LES BASES DE DONNÉES ET LE LANGAGE SQL

TODO :

- B.2 : Modélisation, Merise, modèle entité-relation (à remplir)

v1.1.3.1 – 07/05/2010
peignotc(at)arqendra(dot)net / peignotc(at)gmail(dot)com

TABLE DES MATIÈRES

TABLE DES ANNEXES

TABLE DES ILLUSTRATIONS

1 INTRODUCTION AUX BASES DE DONNÉES

1.1 PRINCIPES ET STRUCTURE D'UNE BASE DE DONNÉES

Une base de données est un objet informatique qui permet de stocker et gérer de manière optimale un ensemble d'éléments d'informations en les structurant.

L'intérêt des bases de données est de permettre d'effectuer assez simplement des dénombrements, des statistiques et/ou des recherches à l'intérieur des données stockées, suivant des critères précis.

Ex. : Sans base de données : informations brutes.

Jean, qui est un jeune homme de 20 ans, habite à Montpellier, dans l'Hérault.
En Haute-Garonne, à Toulouse, habite Nathalie qui est une jeune fille de 18 ans.
C'est à Toulouse, en Haute-Garonne, que vit Sonia, adolescente de 16 ans.
Michel vit à Montpellier, ville de l'Hérault ; c'est un homme de 27 ans.
Paul, adolescent de 14 ans, habite le département de la Haute-Garonne, à Toulouse.

Avec base de données : reformulation et restructuration des informations brutes en ne gardant que les données pertinentes.

Table *infos* :

prenom	age	sexe	ville	departement
Jean	20	M	Montpellier	Hérault
Nathalie	18	F	Toulouse	Haute-Garonne
Sonia	19	F	Toulouse	Haute-Garonne
Michel	27	M	Montpellier	Hérault
Paul	14	M	Toulouse	Haute-Garonne

Dans la terminologie des bases de données, chaque ligne (ex. : Jean – 20 – M – Montpellier – Hérault) est appelée **enregistrement**, et chaque colonne (*prenom*, a*ge*, *sexe*, *ville*, *departement*) est appelée **champ**.

En général, l'un de ces champs [1] permet d'identifier de manière unique chacun des enregistrements et ainsi éviter les doublons ; ce champ est alors appelé **clef primaire**, ne peut être vide [2] et ne peut avoir la même valeur pour 2 enregistrements différents.

Ex. : Dans la table *infos*, la clef primaire est l'information du prénom contenue dans le champ *prenom*.

L'ensemble des champs constitue ce qu'on appelle une **table** et se représente généralement sous forme de tableau à 2 dimensions [3]. Une **base de données** est un ensemble constitué de 1 ou plusieurs tables.

Si un enregistrement ne peut être identifié de manière unique grâce à la clef primaire, on utilise un second champ ; ce champ est alors appelé **clef secondaire**, ne peut être vide et ne peut avoir la même valeur pour 2 enregistrements différents ayant même clef primaire.

La clef secondaire permet de distinguer les enregistrements en cas de doublon sur la clef primaire ; le couple (clef primaire / clef secondaire) permet alors d'identifier assurément de manière unique les enregistrements d'une même table.

[1] 1 ou plus.
[2] Un champ autre qu'une clef primaire peut être vide.
[3] Cette représentation bidimensionnelle est uniquement visuelle (adaptée à l'humain) et ne peut être assimilée à l'objet informatique correspondant. Autrement dit, une table n'est pas stockée informatiquement sous forme de tableau bidimensionnel.

Ex. : Dans la table *infos*, la clef primaire fait référence au prénom. Si on accepte de traiter les cas où des personnes différentes ont le même prénom, il faut donc envisager d'utiliser une clef secondaire pour les différencier. Cela pourrait être par exemple l'information d'âge contenue dans le champ *age*.

En général, si une base comprend plusieurs tables, c'est pour minimiser la taille occupée par celle-ci, ainsi que d'optimiser la gestion ; les différentes tables possèdent alors des champs communs afin de permettre de recouper les informations stockées.

Ex. : La table précédente peut être scindée en deux tables avec un champ commun (*ville*).

Table *etatcivil* :

prenom	age	sexe	ville
Jean	20	M	Montpellier
Nathalie	18	F	Toulouse
Sonia	19	F	Toulouse
Michel	27	M	Montpellier
Paul	14	M	Toulouse

Table *depts* :

ville	departement
Montpellier	Hérault
Toulouse	Haute-Garonne

En terme de taille, si on considère que chaque champ est codé sur 20 octets [1], les tables font :

- solution 1 : 1 table (5 champs, 5 enregistrements) ⇨ 5 x 5 x 20 = 500 octets ;
- solution 2 : 2 tables (4 champs, 5 enregistrements ; 2 champs, 2 enregistrements) ⇨ (4x5 + 2x2) x 20 = 480 octets.

La solution 2 permet donc de diminuer la taille de la base de données [2].

Si on désire mettre à jour la base, en incluant par exemple le numéro du département en correspondance avec le nom du département, le nombre d'opérations à effectuer est :

- solution 1 : 1 table (5 enregistrements) ⇨ 5 accès à la base ;
- solution 2 : 2 tables mais action uniquement sur la table *depts* (2 enregistrements) ⇨ 2 accès à la base.

ville	departement	no departement
Montpellier	Hérault	34
Toulouse	Haute-Garonne	31

Cette solution 2 permet donc aussi de mettre à jour plus rapidement et facilement la base [3].

1.2 EXPLOITATION D'UNE BASE DE DONNÉES

1.2.1 Le SGBD

Utiliser une base de données permet d'effectuer des dénombrements, des statistiques, des recherches au sein des données stockées selon des critères.

Ex. : Savoir qui habite Toulouse ;
Savoir quels sont les hommes de plus de 19 ans ;
Savoir quelles femmes, et quel est leur âge, qui habitent Montpellier ;
etc.

Lorsque la base de données contient beaucoup d'enregistrements [4], il devient difficile de répondre aux questions par une simple observation des données des tables.

[1] Il s'agit d'une approximation, car il convient de définir chaque champ selon le type de données qu'il doit contenir (texte, nombre, date, etc.) ; ainsi différents champs seront codés sur différents nombres d'octets.

[2] Gain en taille de 4% ; pour 100 enregistrements et 10 villes différentes, solution 1 : 5 x 100 x 20 = 10000 octets, solution 2 : (4 x 100 + 2 x 10) x 20 = 8400 octets, gain de 16%.

[3] Gain en nombre d'accès (donc en temps, et en durée de l'opération) de 60% ; pour 100 enregistrements et 5 départements différents, solution 1 : 100 accès, solution 2 : 5 accès, gain de 95%.

[4] Imaginez la base de données des clients de la SNCF par exemple.

C'est pourquoi une base de données peut être manipulée et interrogée [1] directement en lui soumettant les questions, que l'on formule en utilisant le langage SQL. Une question, mise en forme par le langage SQL, est appelée **requête**.

On pourrait sauvegarder une base de données dans un fichier texte, car elle ne contient que des informations textuelles. Mais on ne saurait appliquer facilement le langage SQL pour interroger ce fichier.

C'est là qu'intervient le gestionnaire de base de données (SGBD [2]) qui propose le support du langage SQL et qui, de plus, optimise le stockage des informations ainsi que les accès en lecture et écriture au sein de la base.

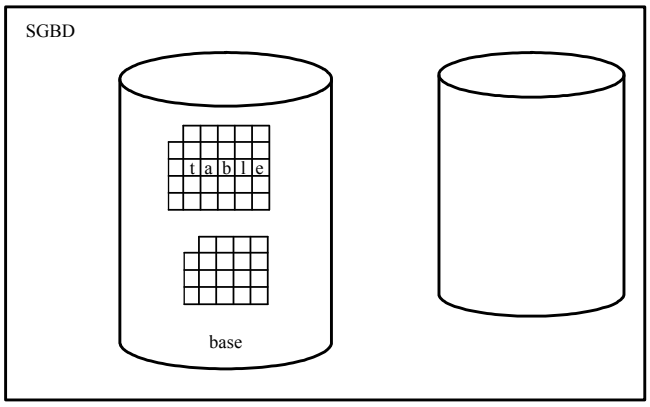

Figure 1.1 : architecture d'un SGBD

Il existe plusieurs SGBD : Access (suite Microsoft Office), Oracle, SQL Server, PostgreSQL, MySQL, HSQL, etc. Même si la plupart des SGBD offrent la possibilité d'interroger une base de données en utilisant le langage SQL, la manière dont chacun stocke les bases lui est propre. C'est pour cela que lorsque l'on parle d'une base, on précise généralement son type (ex. : une base type MySQL).

Parmi ces différents SGBD, on retiendra MySQL car il est gratuit [3], standard et supporté par énormément de systèmes d'exploitation, de langages, librairies et frameworks.

1.2.2 ODBC

La multitude des SGBD disponibles sur le marché et leurs différences, même minimes, sont sources de nombreux problèmes dans l'accès au contenu des bases à partir d'une application. Ces différences rendent parfois l'application peu portable car celle-ci est programmée pour communiquer avec 1 ou plusieurs types de SGBD bien spécifiques.

Plutôt que d'utiliser un driver natif, adapté au SGBD que l'on désire interroger, on peut utiliser une couche intermédiaire afin de pallier ces problèmes. Celle-ci a en charge d'interfacer de façon standard une application à n'importe quel SGBD. Il s'agit de la couche ODBC [4], middleware développé par Microsoft, qui est compatible avec tout SGBD supportant le langage SQL.

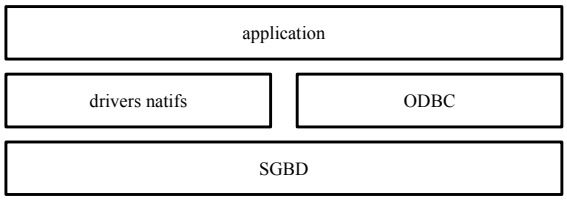

Figure 1.2 : la couche middleware ODBC

---

[1] On parle parfois d'*attaquer une base* lorsque l'on désire l'interroger.

[2] SGBD : Système de Gestion de Base de Données – logiciel gérant les bases de données et permettant de les interroger en utilisant le langage SQL (on parle aussi de SGBDR, Système de Gestion de Base de Données Relationnelles).

[3] Dans une utilisation personnelle ou éducative.

[4] Open DataBase Connectivity (eng) ≡ Connectivité Ouverte des Bases de Données.

2 LE LANGAGE SQL

2.1 DÉFINITIONS

Le langage SQL [1] (Structured Query Language) permet de communiquer avec une base de données afin de la gérer ou de l'interroger. Il s'agit d'un langage déclaratif à la syntaxe très simple, et qui figure parmi les plus utilisés pour l'accès aux bases de données.

SQL permet l'interaction avec le serveur et les informations qu'il héberge en soumettant une commande au SGBD sous la forme d'une requête en suivant la syntaxe :

```
commandeSQL [mot-clef1] paramètre1 [mot-clef2 paramètre2 [...]]
```

Un paramètre peut désigner le nom d'une base, d'une table, d'un champ, une valeur, un critère, etc. Lorsqu'il s'agit d'une valeur alphanumérique, on les encadre par des apostrophes [2] (ex. : `'chaine'`).

2.2 COMMANDES D'ACCÈS À LA BASE

Le langage comprend 4 commandes principales [3] :

- `SELECT` : parcourt la base et lit des enregistrements ;
- `INSERT` : insère un nouvel enregistrement ;
- `UPDATE` : modifie un enregistrement existant ;
- `DELETE` : efface un enregistrement existant.

2.2.1 La commande Select

Pour interroger la base de données, on utilise la commande `SELECT` suivant la syntaxe :

```
SELECT champAExtraire1 [, champAExtraire2 [, ...]]
FROM nomTable
WHERE criteres
```

Le ou les critère(s) de recherche sont définis par la clause `WHERE criteres` suivant la syntaxe `WHERE critere1 [operateurLogique critere2 [...]]` où un critère suit la syntaxe `champCritere operateur valeur`.

La clause `WHERE` peut être omise ; en ce cas les champs indiqués de tous les enregistrements sont lus.
Si on veut lire tous les champs correspondant au(x) critère(s), on peut écrire `*` à la place de la liste des champs.

Ex. : Savoir qui habite Toulouse.

```
SELECT prenom FROM infos WHERE ville='Toulouse'
```

Savoir quel âge ont les hommes de plus de 19 ans.

```
SELECT age FROM infos WHERE sexe='M' AND age>19
```

Savoir quelles femmes, et quel est leur âge, qui habitent Montpellier.

```
SELECT prenom,age FROM infos WHERE sexe='F' AND ville='Montpellier'
```

Connaître toutes les informations concernant les personnes majeures.

---

[1] Structured Query Language (eng) ≡ Langage de Requête Structurée (fr).
[2] Appelées aussi *simple cotes*, en opposition aux guillemets (doubles cotes).
[3] Les commandes SQL sont généralement écrites en majuscules ; mais cela n'est pas une obligation, car SQL ne différencie pas la casse.

```
SELECT * FROM infos WHERE age>=18
```

Connaître tous les prénoms des personnes (sans restriction).

```
SELECT prenom FROM infos
```

Connaître toutes les informations concernant toutes les personnes (soit donc l'intégralité de la table).

```
SELECT * FROM infos
```

Les résultats obtenus correspondent à une portion de la table, que l'on appelle alors **sous-table** ; il s'agit donc d'un tableau à 2 dimensions (même dans le cas où une seule colonne est extraite).

Ex. : Savoir qui habite Toulouse.

```
SELECT prenom FROM infos WHERE ville='Toulouse'
```

prenom	age	sexe	ville	departement
Jean	20	M	Montpellier	Hérault
Nathalie	18	F	Toulouse	Haute-Garonne
Sonia	19	F	Toulouse	Haute-Garonne
Michel	27	M	Montpellier	Hérault
Paul	14	M	Toulouse	Haute-Garonne

La sous-table résultante est donc :

Nathalie
Sonia
Paul

2.2.2 La commande Insert

Pour insérer un nouvel enregistrement dans la base de données, on utilise la commande `INSERT` suivant la syntaxe :

```
INSERT INTO nomTable
VALUES (valeurChamp1, valeurChamp2, ...)
```

Ex. : `INSERT INTO infos VALUES ('Jean', 20, 'M', 'Montpellier', 'Hérault')`

Il faut insérer autant de valeurs qu'il y a de champs dans la table, sinon il faut indiquer les champs que l'on remplit à la suite du nom de la table suivant la syntaxe :

```
INSERT INTO nomTable (nomChamp1 [, nomChamp2 [, ...])
VALUES (valeurChamp1 [, valeurChamp2 [, ...]])
```

Ex. : `INSERT INTO infos (prenom, age, ville) VALUES ('Jean', 20, 'Montpellier')`

2.2.3 La commande Delete

Pour effacer un enregistrement, on utilise la commande `DELETE` suivant la syntaxe :

```
DELETE FROM nomTable
WHERE criteres
```

Ex. : `DELETE FROM infos WHERE prenom='Nathalie'`

2.2.4 La commande Update

Pour mettre à jour un enregistrement, on utilise la commande `UPDATE` suivant la syntaxe :

```
UPDATE nomTable
SET champAModifier1=nouvelleValeur1 [, champAModifier2=nouvelleValeur2 [, ...]]
WHERE criteres
```

Ex. : `UPDATE infos SET age=22 WHERE prenom='Jean'`

2.3 COMMANDES DE GESTION DE BASE

2.3.1 La commande Create

La commande `CREATE` permet de créer une base suivant la syntaxe `CREATE DATABASE nomBase`.

Elle permet également de créer une table en définissant le nom des champs ainsi que le type de chacun d'entre eux suivant la syntaxe :

```
CREATE TABLE nomTable (nomChamp1 type_champ1 [, nomChamp2 type_champ2 [, ...]]).
```

Les différents types de champs possibles sont : `CHAR`, `VARCHAR`, `INT`, etc. [1].

Ex. :
```
CREATE DATABASE cours_php
CREATE TABLE etatcivil
   (prenom VARCHAR(50), age INT, sexe CHAR(1), ville VARCHAR(50))
CREATE TABLE depts (ville VARCHAR(50), departement INT)
```

Nb : Le nombre indiqué entre parenthèses après le type du champ correspond à sa longueur. Ex. : `VARCHAR(50)` indique un champ de type chaîne de caractères variable d'une longueur de 50 caractères.

2.3.2 La commande Drop

La commande `DROP`, utilisée avec la syntaxe `DROP DATABASE nomBase`, permet d'effacer une base ; utilisée avec la syntaxe `DROP TABLE nomTable`, elle permet d'effacer une table.

Ex. :
```
DROP TABLE etatcivil
DROP DATABASE cours_php
```

2.4 LES JOINTURES

Une **jointure** est une mise en relation de deux tables qui sont « jointes » par un champ commun de type clef ; c'est-à-dire que l'une des tables possède un champ qui est aussi la clef primaire d'une autre table. Dans la première table, ce champ est alors qualifié de **clef externe**.

Ex. : Table *etatcivil* :

prenom	age	sexe	ville

Table *depts* :

ville	departement

Le champ *ville* est la clef primaire de la table *depts*, et est aussi présent dans la table *etatcivil* ; le champ *ville* est donc une clef externe pour la table *etatcivil*.

On réalise une jointure avec la commande `SELECT` en mentionnant les 2 tables concernées suivant la syntaxe :

```
SELECT champAExtraire1 [, champAExtraire2 [, ...]]
FROM nomTableX, nomTableY
WHERE criteres
```

Les champs à extraire et utilisés éventuellement dans le ou les critères peuvent être indifféremment de l'une ou l'autre des deux tables.

Généralement on spécifie explicitement la table à laquelle appartient le champ à extraire suivant la syntaxe `nomTable.nomChamp`.

Le critère contient au moins toujours une clause qui met en œuvre la jointure suivant la syntaxe `WHERE tableX.clef_externe = tableY.clef_primaire`. Sans cette clause, la jointure ne sera pas réalisée.

D'autres clauses de sélection peuvent être ajoutées.

[1] cf. A.2.1.

Ex. : Extraire la liste des personnes de plus de 18 ans associées au nom de leur département.

```
SELECT etatcivil.prenom, depts.departement
FROM etatcivil, depts
WHERE etatcivil.ville = depts.ville
AND etatcivil.age > 18
```

`etatcivil.ville = depts.ville` est la clause de jointure ; `etatcivil.age > 18` est une clause de sélection.

2.5 LES SOUS-REQUÊTES

Une **sous-requête** permet de pré-sélectionner un ensemble d'éléments d'une table ou d'utiliser les résultats obtenus pour y effectuer une autre sélection.

Une sous-requête s'écrit exactement comme une requête et donne le même type de résultats, soit donc une sous-table. C'est au sein des données de cette sous-table qu'est alors effectuée la sélection de la requête principale.

Ex. : Savoir qui est plus jeune que l'âge moyen des personnes recensées.

```
SELECT prenom FROM infos WHERE age < (SELECT AVG(age) FROM infos)
```

La sous-requête est `SELECT AVG(age) FROM infos`, elle renvoie la moyenne (`AVG`) des valeurs sélectionnées (`SELECT age from INFOS`) soit donc la moyenne des âges de tout le monde.

A MOTS-CLEF, DÉLIMITEURS ET OPÉRATEURS DU LANGAGE SQL

A.1 ORDRE ALPHABÉTIQUE

Dû à toutes les différences entre les « différents » langages SQL utilisés par les différents SGBDR – on parle parfois de « dialecte » SQL – ; cette liste est donc non-exhaustive.

A	ADD	ALL	ALTER	AND	ANY	AS
	ASC	(AUTOINC)	AVG			
B	BETWEEN	(BIGINT)	(BINARY)	BIT	(BOOLEAN)	(BYTES)
C	CASCADE	CHAR	CHARACTER	CHECK	COLUMN	COMMIT
	CONSTRAINT	COUNT	CREATE			
D	DATABASE	DATE	DATETIME	DEC	DECIMAL	DEFAULT
	DELETE FROM	DESC	DISTINCT	DOUBLE PRECISION		DROP
E	ESCAPE	EXCEPT	EXISTS			
F	FLOAT	FOREIGN KEY	FROM			
G	GRANT	GROUP BY				
H	HAVING					
I	IN	INDEX	INSERT INTO	INT	INTEGER	INTERSECT
	INTERVAL	IS NOT NULL	IS NOT OF	IS NULL	IS OF	
L	LIKE	(LOGICAL)				
M	MAX	(MEDIUMINT)	MIN	MINUS	MODIFY	
N	NATIONAL	NCHAR	NO ACTION	NOCHECK	NOT	NOT NULL
	NULL	NUMERIC				
O	ON	ON DELETE	ON UPDATE	OR	ORDER BY	
P	PRIMARY KEY					
R	REAL	REFERENCES	REVOKE	ROLE	ROLLBACK	
S	SELECT	SET	(SHOW)	SMALLINT	SOME	SUM
T	TABLE	(TEXT)	TIME	TIMESTAMP	(TINYINT)	TO
	TRUNCATE					
U	UNION	UNION ALL	UNIQUE	(UNSIGNED)	UPDATE	USER
V	VALUES	VARCHAR	VARYING	VIEW		
W	WHERE	WITH CHECK	WITH NOCHECK			

`(...)` : non-normalisé.

Les mots-clef sont réservés, donc inutilisables comme noms de base, table, champ, etc.

A.2 CATÉGORIES

A.2.1 Définition des données

Mots-clef LDD (Langage de Définition de Données [1]) : gestion de la structure des bases et des tables.

ADD	Clause d'ajout d'un champ à une table, ou d'une contrainte à un champ (cf. ALTER).
ALTER	Modification d'une base, d'une table ou d'une vue.
ALL	Désignation de toutes les contraintes à activer ou désactiver (cf. ALTER TABLE / CHECK/ NOCHECK CONSTRAINT).
AS	Clause de création d'une vue (à associer avec CREATE ; cf. VIEW).
ASC	Clause d'ordonnancement normal d'une liste ordonnée (à associer avec CREATE INDEX / ON ; cf. DESC) – *par défaut*.
CHECK	Activation d'une ou plusieurs contrainte(s) (cf. ALTER TABLE et CONSTRAINT).
COLUMN	Désignation d'un champ (cf. ALTER).
CONSTRAINT	Désignation d'une contrainte de champ (cf. ALTER TABLE / CHECK/NOCHECK).
CREATE	Création d'une base, d'une table ou d'une vue.
DATABASE	Désignation d'une base (à associer avec ALTER, CREATE, ou DROP).
DEFAULT	Modificateur de spécification d'une valeur par défaut d'un champ.
DESC	Clause d'ordonnancement inverse d'une liste ordonnée (à associer avec CREATE INDEX / ON ; cf. ASC).
DROP	Suppression d'une base, d'une table ou d'une vue / Clause de suppression d'un champ, ou d'une contrainte (cf. ALTER).
INDEX	Désignation d'un index : liste ordonnée d'un ou plusieurs champ(s) d'une table (à associer avec ALTER, CREATE, ou DROP ; cf. ON).
MODIFY	Clause de modification d'un champ d'une table, ou d'une contrainte (cf. ALTER).
NOCHECK	Désactivation d'une ou plusieurs contrainte(s) (cf. ALTER TABLE et CONSTRAINT).
ON	Clause de désignation de la table et du ou des champ(s) sur lesquels construire un index (à associer avec CREATE INDEX).
VIEW	Désignation d'une vue : table virtuelle résultant de l'exécution d'une requête (à associer avec ALTER, CREATE, DROP, ou TRUNCATE ; cf. AS).
TABLE	Désignation d'une table (à associer avec ALTER, CREATE, DROP, ou TRUNCATE).
TRUNCATE	Suppression du contenu d'une table ou d'une vue.
WITH CHECK	Activation d'une contrainte ajoutée à un champ (cf. ALTER TABLE et ADD).
WITH NOCHECK	Désactivation d'une contrainte ajoutée à un champ (cf. ALTER TABLE et ADD).

Contraintes de champs.

CASCADE	Clause d'étendue de modification ou suppression de la clef étrangère d'une table à la clef référente de l'autre table (cf. FOREIGN KEY / REFERENCES / ON DELETE/ON UPDATE).
CHECK	Règle à respecter pour une contrainte de champ (cf. CONSTRAINT).
CONSTRAINT	Définition d'une contrainte de champ (cf. CHECK).
FOREIGN KEY	Modificateur de désignation d'un champ en tant que clef étrangère (cf. REFERENCES).
NO ACTION	Clause de non-étendue de modification ou suppression de la clef étrangère d'une table à la clef référente de l'autre table (cf. FOREIGN KEY / REFERENCES / ON DELETE/ON UPDATE).
NOT NULL	Modificateur d'obligation d'affectation d'une valeur à un champ (valeur non nulle).
NULL	Modificateur de possibilité d'affectation d'une valeur nulle à un champ.
ON DELETE	Clause d'action à effectuer lors de la suppression d'une clef étrangère (à associer avec FOREIGN KEY / REFERENCES ; cf. CASCADE et NO ACTION).
ON UPDATE	Clause d'action à effectuer lors de la modification d'une clef étrangère (à associer avec FOREIGN KEY / REFERENCES ; cf. CASCADE et NO ACTION).
PRIMARY KEY	Modificateur de désignation d'un champ en tant que clef primaire.
REFERENCES	Désignation de la table et du champ à associer à une clef étrangère (cf. FOREIGN KEY).
UNIQUE	Modificateur d'obligation de valeur non-redondante d'un champ.

[1] DDL : Data Definition Language (eng).

Types de données alphanumériques.

CHAR	Valeur alphanumérique de longueur fixe avec indication de la longueur (identique à CHARACTER).
CHARACTER	Valeur alphanumérique de longueur fixe avec indication de la longueur (identique à CHAR).
NATIONAL	Dépendance par rapport au jeu de caractères du pays local (à associer avec BIT, CHAR ou CHARACTER : identique à NCHAR).
NCHAR	Valeur alphanumérique de longueur fixe dépendant du jeu de caractères du pays local (identique à NATIONAL CHAR et NATIONAL CHARACTER).
VARCHAR	Valeur alphanumérique de longueur variable avec indication éventuelle de la longueur maximale (identique à CHAR VARYING, et CHARACTER VARYING).
VARYING	Longueur variable avec indication éventuelle de la longueur maximale (à associer avec BIT, CHAR ou CHARACTER : identique à VARCHAR).

Types de données numériques.

BIT	Chaîne de bits de longueur fixe.
DEC	Décimal avec indication des nombres de chiffres total et décimal (identique à DECIMAL et NUMERIC).
DECIMAL	Décimal avec indication des nombres de chiffres total et décimal (identique à DEC et NUMERIC).
DOUBLE PRECISION	Réel double précision avec indication du nombre de chiffres total.
FLOAT	Réel avec indication des nombres de chiffres total et décimal.
INT	Entier sur 4 octets (identique à INTEGER).
INTEGER	Entier sur 4 octets (identique à INT).
NUMERIC	Décimal avec indication des nombres de chiffres total et décimal (identique à DEC et DECIMAL).
REAL	Réel simple précision avec indication du nombre de chiffres décimal.
SMALLINT	Entier sur 2 octets.

Types de données temporels.

DATE	Date.
DATETIME	Date et heure formatées.
INTERVAL	Intervalle de date et de temps.
TIME	Heure.
TIMESTAMP	Date et heure.

Types de données non-normalisés.

AUTOINC	Entier à incrémentation automatique.
BIGINT	Entier sur 8 octets.
BINARY	Chaîne de bits de longueur variable (identique à BYTES).
BOOLEAN	Booléen (identique à LOGICAL).
BYTES	Chaîne de bits de longueur variable (identique à BINARY).
LOGICAL	Booléen (identique à BOOLEAN).
MEDIUMINT	Entier sur 3 octets.
TEXT	Valeur alphanumérique de longueur variable avec indication éventuelle de la longueur maximale (identique à VARCHAR, CHAR VARYING et CHARACTER VARYING).
TINYINT	Entier sur 1 octet.
UNSIGNED	Type numérique signé (modificateur de type).

A.2.2 Manipulation des données

Mots-clef LMD (Langage de Manipulation de Données [1]) : gestion du contenu des tables.

ALL	Clause de sélection de tous les enregistrements d'une consultation (à associer avec SELECT / FROM ; cf. DISTINCT).
AS	Clause de spécification d'alias (/synonyme) d'une table ou d'un champ (à associer avec SELECT / FROM).
ASC	Clause d'ordonnancement normal d'une extraction selon un ou plusieurs champ(s) (à associer avec SELECT / FROM / ORDER BY ; cf. DESC) – *par défaut*.
DELETE FROM	Effacement d'un ou plusieurs enregistrement(s) d'une table suivant un ou plusieurs critère(s) éventuels.
DESC	Clause d'ordonnancement inverse d'une extraction selon un ou plusieurs champ(s) (à associer avec SELECT / FROM / ORDER BY ; cf. ASC).
DISTINCT	Clause de sélection de tous les enregistrements distincts d'une consultation en excluant les redondances (à associer avec SELECT / FROM ; cf. ALL).
FROM	Désignation de la table à consulter (cf. SELECT).
GROUP BY	Clause de regroupement des résultats d'une consultation selon un ou plusieurs champ(s) (à associer avec SELECT / FROM et AVG, COUNT, MAX, MIN ou SUM ; cf. ASC, DESC et HAVING).
HAVING	Clause de consultation spécifique au regroupement des résultats d'une consultation (à associer avec SELECT / FROM / GROUP BY).
INSERT	Insertion d'un nouvel enregistrement dans une table avec désignation éventuelle des champs de destination (cf. VALUES).
ORDER BY	Clause d'ordonnancement normal ou inverse d'une extraction selon un ou plusieurs champ(s) (à associer avec SELECT / FROM ; cf. ASC, DESC).
SELECT	Consultation d'un ou plusieurs champs parmi tous les enregistrements d'une table suivant un ou plusieurs critère(s) éventuels (cf. FROM).
SET	Désignation du ou des champ(s) à modifier et de la ou des nouvelle(s) valeur(s) (cf. UPDATE).
UPDATE	Mise à jour d'un ou plusieurs champ(s) de un ou plusieurs enregistrement(s) suivant un ou plusieurs critère(s) (cf. SET).
VALUES	Désignation des valeurs des champs du nouvel enregistrement à insérer dans une table (cf. INSERT et INTO).
WHERE	Clause de spécification d'un ou plusieurs critère(s) de consultation (à associer avec SELECT / FROM).

Fonctions d'agrégat des résultats d'une consultation.

AVG	Calcul de la moyenne des valeurs d'un champ (à associer avec SELECT / FROM ou SELECT / FROM / ORDER BY ; cf. COUNT, MAX, MIN et SUM).
COUNT	Comptabilisation du nombre d'enregistrements d'un champ (à associer avec SELECT / FROM ou SELECT / FROM / ORDER BY ; cf. AVG, MAX, MIN et SUM).
MAX	Renvoi le maximum des valeurs d'un champ (à associer avec SELECT / FROM ou SELECT / FROM / ORDER BY ; cf. AVG, COUNT, MIN et SUM).
MIN	Renvoi le minimum des valeurs d'un champ (à associer avec SELECT / FROM ou SELECT / FROM / ORDER BY ; cf. AVG, COUNT, MAX et SUM).
SUM	Calcul de la somme des valeurs d'un champ (à associer avec SELECT / FROM ou SELECT / FROM / ORDER BY ; cf. AVG, COUNT, MAX et MIN).

Clauses de jointures : FULL OUTER JOIN, INNER JOIN, LEFT OUTER JOIN, ON, RIGHT OUTER JOIN, SELF JOIN.

[1] DML : Data Manipulation Language (eng).

A.2.3 Gestion des droits

Mots-clef LCD (Langage de Contrôle de Données [1]) : gestion des privilèges des utilisateurs (droits d'accès).

`ALTER`	Modification d'un utilisateur ou d'un rôle.
`CREATE`	Création d'un utilisateur ou d'un rôle.
`DROP`	Suppression d'un utilisateur ou d'un rôle.
`GRANT`	Attribution d'un ou plusieurs privilège(s) à un utilisateur ou à un rôle sur des instructions ou des objets structurels (cf. `ON` et `TO`).
`FROM`	Désignation du ou des utilisateur(s) concerné(s) par la révocation de privilèges (cf. `REVOKE` ; identique à `TO`).
`ON`	Désignation de ou des objet(s) concerné(s) par l'allocation de privilèges (cf. `GRANT` et `REVOKE`).
`REVOKE`	Révocation d'un ou plusieurs privilège(s) à un utilisateur ou un à rôle sur des instructions ou des objets structurels (cf. `ON` et `TO`).
`ROLE`	Désignation d'un rôle (catégories d'utilisateurs) (à associer avec `ALTER`, `CREATE`, ou `DROP`).
`TO`	Désignation du ou des utilisateur(s) concerné(s) par l'allocation ou la révocation de privilèges (cf. `GRANT` et `REVOKE`).
`USER`	Désignation d'un utilisateur (à associer avec `ALTER`, `CREATE`, ou `DROP`).

Mais aussi : `ACCOUNT`, `ALL`, `ALL PRIVILEGES`, `AS`, `BY`, `DEFAULT TABLESPACE`, `DIRECTORY`, `EXTERNALLY`, `GLOBALLY`, `IDENTIFIED`, `IDENTIFIED BY`, `JAVA`, `NOT IDENTIFIED`, `PASSWORD EXPIRE`, `PROFILE`, `PUBLIC`, `QUOTA`, `UNLIMITED`, `USING`, `WITH GRANT OPTION`, …

A.2.4 Gestion des transactions

Mots-clef LCT (Langage de Contrôle de Transactions [2]) : gestion des transactions (automatisation d'ordres).

`COMMIT`	Sauvegarde des modifications effectuées durant la session courante.
`ROLLBACK`	Annulation des modifications effectuées depuis le début de la session courante.

A.3 DÉLIMITEURS ET OPÉRATEURS

A.3.1 Délimiteurs

`' '`	Encadre une valeur alphanumérique (identique à `" "`).
`" "`	Encadre une valeur alphanumérique (identique à `' '`).
`( )`	Encadre une liste de champs ou de valeurs.
`[ ]`	Encadre et délimite le nom d'une base, d'une table ou d'un champ.
`,`	Sépare deux éléments d'une liste.

[1] DCL : Data Control Language (eng).
[2] TCL : Transaction Control Language (eng).

A.3.2 Opérateurs de critères

=	Égalité.
!=	Différence (identique à <> et ^=).
<>	Différence (identique à != et ^=).
^=	Différence (identique à != et <>).
<	Infériorité stricte.
<=	Infériorité ou égalité.
>	Supériorité stricte.
>=	Supériorité ou égalité.
!<	Non-infériorité stricte (identique à >=).
!>	Non-supériorité stricte (identique à <=).
ALL	Validation du test pour chaque valeur de la première opérande avec toutes les valeurs de la seconde opérande (cf. ANY et SOME).
AND	ET logique.
ANY	Validation du test pour chaque valeur de la première opérande avec au moins l'une des valeurs des deux expressions (cf. ALL ; identique à SOME).
BETWEEN	Test d'appartenance à un intervalle (à associer avec AND).
ESCAPE	Désignation du caractère d'échappement dans le test de conformité avec une expression rationnelle (cf. LIKE).
EXISTS	Test de l'existence dans l'opérande.
IN	Validation du test pour toutes les valeurs communes aux deux opérandes.
IS NOT NULL	Test de non-nullité.
IS NULL	Test de nullité.
IS NOT OF	Test de l'absence d'une donnée du type précisé.
IS OF	Test de la présence d'une donnée du type précisé.
LIKE	Test de correspondance avec une expression relationnelle, avec désignation éventuelle du caractère d'échappement (cf. ESCAPE).
NOT	NON logique.
OR	OU logique.
SOME	Validation du test pour chaque valeur de la première opérande avec au moins l'une des valeurs des deux expressions (cf. ALL ; identique à ANY).
UNIQUE	Test de l'absence de doublons.

A.3.3 Opérateurs de requêtes

EXCEPT	Résultats issus de la première requête et absents de la seconde (identique à MINUS).
INTERSECT	Résultats communs de deux requêtes.
MINUS	Résultats issus de la première requête et absents de la seconde (identique à EXCEPT).
UNION	Résultats combinés de deux requêtes, doublons exclus.
UNION ALL	Résultats combinés de deux requêtes, doublons inclus.

B RÉFÉRENCE DU LANGAGE SQL

B.1 HISTORIQUE

1970 : Introduction par T. Codd de l'algèbre relationnelle dans le traitement automatique de l'information, permettant la formalisation des opérations sur les ensembles, produisant ainsi un futur modèle utilisé par les bases de données relationnelles.
1972 : Début du projet INGRES par l'équipe de G. Wong à l'université de Berkeley.
1975 : Création de QUIEL comme outil d'interrogation des données par l'équipe de G. Wong à l'université de Berkeley.
1976 : Création par P. Chen du modèle entité-relation comme outil de modélisation dans le traitement automatique des données.
1976 : Création du langage d'interrogation SEQUEL par IBM via son projet System R.
1979 : SQL 1.0.
1986 : Standardisation américaine du langage sous l'appellation SQL-ANSI, nommé aussi SQL-86.
1989 : Standardisation internationale du langage sous l'appellation SQL-ISO.
1992 : SQL 2.0 pour SQL-ANSI et SQL-ISO, nommé SQL-92.
1999 : Évolution de SQL 2.0 pour SQL-ANSI et SQL-ISO, nommé SQL-99.
2003 : SQL 3.0, nommé SQL:2003.

L'avenir du langage en tant que tel est très incertain ; son successeur, le langage XQL (XML Query Language) vise la coopération maximale avec le langage XML (eXtended Markup Language), issu du web.

B.2 MODÉLISATION

C BIBLIOGRAPHIE

L'altruiste, *Le langage SQL*, http://www.laltruiste.com, 2006 ;

Wazir Dino, *Cours Java – JDBC*, TS IRIS – LEGT Louis Modeste-Leroy – Évreux, 2002 ;

Développez.com, *Langage SQL*, http://sql.developpez.com/, 2006 ;

Heute Thomas & Damien – ToutEstFacile, *SQL facile*, http://www.sqlfacile.com/, 2007 ;

Muir Christian, *Requêtes SQL*, http://pagesperso.scola.ac-paris.fr/cmuir/sql_01.htm, 2006 ;

Frédéric « OraFrance », *Syntaxe SQL*, http://orafrance.free.fr, 2003 ;

Grange Marc, *Applications des BDD en Sciences Humaines et Sociales*, http://nte-socio.univ-lyon2.fr/Marc_Grange/IntroBD.htm, 2006 ;

Wikipédia – l'encyclopédie libre, http://fr.wikipedia.org, 2006.