

SQL

Dominique Gonzalez
Université Lille3-Charles de Gaulle

SQL

par Dominique Gonzalez

Publié mardi 7 septembre 2010 à 10h21

Copyright © 2010 D.Gonzalez

Ce document est soumis à la licence GNU FDL. Permission vous est donnée de distribuer, modifier des copies de ces pages tant que cette note apparaît clairement.

Table des matières

1. Pourquoi et comment ?	1
1.1. Pourquoi ce document ?	1
1.2. Comment a-t-il été construit ?	1
1.3. QBullets.....	1
2. Qu'est-ce que SQL ?	3
2.1. Avant-propos.....	3
2.2. Intérêt de SQL	3
2.3. SQL dans l'architecture en couches des SGBD.....	3
2.4. SQL : Principes d'une base de données relationnelle.....	4
2.5. Architecture client-serveur et communication par SQL	5
2.5.1. Bases de données en client-serveur	6
2.5.2. Les serveurs de transactions	6
2.6. Structure générale du langage SQL	6
2.7. SQL : un langage algébrique.....	7
3. Une base jouet pour découvrir SQL	9
3.1. Les tables.....	9
3.2. Quelques requêtes	9
4. Présentation	11
4.1. Fonctionnalités générales	11
4.2. Utilisation de cette base	11
5. La base de données	13
5.1. MLD.....	13
5.2. Contenu de la base	13
5.2.1. La table ARTISTE	13
5.2.2. La table MEMBRE	14
5.2.3. La table DISQUE	14
5.2.4. La table GENRE	15
5.2.5. La table ETAT.....	16
5.2.6. La table CHANSON	16
5.2.7. La table LANGUE	17
5.2.8. La table LANGUEARTISTE	17
5.2.9. La table LANGUECHANSON	17
5.2.10. La table INTERPRETE.....	18
5.2.11. La table ECRIT	18
5.2.12. La table DROITS	19
5.2.13. La table AUTORISATIONS	19
6. La commande SELECT, syntaxe de base	21
6.1. SELECT	21
6.2. Suppression des doublons	21
6.3. Restriction.....	22
6.4. Restriction en comparant les colonnes entre elles	22
6.5. Négation, recherche approchée	22
6.6. Tris	22
6.7. Valeurs non renseignées	23
6.8. Expressions arithmétiques	23
6.9. Éviter les valeurs NULL	23
6.10. Arrondis.....	24
6.11. Concaténation	24
6.12. Opérateur conditionnel	24
6.13. Chaînes de caractères.....	24
6.14. Opérations ensemblistes.....	25
7. Les jointures et les sous-requêtes	27
7.1. Jointures	27
7.2. Auto-jointures	27
7.3. Sous-requêtes	27
7.4. LEFT JOIN, RIGHT JOIN.....	28

8. Les dates	31
8.1. Généralités.....	31
8.2. Affichage d'une date.....	31
8.3. Exercices.....	32
9. Les groupes	33
9.1. Utilisation de fonctions de groupe.....	33
9.2. Les groupes.....	33
9.3. La clause <code>HAVING</code>	33
9.4. Exercices.....	34
10. Exercices récapitulatifs	35
11. Réponses aux premiers exercices sur la base <i>jouet</i>	37
12. Réponses aux premiers exercices sur la commande <code>SELECT</code>	39
13. Réponses aux exercices sur les jointures et les sous-requêtes	45
14. Réponses aux exercices sur les dates	51
15. Réponses aux exercices sur les groupes	53
16. Réponses aux exercices récapitulatifs	55
Index	61

Chapitre 1. Pourquoi et comment ?

1.1. Pourquoi ce document ?

Ces pages ont pour origine un cours destiné aux étudiants de 2^{ème} année de l'IUP IIES de l'université de Lille III-Charles de Gaulle, à Villeneuve d'Ascq, pour les années universitaires 2002-2003 et 2003-2004.

Elles ont ensuite été remaniées et augmentées pour un cours destiné aux étudiants de 3^{ème} année de la licence MIASHS.

Ces pages ne sont pas destinées à être un cours autonome : elles ne sont qu'un support de cours, et beaucoup de choses, qui sont transmises à l'oral pendant les cours, ne sont pas écrites.

L'environnement technique du cours est constitué de machines sous `linux`, les bases de données étant hébergées sur un serveur `PostgreSQL`, lui aussi sous `linux`. L'installation de ces logiciels ne sera pas abordée et ne fait pas partie du contenu du cours.

On n'abordera pas non plus la partie *analyse* (Merise, UML, etc.) qui est pourtant fondamentale dans le processus de création d'une base données. Cette partie sera vue ailleurs. Ce cours se limite à la découverte de SQL.

1.2. Comment a-t-il été construit ?

Ce polycopié a été rédigé au format `DocBook` :

- Le texte source a été écrit au format XML avec `emacs` et `Quanta`, en respectant la DTD de `DocBook`.
- Le code source a été compilé au format PDF avec `openjade`, et au format HTML avec `xsltproc`.
- La version que vous avez devant les yeux a été compilée le mardi 7 septembre 2010 à 10h21 .

1.3. QBullets

Les petites images animées qui illustrent les liens de la version web de ce document proviennent de QBullets



1. <http://www.matterform.com/>

Chapitre 1. Pourquoi et comment ?

Chapitre 2. Qu'est-ce que SQL ?

2.1. Avant-propos

Ce premier chapitre est extrait d'un polycopié réalisé par Rémi Gilleron et Marc Tommasi dans le cadre de leur cours sur l'architecture client-serveur.

Je me suis contenté d'en adapter la mise en page.

Vous pouvez consulter l'original sur internet¹.

Ce polycopié était lui-même fortement inspiré de l'excellent ouvrage « SQL 2 », de Christian Marée et Guy Ledant, chez Armand Colin.

2.2. Intérêt de SQL

Tous les systèmes de gestion de données utilisent SQL pour l'accès aux données ou pour communiquer avec un serveur de données. SQL (Standard Query Language) est né à la suite des travaux mathématiques de Codd, travaux qui ont fondé les bases de données relationnelles. SQL, défini d'abord chez IBM, a subi trois tentatives de normalisation en 86, 89 et 92 (SQL 2 ou SQL 92). Nous présentons trois raisons fondamentales qui justifient l'utilisation de SQL.

- D'une part, la structuration et la manipulation des données sont devenues très complexes. Pour une application de taille moyenne, la base de données contient fréquemment plus de trente tables fortement interconnectées. Il est donc hors de question de manipuler les données de façon algorithmique traditionnelle. Une requête SQL dans un langage logique simple remplace donc bien avantageusement plusieurs dizaines de lignes d'un langage de programmation tel C ou COBOL.
- D'autre part, l'architecture client-serveur est omniprésente. Tandis que la station client exécute le code de l'application en gérant, en particulier, l'interface graphique, le serveur optimise la manipulation et le contrôle des données. De plus, les applications doivent être portables et gérer des données virtuelles, c'est-à-dire émanant de n'importe quel serveur. Développer une application dans un environnement hétérogène n'est possible que parce que la communication entre l'applicatif client et le serveur est réalisée par des primitives SQL normalisées.
- Enfin, les applications à développer (même sur un PC) sont devenues de plus en plus complexes. Le profil du programmeur a fortement changé. Il doit maintenant traiter des données de plus en plus volumineuses, intégrer les techniques de manipulation des interfaces, maîtriser la logique événementielle et la programmation orientée objet, tout cela dans un contexte d'architecture client-serveur où se côtoient les systèmes d'exploitation et les protocoles de réseaux hétérogènes. L'accès et la manipulation des données ne sont que l'un des aspects de la conception et de la réalisation de programmes. On cherche donc à acquérir un environnement de développement performant qui prend en charge un grand nombre de tâches annexes. Des outils de développement sont apparus pour permettre au développeur de se concentrer sur l'application proprement dite : générateurs d'écrans, de rapports, de requêtes, d'aide à la conception de programme, de connexion à des bases de données distantes via des réseaux. Dans tous ces outils, la simplicité et la standardisation de SQL font que SQL est utilisé chaque fois qu'une définition, une manipulation, ou un contrôle de données est nécessaire. SQL est donc un élément central entre les divers composants d'un environnement de développement dans une architecture client-serveur.

2.3. SQL dans l'architecture en couches des SGBD

Dans la phase d'analyse de systèmes d'information, on considère différents niveaux d'abstraction du système d'information : le niveau conceptuel, le niveau logique ou organisationnel et enfin, le niveau physique ou opérationnel.

Nous allons considérer ici différents niveaux de perception d'une base de données relationnelle. Un SGBD est fréquemment décrit par une structure en couches correspondant à des perceptions différentes des données, associées à des tâches différentes pour différents acteurs.

1. <http://grappa.univ-lille3.fr/polys/frime>

Pour plus de simplicité, nous distinguerons trois types d'acteurs : les administrateurs de la base de donnée, les développeurs et les utilisateurs. Au niveau externe, proche de l'utilisateur, la perception est totalement indépendante du matériel et des techniques mises en oeuvre, tandis qu'au niveau le plus intérieur, se trouvent les détails de l'organisation sur disque et en mémoire.

Le *schéma logique* est l'ensemble de toutes les données pertinentes, toutes applications confondues. Il est rendu conforme à un modèle de représentation des données, et est totalement indépendant de la technologie utilisée. Nous choisissons le modèle relationnel. Ce niveau a un inconvénient : toutes les données sont accessibles à tout le monde. Cet ensemble vaste de données est trop touffu. Il est préférable de montrer à l'utilisateur (et au programmeur) une vue plus simple des données.

On constitue ainsi des *schémas externes*. Par exemple, le gestionnaire du stock n'est concerné que par les données décrivant les articles en stock. S'il ne manipule que des bordereaux d'entrée, des bordereaux de sortie, et des fiches d'état du stock, ceux-ci constituent son schéma de perception externe.

Le *schéma interne* fournit une perception plus technique des données.

Enfin le *schéma physique* est dépendant du matériel et du logiciel de base.

Niveau externe

Au niveau externe, les utilisateurs et développeurs d'application ont une perception limitée de la base de données. On parle de *vue*. Une vue peut être considérée comme une restriction du schéma logique à un type d'utilisateur. Ce niveau concerne les utilisateurs et les développeurs.

Niveau logique

Traduction dans le modèle relationnel du schéma conceptuel. On précise à ce niveau les tables, les relations entre tables, les contraintes d'intégrité, les vues et les droits par groupe d'utilisateurs. Ce niveau concerne l'administrateur et les développeurs.

Niveau interne

On définit les index et tous les éléments informatiques susceptibles d'optimiser les ressources et les accès aux données. Ce niveau concerne l'administrateur.

Niveau physique

On y précise tout ce qui dépend du matériel et du système d'exploitation. Ce niveau concerne l'administrateur.

Cette découpe en niveaux présente les avantages suivants :

- Les applications développées sont indépendantes du niveau interne. Tout changement de stratégie d'accès, ou d'organisation des données entraîne une modification au niveau interne, mais le schéma logique reste inchangé. Par exemple, une requête SQL précise le QUOI sans se préoccuper du COMMENT.
- La distinction externe/logique assure (en partie) l'indépendance entre les applications et le niveau logique. Par exemple on peut enrichir le schéma logique sans modifier les applications existantes pour toutes les vues non concernées par les modifications apportées au schéma logique.
- La distinction logique/interne permet de modifier les optimisations d'accès aux données. Par exemple, si une application a des performances insuffisantes, il est possible d'optimiser les accès (en introduisant de nouveaux index, par exemple) et d'augmenter les performances sans modifier l'application.
- La distinction interne/physique permet une meilleure portabilité car seule la partie physique est dépendante du matériel et du système d'exploitation.

2.4. SQL : Principes d'une base de données relationnelle

Le terme *relationnel* provient de la définition mathématique d'algèbre relationnelle (Codd 70). Une relation est un ensemble de tuples (tuple est la généralisation de couple à un nombre quelconque d'éléments. Les couples, les triplets, les quadruplets, etc. sont des tuples) de données, on peut alors définir des opérations algébriques sur les relations (voir la section Section 2.7). Nous parlerons dans la suite de *table*, et non pas de *relation*. Une *table* est un ensemble de tuples de données distincts deux à deux. Une *table* est constituée d'une *clef primaire* et de plusieurs *attributs* (ou colonnes) qui dépendent de cette clef. La clef primaire d'une table est un attribut ou un groupe d'attributs de la table qui détermine tous les autres de façon unique. Une

table possède toujours une et une seule clef primaire. Par contre, une table peut présenter plusieurs clefs candidates qui pourraient jouer ce rôle. Le *domaine d'un attribut* est l'ensemble des valeurs que peut prendre cet attribut. Le domaine est constitué d'un type, d'une longueur et de contraintes qui réduisent l'ensemble des valeurs permises. Une clef étrangère dans une table est une clef primaire ou candidate dans une autre table. Les *contraintes d'intégrité* font partie du schéma logique. Parmi celles-ci, on distingue :

- les *contraintes de domaine* qui restreignent l'ensemble des valeurs que peut prendre un attribut dans une table,
- les *contraintes d'intégrité d'entité* qui précisent qu'une table doit toujours avoir une clef primaire et
- les *contraintes d'intégrité référentielle* qui précisent les conditions dans lesquelles peuvent être ajoutés ou supprimés des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clefs étrangères.

Exemple 2-1. Contraintes d'intégrité

```
clients (cltnum, cltnom, cltpnom, cltloc, cltca, clttype)
commandes (cmdnum, cmdcldt, cmddate, cmdvnd)
ligcommandes (ldccmd, ldcart, ldcqte)
articles (artnum, artnom, artpv, artcoult)
```

Les identifiants sont en **gras**, les clefs étrangères en *italiques*. Une contrainte d'intégrité référentielle est, par exemple, l'obligation de la présence d'un client pour une commande. C'est-à-dire encore qu'à un enregistrement dans la table `commandes` doit correspondre un enregistrement de la table `clients` tel que `commandes.cmdcldt=clients.cltnum`.

2.5. Architecture client-serveur et communication par SQL

Nous allons d'abord voir les différents types d'application possibles pour la gestion de données distantes en commençant par les trois formes les plus simples.

Monoposte

La base de données se trouve sur un poste et n'est pas accessible en réseau. Il faut, dans ce cas, penser à la notion de sécurité si plusieurs utilisateurs peuvent interroger la base (suppression accidentelle d'enregistrements).

Multiposte, basée sur des terminaux liés à un site central

C'est l'informatique multiposte traditionnelle. La gestion des données est centralisée. Les applications ont été écrites par le service informatique et seules ces applications peuvent interroger le serveur.

Multiposte, basée sur un serveur de fichiers

C'est la première forme (la plus simple) d'architecture client-serveur. Si l'applicatif sur un PC souhaite visualiser la liste des clients habitant Paris, tous les enregistrements du fichier CLIENT transitent sur le réseau, entre le serveur et le client, la sélection (des clients habitant Paris) est faite sur le PC. Le trafic sur le réseau est énorme et les performances se dégradent lorsque le nombre de clients augmente. Les serveurs de fichiers restent très utilisés comme serveurs d'images, de documents, d'archives.

De nouveaux besoins sont apparus :

- diminuer le trafic sur le réseau pour augmenter le nombre de postes sans nuire au fonctionnement,
- traiter des volumes de données de plus en plus grand,
- accéder de façon transparente à des données situées sur des serveurs différents,
- accéder aux données de façon ensembliste, même si les données sont distantes, afin de diminuer le travail du programmeur,
- adopter des interfaces graphiques de type Windows pour les applicatifs clients.

2.5.1. Bases de données en client-serveur

Dans une architecture client-serveur, un applicatif est constitué de trois parties : l'interface utilisateur, la logique des traitements et la gestion des données. Le client n'exécute que l'interface utilisateur et la logique des traitements, laissant au serveur de bases de données la gestion complète des manipulations de données.

- Le serveur de bases de données fournit des services aux processus clients. Les tâches qu'il doit prendre en compte sont : la gestion d'une mémoire cache, l'exécution de requêtes exprimées en SQL, exécuter des requêtes mémorisées, la gestion des transactions, la sécurité des données.
- Le client doit ouvrir une connection pour pouvoir profiter des services du serveur. Il peut ouvrir plusieurs connections simultanées sur plusieurs serveurs. Le client peut soumettre plusieurs requêtes simultanément au serveur et traiter les résultats de façon asynchrone.
- Communication entre le client et le serveur. Puisque l'application doit pouvoir se connecter à divers serveurs de façon transparente, le langage de communication SQL doit être compatible avec la syntaxe SQL de chaque serveur pressenti. Or, malgré les normes, les dialectes SQL sont nombreux et parfois source d'incompatibilité. La seule façon de permettre une communication plus large est d'adopter un langage SQL standardisé de communication. Une couche fonctionnelle du client traduit les requêtes du dialecte SQL client en SQL normalisé. La requête transformée est envoyée au serveur. Celui-ci traduit la requête dans le dialecte SQL-serveur et l'exécute. Le résultat de la requête suit le chemin inverse. Le langage de communication normalisé le plus fréquent est l'ODBC (Open DataBase Connectivity) de Microsoft. Signalons également IDAPI (Integrated Database Application Programming Interface) de Borland. De plus, ces programmes permettent d'interroger des bases de données autres que relationnelles.

2.5.2. Les serveurs de transactions

Une transaction correspond à une procédure SQL, i.e. un groupe d'instructions SQL. Il y a un seul échange requête/réponse pour la transaction. Le succès ou l'échec concerne l'ensemble des instructions SQL. Ces serveurs sont essentiellement utilisés pour l'informatique de production (ou opérationnelle) pour laquelle la rapidité des temps de réponse est importante sinon essentielle.

2.6. Structure générale du langage SQL

Les instructions essentielles SQL se répartissent en trois familles fonctionnellement distinctes et trois formes d'utilisation :

Selon la norme SQL 92, le SQL *interactif* permet d'exécuter une requête et d'en obtenir immédiatement une réponse. Le SQL *intégré* (ou module SQL) permet d'utiliser SQL dans un langage de troisième génération (C, Cobol, ...), les instructions permettant essentiellement de gérer les curseurs. Le SQL *dynamique* est une extension du SQL intégré qui permet d'exécuter des requêtes SQL non connues au moment de la compilation. Hors norme, SQL est utilisable sous la forme de bibliothèques de fonctions API (exemple : ODBC). Nous nous limitons au SQL interactif.

Dans le SQL interactif, le LDD (Langage de Définition de données) permet la description de la structure de la base (tables, vues, index, attributs, ...). Le dictionnaire contient à tout moment le descriptif complet de la structure de données. Le LMD (Langage de Manipulation de Données) permet la manipulation des tables et des vues. Le LCD (Langage de Contrôle des Données) contient les primitives de gestion des transactions et des privilèges d'accès aux données. Le tableau ci-dessous vous donne les principales primitives SQL et leur classification. Nous nous intéresserons essentiellement au LMD et à la commande *SELECT*. Nous étudierons également quelques problèmes concernant les privilèges d'accès et les transactions.

Tableau 2-1. SQL interactif

LDD	LMD	LCD
create	select	grant

LDD	LMD	LCD
drop	insert	revoke
alter	delete	connect
	update	commit
		rollback
		set

2.7. SQL : un langage algébrique

Pour bien comprendre le langage SQL, nous allons brièvement exposer les principes sur lesquels repose ce langage (algèbre relationnelle).

Une table (relation) est un ensemble de tuples. On peut donc appliquer à une table les opérateurs algébriques usuels. Le résultat d'une opération ou requête est une nouvelle table qui est exploitable à son tour dans une nouvelle opération. Tous les opérateurs peuvent être dérivés de cinq primitives de base : la *projection*, la *sélection*, l'*union*, la *différence* et le *produit*. L'opérateur de *jointure* qui peut être déduit des cinq primitives de base est cependant fondamental (algèbre relationnelle).

- La projection permet de ne conserver que les attributs intéressants d'une table (sélection verticale). De plus, la projection élimine les répétitions de tuples résultant de cette sélection.

Exemple 2-2. Projection

```
CLIENTS2 = PROJECT CLIENTS OVER (cltnom, cltloc)
```

- La sélection permet de ne conserver que les tuples qui respectent une condition définie sur les valeurs des attributs (sélection horizontale).

Exemple 2-3. Sélection

```
BONSCLIENTS = SELECT CLIENTS WHERE cltca>10000
```

- L'union réalise l'union de plusieurs tables.

Exemple 2-4. Union

```
CLIENTS3 = SELECT CLIENTS WHERE cltloc = 'PARIS'
UNION
SELECT BONSCIENTS WHERE cltloc='BRUXELLES'
```

- La différence consiste à prendre les tuples appartenant à une table mais pas à une autre.

Exemple 2-5. Différence

```
CLIENTS4 = SELECT CLIENTS WHERE cltloc = 'BRUXELLES'
EXCEPT BONSCIENTS
```

- Le produit réalise la juxtaposition de tous les tuples de la première table avec chaque tuple de la seconde. Cela signifie que, si les deux tables ont respectivement M et N tuples, la table résultante aura $M \times N$ tuples. Cette opération présente peu d'intérêt mais combinée avec une sélection, on obtient une opération fondamentale : la jointure.

- La jointure n'est possible que sur deux tables possédant un attribut de domaine commun. Elle consiste à juxtaposer les tuples dont la valeur d'un attribut est égal dans les deux tables. C'est une primitive dérivée car elle peut être définie à l'aide des primitives précédentes (exercice laissé au lecteur).

Exemple 2-6. Jointure

```
CMDCLIENTS = COMMANDES JOIN CLIENTS ON cmdclt=cltnum
```

Les primitives peuvent être combinées pour constituer des requêtes plus élaborées. La séquence d'opérateurs permettant de réaliser une requête élaborée devient assez vite complexe. Le langage SQL permet (heureusement) d'exprimer globalement une requête sans faire apparaître les tables et les primitives intermédiaires. Ce sera le moteur SQL qui sera chargé d'optimiser la requête.

Exemple 2-7. Requête

Une requête SQL qui permet de dresser la liste des noms des clients qui ont acheté des articles de moins de 200F est :

```
SELECT DISTINCT cltnom
FROM clients, commandes, ligcommandes, articles
WHERE cltnum=cmdclt
AND artnum=ldcart
AND cmdnum=ldccmd
AND artpv < 200
```

Une combinaison de primitives permettant d'exécuter cette requête est :

```
temp1 = SELECT articles WHERE artpv < 200
temp2 = PROJECT temp1 OVER (artnum)
temp3 = PROJECT clients OVER (cltnum,cltnom)
temp4 = PROJECT commandes OVER (cmdnum, cmdclt)
temp5 = PROJECT ligcommandes OVER (ldccmd,ldcart)
temp6 = temp2 JOIN temp5 ON artnum=ldcart
temp7 = PROJECT temp6 OVER (ldccmd)
temp8 = temp3 JOIN temp4 ON cltnum=cmdclt
temp9 = PROJECT temp8 OVER (cltnom, cmdnum)
temp10 = temp7 JOIN temp9 ON ldccmd=cmdnum
resultat = PROJECT temp10 OVER (cltnom)
```

Une telle séquence n'est, en général, pas unique. La séquence fournie est une des plus efficaces (le lecteur peut s'exercer à en trouver d'autres).

Chapitre 3. Une base *jouet* pour découvrir SQL

3.1. Les tables

Tableau 3-1. La table `UN`

a	b	c
x	m	2
x	n	1
y	m	4
z	p	1

Tableau 3-2. La table `DEUX`

d	e
x	8
y	4
x	1

3.2. Quelques requêtes

Vous trouverez les réponses des exercices au Chapitre 11.

Calculer à la main le résultat des requêtes suivantes.

1. `SELECT * FROM un ;`
2. `SELECT a FROM un ;`
3. `SELECT a FROM un WHERE c=1 ;`
4. `SELECT a FROM un WHERE c=1 OR c=2 ;`
5. `SELECT DISTINCT a FROM un WHERE c=1 OR c=2 ;`
6. `SELECT a FROM un ORDER BY b ;`
7. `SELECT a,e FROM un,deux ;`
8. `SELECT a,e FROM un,deux WHERE c=e ;`

Chapitre 4. Présentation

À partir de maintenant, vous allez utiliser une base de données qui gère le contenu d'une discothèque. Vous manipulerez des informations comme le nom des disques, leurs années de parution, les artistes, les textes des chansons, etc.

4.1. Fonctionnalités générales

- Liste des disques.
- Liste des chansons pour chaque disque.
- Un artiste/groupe pour chaque disque, mais possibilité de préciser des artistes/groupes particuliers pour chaque chanson (compilations, duos, etc.).
- Genres musicaux, langue(s) pour chaque artiste/groupe, langue(s) pour chaque chanson.
- Auteur(s) des chansons.
- Paroles des chansons.
- Appartenance(s) d'un artiste à un groupe.

4.2. Utilisation de cette base

- Obtenir une liste des disques (avec les caractéristiques) et la liste des chansons.
- Obtenir une liste des artistes.
- Rechercher une chanson.
- Quelques statistiques.
- Consultations des paroles des chansons.
- Insertion d'information dans la base (créer un artiste, un genre musical, un disque, une chanson, etc.).
- Gérer les utilisateurs et leurs droits (créer, modifier, supprimer).
- Modifier ou supprimer des données existantes (paroles de chanson, orthographe des noms, etc.).

Chapitre 5. La base de données

L'analyse ne sera pas présentée ici. Nous nous contenterons de donner la structure de la base.

5.1. MLD



5.2. Contenu de la base

Structures et quelques exemples des données figurant dans les différentes tables de la base `Disques`.

Dans toutes les tables suivantes les champs dont le nom se termine par « `_modif` » contiennent la date de dernière modification de l'enregistrement.

5.2.1. La table **ARTISTE**

La table **ARTISTE** contient les artistes utilisés dans la base (interprètes, compositeurs, etc.). Il peut s'agir de personnes, mais aussi de groupes. Les membres du groupes sont identifiés alors grâce à la table **MEMBRE**.

5.2.1.1. Structure de la table **ARTISTE**

```
CREATE TABLE artiste (
    art_num INTEGER NOT NULL PRIMARY KEY,
    art_prenom CHARACTER VARYING,
    art_nom CHARACTER VARYING,
    art_groupe CHARACTER VARYING,
    art_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);
```

5.2.1.2. Quelques enregistrements de la table **ARTISTE**

Tableau 5-1.

Champ	Enreg. n°2	Enreg. n°48	Enreg. n°228
art_num	498	539	187
art_prenom	Brian	Aretha	Keith
art_nom	Eno	Franklin	Richards
art_groupe			& The X-Pensive Winos
art_modif		2009-12-14 00:58:12.061681	2008-06-25 20:23:09.799549

5.2.2. La table **MEMBRE**

La table **MEMBRE** sert à définir une relation d'appartenance entre un artiste au sens propre dans la table **ARTISTE** et un groupe également dans la table **ARTISTE**. Figurent aussi dans cette table les début et fin des appartenances d'un musicien à un groupe. On peut ainsi répertorier plusieurs appartenances successives d'une même personne à un même groupe.

5.2.2.1. Structure de la table **MEMBRE**

```
CREATE TABLE membre (
    mem_membre INTEGER NOT NULL REFERENCES artiste(art_num),
    mem_groupe INTEGER NOT NULL REFERENCES artiste(art_num),
    mem_debut CHARACTER VARYING NOT NULL,
    mem_fin CHARACTER VARYING NOT NULL,
    PRIMARY KEY (mem_membre, mem_groupe, mem_debut, mem_fin));
```

5.2.2.2. Quelques enregistrements de la table **MEMBRE**

Tableau 5-2.

Champ	Enreg. n°6	Enreg. n°11
mem_membre	80	115
mem_groupe	44	93
mem_debut	1962	1976
mem_fin	0	1986

5.2.3. La table DISQUE

Chaque enregistrement correspond à un disque. Champ `dis_cddb` correspond à l'identifiant CDDB¹ ou freedb² du disque. Les champs `dis_artiste` et `dis_etat` font respectivement références aux tables `ARTISTE` et `ETAT`.

5.2.3.1. Structure de la table DISQUE

```
CREATE TABLE disque (
    dis_num INTEGER NOT NULL PRIMARY KEY,
    dis_titre CHARACTER VARYING,
    dis_artiste INTEGER REFERENCES artiste(art_num),
    dis_annee CHARACTER VARYING,
    dis_cddb CHARACTER VARYING,
    dis_anneeachat CHARACTER VARYING,
    dis_prixachat NUMERIC(8,2),
    dis_etat INTEGER REFERENCES etat(eta_num),
    dis_perdu BOOLEAN DEFAULT false,
    dis_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);
```

5.2.3.2. Quelques enregistrements de la table DISQUE

Tableau 5-3.

Champ	Enreg. n°16	Enreg. n°67	Enreg. n°49
<code>dis_num</code>	840	913	893
<code>dis_titre</code>	Les Bons Morceaux	Diabolo Menthe	Dirty Deeds Done Dirt Cheap
<code>dis_artiste</code>	283	556	4
<code>dis_annee</code>	1994	1979	1976
<code>dis_cddb</code>	f20d3712		5f097909
<code>dis_anneeachat</code>	1996	2009	2001
<code>dis_prixachat</code>	13.89	10.54	16.04
<code>dis_etat</code>	3	2	1
<code>dis_perdu</code>		1	
<code>dis_modif</code>	2009-05-29 08:32:27.301759	2009-12-29 10:33:36.441709	2009-03-27 17:11:41.340889

5.2.4. La table GENRE

Table des genres musicaux disponibles.

5.2.4.1. Structure de la table GENRE

```
CREATE TABLE genre (
    gen_num INTEGER NOT NULL PRIMARY KEY,
    gen_nom CHARACTER VARYING,
    gen_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);
```

5.2.4.2. Quelques enregistrements de la table GENRE

Tableau 5-4.

1. http://fr.wikipedia.org/wiki/Compact_Disc_Data_Base
 2. <http://fr.wikipedia.org/wiki/Freedb>

Champ	Enreg. n°1	Enreg. n°2	Enreg. n°3
gen_num	0	4	6
gen_nom	Blues	Jazz	New Age
gen_modif	2008-03-12 09:10:55	2008-03-12 09:22:14	2008-03-12 15:58:58

5.2.5. La table ETAT

Table des états possibles pour un disque.

5.2.5.1. Structure de la table ETAT

```
CREATE TABLE etat (
    eta_num INTEGER NOT NULL PRIMARY KEY,
    eta_nom CHARACTER VARYING);
```

5.2.5.2. Quelques enregistrements de la table ETAT

Tableau 5-5.

Champ	Enreg. n°1	Enreg. n°2	Enreg. n°3	Enreg. n°4
eta_num	1	2	3	4
eta_nom	Neuf	Usagé	Endommagé	Détruit

5.2.6. La table CHANSON

Une fiche par chanson. On parle bien de la chanson en tant qu'œuvre, et non d'un enregistrement donné par un artiste donné. Si une chanson a été enregistrée en plusieurs versions, elle ne figurera qu'une fois. Ce sera à la table *INTERPRETE* de faire le lien entre la chanson et toutes ses interprétations.

La champ booléen *cha_libre* indique si les paroles de la chanson sont libres de droits.

5.2.6.1. Structure de la table CHANSON

```
CREATE TABLE chanson (
    cha_num INTEGER NOT NULL PRIMARY KEY,
    cha_titre CHARACTER VARYING,
    cha_genre INTEGER REFERENCES genre(gen_num),
    cha_texte CHARACTER VARYING,
    cha_libre BOOLEAN,
    cha_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);
```

5.2.6.2. Quelques enregistrements de la table CHANSON

Tableau 5-6.

Champ	Enreg. n°1	Enreg. n°16	Enreg. n°128
cha_num	11712	606	289
cha_titre	La nuit, les désirs	Menuetto - allegretto	The Water Is Wide
cha_genre	5	1	2
cha_texte	la nuit les désirs s'ennr[...]		The water is wide an'[...]

Champ	Enreg. n°1	Enreg. n°16	Enreg. n°128
cha_libre			
cha_modif	2009-12-07 00:09:33.161239	2008-09-10 00:06:47.871009	2009-11-24 19:57:39.458133

5.2.7. La table LANGUE

Table des langues disponibles.

5.2.7.1. Structure de la table LANGUE

```
CREATE TABLE langue (
    lan_num CHARACTER VARYING NOT NULL PRIMARY KEY,
    lan_nom CHARACTER VARYING);
```

5.2.7.2. Quelques enregistrements de la table LANGUE

Tableau 5-7.

Champ	Enreg. n°1	Enreg. n°2	Enreg. n°3	Enreg. n°4
lan_num	fr	en	de	be
lan_nom	Français	Anglais	Allemand	Berbère

5.2.8. La table LANGUEARTISTE

Table dont chaque enregistrement fait la liaison entre un artiste et une langue.

5.2.8.1. Structure de la table LANGUEARTISTE

```
CREATE TABLE langueartiste (
    laa_langue CHARACTER VARYING NOT NULL REFERENCES langue(lan_num),
    laa_artiste INTEGER NOT NULL REFERENCES artiste(art_num),
    laa_maternelle BOOLEAN DEFAULT true,
    PRIMARY KEY (laa_langue, laa_artiste));
```

5.2.8.2. Quelques enregistrements de la table LANGUEARTISTE

Tableau 5-8.

Champ	Enreg. n°1	Enreg. n°2	Enreg. n°3	Enreg. n°4
laa_langue	br	fr	en	fr
laa_artiste	288	288	142	77
laa_maternelle	1		1	1

5.2.9. La table LANGUECHANSON

Table faisant la liaison entre une langue et une chanson.

5.2.9.1. Structure de la table LANGUECHANSON

```
CREATE TABLE languechanson (
    lac_langue CHARACTER VARYING NOT NULL REFERENCES langue(lan_num),
    lac_chanson INTEGER NOT NULL REFERENCES chanson(cha_num),
    lac_principale BOOLEAN DEFAULT true NOT NULL,
    PRIMARY KEY (lac_langue, lac_chanson, lac_principale));
```

5.2.9.2. Quelques enregistrements de la table LANGUECHANSON

Tableau 5-9.

Champ	Enreg. n°25	Enreg. n°89	Enreg. n°103
lac_langue	en	fr	en
lac_chanson	4432	1641	11309
lac_principale	1	1	1

5.2.10. La table INTERPRETE

Table indiquant quel artiste interprète quelle chanson. Le champ `int_numero` indique la place de la chanson sur le disque. C'est grâce à ce champ qu'on pourra établir le contenu ordonné d'un disque.

5.2.10.1. Structure de la table INTERPRETE

```
CREATE TABLE interprete (
    int_artiste INTEGER NOT NULL (int_artiste) REFERENCES artiste(art_num),
    int_chanson INTEGER NOT NULL REFERENCES chanson(cha_num),
    int_disque INTEGER NOT NULL REFERENCES disque(dis_num),
    int_numero INTEGER NOT NULL,
    PRIMARY KEY (int_artiste, int_chanson, int_disque, int_numero));
```

5.2.10.2. Quelques enregistrements de la table INTERPRETE

Tableau 5-10.

Champ	Enreg. n°56	Enreg. n°125	Enreg. n°200
int_artiste	44	72	97
int_chanson	11176	1644	2327
int_disque	869	148	214
int_numero	10	1	6
int_remarque			

5.2.11. La table ECRIT

Table indiquant quel artiste a écrit quelle chanson. On ne préoccupe pas de distinguer compositeur et parolier.

5.2.11.1. Structure de la table ECRIT

```
CREATE TABLE ecrit (
    ecr_artiste INTEGER NOT NULL REFERENCES artiste(art_num),
```

```

ecr_chanson INTEGER NOT NULL REFERENCES chanson (cha_num) ,
PRIMARY KEY (ecr_artiste, ecr_chanson)
);

```

5.2.11.2. Quelques enregistrements de la table **ECRIT**

Tableau 5-11.

Champ	Enreg. n°48	Enreg. n°52	Enreg. n°3
ecr_artiste	52	52	80
ecr_chanson	1469	1474	1034

5.2.12. La table **DROITS**

Cette table stocke les différents niveaux d'autorisations des utilisateurs de la base. Elle est référencée par la table *AUTORISATIONS*. Ces deux tables n'ont aucun lien avec les autres.

5.2.12.1. Structure de la table **DROITS**

```

CREATE TABLE droits (
    dro_num INTEGER NOT NULL PRIMARY KEY,
    dro_nom CHARACTER VARYING,
    dro_niveau INTEGER);

```

5.2.12.2. Quelques enregistrements de la table **DROITS**

Tableau 5-12.

Champ	Enreg. n°1	Enreg. n°2	Enreg. n°3
dro_num	1	2	3
dro_nom	Anonyme	Membre	Administrateur
dro_niveau	0	10	99

5.2.13. La table **AUTORISATIONS**

Cette table stocke les identifications des utilisateurs autorisés à consulter la base (en lien avec un programme chargé de contrôler les accès), ainsi que leurs niveaux de droits, comme défini dans la table *DROITS*.

5.2.13.1. Structure de la table **AUTORISATIONS**

```

CREATE TABLE autorisations (
    aut_login CHARACTER VARYING NOT NULL PRIMARY KEY,
    aut_motdepasse CHARACTER VARYING,
    aut_nom CHARACTER VARYING,
    aut_prenom CHARACTER VARYING,
    aut_droits INTEGER REFERENCES droits (dro_num));

```

5.2.13.2. Quelques enregistrements de la table AUTORISATIONS

Tableau 5-13.

Champ	Enreg. n°12	Enreg. n°53	Enreg. n°213	Enreg. n°289
aut_login	straibie	jairien	jchopine	rmissanc
aut_motdepasse	yizrou	htsewo	myplsc	sscywf
aut_nom	Traibien	Airien Kompry	Chopine	Nissance
aut_prenom	Samira	Johnny	Justine	Rémi
aut_droits	1	1	1	1

Chapitre 6. La commande `SELECT`, syntaxe de base

Vous trouverez les réponses des exercices au Chapitre 12.

6.1. `SELECT`

La commande de recherche est le verbe `SELECT`.

L'étude de la commande `SELECT` et des différentes clauses et fonctions est faite, pour commencer, en considérant une seule table.

Par la suite, l'utilisation de plusieurs tables dans le même `SELECT`, sera prise en compte.

La syntaxe de base est :

```
SELECT colonnes d'une ou plusieurs tables séparées par « , »  
FROM tables séparées par « , »  
WHERE conditions logiques séparées par « AND » ou « OR »  
ORDER BY colonnes séparées par « , »
```

Le caractère « `*` » permet de demander l'affichage de toutes les colonnes d'une table.

1. Afficher toutes les informations concernant les genres musicaux.
2. Afficher toutes les informations concernant les langues disponibles.

A la place de « `*` » on peut donner la liste des colonnes souhaitées, dans l'ordre souhaité, en écrivant la requête sur une ou plusieurs lignes. Il s'agit de la projection vue précédemment. L'écriture sur plusieurs lignes est conseillée en vue de rendre plus simple la relecture et la modification de la requête.

3. Afficher le nom, le prénom et le login de toutes les personnes autorisées.

Le titre des colonnes obtenues par un `SELECT` sont les noms des champs.

On peut cependant changer ces titres en utilisant `AS`. La commande

```
SELECT aut_login AS identifiant FROM autorisations ;
```

produira le même effet que

```
SELECT aut_login FROM autorisations ;
```

à la différence que la colonne ne sera pas intitulé `aut_login` mais `identifiant`.

Si vous voulez utiliser un titre de colonne qui contient autre chose que des lettres non accentuées minuscules (des espaces, des ponctuations, des accents, des majuscules, etc.) il faut l'entourer par des guillemets (« " »). On écrira ainsi :

```
SELECT art_nom AS "Nom de l'artiste" FROM artiste ;
```

6.2. Suppression des doublons

Il peut être utile de supprimer les doublons, d'où utilisation de la clause `DISTINCT`. Elle a pour effet de n'afficher qu'une seule fois les lignes qui seraient semblables à l'affichage.

On l'utilise sous la forme `SELECT DISTINCT ...`

- Afficher les prénoms de toutes les personnes autorisées.
- Afficher les différentes valeurs des prénoms de toutes les personnes autorisées.

6.3. Restriction

Les clauses de restriction s'écrivent derrière `WHERE`. Elle permettent de sélectionner les lignes à afficher. Il s'agit de la sélection telle que vue précédemment.

- Afficher le titre des disques de 1969 dont le code d'état est 2.
- Afficher le nom et l'identifiant des genres musicaux dont l'identifiant est supérieur à 5.
- Afficher le titre, le prix d'achat et le numéro CDDB des disques de l'année 1981, l'état est 2 et le prix d'achat est supérieur à 15.
- Afficher le titre, l'identifiant CDDB et l'année des disques dont le titre est « IV » ou « Killer » ou « No Control ».
- Afficher le titre, l'identifiant CDDB et l'année des disques de l'année 1971, dont le titre est « IV » ou « Killer ».
- Afficher le titre, l'identifiant CDDB et l'année des disques dont le titre est « IV » ou dont le titre est « Killer » dans l'année 1971.
- Afficher le titre, l'identifiant CDDB et le prix des disques dont le prix est compris entre 13.5 et 14.
- Afficher le nom et le prénom des artistes dont le prénom appartient à la liste : « Alain, Louis, Jean ».

6.4. Restriction en comparant les colonnes entre elles

La restriction peut mettre en jeu la comparaison de deux ou plusieurs colonnes entre elles. Il suffit de les appeler par leurs noms.

- Afficher le nom et le prénom des artistes dont le nom est alphabétiquement antérieur au prénom.

6.5. Négation, recherche approchée

Les opérateurs arithmétiques sont niés avec « ! » : par exemple « non égal » s'écrit « != » (ou « <> »).

Les autres opérateurs sont niés par « NOT » : par exemple « pas dans » s'écrit « NOT IN ».

Les caractères *jokers* sont « % » pour une chaîne et « _ » pour un caractère. On les utilise avec l'opérateur `LIKE`.

- Afficher le nom des artistes commençant par la lettre « H ».
- Afficher le nom des artistes se terminant par la lettre « n ».
- Afficher le nom des artistes contenant la lettre « u » en 3^{ème} position.

6.6. Tris

On peut trier (ordonner) l'affichage du résultat selon une ou plusieurs colonnes, voire selon des calculs faits sur les colonnes. Ceci sera réalisé grâce à `ORDER BY`.

18. Afficher le titre et le prix des disques de l'année 1985 classés par prix croissant.
19. Afficher le titre et le prix des disques de l'année 1985 classés par prix décroissant.
20. Afficher le titre et l'année des disques classés par année décroissante et par titre croissant.

6.7. Valeurs non renseignées

En `SQL` il existe une valeur correspondant à la valeur *vide*. C'est l'équivalent de la valeur `Null` de certains langages de programmation (comme `java` ou `python`). C'est la valeur qu'ont les champs non renseignés.

21. Afficher les artistes dont le nom commence par la lettre « A » par ordre de date de modification croissante.

Si on veut utiliser les valeurs non renseignées dans une restriction ce sera avec les clauses `IS NULL` ou `IS NOT NULL`.

22. Afficher les artistes dont le nom commence par la lettre « A » dont la date de modification n'est pas renseignée.
23. Afficher les artistes dont le nom commence par la lettre « A » dont la date de modification est renseignée.
24. Afficher les artistes dont le nom commence par la lettre « A » dont la dernière modification a été faite après le 1^{er} janvier 2009.
25. Afficher les artistes dont le nom commence par la lettre « A » dont la dernière modification a été faite avant le 1^{er} janvier 2009.

6.8. Expressions arithmétiques

Des expressions arithmétiques¹ peuvent être utilisées après : `SELECT`, `WHERE`, et `ORDER BY`.

26. Afficher le titre, le prix en euros, le prix en francs (1€=6,5FF) de tous les disques achetés avant 2002. On renommera les colonnes.

6.9. Éviter les valeurs `NULL`

Le remplacement *automatique* des valeurs non renseignées (`NULL`) se fait par l'utilisation de la fonction `COALESCE`.

```
COALESCE( arg1, arg2, ...)
```

où :

- `arg1` = colonne qui peut être non renseignée
- `arg2` = valeur ou nom d'une autre colonne de substitution
- ...

Le résultat renvoyé est la première valeur non `NULL`.

1. C'est-à-dire des calculs sur les nombres.

27. Afficher les enregistrements de la table `INTERPRETE` qui concernent l'artiste 170, en remplaçant les remarques non renseignées par la phrase « Pas de commentaire ».

6.10. Arrondis

La fonction `ROUND (arg1, arg2)` permet de faire l'arrondi mathématique de `arg1` avec `arg2` décimales.

La fonction `TRUNC (arg1, arg2)` permet de tronquer `arg1` avec `arg2` décimales.

28. Refaire l'exercice 26 mais en arrondissant à deux décimales les prix en francs. Renommer les colonnes.

6.11. Concaténation

L'opérateur de concaténation de chaînes de caractères est « `||` » (2 caractères « pipe »).

29. Afficher le nom, le prénom et le groupe (concaténés) des artistes. Renommer la colonne.

6.12. Opérateur conditionnel

La fonction de codification du SQL standard n'existe pas en PostgreSQL :

```
DECODE (arg1, arg20,arg21, arg30,arg31, arg40)
```

Mais on dispose de la structure `CASE` qui donne le même résultat (et avec plus de possibilités) :

```
CASE WHEN cond1 THEN arg1 WHEN cond2 THEN arg2 ... ELSE arg3 END
```

- si la condition `cond1` est vérifiée, alors le résultat est `arg1`,
- sinon, si la condition `cond2` est vérifiée, alors le résultat est `arg2`,
- ...
- sinon le résultat est `arg3`.

30. Afficher les titres de chansons commençant par la lettre « A » accompagnés de la phrase « paroles inconnues » ou « paroles disponibles » selon que la champ `cha_texte` est renseigné ou pas.

6.13. Chaînes de caractères

La fonction extraction de chaîne de caractères `SUBSTR (arg1, arg2, arg3)` permet d'extraire de la chaîne `arg1` à partir de la position `arg2`, les `arg3` caractères suivants. Si `arg3` n'est pas précisé, on obtient toute la fin de la chaîne de caractères.

La fonction de retour du rang d'une chaîne dans une autre chaîne, `STRPOS (arg1, arg2)` permet de retourner le rang de la chaîne `arg2` dans la chaîne `arg1`.

Les fonctions `UPPER (arg1)` et `LOWER (arg1)` permettent respectivement de forcer à la majuscule ou à la minuscule.

La fonction `LENGTH (arg1)` permet d'obtenir le nombre de caractères d'une chaîne de caractères.

La fonction `TRIM (arg1)` supprime les espaces en début et fin de la chaîne.

31. Afficher les 5 premières lettres du nom des artistes.
32. Afficher le nom et le rang de la lettre « r » à partir de la 3^{ème} lettre dans le nom des artistes.
33. Afficher le nom, le prénom, le prénom en majuscules et le prénom en minuscules des artistes dont le nom est `Russell`.
34. Afficher le nom et le nombre de caractères du nom des artistes.
35. Afficher proprement les noms complets des artistes (prénom+nom+groupe), le nom de famille étant en majuscules.

6.14. Opérations ensemblistes

Utilisation des opérateurs ensemblistes : `UNION`, `INTERSECT`, `EXCEPT`.

`UNION` seul prend les données distinctes uniquement.

Si on veut toutes les données : `UNION ALL`.

36. Afficher les noms des personnes autorisées et des artistes.
37. Afficher les numéros d'artiste communs à la table `ECRIT` et à la table `INTERPRETE`.
38. Afficher les identifiants des langues qui ne correspondent à aucune chanson.

Chapitre 7. Les jointures et les sous-requêtes

Vous trouverez les réponses des exercices au Chapitre 13.

7.1. Jointures

Dans la table `DISQUES` la colonne `dis_artiste` contient des valeurs qui se trouvent dans la colonne `art_num` de la table `ARTISTE` et inversement.

Logiquement chaque ligne de `DISQUE` correspond à une ligne de `ARTISTE` et chaque ligne `ARTISTE` correspond à une ou plusieurs lignes de `DISQUE`.

Cette correspondance se fait par l'opérateur égalité (« = ») entre les valeurs des deux colonnes des deux tables : c'est une *équijointure*.

Après `FROM` les deux tables sont citées, et le critère d'équijointure sert de restriction. Sans cette restriction c'est un produit cartésien de `ARTISTE` et de `DISQUE` qui serait obtenu.

Pour faciliter l'écriture des requêtes, les tables citées derrière `FROM` peuvent être renommées temporairement, et l'alias peut être utilisé pour préfixer les colonnes.

1. Afficher les *login* des personnes autorisées ainsi que leurs droits en toutes lettres.
2. Afficher les titres des disques accompagnés des noms, prénoms et groupes des artistes. La liste sera classée par ordre alphabétique des artistes.

Si le critère de jointure est omis, le résultat est un produit cartésien inutilisable le plus souvent.

3. Afficher le titre et l'état des disques de 1965, ne pas utiliser de critère de jointure (volontairement).
4. Afficher le titre et l'état des disques de 1965

7.2. Auto-jointures

La possibilité de renommer temporairement une table dans une requête permet de faire la jointure d'une table sur elle-même, c'est à dire l'*auto-jointure*.

5. Afficher toutes les appartenances d'un artiste à un groupe.
6. Afficher tous les couples possibles de disques qui sont du même artiste. (Un couple ne peut pas être constitué d'un seul disque...)
7. Idem, mais en écrivant chaque couple une seule fois (c'est-à-dire que si (a,b) est présent, (b,a) n'y sera pas).
8. Rechercher les artistes dont le nom complet (prénom+nom+groupe) est plus long que celui du groupe auquel ils appartiennent. On affichera les noms complets et les longueurs.
9. Rechercher les titres des disques dont le prix est plus grand que celui de `Morrison Hotel`.
10. Rechercher le titre et l'année des disques qui sont du même artiste que `Flowers`.
11. Afficher les titres et années des disques d'artistes différents qui ont le même titre. On affichera aussi le nom complet (prénom+nom+groupe) des artistes.

7.3. Sous-requêtes

Le résultat d'une requête peut servir dans une clause de restriction d'une autre requête, on parlera alors de sous-requête imbriquée.

La recherche suivante peut se faire de deux manières.

- Afficher le titre et l'année des disques qui sont de la même année que `Flowers`.

Dans ce qui précède, la sous-requête retourne une seule valeur, l'opérateur égalité peut être utilisé. Si ce n'est pas le cas il faut utiliser les clauses `ANY` ou `ALL`.

En fait : « `IN` » est équivalent à « `= ANY` », tandis que « `NOT IN` » est équivalent à « `!= ALL` ».

- Afficher le nom complet (prénom+nom+groupe) des artistes dont au moins un disque a un prix plus bas qu'au moins un disque de `Jimi HENDRIX Experience`.
- Afficher les titres des disques plus chers que tous les disques de l'année `1981`.
- Dans la table `AUTORISATIONS` afficher les noms et prénoms des utilisateurs de niveau `membre` dont on trouve aussi le prénom parmi les utilisateurs de niveau `anonyme`.
- Dans la table `AUTORISATIONS` afficher les noms et prénoms des utilisateurs de niveau `membre` dont le prénom ne se trouve pas parmi les utilisateurs de niveau `anonyme`.
- Rechercher le nom et le prénom des personnes qui ont le même niveau d'autorisation que `Sophie FONFEC`.

Renommer une table et utiliser une sous-requête permet de synchroniser une sous-requête avec la requête principale.

- Afficher le titre et le prix des disques qui sont plus chers plus que la moyenne des prix des disques de l'artiste correspondant. (La moyenne s'obtient par la fonction `AVG`.)

L'opérateur `EXISTS` permet de retourner des lignes si la sous-requête en retourne.

- Afficher les artistes dont au moins un disque a été perdu.
- Afficher les artistes n'ayant pas de disque à leur nom.

7.4. LEFT JOIN, RIGHT JOIN

Dans la table `ARTISTE` il y a des artistes dont l'identifiant (`art_num`) ne *correspond* à aucune ligne de `DISQUE`. Ces artistes ne peuvent pas appartenir à une jointure entre les deux tables.

Si on souhaite, malgré tout, obtenir ces artistes dans le résultat d'une jointure on utilise un `LEFT JOIN`.

Pour bien comprendre le sens et la syntaxe de `LEFT JOIN` il peut être utile de revenir sur la syntaxe des requêtes. Il faut savoir que la requête

```
SELECT ...
FROM a,b,c
WHERE a.x=b.y
AND b.z=c.t
...
```

peut s'écrire également

```
SELECT ...
FROM a
JOIN b ON a.x=b.y
JOIN c ON b.z=c.t
WHERE ...
```

Il s'agit d'ailleurs de la syntaxe originale des jointures en `SQL`.

Pour écrire un `LEFT JOIN`, la syntaxe est la même, en remplaçant bien entendu `JOIN` par `LEFT JOIN`.

Qu'est-ce que cela va changer ? On se souvient que pour une jointure *normale* on ne prend que les enregistrements de chaque table qu'on peut relier par la condition de jointure. Avec un `LEFT JOIN` on prendra en plus les enregistrements de la table écrite à gauche (car `LEFT`) de l'expression `LEFT JOIN` et qui ne sont reliés à aucun enregistrement de celle de droite.

Il existe également `RIGHT JOIN` qui fonctionne de manière tout à fait symétrique à droite.

21. Afficher TOUS les artistes (prénom+nom+groupe) dont le nom commence par un « A » avec leurs disques. Ceux qui n'ont pas de disques doivent quand même être affichés.
22. Afficher tous les artistes (prénom+nom+groupe) dont le nom commence par un « A » avec leur appartenance à un groupe (prénom+nom+groupe). Ceux qui n'appartiennent pas à un groupe devront être affichés aussi.

Chapitre 8. Les dates

Vous trouverez les réponses des exercices au Chapitre 14.

8.1. Généralités

Pour avoir la description de la table `DISQUE`, vous pouvez utiliser la commande « `\d disque` ». Cela vous permettra de vérifier que la colonne `dis_modif` est bien de type `date`.

1. Afficher le titre et la date de dernière modification des disques.

Puis recommencez plusieurs fois en faisant précéder votre requête de l'instruction « `SET DATESTYLE TO xxx ;` » avec les valeurs suivantes de `xxx` :

```
'European' 'ISO' 'Postgres' DEFAULT 'German' 'NonEuropean' 'SQL'
```

8.2. Affichage d'une date

L'affichage d'une date peut être modifié en utilisant la fonction `TO_CHAR` :

```
TO_CHAR (arg1, arg2)
```

où `arg1` est une colonne (ou valeur) de type `DATE`, et `arg2` est un masque de sortie (chaîne de caractères).

Tableau 8-1. Liste des masques au format numérique

Masque	Signification
CC	Siècle
YYYY	Année
YYY	3 derniers chiffres de l'année
YY	2 derniers chiffres de l'année
Y	Le dernier chiffre de l'année
Q	Trimestre
WW	Semaine dans l'année
W	Semaine dans le mois
MM	Mois
DDD	Jour dans l'année
DD	Jour dans le mois
D	Jour dans la semaine
HH ou HH12	Heure de la journée (1-12)
HH24	Heure de la journée (0-23)
MI	Minutes
SS	Secondes
SSSSS	Secondes après minuit (0-86399)
J	Jour julien

Exemple 8-1. Utilisation de `TO_CHAR` (numérique)

```
SELECT TO_CHAR(dis_modif,'CC'), TO_CHAR(dis_modif,'W') FROM disque ;
```

Chapitre 8. Les dates

```
SELECT TO_CHAR(dis_modif,'DD/MM/YYYY') FROM disque ;
SELECT TO_CHAR(dis_modif, 'WWème semaine, MMème mois') FROM disque ;
```

Tableau 8-2. Liste des masques au format caractère

Masque	Signification
MONTH	Nom du mois en toutes lettres
MON	3 premières lettres du nom du mois
DAY	Nom du jour en toutes lettres
DY	3 premières lettres du nom du jour
AM ou PM	Avant midi ou après midi
BC ou AD	Avant ou après J.C.

Exemple 8-2. Utilisation de TO_CHAR (caractères)

```
SELECT TO_CHAR(dis_modif,'DAY MONTH YYYY') FROM disque ;
SELECT TO_CHAR(dis_modif,'DY MON YYYY BC') FROM disque ;
```

Tableau 8-3. Masque de suffixe

Masque	Signification
TH	ST, ND, RD, TH après le nombre

Exemple 8-3. Utilisation de TO_CHAR (suffixe)

```
SELECT TO_CHAR(dis_modif, 'WWTH semaine, MMTH mois, CCTH siècle')
FROM disque ;
```

Le masque d'affichage par défaut est YYYY-MM-DD.

2. Rechercher le titre et la date de dernière modification (masque `dd/mm/yy`) des disques de l'année 1981.
3. Rechercher le titre et la date de dernière modification (masque `Day DD Month YYYY`) des disques de l'année 1981.
4. Rechercher le titre et la date de dernière modification (masque `DD MON yyyy HH24:MI:SS`) des disques de l'année 1981.
5. Afficher la date d'aujourd'hui sous la forme `JJ/MM/AAAA` (de deux façons, en utilisant `NOW()` et `CURRENT_DATE`).

8.3. Exercices

6. Afficher la date d'hier (de deux façons, en utilisant `NOW()` et `CURRENT_DATE`).
7. Afficher la date de dimanche dernier.
8. Titre de la chanson qui a été modifiée en dernier, date de cette modification, nombre de jours depuis cette modification.

Chapitre 9. Les groupes

Vous trouverez les réponses des exercices au Chapitre 15.

9.1. Utilisation de fonctions de groupe

Par exemple : AVG (moyenne), MIN (minimum), MAX (maximum), SUM (somme), COUNT (dénombrement)...
Ces fonctions *travaillent* au niveau des groupes de lignes et non plus au niveau des lignes.

Exemple 9-1. Moyenne

Par exemple pour rechercher la moyenne des prix des disques de 1981 :

```
SELECT AVG(dis_prixachat)
FROM disque
WHERE dis_annee = '1981' ;
```

Avec SELECT on ne peut pas *travailler* à la fois au niveau des lignes et des groupes.

Si vous recherchez le titre et la moyenne des prix des disques (cette phrase a-t-elle d'ailleurs un sens ?), vous allez essayer :

```
SELECT dis_titre, AVG(dis_prixachat)
FROM disque ;
```

Ce qui ne produira qu'un message d'erreur.

Par contre avec deux SELECT imbriqués on peut rechercher le titre et le prix des disques dont le prix est le plus grand.

```
SELECT dis_titre, dis_prixachat
FROM disque
WHERE dis_prixachat = (SELECT MAX (dis_prixachat)
                       FROM disque
                       ) ;
```

9.2. Les groupes

Pour exprimer le groupe sur lequel doit porter la fonction de groupe on utilise la clause GROUP BY.

Ces fonctions et clauses peuvent s'utiliser avec une jointure.

Toute colonne qui intervient dans l'affichage sans être utilisée dans une fonction de groupe doit être aussi incluse dans la clause GROUP BY.

Pour rechercher la moyenne des prix des disques de chaque année on écrira :

```
SELECT dis_annee, AVG(dis_prixachat)
FROM disque
GROUP BY dis_annee ;
```

1. Calculer le nombre de disques de chaque année.
2. Calculer la moyenne des prix des disques et leur somme, par artiste (prénom+nom+groupe), rangés en ordre alphabétique.

9.3. La clause HAVING

La clause `WHERE` permet d'écrire une restriction au niveau ligne, la clause `HAVING` permet d'écrire une restriction au niveau groupe.

Pour rechercher les années et le nombre de disques par année, pour les années représentées par plus de 20 disques, on écrira :

```
SELECT dis_annee, COUNT(*)
FROM disque
GROUP BY dis_annee
HAVING COUNT(*) > 20
ORDER BY COUNT(*) DESC ;
```

3. Afficher les artistes (prénom+nom+groupe) qui ont au moins 10 disques à leur nom.
4. Afficher dans l'ordre alphabétique les lettres qui sont l'initiale d'au moins trois noms d'artiste.

9.4. Exercices

5. Rechercher le prix maximum et le prix minimum parmi tous les disques et l'écart entre les deux.
6. Rechercher le nombre d'années différentes pour les disques.
7. Rechercher le nombre de chansons par genre musical et langue.
8. Rechercher les années et la moyenne des prix de disques par année, pour les années dont cette moyenne est supérieure à la moyenne des prix des disques de 1981.
9. Rechercher le nombre de `cha_texte` renseignées, le nombre de champs `cha_libre`, le nombre de champs `cha_modif` ainsi que le nombre de lignes dans la table le nombre de champs `chanson`.

Chapitre 10. Exercices récapitulatifs

Vous trouverez les réponses des exercices au Chapitre 16.

1. Afficher dans l'ordre alphabétique la liste des artistes qui sont référencés pour plus d'une langue.
2. Afficher les titres des disques qui ont le prix minimum et le prix maximum.
3. Afficher les noms des artistes qui ont le plus de disques.
4. Afficher les disques dont le prix est le plus proche du prix moyen des disques.
5. Nombre de disques pour chaque état.
6. Nombre de chansons par genre musical.
7. Nombre de chansons de chaque langue.
8. Nombre de chansons dont les paroles sont libres de droits.
9. Les appartenances de musiciens à des groupes, par ordre alphabétique des groupes, des membres et par ordre chronologique.
10. Liste des utilisateurs, dans l'ordre alphabétique des noms, prénoms, ainsi que leurs *logins* et niveaux d'autorisation.
11. Afficher en français le jour d'aujourd'hui (lundi, mardi, etc.).
12. Afficher l'heure actuelle à la seconde près.
13. Nombre de chansons interprétées par Robert Cray.
14. Nombre d'utilisateurs pour chaque niveau d'autorisation.
15. La liste des disques de Pink Floyd, dans l'ordre chronologique inverse (les plus récents en premier).
16. Les 10 artistes qui ont le plus de disques.
17. Titre des chansons qui contiennent le mot « silver » dans le texte ou dans le titre.
18. Nombre de chansons qui contiennent le mot « love » dans le titre, et nombre de chansons qui contiennent le mot « hate » dans le titre.
19. Les 5 jours où il y a eu le plus de modifications de chansons, et combien.
20. Statistiques horaires des modifications de chansons.
21. Nombre de chansons écrites par chaque artiste.
22. Nombre de chansons de chaque langue.
23. Les disques (titre et artiste) qui comportent moins de 5 chansons.
24. Afficher le titre des chansons dont la dernière modification a été faite au mois de mai.
25. Les chansons dont il existe au moins 5 versions.
26. Le contenu du disque « Déjà Vu ».
27. La discographie complète de Lou Reed, que ce soit seul ou dans un groupe (on précisera).
28. En ordre alphabétique, les musiciens qui font partie d'un groupe qui a chanté des chansons de rock.
29. Les artistes qui ont chanté dans plusieurs langues, et dans combien de langues.

Chapitre 11. Réponses aux premiers exercices sur la base *jouet*

Vous trouverez les énoncés correspondant au Chapitre 3.

Solution de l'exercice 1

a	b	c
x	m	2
x	n	1
y	m	4
z	p	1

Solution de l'exercice 2

a
x
x
y
z

Solution de l'exercice 3

a
x
z

Solution de l'exercice 4

a
x
x
z

Solution de l'exercice 5

a
x
z

Solution de l'exercice 6

a

a
x
y
x
z

Solution de l'exercice 7

a	e
x	8
x	4
x	1
x	8
x	4
x	1
y	8
y	4
y	1
z	8
z	4
z	1

On remarquera que cela donne 12 lignes c'est-à-dire 4×3 , soit le produit du nombre de lignes de UN et du nombre de lignes de DEUX.

Solution de l'exercice 8

a	e
x	1
y	4
z	1

Chapitre 12. Réponses aux premiers exercices sur la commande **SELECT**

Vous trouverez les énoncés correspondant au Chapitre 6.

Solution de l'exercice 1

```
SELECT * FROM genre ;
```

Solution de l'exercice 2

```
SELECT * FROM langue ;
```

Solution de l'exercice 3

```
SELECT aut_nom, aut_prenom, aut_login  
FROM autorisations ;
```

Solution de l'exercice 4

```
SELECT aut_prenom  
FROM autorisations ;
```

Solution de l'exercice 5

```
SELECT DISTINCT aut_prenom  
FROM autorisations ;
```

Solution de l'exercice 6

```
SELECT dis_titre  
FROM disque  
WHERE dis_annee='1969'  
AND dis_etat=2 ;
```

La valeur 1969 est entre deux « ' » car la colonne DIS_ANNEE est alpha-numérique, ce qui n'est pas le cas pour DIS_ETAT qui est numérique.

Solution de l'exercice 7

```
SELECT gen_num, gen_nom  
FROM genre  
WHERE gen_num>5 ;
```

Solution de l'exercice 8

```
SELECT dis_titre, dis_prixachat, dis_cddeb  
FROM disque  
WHERE dis_annee = '1981'  
AND dis_etat = 2  
AND dis_prixachat > 15 ;
```

Solution de l'exercice 9

```
SELECT dis_titre, dis_cddb, dis_annee
FROM disque
WHERE dis_titre='IV'
      OR dis_titre='Killer'
      OR dis_titre='No Control' ;
```

Solution de l'exercice 10

```
SELECT dis_titre, dis_cddb, dis_annee
FROM disque
WHERE dis_annee='1971'
      AND (dis_titre='IV' OR dis_titre='Killer') ;
```

Si les clauses de restriction sont reliées par « AND » ou « OR », ATTENTION aux parenthèses.

Solution de l'exercice 11

```
SELECT dis_titre, dis_cddb, dis_annee
FROM disque
WHERE dis_titre='IV'
      OR (dis_annee='1971' AND dis_titre='Killer') ;
```

Solution de l'exercice 12

```
SELECT dis_titre, dis_cddb, dis_prixachat
FROM disque
WHERE dis_prixachat >= 13.5
      AND dis_prix <= 14 ;
```

La clause « `WHERE dis_prixachat >= 13.5 AND dis_prix <= 14` » peut s'écrire avec l'opérateur « BETWEEN » :

```
SELECT dis_titre, dis_cddb, dis_prixachat
FROM disque
WHERE dis_prixachat BETWEEN 13.5 AND 14 ;
```

Solution de l'exercice 13

```
SELECT art_nom, art_prenom
FROM artiste
WHERE art_prenom='Louis' OR art_prenom='Jean' OR art_prenom='Alain' ;
```

La clause « `WHERE colonne IN ('val1', 'val2')` » permet d'éviter « `WHERE colonne = 'val1' OR colonne = 'val2'` » :

```
SELECT art_nom, art_prenom
FROM artiste
WHERE art_prenom IN ('Louis' , 'Jean' , 'Alain') ;
```

Solution de l'exercice 14

```
SELECT art_nom, art_prenom
FROM artiste
WHERE art_nom < art_prenom ;
```

Solution de l'exercice 15

```
SELECT art_nom
FROM artiste
WHERE art_nom LIKE 'H%' ;
```

Solution de l'exercice 16

```
SELECT art_nom
FROM artiste
WHERE art_nom LIKE '%n' ;
```

Solution de l'exercice 17

```
SELECT art_nom
FROM artiste
WHERE art_nom LIKE '__u%' ;
```

Solution de l'exercice 18

```
SELECT dis_titre,dis_prixachat
FROM disque
WHERE dis_annee='1985'
ORDER BY dis_prixachat ;
```

Solution de l'exercice 19

```
SELECT dis_titre,dis_prixachat
FROM disque
WHERE dis_annee='1985'
ORDER BY dis_prixachat DESC ;
```

Solution de l'exercice 20

```
SELECT dis_titre,dis_annee
FROM disque
ORDER BY dis_annee DESC, dis_titre ;
```

Solution de l'exercice 21

```
SELECT *
FROM artiste
WHERE art_nom LIKE 'A%'
ORDER BY art_modif ;
```

Solution de l'exercice 22

```
SELECT *
FROM artiste
WHERE art_nom LIKE 'A%'
AND art_modif IS NULL ;
```

Solution de l'exercice 23

```
SELECT *
  FROM artiste
 WHERE art_nom LIKE 'A%'
        AND art_modif IS NOT NULL ;
```

Solution de l'exercice 24

```
SELECT *
  FROM artiste
 WHERE art_nom LIKE 'A%'
        AND art_modif > '2009-01-01' ;
```

Par défaut, les colonnes alphanumériques non renseignées ne sont pas considérées comme une suite d'espaces, les colonnes numériques ne sont pas considérées comme égale à 0.

Solution de l'exercice 25

```
SELECT *
  FROM artiste
 WHERE art_nom LIKE 'A%'
        AND art_modif < '2009-01-01' ;
```

Une fois de plus, remarquez le comportement des lignes non renseignées.

Solution de l'exercice 26

```
SELECT dis_titre, dis_prixachat AS "Prix en euros",
       dis_prixachat/6.5 AS "Prix en francs"
  FROM disque
 WHERE dis_anneeachat < '2002' ;
```

Solution de l'exercice 27

```
SELECT int_artiste, int_chanson, int_disque, int_numero,
       COALESCE(int_remarque, 'Aucune remarque disponible') AS "Remarque?"
  FROM interprete
 WHERE int_artiste = 170 ;
```

Solution de l'exercice 28

```
SELECT dis_titre, dis_prixachat AS "Prix en euros",
       ROUND(dis_prixachat/6.5, 2) AS "Prix en francs"
  FROM disque
 WHERE dis_anneeachat < '2002' ;
```

Solution de l'exercice 29

```
SELECT art_prenom || ' ' || art_nom || ' ' || art_groupe AS "Artiste"
  FROM artiste ;
```

Solution de l'exercice 30

```
SELECT cha_titre AS "Titre",
       CASE WHEN cha_texte IS NULL
            THEN 'paroles inconnues'
            ELSE 'paroles disponibles'
            END AS "Paroles ?"
FROM chanson
WHERE cha_titre LIKE 'A%'
ORDER BY 1 ;
```

Solution de l'exercice 31

```
SELECT SUBSTR (art_nom, 1,5)
FROM artiste ;
```

Solution de l'exercice 32

Pour la position à partir du début :

```
SELECT art_nom, STRPOS (art_nom,'r')
FROM artiste ;
```

Pour avoir le résultat à partir de la troisième lettre, il suffit de « supprimer » les deux premières...

```
SELECT art_nom, STRPOS (SUBSTR(art_nom,3),'r')
FROM artiste ;
```

Solution de l'exercice 33

```
SELECT art_nom, art_prenom, UPPER(art_prenom) , LOWER(art_prenom)
FROM artiste
WHERE UPPER(art_nom) = UPPER('Russell') ;
```

Solution de l'exercice 34

```
SELECT nom, LENGTH(nom)
FROM emp ;
```

Solution de l'exercice 35

```
SELECT TRIM(art_prenom || ' ' || UPPER(art_nom) || ' ' || art_groupe) AS "Artiste"
FROM artiste ;
```

Solution de l'exercice 36

```
SELECT aut_nom
FROM autorisations
UNION
SELECT art_nom
FROM artiste ;
```

Solution de l'exercice 37

```
SELECT ecr_artiste AS "Artiste"  
FROM ecrit  
INTERSECT  
SELECT int_artiste AS "Artiste"  
FROM interprete ;
```

Solution de l'exercice 38

```
SELECT lan_num  
FROM langue  
EXCEPT  
SELECT lac_langue  
FROM languechanson ;
```

Chapitre 13. Réponses aux exercices sur les jointures et les sous-requêtes

Vous trouverez les énoncés correspondant au Chapitre 7.

Solution de l'exercice 1

```
SELECT aut_nom, aut_prenom, aut_login, dro_nom
FROM autorisations, droits
WHERE aut_droits = dro_num ;
```

Solution de l'exercice 2

```
SELECT dis_titre, art_prenom, art_nom, art_groupe
FROM artiste, disque
WHERE dis_artiste = art_num
ORDER BY art_nom, art_prenom, art_groupe ;
```

Solution de l'exercice 3

```
SELECT dis_titre, eta_nom
FROM disque, etat
WHERE dis_annee='1965' ;
```

Solution de l'exercice 4

```
SELECT dis_titre, eta_nom
FROM disque, etat
WHERE dis_etat=eta_num
AND dis_annee='1965' ;
```

Solution de l'exercice 5

```
SELECT TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe) AS artiste,
TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe) AS groupe,
mem_debut AS depuis,
mem_fin AS "jusqu'à"
FROM artiste a, artiste g, membre
WHERE a.art_num = mem_membre
AND g.art_num = mem_groupe
ORDER BY g.art_nom, g.art_prenom, a.art_nom, a.art_prenom, a.art_groupe, g.art_groupe, mem_debut ;
```

Solution de l'exercice 6

```
SELECT a.dis_titre, b.dis_titre
FROM disque a, disque b
WHERE a.dis_artiste = b.dis_artiste
AND a.dis_num <> b.dis_num ;
```

Solution de l'exercice 7

```
SELECT a.dis_titre,b.dis_titre
FROM disque a,disque b
WHERE a.dis_artiste = b.dis_artiste
AND a.dis_num <> b.dis_num
AND a.dis_num < b.dis_num ;
```

Solution de l'exercice 8

```
SELECT TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe) AS artiste,
LENGTH(TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe)) AS "longueur artiste",
TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe) AS groupe,
LENGTH(TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe)) AS "longueur groupe"
FROM artiste a, artiste g, membre
WHERE a.art_num = mem_membre
AND g.art_num = mem_groupe
AND LENGTH(TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe))>LENGTH(TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe))
```

Solution de l'exercice 9

```
SELECT d.dis_titre,d.dis_prixachat
FROM disque d,disque m
WHERE d.dis_prixachat > m.dis_prixachat
AND LOWER(m.dis_titre) = 'morrison hotel' ;
```

Solution de l'exercice 10

```
SELECT d.dis_titre,d.dis_annee
FROM disque d, disque f
WHERE d.dis_artiste=f.dis_artiste
AND LOWER(f.dis_titre) = 'flowers' ;
```

Solution de l'exercice 11

```
SELECT da.dis_titre,
TRIM(aa.art_prenom||' '||aa.art_nom||' '||aa.art_groupe),da.dis_annee,
TRIM(ab.art_prenom||' '||ab.art_nom||' '||ab.art_groupe),db.dis_annee
FROM disque da,disque db,artiste aa,artiste ab
WHERE da.dis_titre = db.dis_titre
AND da.dis_artiste = aa.art_num
AND db.dis_artiste = ab.art_num
AND aa.art_num < ab.art_num ;
```

Solution de l'exercice 12

```
SELECT d.dis_titre,d.dis_annee
FROM disque d, disque f
WHERE d.dis_annee=f.dis_annee
AND LOWER(f.dis_titre) = 'flowers' ;
```

Mais aussi... (et c'est mieux !):

```
SELECT dis_titre,dis_annee
FROM disque
WHERE dis_annee = (SELECT dis_annee
FROM disque
WHERE LOWER(dis_titre) = 'flowers') ;
```

Solution de l'exercice 13

```
SELECT DISTINCT TRIM(art_prenom||' '||art_nom||' '||art_groupe)
FROM disque,artiste
WHERE dis_artiste = art_num
      AND dis_prixachat < ANY (SELECT dis_prixachat
                              FROM disque,artiste
                              WHERE dis_artiste = art_num
                                 AND LOWER(art_prenom)='jimi'
                                 AND LOWER(art_nom)='hendrix'
                                 AND LOWER(art_groupe)='experience' ) ;
```

Solution de l'exercice 14

```
SELECT dis_titre
FROM disque
WHERE dis_prixachat > ALL (SELECT dis_prixachat
                          FROM disque
                          WHERE dis_annee = '1981') ;
```

Solution de l'exercice 15

```
SELECT aut_prenom,aut_nom
FROM autorisations,droits
WHERE aut_droits = dro_num
      AND LOWER(dro_nom) = 'membre'
      AND LOWER(aut_prenom) IN (SELECT LOWER(aut_prenom)
                                FROM autorisations,droits
                                WHERE aut_droits = dro_num
                                   AND LOWER(dro_nom) = 'anonyme') ;
```

Solution de l'exercice 16

```
SELECT aut_prenom,aut_nom
FROM autorisations,droits
WHERE aut_droits = dro_num
      AND LOWER(dro_nom) = 'membre'
      AND LOWER(aut_prenom) NOT IN (SELECT LOWER(aut_prenom)
                                    FROM autorisations,droits
                                    WHERE aut_droits = dro_num
                                       AND LOWER(dro_nom) = 'anonyme') ;
```

Solution de l'exercice 17

```
SELECT aut_prenom, aut_nom
FROM autorisations
WHERE aut_droits = (SELECT aut_droits
                   FROM autorisations
                   WHERE LOWER(aut_nom) = 'fonfec'
                      AND LOWER(aut_prenom) = 'sophie') ;
```

Solution de l'exercice 18

```
SELECT dis_titre,dis_prixachat,dis_artiste
FROM disque d
WHERE dis_prixachat > (SELECT AVG(dis_prixachat)
                       FROM disque
                       WHERE dis_artiste = d.dis_artiste)
ORDER BY dis_artiste ;
```

Solution de l'exercice 19

```
SELECT TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe)
FROM artiste a
WHERE EXISTS (SELECT NULL
              FROM disque
              WHERE dis_perdu
              AND dis_artiste = a.art_num) ;
```

Solution de l'exercice 20

```
SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe)
FROM artiste
WHERE art_num NOT IN (SELECT dis_artiste
                     FROM disque) ;
```

Solution de l'exercice 21

```
SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe),dis_titre
FROM artiste
LEFT JOIN disque ON dis_artiste = art_num
WHERE art_nom LIKE 'A%' ;
```

Solution de l'exercice 22

Avec LEFT JOIN cela donne :

```
SELECT TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe) AS artiste,
       TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe) AS artiste
FROM artiste a
LEFT JOIN membre ON a.art_num = mem_membre
LEFT JOIN artiste g ON g.art_num = mem_groupe
WHERE a.art_nom LIKE 'A%'
ORDER BY a.art_nom,a.art_prenom,a.art_groupe ;
```

On peut aussi se passer de LEFT JOIN (certains systèmes ne l'autorisent pas), mais c'est moins simple. Avec une union cela donne :

```
(
  SELECT TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe) AS artiste,
         TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe) AS artiste
  FROM artiste a,membre,artiste g
  WHERE a.art_num = mem_membre
        AND g.art_num = mem_groupe
        AND a.art_nom LIKE 'A%'
  UNION
  SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste,
         NULL AS artiste
  FROM artiste
  WHERE art_nom LIKE 'A%'
)
ORDER BY 1 ;
```

Mais on a une ligne vide pour chaque artiste, même quand ce n'est pas nécessaire. On peut améliorer ainsi :

```
(
  SELECT TRIM(a.art_prenom||' '||a.art_nom||' '||a.art_groupe) AS artiste,
         TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe) AS artiste
    FROM artiste a,membre,artiste g
   WHERE a.art_num = mem_membre
         AND g.art_num = mem_groupe
         AND a.art_nom LIKE 'A%'
 UNION
  SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste,
         NULL AS artiste
    FROM artiste
   WHERE art_nom LIKE 'A%'
         AND art_num NOT IN (SELECT mem_membre
                              FROM membre)
)
ORDER BY 1 ;
```


Chapitre 14. Réponses aux exercices sur les dates

Vous trouverez les énoncés correspondant au Chapitre 8.

Solution de l'exercice 1

```
SELECT dis_titre,dis_modif FROM disque ;

SET DATESTYLE TO 'European' ;
SELECT dis_titre,dis_modif FROM disque ;

SET DATESTYLE TO 'ISO' ;
SELECT dis_titre,dis_modif FROM disque ;

SET DATESTYLE TO 'Postgres' ;
SELECT dis_titre,dis_modif FROM disque ;

SET DATESTYLE TO DEFAULT ;
SELECT dis_titre,dis_modif FROM disque ;

SET DATESTYLE TO 'German' ;
SELECT dis_titre,dis_modif FROM disque ;

SET DATESTYLE TO 'NonEuropean' ;
SELECT dis_titre,dis_modif FROM disque ;

SET DATESTYLE TO 'SQL' ;
SELECT dis_titre,dis_modif FROM disque ;
```

Solution de l'exercice 2

```
SELECT dis_titre,TO_CHAR(dis_modif,'dd/mm/yy')
FROM disque
WHERE dis_annee = '1981' ;
```

Solution de l'exercice 3

```
SELECT dis_titre,TO_CHAR(dis_modif,'Day DD Month YYYY')
FROM disque
WHERE dis_annee = '1981' ;
```

Solution de l'exercice 4

```
SELECT dis_titre,TO_CHAR(dis_modif,'DD MON yyyy HH24:MI:SS')
FROM disque
WHERE dis_annee = '1981' ;
```

Solution de l'exercice 5

```
SELECT TO_CHAR(NOW(),'DD/MM/YYYY') AS "Aujourd'hui",
TO_CHAR(CURRENT_DATE,'DD/MM/YYYY') AS "Aujourd'hui" ;
```

Solution de l'exercice 6

```
SELECT TO_CHAR(NOW()::DATE - 1, 'DD/MM/YYYY') AS "Hier",  
       TO_CHAR(CURRENT_DATE - 1, 'DD/MM/YYYY') AS "Hier" ;
```

Solution de l'exercice 7

```
SELECT CURRENT_DATE - TO_CHAR(CURRENT_DATE, 'D')::INTEGER + 1 AS "Dimanche dernier" ;
```

Solution de l'exercice 8

```
SELECT cha_titre AS titre, TO_CHAR(cha_modif, 'DD/MM/YYYY') AS date,  
       (CURRENT_DATE-cha_modif::date)||' jours' AS Delai  
FROM chanson  
WHERE cha_modif=(SELECT MAX(cha_modif) FROM chanson) ;
```

Chapitre 15. Réponses aux exercices sur les groupes

Vous trouverez les énoncés correspondant au Chapitre 9.

Solution de l'exercice 1

```
SELECT dis_annee, COUNT(*)
FROM disque
GROUP BY dis_annee ;
```

Solution de l'exercice 2

```
SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste,
AVG(dis_prixachat) as moyenne, SUM(dis_prixachat) as somme
FROM disque, artiste
WHERE dis_artiste = art_num
GROUP BY art_num, art_prenom, art_nom, art_groupe
ORDER BY art_nom, art_prenom, art_groupe ;
```

Solution de l'exercice 3

```
SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste, COUNT(*) AS nbre
FROM artiste, disque
WHERE art_num = dis_artiste
GROUP BY art_num, art_prenom, art_nom, art_groupe
HAVING COUNT(*) >= 10
ORDER BY COUNT(*) DESC ;
```

Solution de l'exercice 4

```
SELECT SUBSTR(art_nom, 1, 1), COUNT(*)
FROM artiste
GROUP BY SUBSTR(art_nom, 1, 1)
HAVING COUNT(*) >= 20
ORDER BY 1 ;
```

Solution de l'exercice 5

```
SELECT MAX(dis_prixachat) ,
MIN(dis_prixachat) ,
MAX(dis_prixachat) - MIN(dis_prixachat) AS "Différence"
FROM disque ;
```

Solution de l'exercice 6

```
SELECT COUNT(DISTINCT dis_annee)
FROM disque ;
```

Essayez aussi, pour voir la différence

```
SELECT COUNT(dis_annee)
FROM disque ;
```

Solution de l'exercice 7

```
SELECT gen_nom, lan_nom, COUNT(*)
FROM chanson, genre, langue, languechanson
WHERE gen_num=cha_genre
      AND cha_num=lac_chanson
      AND lac_langue=lan_num
GROUP BY lan_nom, gen_nom, lan_num, gen_num
ORDER BY gen_nom, lan_nom ;
```

Solution de l'exercice 8

```
SELECT dis_annee, AVG(dis_prixachat)
FROM disque
GROUP BY dis_annee
HAVING AVG(dis_prixachat) >= (SELECT AVG(dis_prixachat)
                              FROM disque
                              WHERE dis_annee = '1981')
ORDER BY 2 DESC ;
```

Solution de l'exercice 9

```
SELECT COUNT(cha_texte) AS texte,
       COUNT(cha_libre) AS libre,
       COUNT(cha_modif) AS modif,
       COUNT(*) AS total
FROM chanson ;
```

Chapitre 16. Réponses aux exercices récapitulatifs

Vous trouverez les énoncés correspondant au Chapitre 10.

Solution de l'exercice 1

```
SELECT COUNT(DISTINCT laa_langue) AS "nombre de langues",
       TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste
FROM   artiste,langueartiste
WHERE  art_num=laa_artiste
GROUP BY art_nom,art_prenom,art_groupe,art_num
HAVING COUNT(DISTINCT laa_langue) > 1 ;
```

Solution de l'exercice 2

```
(
  SELECT dis_titre,dis_prixachat,'Minimum' AS "min ou max?"
  FROM   disque
  WHERE  dis_prixachat=(
          SELECT MIN(dis_prixachat)
          FROM   disque
        )
)
UNION
(
  SELECT dis_titre,dis_prixachat,'Maximum' AS "min ou max?"
  FROM   disque
  WHERE  dis_prixachat=(
          SELECT MAX(dis_prixachat)
          FROM   disque
        )
)
ORDER BY 2 ;
```

Solution de l'exercice 3

```
SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste,COUNT(*) AS nombre
FROM   artiste,disque
WHERE  art_num = dis_artiste
GROUP BY art_prenom,art_nom,art_groupe,art_num
HAVING COUNT(*) = (
                SELECT MAX(total)
                FROM (
                        SELECT dis_artiste,COUNT(*) AS total
                        FROM   disque
                        GROUP BY dis_artiste
                    ) nbreParArtiste
            ) ;
```

Solution de l'exercice 4

```
SELECT dis_titre,dis_prixachat
FROM   disque
WHERE  (
        SELECT ABS(dis_prixachat-(SELECT AVG(dis_prixachat)
        FROM   disque))
    )=(
        SELECT MIN(diff)
        FROM (
                SELECT ABS(dis_prixachat-(
                        SELECT AVG(dis_prixachat)
                        FROM   disque
                    )
                )
            )
    ) ;
```

```
        ) AS diff
    FROM disque
) a
) ;
```

ou

```
SELECT dis_titre,dis_prixachat
FROM disque,(
    SELECT AVG(dis_prixachat) AS moy
    FROM disque
) r1,
(
    SELECT MIN(diff) AS mini
    FROM (
        SELECT ABS(dis_prixachat-moy) AS diff
        FROM disque,(
            SELECT AVG(dis_prixachat) AS moy
            FROM disque
        ) r2
    ) r3
) r4
WHERE mini = ABS(dis_prixachat-moy) ;
```

Solution de l'exercice 5

```
SELECT eta_nom,COUNT(*)
FROM disque,etat
WHERE dis_etat=eta_num
GROUP BY eta_nom ;
```

Solution de l'exercice 6

```
SELECT gen_nom,COUNT(*)
FROM chanson,genre
WHERE gen_num=cha_genre
GROUP BY gen_nom,gen_num ;
```

Solution de l'exercice 7

```
SELECT lan_nom,COUNT(*)
FROM chanson,languechanson,langue
WHERE cha_num=lac_chanson
AND lan_num=lac_langue
GROUP BY lan_nom ;
```

Solution de l'exercice 8

```
SELECT COUNT(*)
FROM chanson
WHERE cha_libre ;
```

Solution de l'exercice 9

```
SELECT TRIM(m.art_prenom||' '||m.art_nom||' '||m.art_groupe) AS artiste,
       TRIM(g.art_prenom||' '||g.art_nom||' '||g.art_groupe) AS groupe,
       CASE WHEN mem_debut<>" THEN mem_debut ELSE " END AS depuis,
       CASE WHEN mem_fin<>" THEN mem_fin ELSE " END AS jusque
FROM   artiste m,membre,artiste g
WHERE  m.art_num=mem_membre
       AND mem_groupe=g.art_num
ORDER BY g.art_nom,g.art_prenom,g.art_groupe,
         m.art_nom,m.art_prenom,m.art_groupe,
         mem_debut,mem_fin ;
```

Solution de l'exercice 10

```
SELECT aut_nom, aut_prenom, aut_login, dro_nom
FROM   autorisations, droits
WHERE  aut_droits = dro_num
ORDER BY 1,2 ;
```

Solution de l'exercice 11

```
SELECT CASE
      WHEN TO_CHAR(CURRENT_DATE,'D')='1' THEN 'dimanche'
      WHEN TO_CHAR(CURRENT_DATE,'D')='2' THEN 'lundi'
      WHEN TO_CHAR(CURRENT_DATE,'D')='3' THEN 'mardi'
      WHEN TO_CHAR(CURRENT_DATE,'D')='4' THEN 'mercredi'
      WHEN TO_CHAR(CURRENT_DATE,'D')='5' THEN 'jeudi'
      WHEN TO_CHAR(CURRENT_DATE,'D')='6' THEN 'vendredi'
      ELSE 'samedi'
END AS "jour d'aujourd'hui" ;
```

Solution de l'exercice 12

```
SELECT TO_CHAR(NOW(), 'HH24:MI:SS') AS "Au troisième top..." ;
```

Solution de l'exercice 13

```
SELECT COUNT(*) AS nbre
FROM   artiste,interprete
WHERE  art_num=int_artiste
       AND art_prenom='Robert'
       AND art_nom='Cray'
       AND art_groupe="" ;
```

Solution de l'exercice 14

```
SELECT dro_nom AS Nom,dro_niveau AS niveau,COUNT(*) AS nbre
FROM   autorisations,droits
WHERE  aut_droits=dro_num
GROUP BY dro_nom,dro_niveau ;
```

Solution de l'exercice 15

```
SELECT dis_titre,dis_annee
FROM disque,artiste
WHERE dis_artiste=art_num
AND art_nom='Pink Floyd'
ORDER BY 2 DESC,1 ;
```

Solution de l'exercice 16

```
SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste,COUNT(*)
FROM disque,artiste
WHERE dis_artiste=art_num
GROUP BY art_num,art_prenom,art_nom,art_groupe
ORDER BY COUNT(*) DESC
LIMIT 10 ;
```

Solution de l'exercice 17

```
SELECT cha_titre
FROM chanson
WHERE LOWER(cha_titre) LIKE '%silver%'
OR LOWER(cha_texte) LIKE '%silver%'
ORDER BY cha_titre ;
```

Solution de l'exercice 18

```
SELECT 'Love' AS "Love hate", (SELECT COUNT(*)
FROM chanson
WHERE LOWER(cha_titre) LIKE '%love%') AS nombre
UNION
SELECT 'Hate' AS "Love hate", (SELECT COUNT(*)
FROM chanson
WHERE LOWER(cha_titre) LIKE '%hate%') AS nombre ;
```

Solution de l'exercice 19

```
SELECT TO_CHAR(cha_modif,'DD/MM/YYYY') AS jour,COUNT(*) AS nombre
FROM chanson
GROUP BY TO_CHAR(cha_modif,'DD/MM/YYYY')
ORDER BY COUNT(*) DESC
LIMIT 5 ;
```

Solution de l'exercice 20

```
SELECT TO_CHAR(cha_modif,'HH24') AS heure,COUNT(*) AS nombre
FROM chanson
GROUP BY TO_CHAR(cha_modif,'HH24')
ORDER BY 1 ;
```

Solution de l'exercice 21

```
SELECT TRIM(art_prenom||' '||art_nom||' '||art_groupe) AS artiste,COUNT(*) AS nombre
FROM artiste,ecrit
WHERE art_num=ecr_artiste
GROUP BY art_num,art_prenom,art_nom,art_groupe
ORDER BY art_nom,art_prenom,art_groupe ;
```

Solution de l'exercice 22

```
SELECT lan_nom,COUNT(*)
FROM langue,languechanson
WHERE lan_num=lac_langue
GROUP BY lan_nom
ORDER BY 1,2 ;
```

Solution de l'exercice 23

```
SELECT dis_titre AS disque,
TRIM(art_prenom||' '||UPPER(art_nom)||' '||art_groupe) AS artiste,
COUNT(*) AS Nbre
FROM disque,interprete,artiste
WHERE dis_num=int_disque
AND art_num=dis_artiste
GROUP BY dis_num,dis_titre,art_prenom,art_nom,art_groupe
HAVING COUNT(DISTINCT int_numero)<5 ;
```

Solution de l'exercice 24

```
SELECT cha_titre
FROM chanson
WHERE TO_CHAR(cha_modif,'MM')::INTEGER=5 ;
```

Solution de l'exercice 25

```
SELECT cha_num,cha_titre,COUNT(DISTINCT int_disque||'-'||int_numero)
FROM chanson,interprete
WHERE cha_num=int_chanson
GROUP BY cha_num,cha_titre
HAVING COUNT(DISTINCT int_disque||'-'||int_numero)>=5
ORDER BY COUNT(DISTINCT int_disque||'-'||int_numero) DESC ;
```

Solution de l'exercice 26

```
SELECT int_numero,cha_titre
FROM chanson,interprete,disque
WHERE cha_num=int_chanson
AND int_disque=dis_num
AND LOWER(dis_titre)='déjà vu'
ORDER BY int_numero ;
```

Solution de l'exercice 27

```
(
  SELECT dis_titre AS "Discographie",dis_annee AS "Année",
         'Seul' AS "Conditions"
    FROM artiste,disque
   WHERE art_num=dis_artiste
         AND art_nom='Reed'
         AND art_prenom='Lou'
         AND art_groupe=""
)
UNION
(
  SELECT dis_titre AS "Discographie",dis_annee AS "Année",
         'avec '||TRIM(g.art_prenom)||' '||UPPER(g.art_nom)||' '||g.art_groupe) AS "Conditions"
    FROM artiste a,artiste g,membre,disque
   WHERE g.art_num=dis_artiste
         AND a.art_nom='Reed'
         AND a.art_prenom='Lou'
         AND a.art_groupe=""
         AND g.art_num=mem_groupe
         AND a.art_num=mem_membre
)
ORDER BY 2 ;
```

Solution de l'exercice 28

```
SELECT DISTINCT art_prenom,art_nom,art_groupe
   FROM artiste,membre,interprete,chanson,genre
  WHERE art_num=mem_membre
        AND mem_groupe=int_artiste
        AND int_chanson=cha_num
        AND cha_genre=gen_num
        AND LOWER(gen_nom)='rock'
ORDER BY art_nom,art_prenom,art_groupe ;
```

Solution de l'exercice 29

```
SELECT TRIM(art_prenom)||' '||UPPER(art_nom)||' '||art_groupe) AS artiste,
       count(distinct lan_nom) AS "nombre de langues"
   FROM artiste,interprete,languechanson,langue
  WHERE art_num=int_artiste
        AND int_chanson=lac_chanson
        AND lac_langue=lan_num
GROUP BY art_prenom,art_nom,art_groupe
HAVING COUNT(DISTINCT lan_num)>1
ORDER BY 1,2 ;
```

Index

- administrateur, 4
- affichage d'une , 31
- algèbre relationnelle, 4, 7
- ALL, 28
- ANY, 28
- architecture
 - client-serveur, 3, 3, 5
 - en couches, 3
- ARTISTE
 - les données de la table, 14
- attribut, 4
- auto-jointure, 27
- AUTORISATIONS
 - les données de la table, 19
- AVG, 33
- base de données, 5
 - relationnelle, 4
- base jouet, 9
 - solutions des exercices, 37
- BETWEEN, 40
- CASE, 24
- CHANSON
 - les données de la table, 16
- chaîne de caractères, 24
- clef
 - primaire, 4
 - étrangère, 5
- client-serveur, 5
- COALESCE, 23
- colonne, 4
- concaténation, 24
- Contenu de la base de données, 13
- contrainte
 - d'intégrité, 5
 - d'intégrité d'entité, 5
 - d'intégrité référentielle, 5
 - de domaine, 5
- COUNT, 33
- date, 31
 - affichage d'une -, 31
- DATESTYLE, 31
- DECODE, 24
- DEUX
 - la table, 9
- dictionnaire, 6
- différence, 7
- DISQUE
 - les données de la table, 15
- DISTINCT, 21
- DocBook, 1
- domaine, 5
- Données de la base de données, 13
- DROITS
 - les données de la table, 19
- dénombrement, 33
- développeur, 4
- ECRIT
 - les données de la table, 18
- emacs, 1
- ETAT
 - les données de la table, 16
- exercices récapitulatifs, 35
- EXISTS, 28
- expression arithmétique, 23
- fonctions, 33
- GENRE
 - les données de la table, 15
- Gilleron, Rémi, 3
- GROUP BY, 33
- groupes, 33
- HAVING, 34
- HTML, 1
- IDAPI, 6
- identifiant, 5
- IN, 40
- interface utilisateur, 6
- INTERPRETE
 - les données de la table, 18
- INTERSECT, 25
- jointure, 7, 27
 - solutions des exercices, 45
 - équijointure, 27
- jouet
 - la base, 9, 37
- langage
 - de contrôle des données, 6
 - de définition de données, 6
 - de manipulation de données, 6
- LANGUE
 - les données de la table, 17
- LANGUEARTISTE
 - les données de la table, 17
- LANGUECHANSON
 - les données de la table, 17
- LCD, 6
- LDD, 6
- Ledant, Guy, 3
- LEFT JOIN, 28
- LENGTH, 25
- LMD, 6
- LOWER, 25
- Marée, Christian, 3
- MAX, 33
- maximum, 33
- MEMBRE
 - les données de la table, 14
- MIN, 33
- minimum, 33
- MLD, 13
- moteur SQL, 8
- moyenne, 33
- niveau
 - externe, 4
 - interne, 4
 - logique, 4
 - physique, 4
- NOT IN, 28
- NULL, 23
- ODBC, 6, 6
- openjade, 1
- opérateur
 - algébrique, 7
- PDF, 1
- primitives SQL, 6
- produit, 7
- produit cartésien, 27

- projection, 7, 21
- protocole, 3
- Quanta, 1
- relation, 4, 7
- relationnel, 4
- renommer une table, 28
- requête, 3, 7
- ROUND, 24
- récapitulatifs
 - solutions des exercices, 55
- schéma
 - externe, 4
 - interne, 4
 - logique, 4, 5
 - physique, 4
- SELECT, 6, 21, 33
 - DISTINCT, 21
 - solutions des exercices, 39
- serveur
 - de bases de données, 6
 - de fichiers, 5
- SET DATESTYLE, 31
- SGBD, 3
- somme, 33
- sous-requête, 27
 - synchroniser une, 28
- SQL, 3
 - dynamique, 6
 - interactif, 6
 - intégré, 6
- STRPOS, 24
- structure en couches, 3
- SUBSTR, 24
- SUM, 33
- sécurité, 5, 6
- sélection, 7, 22
- table, 4, 7
- Tommasi, Marc, 3
- transaction, 6
- TRIM, 25
- TRUNC, 24
- tuple, 4, 7
- UN
 - la table, 9
- union, 7, 25
 - ALL, 25
- UPPER, 25
- version, 1
- XML, 1
- xsltproc, 1