREATE TRIGGER MAJ_TYPESCHAMBRE
ON TYPESCHAMBRE
AFTER INSERT, UPDATE, DELETE
AS
BEGIN

DECLARE @i int
DECLARE @d int

SET @i = 0;
SET @d = 0;
SELECT @i = COUNT(1) FROM inserted;
SELECT @d = COUNT(1) FROM deleted;

-- S'il existe des lignes à la fois dans la table inserted et deleted
alors les lignes sont modifiées
-- Utiliser les valeurs des variables pour un résultat fiable
if @i > 0 AND @d > 0
BEGIN
    UPDATE SUIV_MAJ SET NB_TYPESCHAMBRE_MODIFIES = NB_TYPESCHAMBRE_MODIFIES
+ @i
    WHERE date_jour = CAST(getdate() AS date);
END;

/* S'il existe des lignes dans la table inserted mais pas dans la table
deleted alors il s'agit d'un ajout.
Il est important de tester les 2 critères pour ne pas comptabiliser les
lignes modifiées*/
if @i > 0 AND @d = 0
BEGIN
    UPDATE SUIV_MAJ SET NB_TYPESCHAMBRE_AJOUTES = NB_TYPESCHAMBRE_AJOUTES +
    @i
    WHERE date_jour = CAST(getdate() AS date);
END;
```

```
/* S'il n'existe pas de ligne dans la table inserted mais dans la table
```

```
deleted alors il s'agit d'une suppression.
```

```
Il est important de tester les 2 critères pour ne pas
```

```
comptabiliser les lignes modifiées*/
```

```
if @i = 0 AND @d > 0
```

```
BEGIN
```

```
    UPDATE SUIV_MAJ SET NB_TYPESCHAMBRE_SUPPRIMES =
```

```
NB_TYPESCHAMBRE_SUPPRIMES + @d
```

```
    WHERE date_jour = CAST(getdate() AS date);
```

```
END;
```

```
END;
```

Exercices

Premier exercice

Créer une fonction qui calcule l'âge d'un acteur.

Deuxième exercice

Créer une procédure qui affiche la liste des films avec pour chaque film les informations précédées de la description de cette information :

- Titre du film :
- Date de sortie :
- Réalisateur :
- Nom et prénom de l'acteur :
- Date de naissance :
- Age :
- Nombre de films :
- Budget du film :
- Nombre d'entrées :

Troisième exercice

Créer un déclencheur qui, lors de la création d'un acteur, vérifie l'existence de la nationalité. Si celle-ci est inconnue, le code est créé avec le libellé « A COMPLETER ».

Solutions des exercices

Premier exercice

```
CREATE OR REPLACE FUNCTION CALCUL_AGE_ACTEUR
(
  DATE_NAISSANCE IN DATE
) RETURN NUMBER
IS
  AGE_ACTEUR NUMBER(5);
BEGIN

  SELECT (SYSDATE - DATE_NAISSANCE)/365 INTO AGE_ACTEUR FROM DUAL;

  RETURN (AGE_ACTEUR);

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Le numéro de l erreur est : ' ||
      TO_CHAR( SQLCODE )) ;
    DBMS_OUTPUT.PUT_LINE('correspondant à : ' ||
      TO_CHAR( SQLERRM )) ;
END;
```

Deuxième exercice

```
CREATE PROCEDURE LISTE_FILM IS
  -- Déclaration du curseur C_FILMS
  CURSOR C_FILMS IS
    SELECT FILM.TITRE, FILM.DATE_SORTIE,
      REAL.NOM || ' ' || REAL.PRENOM REALISATEUR, ACTEUR.NOM NOM,
      ACTEUR.PRENOM PRENOM, ACTEUR.DATE_NAISSANCE,
      ACTEUR.NB_FILM, STAT.BUDGET, STAT.NB_ENTREE_FRANCE ENTREES,
      CALCUL_AGE_ACTEUR(ACTEUR.DATE_NAISSANCE)
    FROM FILM FILM, REALISATEUR REAL, CASTING CAST,
      ACTEUR ACTEUR, STATISTIQUE STAT
  WHERE
    FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND
    FILM.IDENT_FILM = CAST.IDENT_FILM AND
```

```
FILM.IDENT_FILM      = STAT.IDENT_FILM AND  
CAST.IDENT_ACTEUR    = ACTEUR.IDENT_ACTEUR  
ORDER BY FILM.TITRE;
```

```
/* Déclaration des variables réceptrices
```

```
----- */
```

```
titre_film VARCHAR2(100);
```

```
date_film DATE;
```

```
realisateur VARCHAR2(100);
```

```
nom_acteur VARCHAR2(100);
```

```
prenom_acteur VARCHAR2(100);
```

```
date_naissance DATE;
```

```
nombre_films NUMBER(8);
```

```
budget DECIMAL(10,2);
```

```
nombre_entrees NUMBER(8);
```

```
/* Déclaration des autres variables
```

```
----- */
```

```
age_acteur NUMBER(5);
```

```
BEGIN
```

```
-- Ouverture
```

```
OPEN C_FILMS;
```

```
-- Boucle de lecture
```

```
LOOP
```

```
-- Récupération des éléments ligne par ligne
```

```
FETCH C_FILMS INTO
```

```
titre_film,date_film,
```

```
realisateur,nom_acteur,
```

```
prenom_acteur ,date_naissance ,
```

```
nombre_films,budget,nombre_entrees,age_acteur;
```

```
Exit When C_FILMS%NOTFOUND;
```

```
-- Affichage des éléments récupérés
```



```

DBMS_OUTPUT.PUT_LINE( 'Titre du film   : ' || titre_film);
DBMS_OUTPUT.PUT_LINE( 'Date de sortie  : ' || date_film);
DBMS_OUTPUT.PUT_LINE( 'Realisateur    : ' || realisateur);
DBMS_OUTPUT.PUT_LINE( 'Nom acteur     : ' || nom_acteur);
DBMS_OUTPUT.PUT_LINE( 'Prenom acteur   : ' || prenom_acteur);
DBMS_OUTPUT.PUT_LINE( 'Date de naissance : ' || date_naissance);
DBMS_OUTPUT.PUT_LINE( 'Age de l acteur  : ' || age_acteur);
DBMS_OUTPUT.PUT_LINE( 'Nombre de films : ' || nombre_films);
DBMS_OUTPUT.PUT_LINE( 'Budget du film  : ' || budget);
DBMS_OUTPUT.PUT_LINE( 'Nombre d entrees : ' || nombre_entrees);
DBMS_OUTPUT.PUT_LINE( '----- ');
END LOOP;

-- Fermeture du curseur (libération mémoire)
CLOSE C_FILMS;

```

EXCEPTION

```

    WHEN NO_DATA_FOUND THEN

        DBMS_OUTPUT.PUT_LINE( 'Aucune ligne trouvée' );

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE( 'Le numéro de l erreur est : ' ||
TO_CHAR( SQLCODE )) ;

        DBMS_OUTPUT.PUT_LINE( 'correspondant à : ' ||
TO_CHAR( SQLERRM )) ;

    END;
END;

```

Troisième exercice

```

CREATE OR REPLACE TRIGGER T_INS_ACTEUR
BEFORE INSERT
ON ACTEUR
FOR EACH ROW
DECLARE

    nbvaleur NUMBER(5);

BEGIN

```

```
nbvaleur := 0;

SELECT COUNT(*) INTO nbvaleur FROM PAYS WHERE PAYS.IDENT_PAYS =
:new.NATIONALITE;

IF (nbvaleur = 0) THEN
    DBMS_OUTPUT.PUT_LINE( 'La nationalité est inconnue :
'| | :new.NATIONALITE) ;
    DBMS_OUTPUT.PUT_LINE( 'Une insertion dans la table PAYS va être
réalisée');
    DBMS_OUTPUT.PUT_LINE( 'Merci de compléter le libellé du pays
ultérieurement');
    INSERT INTO PAYS VALUES (:new.NATIONALITE,'A COMPLETER');
END IF;
END;
```

Approfondissement

Les sous-requêtes

Il est possible d'insérer une requête dans une autre. Celle-ci peut se trouver après la clause WHERE ou remplacer une constante derrière un ordre IN ou EXISTS, par exemple.

Il existe deux types de sous-requêtes : imbriquées ou corrélées.

1. Les sous-requêtes imbriquées

En fonction de ce que peut ramener le sous-SELECT, celui-ci ne pourra pas être positionné n'importe où.

Si le résultat de la requête placé dans un sous-SELECT ne ramène qu'une seule ligne, on pourra utiliser la sous-requête à la place de n'importe quelle constante.

Par exemple, si on veut récupérer toutes les chambres qui ont 1 lit simple avec douche, il faut faire un sous-SELECT qui récupère l'identifiant de la table TYPESCHAMBRE qui correspond à la description 1 lit simple avec douche, puis vérifier que la colonne typechambre de la table CHAMBRES correspond à la valeur du sous-SELECT.

Avant de tester la requête complète, il est préférable de tester la sous-requête en premier pour vérifier sa validité et que celle-ci ramène une seule ligne.

Exemple

```
SELECT idtypechambre FROM typeschambre
```

```
where description = '1 lit simple avec douche';
```

affiche :

```
IDTYPECHAMBRE
```

```
-----
```

```
1
```

Ensuite, on peut inclure la sous-requête dans la requête principale ainsi :

```
SELECT hotels.libelle, numchambre FROM Chambres
```

```
INNER JOIN hotels ON hotels.idhotel =chambres.hotel
```

```
where typechambre = (SELECT idtypechambre FROM typeschambre
```

```
where description = '1 lit simple avec douche');
```

Résultat

```
LIBELLE
```

```
NUMCHA
```

```
-----
```

```
Ski Hotel
```

```
1
```

```
Art Hotel
```

```
1
```

```
Rose Hotel
```

```
1
```

```
Lions Hotel
```

```
1
```

Il est également possible d'inclure une sous-requête directement dans le SELECT principal à condition également que celle-ci ne ramène qu'une seule ligne.

Pour récupérer le libellé du pays sans réaliser de jointure, on peut écrire la requête ainsi :

```
SELECT hotels.libelle
, numchambre
,(SELECT description FROM typeschambre
where idtypechambre = chambres.typechambre) descchambre
FROM Chambres
INNER JOIN hotels ON hotels.idhotel =chambres.hotel
WHERE hotels.idhotel = 3;
```

Le lien entre la sous-requête et la requête principale se fait par la clause WHERE qui utilise une colonne de la requête principale : CHAMBRES.TYPECHAMBRE.

Résultat

LIBELLE	NUMCHA	DESCCHAMBRE

Rose Hotel	1	1 lit simple avec douche
Rose Hotel	2	2 lits simples avec douche
Rose Hotel	3	3 lits simples avec douche et WC séparés
Rose Hotel	4	1 lit double avec douche
Rose Hotel	5	1 lit double avec douche et WC séparés
Rose Hotel	6	1 lit double avec bain et WC séparés
Rose Hotel	7	1 lit double large avec bain et WC séparés

7 lignes sélectionnées.

Si maintenant la sous-requête ramène plusieurs occurrences, elle peut être utilisée dans une clause IN. Si on recherche les acteurs américains ou algériens, on écrira la requête ci-dessous.

```
SELECT hotels.libelle, numchambre FROM Chambres
INNER JOIN hotels ON hotels.idhotel =chambres.hotel
where typechambre IN (SELECT idtypechambre FROM typeschambre
where description IN ('1 lit simple avec douche', '1 lit double
avec douche'));
```

Résultat

LIBELLE	NUMCHA
---------	--------

Ski Hotel	1
Ski Hotel	4
Art Hotel	1
Art Hotel	4
Rose Hotel	1
Rose Hotel	4
Lions Hotel	1
Lions Hotel	4

8 lignes sélectionnées.

2. Les sous-requêtes corrélées

Les sous-requêtes peuvent être utilisées également pour tester l'existence d'une valeur dans une table. C'est un peu le même type de recherche que le IN, mais généralement moins rapide à l'exécution.

La même requête que précédemment mais avec la clause EXISTS à la place de la clause IN donne le même résultat :

```
SELECT hotels.libelle, numchambre FROM Chambres
INNER JOIN hotels ON hotels.idhotel =chambres.hotel
where EXISTS (SELECT idtypechambre FROM typeschambre
WHERE description IN ('1 lit simple avec douche', '1 lit double
avec douche')
AND chambres.typechambre = typeschambre.idtypechambre);
```

Autre possibilité, les sous-requêtes avec la même table que la requête principale.

Par exemple, si on veut afficher les chambres qui ont le même type de chambre :

```
SELECT hotels.libelle, numchambre, typechambre FROM CHAMBRES
INNER JOIN hotels ON chambres.hotel =hotels.idhotel
AND hotels.idhotel = 4
WHERE EXISTS (SELECT * FROM chambres C1 WHERE C1.typechambre
=chambres.typechambre
AND c1.numchambre <> chambres.numchambre);
```

Le résultat est « Aucune ligne sélectionnée », étant donné que dans la table CHAMBRES il n'y a pas deux chambres qui ont le même type.

Si maintenant on ajoute une ligne dans la table CHAMBRES avec le type de chambre n° 2 :

```
INSERT INTO CHAMBRES VALUES (29, 4, 2, 8, null);
```

On a maintenant deux types de chambre 2 dans la table, donc le résultat de la requête précédente est :

LIBELLE	NUMCHA	TYPECHAMBRE

Lions Hotel	8	2
Lions Hotel	2	2

Les imports et exports de données

Selon les bases de données, il existe des outils d'import et export de données comme SQL* Loader pour Oracle, décrit ci-dessous. Dans SQL Server, il est possible d'utiliser l'utilitaire bcp ou l'outil d'import/export à partir de SQL Server Management Studio. Nous ne décrivons pas ces outils, dont le dernier s'utilise de manière très intuitive. L'outil le plus adapté pour industrialiser l'import ou l'export de données est un ETL (*Extract Transform and Load*) comme SSIS de l'éditeur Microsoft, inclus dans la licence SQL Server (excepté la version Express), Talend en open source, Oracle Data Integrator, Pentaho, Stambia...

À partir du client Oracle SQL Developer, il suffit de faire un clic droit sur une table et de choisir **Exporter...** ou **Copier dans Oracle** et se laisser guider.

1. Charger des données en masse avec SQL*Loader

Après avoir créé les enveloppes des différentes tables d'une base de données, il faut maintenant les alimenter.

Lorsqu'un historique existe, il peut être intéressant de charger rapidement et en masse tout cet historique.

Avant toute chose, il faut mettre cet historique au format attendu pour que le chargement fonctionne avec l'outil choisi.

Reprenons par exemple la table FILM que l'on a remplie par des INSERT multiples lors du chapitre La manipulation des données (LMD) - Exercices d'application.

TABLE FILM

Requête de création de la table (syntaxe standard) :

```
CREATE TABLE FILM (IDENT_FILM      INTEGER,
                    TTTRE           VARCHAR(50),
                    GENRE1          VARCHAR(20),
                    GENRE2          VARCHAR(20),
                    DATE_SORTIE     DATE,
                    PAYS             SMALLINT,
                    IDENT_REALISATEUR INTEGER,
                    DISTRIBUTEUR    VARCHAR(50),
                    RESUME           VARCHAR(2000));
```

Requête d'insertion de lignes (syntaxe Oracle) :

```
INSERT INTO FILM VALUES
```

```
(1,'SUBWAY','POLICIER','DRAME',TO_DATE('10/04/1985','DD/MM/YYYY'),
```

1,1,'GAUMONT','Conte les aventures de la population souterraine
dans les couloirs du métro parisien');

INSERT INTO FILM VALUES

(2,'NIKITA','DRAME','ROMANTIQUE',TO_DATE('21/02/1990','DD/MM/YYYY'
) ,1,1,'GAUMONT','Nikita condamnée à la prison à perpétuité est
contrainte à travailler secrètement pour le gouvernement en tant
que agent hautement qualifié des services secrets.');

INSERT INTO FILM VALUES (3,'STAR WARS 6 : LE RETOUR DU
JEDI','ACTION','SF',TO_DATE('19/10/1983','DD/MM/YYYY'),2,2,'20th
Century Fox ','L"Empire galactique est plus puissant que jamais :
la construction de la nouvelle arme, l"Etoile de la Mort, menace
l'univers tout entier.');

INSERT INTO FILM VALUES

(4,'AVATAR','ACTION','SF',TO_DATE('16/10/2009','DD/MM/YYYY'),2,3,'
20th Century Fox ','Malgré sa paralysie, Jake Sully, un ancien
marine immobilisé dans un fauteuil roulant, est resté un
combattant au plus profond');

INSERT INTO FILM VALUES (5,'BIENVENUE CHEZ LES
CH" TIS','COMEDIE','',TO_DATE('27/02/2008','DD/MM/YYYY'),1,4,'PATH
E','Philippe Abrams est directeur de la poste de Salon-de-Provence
est muté dans le Nord.');

Pour la syntaxe MySQL, il suffit de changer le TO_DATE('10/04/1985','DD/MM/YYYY') par
STR_TO_DATE('10/04/1985','%d/%m/%Y').

On peut imaginer que les films étaient auparavant stockés dans un fichier Excel ou tout simplement dans un
fichier texte.

Avant de pouvoir charger en une seule fois les lignes, il faut formater un fichier pour qu'il contienne une ligne
par enregistrement et chaque colonne séparée par une virgule ou un point-virgule par exemple.

Il faut également que toutes les zones de type caractère soient encadrées par une quote. Les dates doivent toutes
avoir le même format.

Ce qui donne ceci :

1,'SUBWAY','POLICIER','DRAME',10/04/85,1,1,'GAUMONT','Conte les

aventures de la population souterraine dans les couloirs du métro parisien'

2,'NIKITA','DRAME','ROMANTIQUE',21/02/90,1,1,'GAUMONT','Nikita condamnée à la prison à perpétuité est contrainte à travailler secrètement pour le gouvernement en tant que agent hautement qualifié des services secrets.'

3,'STAR WARS 6 : LE RETOUR DU JEDI','ACTION','SF',19/10/83,2,2,'20th Century Fox ','L'Empire galactique est plus puissant que jamais : la construction de la nouvelle arme, l'"Etoile de la Mort, menace l'univers tout entier.'

4,'AVATAR','ACTION','SF',16/10/09,2,3,'20th Century Fox ','Malgré sa paralysie, Jake Sully, un ancien marine immobilisé dans un fauteuil roulant, est resté un combattant au plus profond'

5,'BIENVENUE CHEZ LES CH'TIS','COMEDIE','",27/02/08,1,4,'PATHE','Philippe Abrams est directeur de la poste de Salon-de-Provence est muté dans le Nord.'

Avec Oracle, les dates devront être dans le fichier en entrée au même format que la variable NLS_DATE_FORMAT.

Une fois ce fichier obtenu, il faut maintenant utiliser l'outil de chargement SQL*Loader proposé par Oracle.

Pour fonctionner, il faut fournir à l'outil un fichier dit de contrôle qui indique comment sont formatées les données en entrée. Classiquement on appelle ce fichier <Table>.ctl.

Exemple d'un fichier de contrôle pour un fichier avec séparateur virgule : charge_film.csv

```
LOAD DATA
INFILE 'Charge_film.dat'
BADFILE 'Charge_film.dat.bad'
DISCARDFILE 'Charge_film.dsc'
DISCARDMAX 999
TRUNCATE
INTO TABLE FILM
FIELDS TERMINATED BY ','
(IDENT_FILM,TITRE,GENRE1,GENRE2,
DATE_SORTIE,PAYS,IDENT_REALISATEUR ,DISTRIBUTEUR,RESUME )
```

INFILE : fichier qui contient les données à charger.

BADFILE : fichier qui contiendra les lignes rejetées.

DISCARDFILE : fichier qui contiendra les erreurs rencontrées.

TRUNCATE : vide la table avant le chargement.

DISCARDMAX : nombre maximum d'erreurs autorisées.

FIELDS TERMINATED BY ',' : indique que le séparateur de champs est la virgule.

Syntaxe de lancement du SQL*Loader

```
sqlldr data=Charge_film.dat control=Charge_film.csv
```

```
log=Charge_film.log
```

```
bad=Charge_film.bad
```

```
discard=Charge_film.dsc
```

```
userid=user/passwd
```

Cette commande se lance sur la ligne de commande du système d'exploitation.

Résultat dans la log

SQL*Loader: Release 10.2.0.1.0 - Production on Ven. Juin 24

16:29:01 2011

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Fichier de contrôle : Charge_film.csv

Fichier de données : Charge_film.dat

Fichier BAD : Charge_film.bad

Fichier DISCARD : Charge_film.dsc

(Autoriser 999 rebuts)

Nombre à charger : ALL

Nombre à sauter: 0

Erreurs permises: 50

Tableau de liens : 64 lignes, maximum de 256000 octets

Continuation : aucune spécification

Chemin utilisé: Classique

Table FILM, chargé à partir de chaque enregistrement physique.

Option d'insertion en vigueur pour cette table : TRUNCATE

Nom de colonne Position Long. Séparat.

Encadrem. Type de données

IDENT_FILM FIRST * ,
CHARACTER

TITRE NEXT * ,
CHARACTER

GENRE1 NEXT * ,
CHARACTER

GENRE2 NEXT * ,
CHARACTER

DATE_SORTIE NEXT * ,
CHARACTER

PAYS NEXT * ,
CHARACTER

IDENT_REALISATEUR NEXT * ,
CHARACTER

DISTBIUTEUR NEXT * ,
CHARACTER

RESUME NEXT * ,
CHARACTER

Table FILM :

Chargement réussi de 5 lignes.

0 Lignes chargement impossible dû à des erreurs de données.

0 Lignes chargement impossible car échec de toutes les clauses
WHEN.

0 Lignes chargement impossible car tous les champs étaient non
renseignés.

SQL*Loader indique « Chargement réussi de 5 lignes. » et il n'y a aucun enregistrement dans les fichiers .log et .bad donc le chargement s'est bien déroulé.

Il est également possible de charger un fichier qui n'a pas de séparateur en indiquant à SQL*Loader les positions début et fin de chaque zone.

Il faut également que pour une même colonne celle-ci soit toujours de la même taille, en effet on indique la position de début et la position de fin de la zone. Les zones en CHAR seront complétées par des blancs si nécessaire.

Voici quelques lignes du fichier en entrée, sans séparateur :

```
1SUBWAY                POLICIERDRAME
10/04/8511GAUMONT      Conte les aventures de la population
souterraine dans les couloirs du métro parisien
2NIKITA                DROME
ROMANTIQUE21/02/9011GAUMONT      Nikita condamnée à la prison
à perpétuité est contrainte à travailler secrètement pour
le gouvernement en tant que agent hautement qualifié des services
secrets.
3STAR WARS 6 : LE RETOUR DU JEDI ACTION SF      19/10/832220th
Century Fox L"Empire galactique est plus puissant que jamais : la
construction de la nouvelle arme, l'Etoile de la Mort, menace
l'univers tout entier.
4AVATAR                ACTION SF      16/10/092320th
Century Fox Malgré sa paralysie, Jake Sully, un ancien marine
immobilisé dans un fauteuil roulant, est resté un combattant au
plus profond
5BIENVENUE CHEZ LES CH'TIS COMEDIE      27/02/0814PATHE
Philippe Abrams est directeur de la poste de Salon-de-Provence est
muté dans le Nord.
```

Le fichier de contrôle sera bien sûr différent de la version avec séparateurs.

Exemple d'un fichier de contrôle avec pour chaque colonne de la table la position début et la position fin dans le fichier en entrée : charge_film_fixe.csv

```
LOAD DATA
INFILE 'Charge_film_fixe.dat'
BADFILE 'Charge_film_fixe.dat.bad'
DISCARDFILE 'Charge_film_fixe.dsc'
DISCARDMAX 999
```

```
TRUNCATE
INTO TABLE FILM
(IDENT_FILM      POSITION ( 1 : 1 ) INTEGER EXTERNAL,
TITRE           POSITION ( 2 : 32 ) CHAR,
GENRE1          POSITION ( 33 : 40 ) CHAR,
GENRE2          POSITION ( 41 : 50 ) CHAR,
DATE_SORTIE     POSITION ( 51 : 58 )
"TO_DATE(:DATE_SORTIE,'DD/MM/YY')",
PAYS            POSITION ( 59 : 59 ) INTEGER EXTERNAL,
IDENT_REALISATEUR POSITION ( 60 : 60 ) INTEGER EXTERNAL,
DISTRIBUTEUR    POSITION ( 61 : 77 ) CHAR,
RESUME          POSITION ( 78 : 239 ) CHAR)
```

sqlldr data=Charge_film_fixe.dat control=Charge_film_fixe.csv

log=Charge_film_fixe.log bad=Charge_film_fixe.bad

discard=Charge_film_fixe.dsc userid=user/passwd

Pour les dates, il est nécessaire de préciser le format.

Résultat :

SQL*Loader: Release 10.2.0.1.0 -

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Fichier de contrôle : Charge_film_fixe.csv

Fichier de données : Charge_film_fixe.dat

Fichier BAD : Charge_film_fixe.bad

Fichier DISCARD : Charge_film_fixe.dsc

(Autoriser 999 rebuts)

Nombre à charger : ALL

Nombre à sauter: 0

Erreurs permises: 50

Tableau de liens : 64 lignes, maximum de 256000 octets

Continuation : aucune spécification

Chemin utilisé: Classique

Table FILM, chargé à partir de chaque enregistrement physique.

Option d'insertion en vigueur pour cette table : TRUNCATE

Nom de colonne	Position	Long.	Séparat.	Encadrem.
----------------	----------	-------	----------	-----------

Type de données

IDENT_FILM	1:1	1		CHARACTER
------------	-----	---	--	-----------

TITRE	2:32	31		CHARACTER
-------	------	----	--	-----------

GENRE1	33:40	8		CHARACTER
--------	-------	---	--	-----------

GENRE2	41:50	10		CHARACTER
--------	-------	----	--	-----------

DATE_SORTIE	51:58	8		CHARACTER
-------------	-------	---	--	-----------

chaîne SQL pour la colonne : "TO_DATE(:DATE_SORTIE,'DD/MM/YY')"

PAYS	59:59	1		CHARACTER
------	-------	---	--	-----------

IDENT_REALISATEUR	60:60	1		CHARACTER
-------------------	-------	---	--	-----------

DISTRIBUTEUR	61:77	17		CHARACTER
--------------	-------	----	--	-----------

RESUME	78:239	162		CHARACTER
--------	--------	-----	--	-----------

Table FILM :

Chargement réussi de 5 lignes.

0 Lignes chargement impossible dû à des erreurs de données.

0 Lignes chargement impossible car échec de toutes les clauses

WHEN.

0 Lignes chargement impossible car tous les champs étaient non
renseignés.

Espace affecté au tableau de liens : 16768 octets(64 lignes)

Octets de tampon de lecture : 1048576

Nombre total d'enregistrements logiques ignorés : 0
Nombre total d'enregistrements logiques lus : 5
Nombre total d'enregistrements logiques rejetés : 0
Nombre total d'enregistrements logiques mis au rebut : 0

Temps écoulé (ELAPSED) : 00:00:06.00

Temps processeur (CPU) : 00:00:00.14

2. Les imports et exports de tables avec Oracle

Les SGBDR proposent en général un mécanisme d'export et d'import de table ou de toute la base. Ces procédures sont utilisées principalement pour sauvegarder les bases ou transférer des bases d'un serveur à un autre.

Le format des fichiers extraits est souvent inexploitable par une autre application, notamment avec Oracle. C'est un format propriétaire de la base. Il faut également éviter de compresser ces fichiers, au risque de les rendre inexploitables.

a. Les exports de tables

Avec Oracle, on utilisera l'exécutable « exp » et pour MySQL « mysqldump ». Ces outils sont fournis lors de l'installation du SGBDR.

Syntaxe Oracle pour exporter la table FILM

exp <user/mot de passe> file=FILM.dmp TABLES=FILM LOG=FILM.log

Cette commande se lance sur la ligne de commande du système d'exploitation.

Rapport d'exécution indiquant que cinq lignes de la table FILM ont été exportées :

Export: Release 10.2.0.1.0 - Production on Ven. Juil. 1 15:26:15

2011

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Connecté Ó : Oracle Database 10g Express Edition Release

10.2.0.1.0 - Production

Export fait dans le jeu de car WE8MSWIN1252 et jeu de car NCHAR

AL16UTF16

Prêt à exporter les tables spécifiées ... via le chemin direct...

.. export de la table

FILM 5

lignes exportées <

Procédure d'export terminée avec succès sans avertissements.

On peut également ajouter une condition à l'exportation. Par exemple si l'on ne veut que les films dont l'identifiant est supérieur à 3, on ajoutera la clause QUERY :

```
exp <user/mot de passe> file=FILM.dmp TABLES=FILM LOG=FILM.log
```

```
query="'where IDENT_FILM > 3'
```

L'export n'a extrait que deux lignes :

Prêt à exporter les tables spécifiées ... via le chemin direct...

.. export de la table

FILM 2

lignes exportées <

Il est également possible d'exporter toute la base de données en ajoutant l'option FULL=Y. Cette option est réservée à l'administrateur de la base.

```
exp <user/mot de passe> file=TOUTBASE.dmp LOG=TOUTBASE.log full=y
```

Dernière remarque, il est possible de mettre tous les paramètres de l'export dans un fichier de paramètres de cette façon :

```
exp parfile=param_chgt_film.prm
```

Consultez la documentation de la base de données afin de visualiser toutes les possibilités offertes par l'export.

Syntaxe MySQL pour exporter la table FILM

```
mysqldump -h<nom ou adresse machine, par défaut localhost> -u
```

```
<user> -p<mot de passe> -rFILM.dmp <nom base> FILM
```

-r : nom du fichier résultat suivi du nom de la base et du nom de(s) table(s) exportée(s).

Contrairement à Oracle, le fichier issu de l'export MySQL est exploitable. On retrouve les ordres de création des tables ainsi que les Inserts des lignes en clair.

Fichier export MySQL :

```
-- MySQL dump 10.13 Distrib 5.1.54, for Win32 (ia32)
```

```
--
```

```
-- Host: localhost Database: TEST
```

```
-- -----
```

```
-- Server version 5.1.54-community
```

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
```

```
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
```

```
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
```

```
/*!40101 SET NAMES utf8 */;
```

```

/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;

/*!40103 SET TIME_ZONE='+00:00' */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;

/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;

/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--

-- Table structure for table `film`

--

DROP TABLE IF EXISTS `film`;

/*!40101 SET @saved_cs_client = @@character_set_client */;

/*!40101 SET character_set_client = utf8 */;

CREATE TABLE `film` (
  `IDENT_FILM` int(11) DEFAULT NULL,
  `TITRE` varchar(50) DEFAULT NULL,
  `GENRE1` varchar(20) DEFAULT NULL,
  `GENRE2` varchar(20) DEFAULT NULL,
  `DATE_SORTIE` date DEFAULT NULL,
  `PAYS` smallint(6) DEFAULT NULL,
  `IDENT_REALISATEUR` int(11) DEFAULT NULL,
  `DISTRIBUTEUR` varchar(50) DEFAULT NULL,
  `RESUME` varchar(2000) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*!40101 SET character_set_client = @saved_cs_client */;

--

-- Dumping data for table `film`

--

```


LOCK TABLES `film` WRITE;

/*!40000 ALTER TABLE `film` DISABLE KEYS */;

INSERT INTO `film` VALUES (1,'SUBWAY','POLICIER','DRAME','1985-04-

10',1,1,'GAUMONT','Conte les aventures de la population

souterraine dans les couloirs du métro parisien'),

(2,'NIKITA','DRAME','ROMANTIQUE','1990-02-

21',1,1,'GAUMONT','Nikita condamnée à la prison

perpétratrice est contrainte de travailler secrètement pour le

gouvernement en tant qu'agent hautement qualifié des services

secrets.),(3,'STAR WARS 6 : LE RETOUR DU

JEDI','ACTION','SF','1983-10-19',2,2,'20th Century Fox

','L'Empire galactique est plus puissant que jamais : la

construction de la nouvelle arme, l'Etoile de la Mort, menace

l'univers tout entier.),(4,'AVATAR','ACTION','SF','2009-10-

16',2,3,'20th Century Fox','Malgré sa paralysie, Jake Sully, un

ancien marine immobilisé dans un fauteuil roulant, est resté

un combattant au plus profond'),(5,'BIENVENUE CHEZ LES

CH'TIS','COMEDIE','','2008-02-27',1,4,'PATHE','Philippe Abrams

est directeur de la poste de Salon-de-Provence est muté dans le

Nord.),(6,'DIE HARD 4','ACTION','','2007-07-04',2,5,'Twentieth

Century Fox Film Corporation','Pour sa quatrième aventure,

l'inspecteur John McClane se trouve confronté à un nouveau

genre de terrorisme. Le réseau informatique national qui

contrôle absolument toutes les communications, les transports et

l'énergie des Etats-Unis, est détruit de façon

systématique, plongeant le pays dans le chaos.');

/*!40000 ALTER TABLE `film` ENABLE KEYS */;

UNLOCK TABLES;

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;

/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;

/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;

```

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

```

-- Dump completed on 2011-07-01 16:32:10

Avec MySQL, il est également possible d'extraire toute la base en ne précisant aucun nom de tables.

mysqldump -h<nom ou adresse machine, par défaut localhost> -u

<user> -p<mot de passe> <nom base> >TOUTBASE.dmp

b. Les imports de tables

Une fois la base exportée, il faut pouvoir importer les résultats dans une autre base ou dans la même. Pour cela, on utilisera les mécanismes d'importation.

Avec Oracle, on utilisera l'exécutable « imp » et pour MySQL la commande en ligne « mysql ».

Exemple d'import pour une base Oracle :

```

imp <user/mot de passe> file=FILM.dmp TABLES=FILM
LOG=IMPORTFILM.log FROMUSER=<user> TOUSER=<user>

```

Cette commande se lance sur la ligne de commande du système d'exploitation.

Import: Release 10.2.0.1.0 - Production on Ven. Juil. 1 17:01:05

2011

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Connecté à : Oracle Database 10g Express Edition Release

10.2.0.1.0 - Production

Fichier d'export créé par EXPORT:V10.02.01 via le chemin classique

import effectué dans le jeu de caractères WE8MSWIN1252 et le jeu

NCHAR AL16UTF16

. Import d'objets <USER> dans <USER>

.. Import de la table "FILM" 5

lignes importé

es <

Exemple d'import pour une base MySQL :

mysql -hlocalhost -u <user> -p<mot de passe> <Nom base de données>

< FILM.dmp

Contrairement à Oracle, il n'est pas nécessaire de dropper les tables en amont. Nous avons vu lors de l'export que l'ordre DROP est inclus dans le script de rechargement.

DROP TABLE IF EXISTS `film`;

/*!40101 SET @saved_cs_client = @@character_set_client */;

/*!40101 SET character_set_client = utf8 */;

CREATE TABLE `film` (

`IDENT_FILM` int(11) DEFAULT NULL,

...

3. Extraire les données d'une table dans un fichier à plat

Avec SQL Server, les instructions OPENROWS et OPENDATASOURCE permettent de travailler avec des fichiers.

Il est nécessaire de modifier la configuration du serveur pour les utiliser avec ce script :

```
sp_configure 'show advanced options', 1
```

```
GO
```

```
RECONFIGURE
```

```
GO
```

```
sp_configure 'Ad Hoc Distributed Queries', 1
```

```
GO
```

```
RECONFIGURE
```

Exemple

```
INSERT INTO OPENROWSET('Microsoft.ACE.OLEDB.12.0',  
'Text;DATABASE=C:\Test\;HDR=YES', 'Select * from Film.csv')  
(Titre, Genre1, Distributeur)  
SELECT Titre, Genre1, Distbiuteur FROM FILM
```

Exemple avec Excel

Ce script fonctionne à condition que les drivers Excel soient installés sur la machine.

```
INSERT INTO OPENROWSET('Microsoft.ACE.OLEDB.12.0',  
'Excel 12.0;DATABASE=C:\Test\Film.xls', 'Select * from [Sheet1$]')  
(Titre, Genre1, Distributeur)  
SELECT Titre, Genre1, Distbiuteur FROM FILM
```

Il est aussi possible d'exécuter une commande PowerShell après avoir modifié la configuration du serveur avec ce script :

```
sp_configure 'show advanced options', 1
```

```
GO
```

```
RECONFIGURE
```

```
GO
```

```
sp_configure 'xp_cmdshell', 1
```

```
GO
```

```
RECONFIGURE
```

```
EXEC xp_cmdshell 'SQLCMD -S . -d CINEMA -Q "SELECT Titre, Genre1,  
Distributeur FROM FILM" -s "," -o "C:\Test\Film.csv";
```

Avec Oracle, pour extraire le contenu des colonnes d'une table dans un fichier texte, il faut utiliser soit la commande SPOOL pour Oracle, soit le SELECT ... INTO OUTFILE pour MySQL.

Ces deux fonctions ont la même finalité : disposer des données dans un fichier pour pouvoir les utiliser dans un traitement de texte ou un tableur, par exemple.

Exemple Oracle : commande SPOOL

```
SET HEADING OFF FEEDBACK OFF ECHO OFF VERIFY OFF SPACE 0 PAGESIZE
```

```
0 TERMOUT OFF WRAP OFF
```

```
SET LINESIZE 300
```

```
SET COLSEP ","
```

```
SPOOL Extrait_Acteurs.TXT
```

```
SELECT * FROM ACTEUR;
```

```
SPOOL OFF
```

```
SET TERMOUT ON
```

Les ordres SET permettent de formater le résultat. Les options sont très nombreuses, les plus connues étant :

- LINESIZE : largeur de la ligne.
- PAGESIZE : nombre de lignes par page.
- HEADING : avec (ON) ou sans (OFF) les en-têtes de colonnes.
- ECHO : affiche (ON) ou n'affiche pas (OFF) le résultat à l'écran.
- COLSEP : caractère de séparation entre les colonnes.
- etc.

Résultat obtenu

```
SQL> SELECT * FROM ACTEUR;
```

```
1;ADJANI ;ISABELLE ;
```

```

27/06/55;    42;    1
2;LAMBERT          ;CHRISTOPHE  ;
29/03/57;    64;    1
3;BOHRINGER        ;RICHARD    ;
16/06/42;    132;    1
4;GALABRU          ;MICHEL    ;
27/10/22;    277;    1
5;PARILLAUD        ;ANNE      ;
06/05/60;    35;    1
6;FORD             ;HARRISON  ;
13/06/42;    64;    2
7;FISHER           ;CARRIE    ;
21/10/56;    74;    2
8;SALDANA          ;ZOE       ;
19/06/78;    31;    2
9;WEAVER           ;SIGOURNEY
;08/10/49;    66;    2
10;RENO            ;JEAN
;30/06/48;    75;    1
11;BOON            ;DANY
;26/06/66;    23;    1
12;MERAD           ;KAD
;27/03/64;    55;    3
SQL>
SQL> SPOOL OFF

```

À noter qu'on retrouve également toutes les commandes passées en ligne de commande.

Exemple MySQL : commande INTO OUTFILE

```
SELECT * FROM ACTEUR INTO OUTFILE 'C:\\Extrait_Acteurs.TXT';
```

Il existe également des options pour formater le résultat, comme par exemple :

```
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
```

```

LINES TERMINATED BY '\n' etc ...

```

4. Importer les données d'une autre source de données

Dans SQL Server, l'instruction BULK INSERT permet d'importer des données d'un fichier.

Il faut d'abord créer la table qui va recevoir les données, par exemple :

```
CREATE TABLE [FILMImport](  
    [TITRE] [varchar](50) NULL,  
    [GENRE1] [varchar](20) NULL,  
    [DISTRIBUTEUR] [varchar](50) NULL,  
)
```

Puis on importe les données dans la table :

```
BULK INSERT FILMImport  
FROM 'C:\Test\Film.csv'  
WITH  
(  
    FIELDTERMINATOR = ',',  
    ROWTERMINATOR = '\n'  
)
```

Il est aussi possible d'utiliser un serveur lié avec la syntaxe suivante :

```
SELECT * FROM MonServeurlié.MaBase.schéma.table
```

5. Importer les données d'un fichier XML

L'import de données à partir d'un fichier XML se réalise en deux temps. Dans un premier temps, on charge les données brutes dans une table qui contient une colonne au format XML. Puis on transforme les données XML en format relationnel.

Exemple

Création de la table qui va recevoir les données brutes :

```
CREATE TABLE timport(c1 int identity, c2 xml)
```

Puis import des données dans cette table :

```
INSERT INTO timport(c2)  
SELECT * FROM OPENROWSET(BULK 'C:\Test\Film.xml', SINGLE_BLOB)  
AS t
```

Pour transformer les données au format XML en données relationnelles, SQL Server propose la fonction XQuery nodes.

Pour obtenir 6 enregistrements dans une seule colonne, voici le script :

```
SELECT t2.item.query('.')  
FROM timport  
CROSS APPLY c2.nodes('/list/FILM') AS t2(item)
```

Pour obtenir 6 enregistrements sur 3 colonnes, voici le script :

```

SELECT t2.item.value('/Titre[1]', 'varchar(50)') AS Titre
, t2.item.value('/Genre[1]', 'varchar(20)') AS Genre
, t2.item.value('/Distributeur[1]', 'varchar(50)') AS Distributeur
FROM timport
CROSS APPLY c2.nodes('/list/FILM') AS t2(item)

```

Pour insérer les données :

```

INSERT INTO FILMImport
SELECT t2.item.value('/Titre[1]', 'varchar(50)') AS Titre
, t2.item.value('/Genre[1]', 'varchar(20)') As Genre
, t2.item.value('/Distributeur[1]', 'varchar(50)') As Distributeur
FROM timport
CROSS APPLY c2.nodes('/list/FILM') AS t2(item)

```

Ensuite, on supprime la table timport qui est inutile :

```

DROP TABLE timport

```

Quelques notions de performances

Dans l'utilisation d'une base de données, on rencontre souvent des problèmes de temps de réponse importants sur une requête ou sur une autre.

Les raisons sont multiples, il peut s'agir d'une requête qui n'utilise aucun index, d'une table très importante, de jointures multiples, de problèmes d'accès disque ou de capacité mémoire, etc.

Ce que l'on appelle le « tuning » d'une base de données est très complexe et nécessite beaucoup d'expériences et de connaissances multiples en bases de données et systèmes d'exploitation.

Les règles de base lorsque l'on écrit une requête est de :

- Vérifier que les critères de recherche (WHERE) utilisent des index.
- Vérifier que les jointures entre tables se font bien sur les clés des tables et que des index ont bien été posés sur ces tables.
- Vérifier que la sélection ne ramène pas des millions de lignes.
- Vérifier que les statistiques de la base de données ont été activées et mises à jour régulièrement (surtout avec Oracle).
- Ne pas utiliser trop de fonctions dans un même SELECT.

Les statistiques sont des données qui servent à la base de données pour savoir quel chemin est le plus optimisé pour atteindre une donnée.

1. Utilisation de EXPLAIN PLAN

Il existe un moyen de connaître le chemin utilisé par le SGBDR pour atteindre un élément. Il faut utiliser la commande EXPLAIN PLAN qui analyse l'ordre et indique ensuite le chemin pris. Pour cela, il stocke des éléments dans une table : PLAN_table sous Oracle.

La syntaxe à utiliser est celle-ci :

```
EXPLAIN PLAN SET STATEMENT_ID='<identifiant>' INTO PLAN_TABLE FOR  
SELECT ... .. ;
```

On indique au SGBDR de stocker dans une table nommée « PLAN_TABLE » sous l'identifiant choisi (STATEMENT_ID) les analyses réalisées sur la requête que l'on indique après le SELECT.

Exemple avec un SELECT sur trois tables, identifiant choisi 'TEST-PERF'

```
DELETE FROM PLAN_TABLE WHERE STATEMENT_ID='TEST-PERF';  
EXPLAIN PLAN SET STATEMENT_ID='TEST-PERF' INTO PLAN_TABLE FOR  
SELECT Hotels.libelle  
, Chambres.NumChambre  
, TypesChambre.TypeLit  
, TypesChambre.NombreLit  
, TypesChambre.Description  
FROM Chambres, TypesChambre, HOTELS  
WHERE Chambres.TypeChambre = TypesChambre.idTypeChambre  
AND hotels.idhotel = chambres.Hotel  
AND nombrelit in (1, 3);
```

On commence par supprimer de la table les lignes qui sont éventuellement déjà présentes avec cet identifiant 'TEST-PERF'.

Pour ensuite afficher le résultat de l'analyse réalisée par le SGBDR, il faut réaliser un SELECT dans la table PLAN_TABLE.

Exemple de sélection dans la table PLAN_TABLE

```
SET LINES 132  
SET PAGES 50  
SET LONG 500  
COL PLAN FOR A45  
COL AUTRE FOR A85 WRAP  
SELECT OBJECT_NAME, OPERATION || ' ' || OPTIONS || ' ' || OBJECT_NAME  
|| ' ' || DECODE(ID, 0, 'COST = ' || POSITION) PLAN  
FROM PLAN_TABLE WHERE STATEMENT_ID = 'TEST-PERF';
```


Le résultat est de la forme :

OBJECT_NAME	PLAN

	SELECT STATEMENT Cost = 9
	HASH JOIN
	MERGE JOIN CARTESIAN
CHAMBRES	TABLE ACCESS FULL TYPE_TEL
	BUFFER SORT
TYPESCHAMBRE	TABLE ACCESS FULL MARQUE_TEL
HOTELS	TABLE ACCESS FULL TELEPHONE

Ce qu'il est important de surveiller, c'est notamment les tables en ACCESS FULL ; cela signifie que le SGBDR va parcourir l'ensemble de la table lors de la requête.

On voit dans l'exemple que les trois tables sont en ACCESS FULL, ce qui peut provoquer un ralentissement de la requête sur les tables qui ont un nombre de lignes important. L'ACCESS FULL indique que le SGBDR n'utilise pas du tout les index pour accéder aux données.

Dans le cas de petites tables, ce n'est pas gênant, mais dès lors qu'au moins une des tables a un nombre de lignes important, il faut que le SGBDR utilise un index.

On peut constater également une information nommée « Cost », celle-ci nous donne une indication du « coût » de la requête. Ce « coût » est calculé par Oracle à l'aide de plusieurs paramètres, notamment le nombre des lectures E/S, le temps nécessaire à la lecture d'un bloc, le nombre de cycles CPU par seconde, etc.

L'unité n'est pas vraiment importante. L'important est de comparer le coût au fur et à mesure que l'on optimise une requête ou que l'on ajoute des index. Ainsi, cela permet de vérifier que l'on est sur la bonne voie en termes d'optimisation.

Reprenons l'exemple au-dessus en ajoutant des index sur les tables, et particulièrement sur la colonne NOMBRELIT de la table TYPESCHAMBRE.

Création d'un index sur la table TYPESCHAMBRE

```
CREATE INDEX I_NBLIT ON TYPESCHAMBRE (NOMBRELIT);
```

Relance de l'analyse de l'ordre. Ce qui donne ce nouveau résultat :

OBJECT_NAME	PLAN

	SELECT STATEMENT COST = 8
	NESTED LOOPS
	NESTED LOOPS
CHAMBRES	TABLE ACCESS FULL CHAMBRES
	INLIST ITERATOR
TYPESCHAMBRE	TABLE ACCESS BY INDEX ROWID BATCHED

TYPESCHAMBRE

I_NBLIT INDEX RANGE SCAN I_NBLIT

HOTELS TABLE ACCESS BY INDEX ROWID HOTELS

PK_TYPESCHAMBRE INDEX UNIQUE SCAN PK_TYPESCHAMBRE

On constate tout de suite que sur la table TYPESCHAMBRE, le SGBDR utilise le nouvel index (I_NBLIT) en indiquant « TABLE ACCESS BY INDEX ROWID ».

Il accède également à la table HOTEL sur l'index ROWID et il parcourt la clé primaire de la table TYPESCHAMBRE (INDEX UNIQUE SCAN PK_TYPESCHAMBRE).

Par rapport à la première analyse, on voit tout de suite une amélioration notable des accès aux données, ce qui va se traduire par une réponse plus rapide de la requête.

Le coût (cost) est descendu à 8, indiquant que l'optimisation apporte un plus. Le nombre de lignes par table étant très faible, il est normal que la différence entre avant et après optimisation ne soit pas importante. Si on avait eu des tables de plusieurs milliers de lignes, la différence aurait été plus importante.

L'optimisation d'une requête doit être menée pas à pas en consultant après chaque modification l'impact sur les chemins pris et sur le coût constaté. En effet, certaines actions peuvent être contradictoires en termes de chemin d'accès et ainsi dégrader le temps de réponse.

2. Utilisation du package DBMS_XPLAN.DISPLAY

On peut également afficher le résultat d'une analyse en utilisant un package fourni par Oracle.

Ce package fournit sous une autre forme les mêmes informations que l'EXPLAIN PLAN.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Ce qui donne un résultat de cette forme :

PLAN_TABLE_OUTPUT

Plan hash value: 257614804

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time

0	SELECT STATEMENT		2	80	5(0)	00:00:01
1	NESTED LOOPS		2	80	5(0)	00:00:01
2	NESTED LOOPS		2	58	4(0)	00:00:01
3	TABLE ACCESS	CHAMBRES	5	55	2(0)	00:00:01
	FULL					
4	INLIST					
	ITERATOR					

* 5	TABLE ACCESS	TYPESCHAMBRE	1	18	2(0)	00:00:01
	BY INDEX ROWID					
* 6	INDEX RANGE	I_NBLIT	2		1(0)	00:00:01
	SCAN					
7	TABLE ACCESS	HOTEL	1	11	1(0)	00:00:01
	BY INDEX ROWID					
* 8	INDEX UNIQUE	PK_Typeschambre	1		0(0)	00:00:01
	SCAN					

Sous cette forme, on peut vérifier le coût par accès. En effet, la colonne « cost » précise le coût pour chaque action.

3. Optimisation des requêtes par l'utilisation des HINTS

Dans la majorité des cas, Oracle utilise le bon index lors de l'accès aux données.

Néanmoins dans certains cas, si on souhaite « forcer » l'utilisation d'un index particulier, il est possible de l'indiquer dans la requête.

Pour cela, il faut utiliser des ordres nommés HINTS qui s'insèrent dans la requête et qui indiquent à Oracle d'utiliser tel ou tel index ou une méthode d'accès particulière.

Dans cet exemple, on cherche les types de chambre de type « lit double » :

```
SELECT TYPELIT,
       NOMBRELIT,
       DESCRIPTION
FROM   TYPESCHAMBRE
WHERE  TYPELIT = 'lit double';
```

On crée un index sur le type.

```
CREATE INDEX I_TYPE ON TYPESCHAMBRE (TYPELIT);
```

Le plan d'accès initial est :

OBJECT_NAME	PLAN

	SELECT STATEMENT
	COST = 2
	TYPESCHAMBRE
	TABLE ACCESS FULL TYPESCHAMBRE

Il existe un index sur la colonne TYPE mais Oracle ne l'utilise pas. Même avec une restriction sur TYPELIT, Oracle peut quand même réaliser un ACCESS FULL s'il estime que le nombre de lignes est insuffisant, l'accès complet de la table sera plus rapide.

Maintenant, si l'on veut absolument passer par l'index I_TYPE, on peut écrire la requête de cette façon :

```
SELECT /*+ INDEX(TYPESCHAMBRE I_TYPE) */
```

```

TYPELIT,
NOMBRELIT,
DESCRIPTION
FROM TYPESCHAMBRE
WHERE TYPELIT = 'lit double'

```

Le plan est maintenant :

```

OBJECT_NAME  PLAN
-----
SELECT STATEMENT  COST = 2
TYPESCHAMBRE      TABLE ACCESS BY INDEX ROWID TYPESCHAMBRE
I_TYPE            INDEX RANGE SCAN I_TYPE

```

Dans certains cas, on peut aussi préférer réaliser un ACCESS FULL alors qu'Oracle utilise un index.

Dans ce cas, on écrira :

```

SELECT /*+ FULL(TYPESCHAMBRE) */
TYPELIT,
NOMBRELIT,
DESCRIPTION
FROM TYPESCHAMBRE
WHERE TYPELIT = 'lit double'

```

La syntaxe est donc :

```

SELECT /*+ <ORDRE>(<NOM TABLE> <NOM INDEX> */ <colonne 1>, <colonne 2>
FROM TABLE;

```

L'utilisation des HINTS doit se faire avec précaution. Si les statistiques sont à jour, Oracle utilisera le chemin le plus optimisé pour répondre à la requête.

Dans SQL Server, à partir de Management Studio, il est possible d'afficher le plan d'exécution lorsqu'on exécute une requête en cliquant sur l'icône **Inclure le plan d'exécution actuel**, disponible également à partir du menu **Requête**.

4. Conclusion

En conclusion, il faut préciser que l'optimisation d'une requête SQL peut se révéler complexe et qu'il ne faut pas hésiter à demander de l'aide aux DBA qui pourront analyser les résultats des requêtes et donner des conseils précieux.

Il faut également se rapprocher des métiers (MOA) afin de connaître les types d'accès qui sont réalisés par les utilisateurs sur telle ou telle table.

Les tables système

Les SGDBR utilisent pour leurs besoins un ensemble de tables pour stocker tous les éléments créés par un utilisateur. Tous les objets sont stockés dans des tables dites système.

Celles-ci sont accessibles simplement par la commande :

```
SELECT * FROM <Nom table>;
```

1. Tables système pour les tables et colonnes

a. Oracle

Table	Contenu
ALL_COL_COMMENTS	Liste tous les commentaires sur les colonnes des tables.
ALL_TABLES	Liste toutes les tables.
ALL_TAB_COLUMNS	Liste toutes les colonnes des tables.

b. MySQL

Table	Contenu
INFORMATION_SCHEMA.TABLES	Liste toutes les tables.
INFORMATION_SCHEMA.COLUMNS	Liste toutes les colonnes des tables.

c. SQL Server

SQL Server stocke ces tables dans une base de données système nommée master.

Table	Contenu
sys.tables	Liste toutes les tables.
sys.all_columns	Liste toutes les colonnes des tables.

2. Tables système pour les index et les vues

a. Oracle

Table	Contenu
ALL_INDEXES	Liste tous les index.
ALL_IND_COLUMNS	Liste toutes les colonnes des index.
ALL_VIEWS	Liste toutes les vues.

b. MySQL

Table	Contenu
INFORMATION_SCHEMA. STATISTICS	Liste toutes les informations sur les index.
INFORMATION_SCHEMA. VIEWS	Liste toutes les vues.

c. SQL Server

Table	Contenu
sys.indexes	Liste toutes les informations sur les index.
sys.views	Liste toutes les vues utilisateurs.
sys.all_views	Liste toutes les vues.

3. Les autres tables système

a. Oracle

Table	Contenu
ALL_CATALOG	Liste toutes les tables, vues, séquences et synonymes.
ALL_CONSTRAINTS	Liste les contraintes.
ALL_OBJECTS	Liste tous les objets accessibles par l'utilisateur.
ALL_SEQUENCES	Liste les séquences.
ALL_SYNONYMS	Liste les synonymes.
ALL_TRIGGERS	Liste tous les triggers.
ALL_TRIGGERS_COLS	Liste toutes les colonnes des triggers.
ALL_USERS	Liste les utilisateurs déclarés.

b. MySQL

Table	Contenu
INFORMATION_SCHEMA. SCHEMATA	Liste toutes les tables, vues, séquences et synonymes.
INFORMATION_SCHEMA. CONSTRAINTS	Liste les contraintes.
INFORMATION_SCHEMA.COLUMN_PRIVILEGES	Liste tous les objets accessibles par l'utilisateur.
INFORMATION_SCHEMA.USER_PRIVILEGES	Liste les utilisateurs déclarés.
INFORMATION_SCHEMA.COLUMN_PRIVILEGES	Liste les privilèges sur les colonnes.

Table	Contenu
INFORMATION_SCHEMA.TABLE_PRIVILEGES	Liste les privilèges sur les tables.
INFORMATION_SCHEMA.ROUTINES	Liste les fonctions et les procédures stockées.
INFORMATION_SCHEMA.TRIGGERS	Liste toutes les informations sur les triggers.

c. SQL Server

Table	Contenu
sys.all_objects	Liste toutes les tables, vues, séquences et synonymes.
sys.check_constraints sys.default_constraints sys.key_constraints	Liste les contraintes.
sys.databases	Liste toutes les bases de données de l'instance.
sys.sysusers	Liste les utilisateurs déclarés.
sys.procedures	Liste les procédures stockées.
sys.triggers	Liste toutes les informations sur les triggers.

Les métadonnées, fonctions et procédures système SQL Server

Procédures système de description complète :

- `exec sp_helpdb`
- `exec sp_help 'Hotels'`
- `exec sp_helpdb 'RESAHOTEL'`
- `exec sp_linkedservers`

Fonctions système :

- `select DB_NAME()`
- `select db_id()`
- `select db_name(2)`
- `select DB_ID('RESAHOTEL')`
- `select SUSER_NAME()`
- `select getdate()`
- `select SYSDATETIME()`
- `select HOST_NAME() --machine`

Variables système :

- select @@SERVERNAME --instance
- select @@VERSION

Quelques scripts bien utiles

1. Connaître la taille réelle d'une colonne

Sur une colonne déclarée en VARCHAR, il peut être intéressant de connaître la taille réelle de chaque valeur.

Cette requête permet en plus de trier le résultat.

Syntaxe

```
SELECT <nom de colonne>, LENGTH (TRIM(<nom de colonne>))
FROM <nom table> WHERE ..
ORDER BY LENGTH (TRIM(<nom de la colonne>)),<nom de colonne> ;
```

Exemple Oracle

```
SELECT LENGTH(TRIM(description)) as longueurdesc, description
FROM typeschambre
ORDER BY longueurdesc;
```

Exemple SQL Server

```
SELECT LEN(TRIM(description)) as longueurdesc, description
FROM typeschambre
ORDER BY longueurdesc;
```

Résultat

LONGUEURDESC DESCRIPTION

24	1 lit simple avec douche
24	1 lit double avec douche
25	2 lits double avec douche
26	2 lits simples avec douche
34	1 lit XL et 1 lit simple avec bain
35	1 lit double avec bain et WC séparés
36	2 lits double avec bain et WC séparés
37	1 lit double avec douche et WC séparés
38	2 lits double avec douche et WC séparés

2. Rechercher et supprimer des doublons dans une table

Souvent on se retrouve avec des lignes en double dans une table suite à une mauvaise manipulation ou suite à un bug dans l'applicatif qui ne contrôle pas les doublons.

Si l'on reprend la table TYPECHAMBRE et que l'on ajoute la ligne n°13 avec 1 lit simple avec douche qui existe déjà en ligne 1.

```
INSERT INTO typeschambre VALUES (13, 1, 'lit simple', '1 lit simple avec douche');
```

Contenu de la table TYPECHAMBRE

IDTYPECHAMBRE	NOMBRELIT	TYPELIT	DESCRIPTION
1	1 lit simple	1 lit simple avec douche	
2	2 lit simple	2 lits simples avec douche	
3	3 lit simple	3 lits simples avec douche et WC séparés	
4	1 lit double	1 lit double avec douche	
5	1 lit double	1 lit double avec douche et WC séparés	
6	1 lit double	1 lit double avec bain et WC séparés	
7	1 lit XL	1 lit double large avec bain et WC séparés	
8	2 lit XL	1 lit XL et 1 lit simple avec bain	
9	2 lit double	2 lits double avec douche	
10	2 lit double	2 lits double avec douche et WC séparés	
11	2 lit double	2 lits double avec bain et WC séparés	
12	2 lit double	1 lit double et 1 lit simple avec douche	
13	1 lit simple	1 lit simple avec douche	

13 lignes sélectionnées.

Tout d'abord, il faut rechercher les lignes potentiellement en double à l'aide de la colonne ROWID qui est l'identifiant unique géré par la base Oracle pour toutes les lignes de la base.

Syntaxe

```
SELECT * FROM <nom table> A WHERE A.ROWID > ANY
(SELECT B.ROWID FROM <nom table> B WHERE A.<1ère clé commune> =
B.<1ère clé commune>) AND A.<2ème clé commune> = B.<2ème clé
commune>);
```

Cette syntaxe fonctionne uniquement avec Oracle, MySQL ne connaissant pas cette notion de ROWID.

Exemple Oracle sur la table TYPESCHAMBRE

```
SELECT * FROM TYPESCHAMBRE A WHERE A.ROWID > ANY  
(SELECT B.ROWID FROM TYPESCHAMBRE B WHERE A.DESCRPTION = B.DESCRPTION);
```

Exemple SQL Server

```
SELECT * FROM TypesChambre WHERE idTypeChambre NOT IN  
(SELECT MIN(idTypeChambre) FROM TypesChambre AS TC  
GROUP BY Description)
```

Résultat

IDTYPECHAMBRE	NOMBRELIT	TYPELIT	DESCRIPTION
---------------	-----------	---------	-------------

13	1 lit simple	1 lit simple avec douche
----	--------------	--------------------------

Maintenant pour détruire la ligne en trop (Oracle) :

```
DELETE FROM TYPESCHAMBRE A WHERE A.ROWID > ANY  
(SELECT B.ROWID FROM TYPESCHAMBRE B WHERE A.DESCRPTION = B.DESCRPTION);
```

SQL Server

```
DELETE FROM TypesChambre WHERE idTypeChambre NOT IN  
(SELECT MIN(idTypeChambre) FROM TypesChambre AS TC  
GROUP BY Description)
```

3. Afficher le contenu d'une table sans connaître sa structure

En utilisant les tables internes d'Oracle, on peut retrouver toutes les informations sur une table sans en connaître la structure.

Si par exemple on applique le script Oracle ci-dessous sur la table TYPESCHAMBRE, celui-ci va générer un ordre SELECT de la table que l'on peut ensuite réutiliser.

```
COLUMN VARIAB1 NOPRINT  
COLUMN VARIAB2 NOPRINT  
SELECT 'A' VARIAB1, 0 VARIAB2, 'SELECT '  
FROM DUAL  
UNION  
SELECT 'B', COLUMN_ID, DECODE(COLUMN_ID, 1, ' ', ' ', ' )  
|| DECODE(DATA_TYPE, 'DATE', 'TO_CHAR(' || ' ' ||  
COLUMN_NAME || ' ' || ' , "YYYYMMDDHH24MISS") ' || ' ' ||  
COLUMN_NAME || ' ' || ' , ' || ' ' || COLUMN_NAME || ' ' || ' )
```

```

FROM USER_TAB_COLUMNS
WHERE TABLE_NAME=UPPER('TYPESCHAMBRE')
UNION
SELECT 'C', 0, 'FROM TYPESCHAMBRE'
FROM DUAL
UNION
SELECT 'D', 0, ';'
FROM DUAL
ORDER BY 1,2
/

```

Voici le résultat de la sélection précédente :

```

SELECT
  "IDTYPECHAMBRE"
, "NOMBRELIT"
, "TYPELIT"
, "DESCRIPTION"
FROM TYPESCHAMBRE
;

```

L'exécution de la commande affiche bien les éléments de la table TYPESCHAMBRE.

IDTYPECHAMBRE	NOMBRELIT	TYPELIT	DESCRIPTION
---------------	-----------	---------	-------------

1	1 lit simple	1 lit simple avec douche	
2	2 lit simple	2 lits simples avec douche	
3	3 lit simple	3 lits simples avec douche et WC séparés	
4	1 lit double	1 lit double avec douche	
5	1 lit double	1 lit double avec douche et WC séparés	
6	1 lit double	1 lit double avec bain et WC séparés	
7	1 lit XL	1 lit double large avec bain et WC séparés	
8	2 lit XL	1 lit XL et 1 lit simple avec bain	
9	2 lit double	2 lits double avec douche	
10	2 lit double	2 lits double avec douche et WC séparés	
11	2 lit double	2 lits double avec bain et WC séparés	
12	2 lit double	1 lit double et 1 lit simple avec douche	

Script SQL Server

```
DECLARE C1 cursor FOR
SELECT C.name FROM sys.columns C
INNER JOIN sys.tables T ON C.object_id = T.object_id
WHERE T.name = 'Chambres'

DECLARE @vsq1 varchar(MAX), @vcolonne varchar(MAX), @i int
SET @i = 0
OPEN C1
FETCH NEXT FROM C1 INTO @vcolonne
WHILE @@FETCH_STATUS = 0
BEGIN
    IF @i = 0
        BEGIN
            SET @vsq1 = 'SELECT ' + @vcolonne
        END
    IF @i > 1
        BEGIN
            SET @vsq1 = @vsq1 + ', ' + @vcolonne
        END
    SET @i = @i + 1
    FETCH NEXT FROM C1 INTO @vcolonne
END
CLOSE C1
DEALLOCATE C1

SET @vsq1 = @vsq1 + ' FROM Chambres'

print @vsq1
```

En remplaçant le nom de la table par une variable (&1), on peut ainsi visualiser l'ensemble des structures de tables en ne connaissant que le nom des tables.

4. Générer les ordres d'insertion dans une table à partir d'un fichier Excel

Il peut être fastidieux de créer les ordres INSERT un par un à partir d'un fichier Excel qui contient les données à créer.

En organisant le fichier Excel dans l'ordre des colonnes de la table on peut générer assez facilement les ordres nécessaires.

Exemple de fichier Excel

	A	B	C	D	E	F	G	H	I
1	Numéro	Titre	Genre 1	Genre 2	Datede Sortie	Pays	Réalisateur	Distributeur	Resume
2	1	SUBWAY	POLICIER	DRAME	10/04/85	1	1	GAUMONT	Conte les aventures de la population souterraine dans les couloirs du métro parisien
3	2	NIKITA	DRAME	ROMANTIQUE	21/02/90	1	1	GAUMONT	Nikita condamné à la prison à perpétuité est contraint à travailler secrètement pour le gouvernement en tant que agent hautement qualifié des services secrets.
4	3	STAR WARS 6 : LE RETOUR DU JEDI	ACTION	SF	19/10/83	2	2	20th Century Fox	L'Empire galactique est plus puissant que jamais : la construction de la nouvelle

	A	B	C	D	E	F	G	H	I
									arme, l'Etoile de la Mort, menace l'univers tout entier.
5	4	AVATAR	ACTION	SF	16/10/09	2	3	20th Century Fox	Malgré sa paralysie, Jake Sully, un ancien marine immobilisé dans un fauteuil roulant, est resté un combattant au plus profond
6	5	BIENVENUE CHEZ LES CH'TIS	COMEDIE		27/02/08	1	4	PATHE	Philippe Abrams est directeur de la poste de Salon- de- Provence est muté dans le Nord.

En face de chaque ligne (en colonne J), il faut écrire une formule de ce type :

Colonne J2

=INSERT INTO FILM VALUES

("&A2&","&B2&","&C2&","&D2&",TO_DATE("&E2&",'DD/MM/YYYY'),
"&F2&","&G2&","&H2&","&I2&");"

Colonne J3

=INSERT INTO FILM VALUES

("&A3&","&B3&","&C3&","&D3&",TO_DATE("&E3&",'DD/MM/YYYY'),
"&F3&","&G3&","&H3&","&I3&");"

Colonne J4

```
= "INSERT INTO FILM VALUES  
("&A4&","&B4&","&C4&","&D4&","TO_DATE("&E4&","DD/MM/YYYY"),  
"&F4&","&G4&","&H4&","&I4&");"
```

Colonne J5

```
= "INSERT INTO FILM VALUES  
("&A5&","&B5&","&C5&","&D5&","TO_DATE("&E5&","DD/MM/YYYY"),  
"&F5&","&G5&","&H5&","&I5&");"
```

Colonne J6

```
= "INSERT INTO FILM VALUES  
("&A6&","&B6&","&C6&","&D6&","TO_DATE("&E6&","DD/MM/YYYY"),  
"&F6&","&G6&","&H6&","&I6&");"
```

Chaque valeur des colonnes est ainsi interprétée et donne le résultat suivant :

```
INSERT INTO FILM VALUES  
(1,'SUBWAY','POLICIER','DRAME',TO_DATE('10/04/1985','DD/MM/YYYY'),  
1,1,'GAUMONT','Conte les aventures de la population souterraine  
dans les couloirs du métro parisien');  
INSERT INTO FILM VALUES  
(2,'NIKITA','DRAME','ROMANTIQUE',TO_DATE('21/02/1990','DD/MM/YYYY')  
,1,1,'GAUMONT','Nikita condamnée à la prison à perpétuité est  
contrainte à travailler secrètement pour le gouvernement en tant  
que agent hautement qualifié des services secrets.');
```

```
INSERT INTO FILM VALUES (3,'STAR WARS 6 : LE RETOUR DU  
JEDI','ACTION','SF',TO_DATE('19/10/1983','DD/MM/YYYY'),2,2,'20th
```

Century Fox ','L'"Empire galactique est plus puissant que jamais :
la construction de la nouvelle arme, l'"Etoile de la Mort, menace
l'univers tout entier.');

```
INSERT INTO FILM VALUES  
(4,'AVATAR','ACTION','SF',TO_DATE('16/10/2009','DD/MM/YYYY'),2,3,'  
20th Century Fox ','Malgré sa paralysie, Jake Sully, un ancien  
marine immobilisé dans un fauteuil roulant, est resté un  
combattant au plus profond');
```

```
INSERT INTO FILM VALUES (5,'BIENVENUE CHEZ LES  
CH"GIS','COMEDIE','',TO_DATE('27/02/2008','DD/MM/YYYY'),1,4,'PATH  
E','Philippe Abrams est directeur de la poste de Salon-de-Provence  
est muté dans le Nord.');
```

Il suffit ensuite de copier/coller les colonnes J2 à J6 dans un éditeur de texte quelconque (Notepad, Textpad, etc.) et de créer un fichier avec l'extension .sql et de l'exécuter dans SQL*Plus par @<nom script>.

Ceci est une alternative à l'utilisation de SQL*Loader ou l'outil d'import/export de SQL Server lorsqu'il n'y a que quelques lignes à créer. On peut utiliser cette méthode pour réaliser des UPDATE également.

Exercices

Premier exercice

Créer une requête qui récupère tous les films qui ont dans leur casting un acteur français.

Deuxième exercice

Afficher les acteurs qui portent le même prénom qu'un autre acteur.

Solutions des exercices

Premier exercice

```
SELECT SUBSTR(FILM.TITRE,1,20) TITRE, FILM.DATE_SORTIE,  
       SUBSTR((REAL.NOM || ' ' || REAL.PRENOM),1,25) REALISATEUR,  
       SUBSTR(RTRIM(ACTEUR.NOM || ' ' || ACTEUR.PRENOM),1,25) ACTEUR,  
       ACTEUR.DATE_NAISSANCE NE,ACTEUR.NB_FILM NBFILMS,  
       STAT.BUDGET,STAT.NB_ENTREE_FRANCE NBENTREE  
FROM   FILM FILM, REALISATEUR REAL, CASTING CAST,  
       ACTEUR ACTEUR, STATISTIQUE STAT  
WHERE  
  
FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND  
FILM.IDENT_FILM        = CAST.IDENT_FILM AND  
FILM.IDENT_FILM        = STAT.IDENT_FILM AND  
CAST.IDENT_ACTEUR      = ACTEUR.IDENT_ACTEUR AND  
ACTEUR.NATIONALITE = (SELECT PAYS.IDENT_PAYS FROM PAYS WHERE  
                       PAYS.LIBELLE = 'FRANCE')  
ORDER BY FILM.TITRE, ACTEUR.NOM;
```


Deuxième exercice

```
SELECT SUBSTR(FILM.TITRE,1,20) TITRE, FILM.DATE_SORTIE,
       SUBSTR((REAL.NOM || ' ' || REAL.PRENOM),1,25) REALISATEUR,
       SUBSTR(RTRIM(ACT1.NOM || ' ' || ACT1.PRENOM),1,25) ACTEUR,
       ACT1.DATE_NAISSANCE NE,ACT1.NB_FILM NBFILMS,
       STAT.BUDGET,STAT.NB_ENTREE_FRANCE NBENTREE
FROM   FILM FILM, REALISATEUR REAL, CASTING CAST,
       ACTEUR ACT1, STATISTIQUE STAT
WHERE
       FILM.IDENT_REALISATEUR = REAL.IDENT_REALISATEUR AND
       FILM.IDENT_FILM        = CAST.IDENT_FILM AND
       FILM.IDENT_FILM        = STAT.IDENT_FILM AND
       CAST.IDENT_ACTEUR      = ACT1.IDENT_ACTEUR AND
       EXISTS (SELECT * FROM ACTEUR ACT2 WHERE ACT2.PRENOM =
ACT1.PRENOM
              AND ACT2.NOM <> ACT1.NOM)
ORDER BY FILM.TITRE, ACT1.NOM;
```

Les erreurs les plus couramment rencontrées

Introduction

Les erreurs Oracle d'accès aux données sont au format ORA-nnnnn. Il en existe plusieurs milliers. Dans ce chapitre seront évoquées quelques erreurs les plus couramment rencontrées.

D'autres erreurs Oracle existent et sont classées par types de demande. Par exemple, les erreurs spécifiques au PLS/SQL seront notées PLS-nnnnn, les erreurs liées aux sauvegardes seront notées RMAN-nnnnn, les erreurs de chargement avec SQL*Loader seront notées SQL*LOADERnnnnn, etc.

Vous trouverez sur Internet des sites répertoriant l'ensemble des erreurs existantes. En voici quelques-uns :

- http://www.ora-error.com/error_info_window.php
- http://docs.oracle.com/cd/B28359_01/server.111/b28278/toc.htm
- www.ora-error.com

Sur les accès aux données (LDD/LMD)

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00001	unique constraint (string.string) violated	Un UPDATE ou un INSERT provoque une clé dupliquée.	Supprimez la contrainte UNIQUE sur la clé, ou corrigez l'INSERT ou l'UPDATE.
ORA-00051	timeout occurred while waiting for a resource	Dépassement de délai lors du lancement d'une commande. Une ligne dans une table est réservée par un autre utilisateur et, passé un certain délai d'attente, Oracle ramène cette erreur.	Relancez la commande plus tard.
ORA-00054	resource busy and acquire with NOWAIT specified	La table ou les lignes que l'on souhaite accéder est réservée (lockée) par un autre utilisateur et le paramètre NOWAIT est activé. Cela signifie que Oracle n'attend pas dans ce cas.	Réessayez la commande après avoir attendu quelques minutes ou enlevez le paramètre NOWAIT pour qu'Oracle attende la libération de la ressource.
ORA-00060	deadlock detected while waiting for resource	Vous essayez de mettre à jour une ligne qui est également mise à jour par une autre session utilisateur.	Il faut que l'une ou l'autre des sessions réalise un ROLLBACK ou un COMMIT pour libérer la ressource.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00100	no data found	Aucune ligne n'est ramenée par le SELECT demandé.	Vérifiez la clause WHERE et les données de la table.
ORA-00900	invalid SQL statement	La syntaxe de votre requête ou de votre procédure stockée n'est pas correcte.	Corrigez la syntaxe, vérifiez notamment le format des dates, par rapport à la variable NLS_DATE_FORMAT déclarée dans la base.
ORA-00900	to ORA-01499 3-7statement. ORDER BY cannot be used to create an ordered view or to insert in acertain order.	L'ordre SQL ORDER BY n'est pas possible avec un CREATE VIEW or INSERT.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00901	invalid CREATE command	La commande CREATE n'a pas la bonne syntaxe.	Corrigez la syntaxe.
ORA-00902	invalid datatype	Lors de la création ou de la modification d'une table, vous n'avez pas utilisé un type de colonne Oracle connu.	Vérifiez les formats attribués aux colonnes (CHAR, NUMBER, etc.). Vérifiez le contenu de vos variables lors d'un UPDATE ou d'un INSERT.
ORA-00903	invalid table name	Nom de table invalide. Un nom de table doit faire au maximum 30 caractères et contenir uniquement des caractères alphanumériques. Ou la table demandée n'existe pas ou n'est pas accessible avec vos droits.	Vérifiez le nom de la table et/ou demandez au DBA si vous avez les accès.
ORA-00904	string: invalid identifier	Nom de colonne invalide. Un nom de colonne doit faire au maximum 30 caractères et contenir uniquement des caractères alphanumériques.	Vérifiez le nom de la colonne.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00905	missing keyword	Problème de syntaxe dans la commande.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00906	missing left parenthesis	Il manque une parenthèse à gauche.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00907	missing right parenthesis	Il manque une parenthèse à droite.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00908	missing NULL keyword	Manque le mot NULL dans la requête. Exemple : Select toto from table where toto is not;	Vérifiez la syntaxe de l'ordre demandé.
ORA-00909	invalid number of arguments	Il manque un argument à la fonction demandée.	Vérifiez la syntaxe de la fonction demandée.
ORA-00910	specified length too long for its datatype	La taille demandée pour une colonne dépasse la limite maximum autorisée.	Vérifiez la taille maximum des colonnes par type de données (VARCHAR, CHAR, etc.).
ORA-00911	invalid character	Un caractère invalide a été trouvé dans votre requête (par exemple un " ou un tiret (-)).	Vérifiez la syntaxe de l'ordre demandé.
ORA-00913	too many values	Erreur dans le nombre d'arguments lors d'une requête INSERT par exemple. Il y a plus d'arguments que de colonnes dans la table.	Vérifiez la structure de la table souhaitée.
ORA-00914	missing ADD keyword	Il manque le mot-clé ADD dans l'ordre ALTER TABLE.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00917	missing comma	Il manque la virgule dans la liste des valeurs. Sur un ordre INSERT par exemple, les arguments doivent être indiqués ainsi : (C,D,E,F, ...).	Vérifiez la syntaxe de l'ordre demandé.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00918	column ambiguously defined	Lors d'une jointure entre deux tables, le même nom de colonne existe dans chaque table.	Ajoutez un alias devant le nom de la colonne pour préciser de quelle table il s'agit : Table1.col1, Table2.col1
ORA-00919	invalid function	La fonction demandée n'existe pas.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00920	invalid relational operator	Opérateur invalide. Doit être dans la liste =, !=, ^=, <>, >, <, >=, <=, ALL, ANY, [NOT] BETWEEN, EXISTS, [NOT] IN, IS [NOT] NULL ou [NOT] LIKE.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00921	unexpected end of SQL command	La commande SQL n'est pas complète, il manque des éléments à la fin.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00922	missing or invalid option	Une option invalide a été indiquée dans la création ou la modification d'une table sur une colonne. Par exemple, mettre l'option NOT NULL alors que des valeurs NULL existent dans la table.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00923	FROM keyword not found where expected	Il manque l'ordre FROM dans la requête avec SELECT.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00924	missing BY keyword	Il manque l'ordre BY dans la requête avec SELECT.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00925	missing INTO keyword	Il manque l'ordre INTO dans la requête avec SELECT.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00926	missing VALUES keyword	Il manque l'ordre VALUES dans la requête avec INSERT ou UPDATE.	Vérifiez la syntaxe de l'ordre demandé.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00927	missing equal sign	Il manque le signe = dans la requête.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00932	inconsistent datatypes: expected string got string	Ordre incompatible entre deux types différents. Par exemple ajouter un caractère à une date.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00933	SQL command not properly ended	L'ordre SQL utilise un ordre inapproprié. Par exemple une clause ORDER BY dans un ordre CREATE VIEW or INSERT.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00934	group function is not allowed here	Il n'est pas possible d'utiliser une fonction du type AVG, COUNT, MAX, MIN, SUM, STDDEV ou VARIANCE sans utiliser l'ordre GROUP BY.	Ajoutez le GROUP BY sur la colonne spécifiée.
ORA-00936	missing expression	L'ordre SELECT n'est pas complet ou syntaxiquement incorrect.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00937	not a single-group group function	Une des colonnes est dans un GROUP BY, il faut donc qu'elle soit utilisée avec une fonction du type AVG, COUNT, MAX, MIN, SUM, STDDEV ou VARIANCE.	Vérifiez la syntaxe de l'ordre demandé.
ORA-00938	not enough arguments for function	Vous utilisez une fonction qui requiert des arguments complémentaires.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00939	too many arguments for function	Vous utilisez une fonction qui requiert moins d'arguments.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00940	invalid ALTER command	Dans l'ordre ALTER vous utilisez une commande inconnue ou interdite.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00941	missing cluster name	Le nom du cluster est invalide ou incorrect.	Un nom de cluster doit faire au maximum 30 caractères et contenir uniquement des caractères alphanumériques. Ou le cluster demandé n'existe pas ou n'est pas accessible avec vos droits.
ORA-00942	table or view does not exist	Le nom de la table ou de la vue n'existe pas.	La table demandée n'existe pas ou n'est pas accessible avec vos droits.
ORA-00943	cluster does not exist	Le nom du cluster n'existe pas.	Le cluster demandé n'existe pas ou n'est pas accessible avec vos droits.
ORA-00946	missing TO keyword	Vous avez oublié le TO dans l'ordre GRANT ou la syntaxe est incorrecte.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00947	not enough values	Il manque une valeur dans un ordre INSERT ou dans un ordre SELECT. Vous avez indiqué plus de colonnes que de valeurs dans la clause VALUE ou dans la clause INTO.	Vérifiez que vous avez le même nombre de colonnes que de valeurs.
ORA-00950	invalid DROP option	Vous utilisez le DROP sur un élément interdit. Il s'applique sur les types CLUSTER, DATABASE LINK, INDEX, ROLLBACK SEGMENT, SEQUENCE, SYNONYM, TABLE, TABLESPACE, ou VIEW.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00952	missing GROUP keyword	Il manque la clause GROUP dans l'ordre demandé.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00953	missing or invalid index name	Il manque le nom de l'index ou celui-ci n'est pas valide dans une clause CREATE INDEX ou DROP INDEX.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00955	name is already used by an existing object	Vous essayez de créer un objet (table, vue, index, etc.) qui existe déjà dans la base.	Faites un SELECT sur ALL_TABLES ou ALL_VIEWS ou sur une autre table système pour vérifier les éléments existants dans la base.
ORA-00957	duplicate column name	Le nom des colonnes dans une table doit être unique.	Changez le nom de la colonne impliquée.
ORA-00967	missing WHERE keyword	Vous avez oublié le WHERE dans l'ordre SELECT ou la syntaxe est incorrecte.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00968	missing INDEX keyword	Vous avez oublié l'INDEX dans l'ordre CREATE ou la syntaxe est incorrecte.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00969	missing ON keyword	Vous avez oublié le ON dans l'ordre ou la syntaxe est incorrecte.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00978	nested group function without GROUP BY	Vous utilisez une fonction sur une colonne mais vous n'avez pas de clause GROUP BY associée.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00979	not a GROUP BY expression	Vous utilisez une colonne dans une clause GROUP BY. Cette colonne n'est pas associée à une fonction	Ajouter une fonction sur la colonne de type AVG, COUNT, MAX, MIN, SUM, STDDEV, ou VARIANCE.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
		en lien avec le GROUP BY.	
ORA-00984	column not allowed here	Une colonne est utilisée à tort dans une requête.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-00996	the concatenate operator is , not	Pour concaténer deux colonnes, vous devez utiliser deux "pipe" et non un seul.	Corrigez l'ordre.
ORA-01002	fetch out of sequence	Vous avez ouvert un curseur. Vous réalisez des FETCH pour le parcourir. Le curseur est arrivé à la fin de sa lecture et vous demandez un FETCH supplémentaire.	Testez la valeur de fin de lecture (ORA-01403) avant de réaliser le FETCH.
ORA-01006	bind variable does not exist	Vous utilisez une variable externe qui n'est pas déclarée.	Souvent utilisé avec un langage de type C ou COBOL. Toutes les variables utilisées dans un ordre SQL doivent être déclarées au préalable.
ORA-01012	not logged on	Vous essayez des ordres SQL mais vous n'êtes pas connecté à la base.	Loggez-vous à Oracle.
ORA-01329	unable to truncate required build table	Impossible de vider la table, celle-ci est utilisée par un autre utilisateur.	Attendez la disponibilité de la table.
ORA-01400	cannot insert NULL into	Vous ne pouvez pas mettre la valeur NULL dans cette colonne.	Modifiez votre ordre INSERT.
ORA-01401	inserted value too large for column	Lors d'un ordre INSERT ou UPDATE vous affectez une valeur dans une colonne qui ne correspond pas à la taille de la colonne.	Vérifiez la définition des colonnes par rapport à vos valeurs.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-01403	no data found	Aucune donnée ramenée par l'ordre SELECT.	Vérifiez la clause WHERE ou la table est vide.
ORA-01405	fetched column value is NULL	Dans un curseur, vous réalisez un ordre INTO entre des colonnes de la table et des variables. Une des colonnes a la valeur NULL donc vous ne pouvez pas affecter la variable.	Utilisez la clause NVL pour indiquer la valeur par défaut que doit prendre la variable quand la colonne est vide. Exemple : SELECT NVL(COL1, 0) INTO VARIAB
ORA-01406	fetched column value was truncated	La taille de la variable réceptrice est inférieure à la taille de la colonne de la table.	Vérifiez la définition des colonnes par rapport à vos variables.
ORA-01407	cannot update to NULL	Vous ne pouvez pas mettre la valeur NULL dans cette colonne.	Modifiez votre ordre UPDATE.
ORA-01438	value larger than specified precision allows for this column	La colonne numérique est plus petite que la valeur utilisée dans l'ordre INSERT ou UPDATE.	Adaptez vos variables ou modifiez la colonne dans la table.
ORA-01452	cannot CREATE UNIQUE INDEX-duplicate keys found	Vous ne pouvez pas créer un index UNIQUE sur cette table car elle contient des valeurs en double sur les colonnes utilisées dans l'index.	Faites le ménage dans la table avant de créer l'index ou modifiez les colonnes contenues dans l'index.
ORA-01555	snapshot too old: rollback segment number string with name "string"too small	Vous avez ouvert un curseur depuis un certain temps et le contenu de la base a changé entre-temps. Oracle ne peut donc plus assurer la cohérence des données.	Modifiez le curseur pour qu'il ramène moins de lignes ou fermez-le régulièrement en appliquant des COMMIT puis rouvrez-le.
ORA-01562	failed to extend rollback segment number string	L'ordre de mise à jour (UPDATE, INSERT, DELETE) s'applique sur énormément de lignes et Oracle n'a pas assez de place pour	Ajoutez des WHERE plus restrictifs pour diminuer le nombre de mises à jour simultanées. Faites des COMMIT plus régulièrement,

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
		sauvegarder toutes les mises à jour.	contactez le DBA éventuellement.
ORA-01650	unable to extend rollback segment string by string in tablespace string	L'ordre de mise à jour (UPDATE, INSERT, DELETE) s'applique sur énormément de lignes et Oracle n'a pas assez de place pour sauvegarder toutes les mises à jour.	Ajoutez des WHERE plus restrictifs pour diminuer le nombre de mises à jour simultanées. Faites des COMMIT plus régulièrement, contactez le DBA éventuellement.
ORA-01730	invalid number of column names specified	Lors d'une clause CREATE TABLE AS SELECT xxx, le nombre de colonnes du SELECT ne correspond pas au nombre de colonnes de la table réceptrice.	Vérifiez la syntaxe de l'ordre demandé.
ORA-01821	date format not recognized	Le format de la date que vous avez indiqué n'est pas valide.	Vérifiez la syntaxe de l'ordre demandé.
ORA-01839	date not valid for month specified	Le jour indiqué ne correspond pas au mois (exemple 30/02).	Corrigez vos valeurs.
ORA-01843	not a valid month	Nom de mois invalide.	Corrigez vos valeurs.
ORA-01847	day of month must be between 1 and last day of month	Le jour doit correspondre au mois indiqué (31/02 ou 31/04 sont invalides par exemple).	Corrigez vos valeurs.
ORA-01848	day of year must be between 1 and 365 (366 for leap year)	Le jour doit être compris en 1 et 365.	Corrigez vos valeurs.
ORA-01849	hour must be between 1 and 12	L'heure doit être comprise en 1 et 12.	Corrigez vos valeurs.
ORA-01850	hour must be between 0 and 23	L'heure doit être comprise en 1 et 23.	Corrigez vos valeurs.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-02112	SELECT..INTO returns too many rows	Par exemple : Vous avez réalisé un SELECT qui est censé ramener une ligne et il en ramène plusieurs.	Il faut que vos variables réceptrices soient déclarées en tableau ou ouvrez un curseur pour ramener les lignes une par une.
ORA-02289	sequence does not exist	La séquence demandée n'existe pas ou n'est pas accessible avec vos droits.	Vérifiez le nom de votre séquence.

Sur les transactions et les sessions (TCL/DCL)

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00018	maximum number of sessions exceeded	Dépassement du nombre de sessions possibles.	Augmentez la valeur du nombre de sessions maximum : paramètre SESSIONS alter system set sessions=250 scope=spfile;
ORA-00019	maximum number of session licenses exceeded	Toutes les licences sont utilisées.	Augmentez la valeur du nombre de sessions maximum : paramètre LICENSE_MAX_SESSIONS
ORA-00021	session attached to some other process cannot switch session	Une session utilisateur est utilisée par un autre utilisateur.	Contrôlez les sessions actives.
ORA-00022	invalid session ID- access denied	La session n'existe pas ou vous n'avez pas les droits pour l'utiliser.	Utilisez une session valide ou vérifiez les droits utilisateurs.
ORA-00025	failed to allocate string	Manque de mémoire pour une chaîne de caractères.	Augmentez la taille mémoire de la SGA.
ORA-00026	missing or invalid session ID	La commande ALTER SYSTEM KILL SESSION indique que l'ID n'existe pas ou n'est pas présent.	Relancez la commande avec un ID valide.
ORA-00027	cannot kill current session	Impossible de tuer la session courante.	Utilisez un autre utilisateur pour tuer la session souhaitée.

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00028	your session has been killed	Votre session a été tuée par un autre utilisateur.	Reconnectez-vous.
ORA-00029	session is not a user session	La commande ALTER SYSTEM KILL SESSION indique que l'ID n'existe pas.	Utilisez un numéro ID valide.
ORA-00030	User session ID does not exist.	La session a déjà été supprimée.	Vérifiez le numéro de session demandé.
ORA-00032	invalid session migration password	Modification de mot de passe invalide, peut-être trop long.	Réessayez avec un nouveau mot de passe avec moins de 30 caractères.
ORA-00150	duplicate transaction ID	Vous essayez de démarrer une nouvelle transaction avec un numéro existant déjà.	Changez le numéro ou arrêtez la session de l'autre utilisateur.
ORA-00151	invalid transaction ID	Le numéro de la transaction n'existe pas.	Changez le numéro.
ORA-00987	missing or invalid username(s)	Il manque le nom de l'utilisateur dans une clause GRANT ou la syntaxe ne respecte pas les 30 caractères maximum ou ne contient pas que des caractères alphanumériques.	Indiquez un nom d'utilisateur correct.
ORA-00992	invalid format for REVOKE command	La commande REVOKE ne respecte pas la syntaxe.	Vérifiez la syntaxe de l'ordre demandé et consultez la documentation Oracle.
ORA-01014	ORACLE shutdown in progress	Oracle est en train de s'arrêter donc vous ne pouvez lancer aucun ordre SQL.	Attendez le redémarrage de la base.
ORA-01017	invalid username/password-logon denied	Le nom de l'utilisateur ou le mot de passe est incorrect. Connexion non possible.	Vérifiez votre nom d'utilisateur et/ou votre mot de passe.
ORA-01435	user does not exist	Dans un ordre GRANT ou REVOKE vous indiquez un utilisateur inexistant.	Vérifiez la validité de l'utilisateur.

Sur les composants internes (mémoire, système)

Code de l'anomalie	Message Oracle	Cause de l'anomalie	Solution
ORA-00058	DB_BLOCK_SIZE must be string to mount this database (not string)	La valeur indiquée pour le paramètre DB_BLOCK_SIZE lors du démarrage de la base n'est pas celui qui a servi lors de la création de la base.	Corrigez le paramètre DB_BLOCK_SIZE.
ORA-00059	maximum number of DB_FILES exceeded	Le nombre de fichiers physiques maximum (DB_FILES) contenant les données de la base est dépassé.	Incrémentez le paramètre DB_FILES et redémarrez la base.
ORA-00063	maximum number of LOG_FILES exceeded	Le nombre de fichiers journaux (log) maximum est dépassé.	Incrémentez le paramètre LOG_FILES.
ORA-00065	initialization of FIXED_DATE failed	La variable FIXED_DATE qui est la date système Oracle n'est pas au format date (yyyy-mm-dd:hh24:mi:ss).	Modifiez le paramètre pour le mettre au bon format et relancez la base.
ORA-00483	During shutdown a process abnormally terminated	Lors de l'arrêt de la base, un processus s'est mal terminé.	Vérifiez les logs pour détecter quel processus est concerné.

Annexes

Récapitulatif des ordres principaux

1. Les principaux ordres du LDD (langage de définition de données) ou DDL (Data Definition Language)

Création d'une table

```
CREATE TABLE nom_de_table (Nom_colonne Type_colonne,  
                             Nom_colonne Type_colonne,  
                             Nom_colonne Type_colonne,  
                             ... );
```

```
CREATE TABLE Tarifs(  
    idTarif int,  
    hotel int,  
    typeChambre int,  
    DateDebut date,  
    DateFin date,  
    Prix money);;
```

Création d'une table à partir d'une autre

```
CREATE TABLE <Nouvelle table> AS SELECT <nom colonne1>,<nom  
colonne2>... ou <*> FROM <Table à copier> WHERE ... ... ;
```

```
CREATE TABLE SAUV_Chambres AS SELECT idchambre, hotel,  
numchambre FROM Chambres WHERE typechambre = 3;
```

Ajout d'un commentaire sur une table ou sur une colonne

```
COMMENT ON <'COLUMN' ou 'TABLE'> <Nom colonne ou nom de table> IS  
'libellé libre';
```

```
COMMENT ON TABLE Tarifs IS 'Tarif par hotel et type de chambre';
```

```
COMMENT ON COLUMN Hotels.idHotel IS Numéro de l'hotel';
```

Suppression d'une table

```
DROP TABLE nom_de_table;
```

```
DROP TABLE Hotels;
```

Création d'un synonyme

```
CREATE SYNONYM <Nom synonyme> FOR <Nom table>;
```

```
CREATE SYNONYM PRIX FOR Tarifs;
```

Modification d'une colonne ou d'une contrainte d'une table

```
ALTER TABLE nom_de_table [ADD nom_de_colonne Type_colonne]  
                             [,DROP COLUMN nom_de_colonne]  
                             [,ADD CONSTRAINT nom_contrainte]  
                             [,DROP CONSTRAINT nom_contrainte]
```

```
ALTER TABLE Chambres ADD Vue VARCHAR(20);
```

Renommer une table

RENAME nom_de_table_ancien TO nom_de_table_nouveau;

RENAME Chambres TO SAUV_Chambres;

Création d'une séquence

CREATE SEQUENCE <nom séquence>;

CREATE SEQUENCE S_NUMERO START WITH 5 INCREMENT BY 1
MINVALUE 2 MAXVALUE 999999 CYCLE;

Création d'une vue

CREATE VIEW <Nom_Vue> AS SELECT ...

CREATE VIEW V_Chambre_desc AS
SELECT Chambres.Hotel, Chambres.NumChambre,
TypesChambre.TypeLit,
TypesChambre.NombreLit, TypesChambre.Description
FROM Chambres, TypesChambre
WHERE Chambres.TypeChambre = TypesChambre.idTypeChambre;

Suppression d'une vue

DROP VIEW nom_vue

DROP VIEW V_Chambres_desc

Création d'un index

CREATE [UNIQUE] INDEX <nom index> ON <nom table>
<nom colonne 1> [ASC|DESC], <nom colonne 2> [ASC|DESC],

CREATE INDEX I3_TypHotel ON Chambres (TypeChambre, Hotel);

Suppression d'un index

DROP INDEX <nom_index>

2. Les principaux ordres du LMD (langage de manipulation de données) ou DML (Data Manipulation Language)

L'ordre SELECT : sélectionner des données dans une ou plusieurs tables

SELECT <nom colonne1>, <nom colonne 2> ,
[SUM/COUNT/AVG/MIN/MAX](<nom colonne 3>), [SUM/COUNT/AVG/MIN/MAX]
(<nom colonne 4>)
FROM <table 1> , <table 4> ,
JOIN <table 2> ON <table1.colonne1> = <table2.colonne1>
JOIN <table 3> ON <table1.colonne2> = <table3.colonne2>
WHERE
GROUP BY <nom colonne1>, <nom colonne 2>
HAVING [SUM/COUNT/AVG/MIN/MAX]<nom colonne 4> <opérateur
arithmétique> <valeur>
ORDER BY <nom colonne1>, <nom colonne 2>

SELECT H.Libelle, TYP.Description, AVG(T.Prix) as moyenne
FROM Tarifs AS T INNER JOIN Hotels AS H ON H.idHotel = T.hotel
INNER JOIN TypesChambre AS TYP ON TYP.idTypeChambre =
T.typeChambre
INNER JOIN Chambres AS CH ON CH.Hotel = H.idHotel AND


```
CH.TypeChambre = TYP.idTypeChambre
WHERE H.idhotel = 2
GROUP BY TYP.Description
HAVING AVG(T.Prix) > 75
ORDER BY moyenne DESC;
```

L'opérateur UNION

```
SELECT <colonne 1>, <colonne 2>, ... FROM <table1>
WHERE ...
UNION
SELECT <colonne 1>, <colonne 2>, ... FROM <table2>
WHERE ...
ORDER BY <colonne 1>, <colonne 2>, ...
```

```
SELECT Etoile FROM Hotels_FR
UNION
SELECT Etoile FROM Hotels_GB;
```

L'opérateur INTERSECT

```
SELECT <colonne 1>, <colonne 2>, ... FROM <table1>
WHERE ...
INTERSECT
SELECT <colonne 1>, <colonne 2>, ... FROM <table2>
WHERE ...
ORDER BY <colonne 1>, <colonne 2>, ...
```

```
SELECT Libelle, Etoile FROM Hotels_FR
INTERSECT
SELECT Libelle, Etoile FROM Hotels_GB;
```

L'opérateur EXCEPT

```
SELECT <colonne 1>, <colonne 2>, ... FROM <table1>
WHERE ...
EXCEPT
SELECT <colonne 1>, <colonne 2>, ... FROM <table2>
WHERE ...
ORDER BY <colonne 1>, <colonne 2>, ...
```

```
SELECT Libelle, Etoile FROM Hotels_FR
EXCEPT
SELECT Libelle, Etoile FROM Hotels_GB;
```

L'ordre INSERT

```
INSERT INTO <nom table> (<nom colonne1>, <nom colonne 2> ... ..)
VALUE (<valeur 1>, <valeur 2> ... ..)
ou
INSERT INTO <nom table> (<nom colonne1>, <nom colonne 2> ... ..)
SELECT <nom colonne1>, <nom colonne 2> ... ..
FROM <table 2>
[JOIN <table 3> ON <table2.colonne1> = <table3.colonne1>]
[JOIN <table 4> ON <table2.colonne2> = <table4.colonne2>]
[WHERE ... .. ]
[GROUP BY <nom colonne1>, <nom colonne 2> ... .. ]
[HAVING [SUM/COUNT/AVG/MIN/MAX]<nom colonne 4> <opérateur
arithmétique> <valeur> ... .. ]
```

[ORDER BY <nom colonne1>, <nom colonne 2> ...]

INSERT INTO Tarifs

(idTarif, hotel, typeChambre, DateDebut, DateFin, Prix)

VALUES

(58, 1, 2, '2022-04-15', '2022-09-30', 69.99);

L'ordre DELETE : suppression d'une table

DELETE FROM <nom table> WHERE ...

DELETE FROM Chambres

WHERE idChambre > 28;

L'ordre TRUNCATE : vidage d'une table

TRUNCATE TABLE <nom table>;

TRUNCATE TABLE Chambres;

Jointure interne

SELECT <colonne 1>, <colonne 2>, etc ...

FROM <table gauche> [INNER] JOIN <table droite 1> ON <critères 1>,

[INNER] JOIN <table droite 2> ON <critères 2>,

etc ...

ou

SELECT <colonne 1>, <colonne 2>, etc ...

FROM <table gauche>, <table droite 1>, <table droite 2> etc ...

WHERE <critères 1> AND <critères 2> etc ...

SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre

FROM Tarifs T INNER JOIN Hotels H ON H.idHotel = T.hotel

INNER JOIN TypesChambre TYP ON TYP.idTypeChambre =

T.typeChambre

INNER JOIN Chambres CH ON CH.Hotel = H.idHotel AND

CH.TypeChambre

= TYP.idTypeChambre;

Jointure externe

SELECT <colonne 1>, <colonne 2>, etc ...

FROM <table gauche> [LEFT|RIGHT|FULL] [OUTER] JOIN <table droite

1> ON <critères>,

[LEFT|RIGHT|FULL] [OUTER] JOIN <table droite

2> ON <critères>,

etc ...

SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre

FROM Tarifs T LEFT OUTER JOIN Hotels H ON H.idHotel = T.hotel

LEFT OUTER JOIN TypesChambre TYP ON TYP.idTypeChambre = T.typeChambre

LEFT OUTER JOIN Chambres CH ON CH.Hotel = H.idHotel AND CH.TypeChambre =

TYP.idTypeChambre;

Jointure naturelle

SELECT <colonne 1>, <colonne 2>, etc ...

FROM <table gauche> NATURAL JOIN <table droite 1>,

NATURAL JOIN <table droite 2>,

etc ...

SELECT H.Libelle, TYP.Description, T.Prix, CH.NumChambre

```
FROM Tarifs T NATURAL JOIN Hotels H
NATURAL JOIN TypesChambre TYP
NATURAL JOIN Chambres CH;
```

Jointure croisée

```
SELECT <colonne 1>, <colonne 2>, etc ...
FROM <table gauche> CROSS JOIN <table droite 1>,
      CROSS JOIN <table droite 2>,
      etc ...
```

ou

```
SELECT <colonne 1>, <colonne 2>, etc ...
FROM <table gauche>, <table droite 1>, <table droite 2> etc ...
```

```
SELECT Libelle, description FROM hotels CROSS JOIN typeschambre;
```

3. Les principaux ordres du LCD (langage de contrôle des données) ou DCL (Data Control Language)

Création d'un utilisateur (Oracle)

```
CREATE <user> IDENTIFIED BY <Mot de passe>;
```

```
CREATE USER ASMITH IDENTIFIED BY ASMITH;
```

Création d'un utilisateur (MySQL)

```
GRANT ALL PRIVILEGES ON *.* TO '<user>'@'localhost'
IDENTIFIED BY '<Mot de passe>' WITH GRANT OPTION;
```

Changer un mot de passe (Oracle)

```
ALTER USER <user> IDENTIFIED BY <Nouveau Mot de passe>;
```

```
ALTER USER ASMITH IDENTIFIED BY ABCD12E;
```

Changer un mot de passe (MySQL)

```
SET PASSWORD FOR '<user>'@'<host>' = PASSWORD('mypass');
```

Attribuer des droits

```
GRANT <droit1>, <droit2>, ...
ON TABLE <nom table>
TO <user1>, <user2> ...
[ WITH GRANT OPTION];
```

```
GRANT SELECT ON TABLE TARIFS TO ASMITH;
```

Attribuer tous les droits (MySQL)

```
GRANT ALL PRIVILEGES
ON TABLE <table1>, <table2>, ...
[ WITH GRANT OPTION] TO <user1>, <user2> ...
```

Attribuer tous les droits (Oracle)

```
GRANT ALL PRIVILEGES
ON <table1>, <table2>, ...
[ WITH GRANT OPTION] TO <user1>, <user2> ...
```

```
GRANT ALL PRIVILEGES ON TARIFS TO ASMITH;
GRANT ALL PRIVILEGES ON HOTELS TO ASMITH;
GRANT ALL PRIVILEGES ON TYPESCHAMBRE TO ASMITH;
```

Supprimer des droits

```
REVOKE <droit1>, <droit2>, ...  
ON TABLE <nom table>  
FROM <user1>, <user2> ...;  
REVOKE ALL PRIVILEGES ON UTILISATEURS FROM ASMITH;
```

Créer un rôle

```
CREATE ROLE <nom rôle>;
```

Exemple Oracle

```
CREATE ROLE CONTROLE_GESTION;
```

Attribuer un rôle aux utilisateurs

```
GRANT CONTROLE_GESTION TO ASMITH;  
GRANT CONTROLE_GESTION TO BMARTIN;  
etc ...
```

Supprimer un rôle

```
DROP ROLE <nom rôle>;
```

Exemple Oracle

```
DROP ROLE CONTROLE_GESTION;
```

4. Les principaux ordres du LCT (langage de contrôle des transactions) ou TCL (Transaction Control Language)

Mise en œuvre d'un mode de verrouillage au niveau d'une session

Syntaxe MySQL

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL  
{ READ UNCOMMITTED  
| READ COMMITTED  
| REPEATABLE READ  
| SERIALIZABLE  
};
```

Syntaxe Oracle

```
SET TRANSACTION ISOLATION LEVEL  
{ READ COMMITTED  
| SERIALIZABLE  
};
```

Mise en œuvre d'un mode de verrouillage au niveau des tables

Syntaxe MySQL

```
SELECT ... FROM .... WHERE ....  
FOR UPDATE;
```

ou

```
SELECT ... FROM .... WHERE ....  
LOCK IN SHARE MODE;
```

Syntaxe Oracle

```
SELECT ... FROM .... WHERE ....  
FOR UPDATE;
```

Valider des modifications dans une base

```
COMMIT;
```

Annuler des modifications dans une base

```
ROLLBACK TO [SAVEPOINT] <nom du point de contrôle>;
```

Poser un point de synchronisation

```
SAVEPOINT <nom du point de contrôle>;
```

Exemple

SAVEPOINT AVANT_INSERT;

5. La création de procédures et de fonctions

Création d'une procédure stockée

```
CREATE OR REPLACE PROCEDURE <nom procédure>
([<variable entrée 1> IN <format>,
 <variable entrée 2> IN <format>,
 ... ...]
 <variable sortie> OUT <format>)
IS
BEGIN
... ...

[EXCEPTION
... ...
]
END;
```

Création d'une fonction stockée

```
CREATE OR REPLACE FUNCTION <nom fonction>
([<variable entrée 1> IN <format>,
 <variable entrée 2> IN <format>,
 ... ...])
RETURN <format>
IS
<variable sortie> <format>;
BEGIN
... ...

[EXCEPTION
... ...
]
END;
```

Création d'un package

```
CREATE OR REPLACE PACKAGE <nom package> IS
    PROCEDURE <nom procédure 1>;
    FUNCTION <nom fonction 1> (<variable 1> IN <format>) RETURN
<format>;
END;
/
CREATE OR REPLACE PACKAGE BODY <nom package> IS

    FUNCTION <fonction 1>
    ... ...
END;

    PROCEDURE <procédure 1> IS
    ... ...
```

END;
END;
/

Syntaxe pour la compilation d'une procédure, d'une fonction ou d'un package

ALTER <'PROCEDURE' ou 'FUNCTION' ou 'PACKAGE'> <Nom procédure ou fonction ou package> COMPILE;

Syntaxe pour la suppression d'une procédure, d'une fonction ou d'un package

DROP <'PROCEDURE' ou 'FUNCTION' ou 'PACKAGE'> <Nom procédure ou fonction ou package>;

Fonctions SQL présentées dans ce livre

COUNT	Compter des lignes
SUM	Sommer des valeurs
MAX ET MIN	Valeurs maximum et minimum
AVG	Moyenne
ABS	Valeur absolue
ASCII	Valeur ASCII d'un caractère
COS	Cosinus
SIN	Sinus
MOD(<colonne>,<valeur>)	Modulo
ROUND(<colonne>,[<précision>])	Arrondi
SQRT	Racine carrée
IN - NOT IN	Dans la liste ou pas
EXISTS - NOT EXISTS	Existence ou non
BETWEEN	Rechercher entre deux valeurs
LIKE	Rechercher les éléments qui contiennent une partie de valeur
CURRENT_DATE	Date du jour
CURRENT_TIMESTAMP	Date et heure du jour
LOWER / UPPER / UCASE / LCASE	Transformer les caractères en minuscules et majuscules
TRIM / LTRIM / RTRIM	Supprimer les blancs à droite ou gauche d'une chaîne de caractères
TO_CHAR	Transformer un numérique ou une date en caractère
INSTR	Trouver la position d'une chaîne de caractères dans chaîne
LPAD / RPAD	Ajouter des caractères avant ou après une chaîne
SUBSTR	Extraire une partie d'une chaîne de caractères
NVL	Tester une colonne à null
NULLIF	Comparer deux colonnes
COALESCE	Tester plusieurs valeurs
CAST	Changer le type d'une colonne
DECODE	Conditionner et tester les données
CASE	Conditionner et tester les données

Glossaire

SQL	<i>Structured Query Language</i>
DDL	<i>Data Definition Language</i>
LDD	Langage de définition de données
DML	<i>Data Manipulation Language</i>
LMD	Langage de manipulation de données
DCL	<i>Data Control Language</i>
LCD	Langage de contrôle des données
TCL	<i>Transaction Control Language</i>
LCT	Langage de contrôle des transactions
BDD ou BD	Base de données
SGBD	Système de gestion de bases de données
SGBDR	Système de gestion de bases de données relationnelles
DBA	<i>DataBase Administrator</i>
MOA	Maîtrise d'ouvrage
MOE	Maîtrise d'œuvre