LE LANGAGE SQL2

1. INTRODUCTION

Les serveurs de données relationnels présentent aujourd'hui une interface externe sous forme d'un langage de recherche et mise à jour, permettant de spécifier les ensembles de données à sélectionner ou à mettre à jour à partir de propriétés des valeurs, sans dire comment retrouver les données. Ainsi, les opérations directement utilisables par les usagers sont en général celles des langages dits assertionnels. Plusieurs langages assertionnels permettant de manipuler des bases de données relationnelles ont été proposés, en particulier QUEL [Zook77], QBE [Zloof77] et SQL [IBM82, IBM87]. Aujourd'hui, le langage SQL est normalisé [ISO89, ISO92] et constitue le standard d'accès aux bases de données relationnelles. Les autres interfaces par menus, fenêtres, grilles, etc., ou de programmation type langage de 3^{e} ou 4^{e} génération, sont le plus souvent offertes au-dessus du langage SQL. Celui-ci constitue donc le point d'entrée obligatoire des SGBD relationnels. QUEL était le langage proposé par l'université de Berkeley pour son système INGRES : il est aujourd'hui peu utilisé dans l'industrie. QBE est un langage par grille dérivé de la logique qui est souvent offert au-dessus de SQL.

De manière générale, SQL comme les autres langages qui ont été proposés (e.g., QUEL) utilisent tous des critères de recherche (encore appelés qualifications) construits à partir de la logique des prédicats du premier ordre. Ils comportent quatre opérations de base :

- **la recherche** (mot clé SELECT en SQL, RETRIEVE en QUEL) permet de retrouver des tuples ou parties de tuples vérifiant la qualification citée en arguments ;
- **l'insertion** (mot clé INSERT en SQL, APPEND en QUEL) permet d'ajouter des tuples dans une relation ; les tuples peuvent être fournis par l'utilisateur ou construits à partir de données existant déjà dans la base ;
- **la suppression** (mot clé DELETE en SQL, SUPPRESS en QUEL) permet de supprimer d'une relation les tuples vérifiant la qualification citée en argument ;

- **la modification** (mot clé UPDATE en SQL, REPLACE en QUEL) permet de mettre à jour les tuples vérifiant la qualification citée en argument à l'aide de nouvelles valeurs d'attributs ou de résultats d'opérations arithmétiques appliquées aux anciennes valeurs.

Le langage assertionnel SQL fut introduit commercialement tout d'abord par IBM [IBM82]. Il résultait alors d'une évolution du langage SEQUEL 2 [Chamberlin76] initialement développé au centre de recherches de San-José comme un langage expérimental appelé SQUARE [Boyce75] pour paraphraser en anglais les expressions de l'algèbre relationnelle. Aujourd'hui, l'ISO a normalisé le langage SQL pour manipuler les bases de données relationnelles et ceci à plusieurs niveaux. Le niveau SQL1 [ISO89] correspond à la norme de base acceptée en 1989, telle qu'elle est aujourd'hui appliquée par la plupart des constructeurs. SQL1 permet l'expression des requêtes composées d'opérations de l'algèbre relationnelle et d'agrégats. En plus des fonctionnalités de définition, de recherche et de mise à jour, SQL1 comporte aussi des fonctions de contrôle qui n'appartiennent pas à proprement parler au modèle relationnel, mais qui sont nécessaires pour programmer des applications transactionnelles.

Le niveau SQL2 [ISO92] est sous-divisé en trois niveaux, respectivement entrée, intermédiaire et complet. Le niveau entrée peut être perçu comme une amélioration de SQL1, alors que les niveaux intermédiaire et complet permettent de supporter totalement le modèle relationnel avec des domaines variés, tels date et temps. SQL3 est constitué d'un ensemble de propositions nouvelles traitant plus particulièrement des fonctionnalités objets et déductives.

Ce chapitre est organisé comme suit. Les quatre sections qui suivent sont consacrées à l'étude du standard SQL1. Nous étudions successivement la définition de schéma, la recherche de données, l'expression des mises à jour et l'intégration aux langages de programmation pour écrire des transactions. La section qui suit présente les fonctionnalités du nouveau standard SQL2. En conclusion, nous résumons brièvement les propositions émises dans le cadre SQL3 et nous soulignons l'importance de SQL.

Nous utilisons des notations syntaxiques en BNF (*Backus-Naur Form*), avec les extensions suivantes :

- les crochets ([]) indiquent des éléments optionnels ;
- les pointillés suivent un élément qui peut être répété plusieurs fois ;
- les accolades groupent comme un seul élément une séquence d'éléments ;
- un exposant plus ($^+$) indique que l'élément qui précède peut être répété n fois (n>0), chaque occurrence étant séparée de la précédente par une virgule ;
- un exposant multiplié (*) indique que l'élément qui précède peut être répété n fois (n≥0), chaque occurrence étant séparée de la précédente par une virgule.

2. SQL1 : LA DEFINITION DE SCHEMAS

SQL1 permet de définir des schémas de bases de données composés de tables et de vues. Un schéma est simplement identifié par un identifiant autorisant l'accès à la base. Au niveau d'un

schéma sont associés des autorisations d'accès aux tables. Au niveau de la table, des contraintes d'intégrité peuvent être définies.

2.1 Création de tables

SQL1 permet de créer des relations sous forme de tables et de définir lors de la création des contraintes d'intégrité variés sur les attributs. Ainsi, une commande de création permet de spécifier le nom de la table et de définir les éléments de table correspondant aux colonnes ou aux contraintes, selon la syntaxe suivante :

CREATE TABLE <NOM DE TABLE> (<ELEMENT DE TABLE>$^{+}$)

Un nom de table peut être un nom simple (par exemple VINS) ou un nom composé d'un nom de schéma (identifiant d'autorisation d'accès à la base, par exemple DEGUSTATION) suivi par le nom simple de table en notation pointée (par exemple, DEGUSTATION.VINS).

Un élément de table est soit une définition de colonne, soit une définition de contrainte, comme suit :

<ELEMENT DE TABLE> ::= <DEFINITION DE COLONNE> |
<CONTRAINTE DE TABLE>

Une colonne est définie par un nom et un type de données. Une valeur par défaut peut être précisée. Une contrainte de colonne peut aussi être définie à ce niveau. On obtient donc la syntaxe suivante :

<DEFINITION DE COLONNE> : := <NOM DE COLONNE> <TYPE DE DONNEES>
[<CLAUSE DEFAUT>] [<CONTRAINTE DE COLONNE>]

Les types de données supportés sont les chaînes de caractères de longueurs fixes — CHAR(<longueur>) —, la valeur par défaut de la longueur étant 1, les numériques exactes — NUMERIC et DECIMAL avec précision et échelle optionnelles, INTEGER et SMALLINT —, les numériques approchés — FLOAT avec précision optionnelle, REAL et DOUBLE PRECISION.

La clause défaut permet simplement de spécifier une valeur par défaut selon la syntaxe DEFAULT <valeur>, la valeur NULL étant permise. Nous examinerons les contraintes d'intégrité de table et de colonne dans la section qui suit.

Afin d'illustrer les possibilité introduites, la figure VII.1 présente les commandes permettant de créer la base dégustation, pour l'instant sans contrainte d'intégrité. Le schéma de la base obtenu est composé des trois relations suivantes :

VINS (NV, CRU, MILLESIME, DEGRE, QUALITE)
BUVEURS (NB, NOM, ADRESSE, TYPE)
ABUS (NB, NV, DATE, QUANTITE).

```
CREATE  SCHEMA AUTHORIZATION DEGUSTATION
CREATE  TABLE VINS (NV INT, CRU CHAR(12), MILLESIME INT, DEGRE
     DEC(3,1), QUALITE CHAR)
CREATE TABLE BUVEURS (NB INT, NOM CHAR(20), ADRESSE CHAR(30),
     TYPE CHAR(4))
CREATE TABLE ABUS(NB INT, NV INT, DATE DEC(6),
     QUANTITE SMALLINT)
```

Figure VII.1 — Création de la base Dégustation

2.2 Expression des contraintes d'intégrité

Les **contraintes de colonnes** permettent de spécifier différentes contraintes d'intégrité portant sur un seul attribut, y compris les contraintes référentielles. Les différentes variantes possibles sont :

- valeur nulle impossible (syntaxe `NOT NULL`),
- unicité de l'attribut (syntaxe `UNIQUE` ou `PRIMARY KEY`),
- contrainte référentielle — syntaxe `REFERENCES` <table référencée> [(<colonne référencée>)] —, le nom de la colonne référencée étant optionnel s'il est identique à celui de la colonne référençante,
- contrainte générale (syntaxe `CHECK` <condition>) ; la condition est une condition pouvant spécifier des plages ou des listes de valeurs possibles (voir condition de recherche ci-dessous).

Les **contraintes de relations** peuvent porter sur plusieurs attributs. Ce peut être des contraintes d'unicité, référentielles ou générales. Elles sont exprimées à l'aide des phrases suivantes :

- contrainte d'unicité `UNIQUE` <attribut>$^+$,
- contrainte référentielle, permettant de spécifier quelles colonnes référencent celles d'une autre table, [`FOREIGN KEY` (<colonne référençante>$^+$)] `REFERENCES` <table référencée> [(<colonne référencée>$^+$)],
- contrainte générale `CHECK` <condition>.

Par exemple, la création de la relation `ABUS` avec contraintes d'intégrité pourra être effectuée par la commande de la figure VII.2. Cette commande précise que les attributs `NB` et `NV` ne doivent pas être nuls et doivent exister dans les relations `BUVEURS` et `VINS` respectivement, que la date est comprise entre le premier janvier 80 et le 31 décembre 99, que la quantité doit être comprise entre 1 et 100 avec une valeur par défaut de 1.

```
CREATE TABLE ABUS (
NB INT NOT NULL,
NV INT NOT NULL REFERENCES VINS(NV),
DATE DEC(6) CHECK (DATE BETWEEN 010180 AND 311299),
QUANTITE SMALLINT DEFAULT 1,
PRIMARY KEY(NB, NV, DATE),
FOREIGN KEY NB REFERENCES BUVEURS,
CHECK (QUANTITE BETWEEN 1 AND 100) )
```

Figure VII.2 — Création de la table ABUS avec contraintes d'intégrité

2.3 Définition des vues

SQL1 permet de définir des vues au niveau du schéma. Rappelons qu'une vue est une table virtuelle calculée à partir des tables de base par une question. La syntaxe de la commande de création de vues est la suivante :

```
CREATE VIEW <NOM DE TABLE> [(<NOM DE COLONNE>+)]
AS <SPECIFICATION DE QUESTION>
[WITH CHECK OPTION]
```

Une vue est modifiable s'il est possible d'insérer et de supprimer des tuples dans la base au-travers de la vue. Dans ce cas, les tuples sont insérés ou supprimés dans la première table référencée par la question. La vue doit alors contenir toutes les colonnes de cette table. L'option `WITH CHECK OPTION` permet de s'assurer que les tuples insérés vérifient les conditions exprimées dans la question, c'est-à-dire qu'ils appartiennent bien à la vue. Cela permet d'imposer des contraintes d'intégrité lors des mises à jour au travers de la vue (par exemple, le fait qu'un tuple de la vue doit référencé un tuple d'une autre table).

À titre d'illustration, voici une vue `GROS-BUVEURS` définie simplement comme la table virtuelle contenant le nom et le prénom des buveurs de type « `GROS` ». Cette vue n'est pas modifiable. Cette définition montre déjà un premier exemple de question SQL très simple, de type sélection.

```
CREATE VIEW GROS-BUVEURS (NOM, PRENOM)
AS    SELECT NOM, PRENOM
FROM BUVEURS
WHERE TYPE = "GROS"
```

Figure VII.3 — Exemple de définition de vue

2.4 Suppression des tables

La suppression des tables n'est pas permise en SQL1. La plupart des systèmes permettent cependant de détruire une relation par la commande :

```
DROP TABLE <NOM DE TABLE>.
```

Par exemple, la destruction de la relation VINS s'effectuera par la commande :

```
DROP TABLE VINS.
```

2.5 Droits d'accès

La gestion des droits d'accès aux tables est décentralisée : il n'existe pas d'administrateur global attribuant des droits. Chaque créateur de table obtient tous les droits d'accès à cette table, en particulier les droits d'effectuer les actions de sélection (SELECT), d'insertion (INSERT), de suppression (DELETE), de mise à jour (UPDATE) et aussi de référencer la table dans une contrainte (REFERENCES). Il peut ensuite passer ses droits sélectivement à d'autres utilisateurs ou à tous le monde (PUBLIC). Un droit peut être passé avec le droit de le transmettre (WITH GRANT OPTION) ou non.

SQL1 propose ainsi une commande de passation de droits dont la syntaxe est la suivante :

```
GRANT <PRIVILEGES> ON <NOM DE TABLE> TO <RECEPTEUR>+
[WITH GRANT OPTION]
```

avec :

```
<PRIVILEGES> ::= ALL PRIVILEGES | <ACTION>+
<ACTION>     ::= SELECT | INSERT | UPDATE [(<NOM DE COLONNE>+)]
                 | REFERENCE [(<NOM DE COLONNE>+)]
<RECEPTEUR>  ::= PUBLIC | <IDENTIFIANT D'AUTORISATION>
```

L'ensemble des privilèges (ALL PRIVILEGES) inclut les droits d'administration (changement de schéma et destruction de la relation). Le récepteur peut être un utilisateur ou un groupe d'utilisateurs, selon l'identifiant d'autorisation donné.

Par exemple, la passation des droits de consultation et mise à jour de la table VINS à l'utilisateur Poivrot s'effectuera comme indiqué ci-dessous. La présence de l'option de passation (GRANT OPTION) permet à Poivrot de passer ce droit.

```
GRANT SELECT, UPDATE ON VINS TO POIVROT
WITH GRANT OPTION
```

Bien que non prévue dans la norme de 1989, la commande REVOKE permet de retirer un droit à un utilisateur. Sa syntaxe est la suivante :

REVOKE <privilèges> **ON** <nom de table> **FROM** <récepteur>.

Par exemple, la commande qui suit permet de retirer le droit donné ci-dessus ainsi que tous les droits qui en dépendent (c'est-à-dire ceux de sélection ou mise à jour de la table VINS passés par Poivrot).

```
REVOKE SELECT, UPDATE ON VINS FROM POIVROT.
```

3. SQL1 : LA RECHERCHE DE DONNEES

Dans cette section, nous étudions la requête de recherche qui est à la base de SQL, le fameux SELECT. Nous commençons par des exemples à partir de cas simples dérivés de l'algèbre relationnelle pour aboutir au cas général.

3.1 Expression des projections

Rappelons qu'une projection effectue l'extraction de colonnes (attributs) spécifiées d'une relation, puis élimine les tuples en double. SQL n'élimine pas les doubles, à moins que cela soit explicitement demandé par le mot clé `DISTINCT`, l'option par défaut étant `ALL`. SQL généralise la projection en ce sens qu'il est possible d'appliquer des fonctions de calculs sur les colonnes extraites. Les fonctions de calculs permises sont en particulier les fonctions arithmétiques d'addition, soustraction, multiplication et division. Une projection s'exprime à l'aide du langage SQL par la clause :

SELECT [ALL|DISTINCT] <EXPRESSION DE VALEURS>$^+$
FROM <NOM DE TABLE> [<NOM DE VARIABLE>]

Une expression de valeurs est une expression arithmétique (composée avec les opérateurs binaires +, -, * et /), éventuellement parenthésée, de spécifications de constantes ou de colonnes. Une spécification de constante est soit une constante, une variable de programme ou le nom de l'usager (mot clé `USER`). Une spécification de colonne désigne le nom d'une colonne précédé d'un désignateur de relation éventuel (nom de table ou variable), comme suit :

<SPECIFICATION DE COLONNE> ::= [<DESIGNATEUR>.] <NOM DE COLONNE>
<DESIGNATEUR> ::= <NOM DE TABLE> | <NOM DE VARIABLE>

L'usage d'un nom de variable nécessite de définir dans la clause `FROM` une variable dite de corrélation, permettant de désigner la table par cette variable. De telles variables évitent de répéter le nom de la table dans le cas de questions portant sur plusieurs tables, comme nous le verrons ci-dessous. Notez aussi qu'il est possible d'utiliser une étoile (*) à la place de la liste d'expression de valeurs, cela signifiant simplement que l'on désire lister tous les attributs de la table référencée dans la clause `FROM`.

Nous illustrons la projection en utilisant la table `VINS` dont le schéma a été défini ci-dessus. La première question (Q1) indiquée figure VII.4 permet d'obtenir pour tous les vins les crus, millésimes et quantité d'alcool pur contenue dans 1000 litres, avec doubles éventuels. La deuxième question (Q2) effectue la recherche de tous les tuples de la table `VINS` sans double.

```
(Q1)   SELECT CRU, MILLESIME, (DEGRE/100)*1000
       FROM VINS

(Q2)   SELECT DISTINCT *
       FROM VINS
```

Figure VII.4 — Exemples de projections

3.2 Expression des sélections

Une sélection est une combinaison d'une restriction suivie d'une projection. Une sélection s'exprime comme une projection avec en plus une condition de recherche selon la syntaxe suivante :

SELECT [ALL|DISTINCT] { <expression de valeurs>$^+$ | *}

FROM <nom de table> [<nom de variable>]

WHERE <condition de recherche>

Une condition de recherche définit un critère, qui appliqué à un tuple, est vrai, faux ou inconnu, selon le résultat de l'application d'opérateurs booléens (`ET, OU, NOT`) à des conditions élémentaires. L'expression booléenne de conditions élémentaires peut être parenthésée. La figure VII.5 donne les tables de vérité permettant de calculer la valeur de vérité d'une condition de recherche. En principe, les seuls tuples satisfaisant la condition de recherche sont sélectionnés par la requête.

AND	***VRAI***	***FAUX***	***INCONNU***
VRAI	VRAI	FAUX	INCONNU
FAUX	FAUX	FAUX	FAUX
INCONNU	INCONNU	FAUX	INCONNU

OR	***VRAI***	***FAUX***	***INCONNU***
VRAI	VRAI	VRAI	VRAI
FAUX	VRAI	FAUX	INCONNU
INCONNU	VRAI	INCONNU	INCONNU

NOT	***VRAI***	***FAUX***	***INCONNU***
	FAUX	VRAI	INCONNU

Figure VII.5 — Calcul de la valeur d'une condition de recherche

Une condition de recherche élémentaire est appelée **prédicat** en SQL. Un prédicat de restriction permet de comparer deux expressions de valeurs ; la première contenant des spécifications de colonnes est appelée **terme** ; la seconde contenant seulement des spécifications de constantes est appelée **constante**. Il existe une grande diversité de prédicats en SQL1. On trouve en effet :

1. un prédicat de comparaison permettant de comparer un terme à une constante à l'aide des opérateurs {=, ≠, <, >, <=, >=} ;

2. un prédicat d'intervalle `BETWEEN` permettant de tester si la valeur d'un terme est comprise entre la valeur de deux constantes ;

3. un prédicat de comparaison de texte `LIKE` permettant de tester si un terme de type chaîne de caractères contient une ou plusieurs sous-chaînes ;

4. un prédicat de test de nullité qui permet de tester si un terme a une valeur convenue `NULL`, signifiant que sa valeur est inconnue ;

5. un prédicat d'appartenance `IN` qui permet de tester si la valeur d'un terme appartient à une liste de constantes.

La figure VII.6 donne quelques exemples de sélections illustrant ces différents types de prédicats. La question (Q3) effectue la restriction de la relation `VINS` par la qualification Millésime = 1977 et Degré > 13. La question (Q4) délivre les crus et degrés des vins de millésime 1977 et de degré compris entre 11 et 13. La question (Q5) retrouve les crus, années de production (millésime diminué de 1900) et qualité des Beaujolais. Elle illustre le prédicat de recherche textuel (`LIKE`) ; le caractère % signifie dans le cas du `LIKE` une quelconque sous-

chaîne de caractères ; par exemple, BEAUJOLAIS NOUVEAUX et NOUVEAUX BEAUJOLAIS rendent le prédicat CRU LIKE "%BEAUJOLAIS%" vrai. La question (Q6) recherche tous les vins de degré nul. La question (Q7) délivre les crus des vins de qualité A, B ou C.

```
(Q3)   SELECT     *
       FROM       VINS
       WHERE      MILLESIME = 1977
       AND        DEGRE > 13

(Q4)   SELECT     CRU, DEGRE
       FROM       VINS
       WHERE      MILLESIME = 1977
       AND        DEGRE BETWEEN 11 AND 13

(Q5)   SELECT     DISTINCT CRU, MILLESIME - 1900, QUALITE
       FROM       VINS
       WHERE      CRU LIKE

(Q6)   SELECT     *
       FROM       VINS
       WHERE      CRU IS NULL

(Q7)   SELECT     CRU
       FROM       VINS
       WHERE      QUALITE IN (A,B,C)
```

Figure VII.6 — Exemples de sélections

3.3 Expression des jointures

Un cas particulier simple de jointure sans qualification est le produit cartésien. Celui-ci s'exprime très simplement en incluant plusieurs relations dans la clause FROM. Par exemple, le produit cartésien des relations VINS et ABUS se calcule à l'aide de la question (Q8) représentée figure VII.7.

```
(Q8)   SELECT     *
       FROM VINS, ABUS
```

Figure VII.7 — Exemple de produit cartésien

La jointure avec qualification peut s'exprimer de plusieurs manières. Une première expression naturelle est la restriction du produit cartésien par un prédicat permettant de comparer deux termes. Les prédicats de comparaison (=, ≠, ≤, ≥, <, >), d'intervalle (BETWEEN), d'appartenance (IN) et de comparaison textuelle (LIKE) sont utilisables. La combinaison des opérations de jointures, restrictions et projections peut être effectuée à l'intérieur d'un même bloc SELECT.

La figure VII.8 illustre différents cas de jointures. La question (Q9) effectue la jointure de VINS et ABUS sur l'attribut numéro de vins (NV) et permet de lister en préfixe de chaque tuple de ABUS le vin correspondant. La question (Q10) recherche les buveurs ayant un nom similaire à celui d'un cru. La question (Q11) retourne le nom des buveurs ayant bu du Chablis, sans double. Notez l'usage des variables B, V et A comme alias des relations BUVEURS, VINS et ABUS, afin d'éviter de répéter le nom complet des relations dans le critère.

```
(Q9)   SELECT     *
       FROM       VINS, ABUS
       WHERE      VINS.NV = ABUS.NV

(Q10)  SELECT     NB, NOM
       FROM       BUVEURS, VINS
       WHERE      NOM LIKE CRU

(Q11)  SELECT     DISTINCT NOM
       FROM       BUVEURS B, VINS V, ABUS A
       WHERE      B.NB = A. NB
       AND        A.NV = V.NV
       AND        V.CRU = "CHABLIS"
```

Figure VII.8 — Exemples de jointures

3.4 Sous-questions

SQL permet l'imbrication de sous-questions au niveau de la clause WHERE, si bien que l'on pet écrire des questions du type SELECT … FROM … WHERE … SELECT …. . En effet, le résultat d'une question peut être considéré comme une valeur simple ou comme un ensemble de valeurs avec doubles éventuels (multi-ensemble) ; dans ce dernier cas, chaque valeur de l'ensemble correspond à un tuple du résultat. Ainsi, il est possible de considérer une sous-question comme argument particulier des prédicats de comparaison (=,≠,<,>,≥,≤) et d'appartenance à une liste (IN). Toute sous-question peut elle-même invoquer des sous-questions, si bien qu'il est possible d'imbriquer des blocs SELECT à plusieurs niveaux. L'imbrication de blocs SELECT par le prédicat IN permet d'exprimer en particulier des jointures d'une manière plus procédurale.

Par exemple, la question (Q12) de la figure VII.9 effectue la jointure des tables VINS et ABUS sur numéro de vin et sélectionne les crus des vins résultants sans double. Elle est équivalente à la question (Q13). Il est aussi possible d'utiliser des variables définies dans un bloc interne au niveau d'un bloc externe. On parle alors de variable de corrélation. La question (Q14) illustre l'usage d'une variable de corrélation pour retrouver cette fois le cru du vins mais aussi la quantité bue à partir de la jointure de VINS et ABUS. Finalement, la question (Q15) recherche les noms des buveurs de Chablis. Elle est équivalente à la question (Q11), mais est écrite de manière plus procédurale avec trois blocs imbriqués.

```
(Q12)  SELECT     DISTINCT CRU
       FROM       VINS
       WHERE      NV IN
                  SELECT NV
                  FROM ABUS

(Q13)  SELECT     DISTINCT CRU
       FROM       VINS V, ABUS A
       WHERE      V.NV = A.NV

(Q14)  SELECT     DISTINCT CRU, A.QUANTITE
       FROM       VINS
       WHERE      NV IN
                  SELECT NV
                  FROM ABUS A

(Q15)  SELECT     DISTINCT NOM
       FROM       BUVEURS
       WHERE      NB IN
                  SELECT     NB
                  FROM       ABUS
                  WHERE      NV IN
                             SELECT     NV
                             FROM       VINS
                             WHERE      CRU= "CHABLIS"
```

Figure VII.9 — Exemples de blocs imbriqués

3.5 Questions quantifiées

Il est aussi possible de vouloir comparer une expression de valeurs à tous les résultats d'une sous-question ou seulement à l'une quelconque des valeurs générées. SQL propose pour cela l'usage de sous-questions quantifiées par « quel que soit » (ALL) ou « il existe » (ANY ou SOME). Ces quantificateurs permettent de tester si la valeur d'un terme satisfait un opérateur de comparaison avec tous (ALL) ou au moins un (ANY ou SOME) des résultats d'une sous-question. Un prédicat quantifié par ALL est vrai s'il est vérifié pour tous les éléments de l'ensemble. Un prédicat quantifié par ANY ou SOME est vrai s'il est vérifié par au moins un élément de l'ensemble.

Ainsi, la question (Q16) recherche les noms des buveurs n'ayant commis que des abus en quantité supérieure ou égale à toutes les quantités bues, alors que la question (Q17) recherche ceux ayant commis au moins un abus en quantité supérieure ou égale à toutes les quantités bues. La première n'aura probablement pas de réponse alors que la deuxième éditera le nom de la personne ayant effectué le plus gros abus.

SQL offre une autre possibilité de quantification pour tester si le résultat d'une sous question est vide ou non. Il s'agit du prédicat d'existence EXISTS. EXISTS <sous-question> est vrai si et seulement si le résultat de la sous-question est non vide. Ainsi, la question (Q18) recherche les buveurs ayant bu du Volnay alors que la question (Q19) recherche ceux n'ayant bu que du Volnay.

```
(Q16)  SELECT     NOM
       FROM       BUVEURS B, ABUS A
       WHERE      B.NB = A.NB
       AND        A.QUANTITE ≥ ALL
                  SELECT     QUANTITE
                  FROM       ABUS

(Q17)  SELECT     B.NOM
       FROM       BUVEURS B, ABUS A
       WHERE      B.NB = A.NB
       AND        A.QUANTITE ≥ ANY
                  SELECT     QUANTITE
                  FROM       ABUS

(Q18)  SELECT     B.NOM
       FROM       BUVEURS B
       WHERE      EXISTS
                  SELECT     *
                  FROM       ABUS A, VINS V
                  WHERE      A.NV = V.NV
                  AND        A.NB = B.NB
                  AND        CRU = "VOLNAY"

(Q19)  SELECT     B.NOM
       FROM       BUVEURS B
       WHERE      NOT EXISTS
                  SELECT     *
                  FROM       ABUS A, VINS V
                  WHERE      A.NV = V.NV
                  AND        A.NB = B.NB
                  AND        CRU ≠ "VOLNAY"
```

Figure VII.10 — Exemples de questions quantifiées

3.6 Expression des unions

SQL1 permet également d'exprimer l'opération d'union. Par exemple, l'obtention des crus de degré supérieur à 13 ou de millésime 1977 peut s'effectuer par la question (Q20) représentée figure VII.11.

```
(Q20)  SELECT     CRU
       FROM       VINS
       WHERE      DEGRE > 13
       UNION
       SELECT     CRU
       FROM       VINS
       WHERE      MILLESIME = 1977
```

Figure VII.11 — Exemple d'union

3.7 Fonctions de calculs et agrégats

Au-delà de l'algèbre relationnelle, des possibilités de calcul de fonctions existent. Les fonctions implantées sont :

- `COUNT` qui permet de compter le nombre de valeurs d'un ensemble ;
- `SUM` qui permet de sommer les valeurs d'un ensemble ;
- `AVG` qui permet de calculer la valeur moyenne d'un ensemble ;
- `MAX` qui permet de calculer la valeur moyenne d'un ensemble ;
- `MIN` qui permet de calculer la valeur moyenne d'un ensemble.

Les fonctions peuvent être utilisées dans la clause `SELECT`, par exemple pour calculer le degré moyen des « Chablis » comme dans la question (Q21) figure VII.12. Les fonctions sont aussi utilisables pour effectuer des calculs d'agrégats.

Rappelons qu'un agrégat est un partitionnement horizontal d'une table en sous-tables en fonction des valeurs d'un ou de plusieurs attributs de partitionnement, suivi de l'application d'une fonction de calculs à chaque attribut des sous-tables obtenues. Cette fonction est choisie parmi celles indiquées ci-dessus. En SQL, le partitionnement s'exprime par la clause :

GROUP BY <spécification de colonne>$^{+}$

Cette dernière permet de préciser les attributs de partitionnement, alors que les fonctions de calculs appliquées aux ensembles générés sont directement indiquées dans les expressions de valeurs suivant le `SELECT`.

Une restriction peut être appliquée avant calcul de l'agrégat au niveau de la clause `WHERE`, mais aussi après calcul de l'agrégat sur les résultats de ce dernier. Pour cela, une clause spéciale est ajoutée à la requête `SELECT` :

HAVING <EXPRESSION DE VALEURS>$^{+}$

La figure VII.12 propose quelques exemples de calculs d'agrégats. La question (Q21) calcule simplement la moyenne des degrés des Chablis. La question (Q22) mixte jointure et agrégat : elle affiche pour chaque cru la somme des quantités bues ainsi que la moyenne des degrés des vins du cru. Les questions (Q23) et (Q24) combinent agrégats et restrictions : (Q23) calcule la moyenne des degrés pour tous les crus dont le degré minimal est supérieur à 12 ; (Q24) recherche tous les numéros de vins bus en quantité supérieure à 10 par plus de 100 buveurs. La question (Q25) démontre une combinaison de blocs imbriqués et d'agrégats avec restriction portant sur des calculs ; elle retrouve les noms des buveurs ayant bu depuis 1983 une quantité d'alcool supérieur à 100.

```
(Q21)  SELECT     AVG(DEGRE)
       FROM       VINS
       WHERE      CRU = "CHABLIS"

(Q22)  SELECT     CRU, SUM(QUANTITE), AVG(DEGRE)
       FROM       VINS V, ABUS A
       WHERE      V.NV = A.NV
       GROUP BY   CRU

(Q23)  SELECT     CRU, AVG(DEGRE)
       FROM       VINS
       GROUP BY   CRU
       HAVING     MIN(DEGRE)>12

(Q24)  SELECT     NV
       FROM       ABUS
       WHERE      QUANTITE>10
       GROUP BY   NV
       HAVING     COUNT(NB)>100

(Q25)  SELECT     NOM
       FROM       BUVEURS
       WHERE      NB IN
                  SELECT     NB
                  FROM       ABUS A, VINS V
                  WHERE      A.NV = V.NV
                  AND        DATE > 01-01-83
                  GROUP BY   NB
                  HAVING     SUM(QUANTITE*DEGRE/100)>100
```

Figure VII.12 — Exemple de calculs d'agrégats

4. SQL1 : LES MISES A JOUR

SQL1 offre trois commandes de mise à jour : INSERT (insérer), DELETE (supprimer) et UPDATE (modifier). Toute mise à jour s'effectue par recherche des tuples à modifier et application des modifications. La recherche peut s'effectuer directement par une question intégrée dans la commande de mise à jour ou préalablement par utilisation d'un curseur depuis un programme traditionnel. Nous détaillons tout d'abord les commandes de mise à jour à partir de questions. La syntaxe des mises à jour est cependant précisée pour l'utilisation de curseurs, que nous verrons dans la section suivante.

4.1 Insertion de tuples

L'insertion de tuples dans une relation permet de créer de nouvelles lignes. Elle peut s'effectuer soit par fourniture directe au terminal d'un tuple à insérer (ou d'une partie de tuple, les valeurs inconnues étant positionnées à NULL), soit par construction à partir d'une question des tuples à insérer. La première variante est l'insertion directe, la deuxième l'insertion via question. La syntaxe de la commande d'insertion de SQL1 est :

INSERT INTO <NOM DE TABLE> [(<NOM DE COLONNE> $^{+}$)]

{**VALUES** (<CONSTANTE>$^{+}$) | <COMMANDE DE RECHERCHE>}

Dans le cas où la liste de colonnes n'est pas spécifiée, tous les attributs de la relation doivent être fournis dans l'ordre de déclaration. Si seulement certaines colonnes sont spécifiées, les autres sont insérées avec la valeur nulle.

Une insertion à partir d'une commande de recherche permet de composer une relation à partir des tuples d'une relation existante, par recherche dans la base.

En guise d'illustration, la commande d'insertion d'un Juliénas 1983 de degré inconnu sous le numéro de vins 112 est représentée figure VII.13 (requête (I1)). Nous donnons aussi la commande (I2) insérant dans une table BONSBUVEURS tous les buveurs ayant bu des vins de qualité A. La table BONSBUVEURS de schéma (NB, NOM, PRENOM) a du être créée auparavant.

```
(I1)  INSERT INTO VINS (NV, CRU, MILLESIME)
      VALUE (112, JULIENAS, 1983)


(I2)  INSERT      INTO BONSBUVEURS
      SELECT      NB, NOM, PRENOM
      FROM        BUVEURS B, ABUS A, VINS V
      WHERE       B.NB = A.NB
      AND         A.NV = V.NV
      AND         V.QUALITE = "A"
```

Figure VII.13 — Exemples de commandes d'insertion

4.2 Mise à jour de tuples

La mise à jour permet de changer des valeurs d'attributs de tuples existants. La mise à jour d'une relation peut s'effectuer soit par fourniture directe des valeurs à modifier, soit par

l'élaboration de ces valeurs à partir d'une expression. Les seuls tuples mis à jour sont ceux vérifiant une condition de recherche optionnelle fournie en argument d'une clause WHERE. Il est aussi possible de faire une mise à jour d'un seul tuple pointé par un curseur préalablement ouvert.

La syntaxe de la commande de mise à jour SQL1 est :

```
UPDATE <NOM DE TABLE>
SET {<NOM DE COLONNE> = {<EXPRESSION DE VALEUR> | NULL}}+
WHERE {<CONDITION DE RECHERCHE> | CURRENT OF <NOM DE CURSEUR>}
```

Par exemple, la mise à jour du degré du Juliénas 1983 par la valeur 13 s'effectuera par la requête (U1). L'accroissement des quantités bues de Volnay 1983 de 10% s'effectuera par la commande (U2).

```
(U1)  UPDATE      VINS
      SET         DEGRE = 13
      WHERE       CRU = "JULIENAS" AND MILLESIME = 1983


(U2)  UPDATE      ABUS
      SET         QUANTITE = QUANTITE * 1.1
      WHERE       NV IN
                  SELECT      NV
                  FROM        VINS
                  WHERE       CRUS = "JULIENAS"
                  AND         MILLESIME = "1983"
```

Figure VII.14 — Exemples de commandes de mise à jour

4.3 Suppression de tuples

L'opération de suppression permet d'enlever d'une relation des tuples existants. Les tuples sont spécifiés à l'aide d'une condition de recherche, à moins que l'on désire supprimer tous les tuples d'une relation. La syntaxe de la commande de suppression est :

```
DELETE FROM <NOM DE TABLE>
[WHERE {<CONDITION DE RECHERCHE> | CURRENT OF <NOM DE CURSEUR>}]
```

Par exemple, la suppression de tous les abus de vins de degré inconnu s'effectuera par la commande (D1), et la suppression de tous les vins bus par MARTIN par la commande (D2).

```
(D1)   DELETE
       FROM        ABUS
       WHERE       NV IN
                   SELECT      NV
                   FROM        VINS
                   WHERE       DEGRE IS NULL

(D2)  DELETE
      FROM         VINS
      WHERE        NV  IN
                   SELECT  NV
                   FROM BUVEURS, ABUS
                   WHERE  ABUS.NB = BUVEURS.NB
                   AND BUVEURS.NOM = "MARTIN"
```

Figure VII.15 — Exemples de commandes de suppression

5. SQL1 : LA PROGRAMMATION DE TRANSACTIONS

Une transaction est une séquence d'opérations, incluant des opérations bases de données, qui est atomique vis-à-vis des problèmes de concurrence et reprise après panne. Une transaction est généralement programmée dans un langage hôte. Dans cette section, nous présentons les commandes de gestion de transactions incluses dans SQL1 et l'intégration de SQL dans un langage de programmation.

5.1 Commandes de gestion de transaction

Une transaction est initiée par un appel de procédure base de données, si une transaction n'est pas déjà active. Elle est terminée soit avec succès par un ordre de validation des mises à jour `COMMIT WORK`, soit en échec suite à un problème par un ordre `ROLLBACK WORK`. Tout se passe comme si les mises à jour étaient préparées seulement en mémoire lors des commandes `UPDATE`, `INSERT` et `DELETE`, puis intégrées de manière atomique à la base de données par l'ordre `COMMIT WORK`, ou annulées par l'ordre `ROLLBACK WORK` (voir figure VII.16).

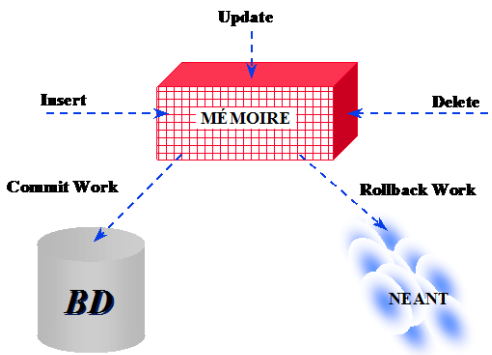

Figure VII.16 — Terminaison avec succès ou échec d'une transaction

5.2 Curseurs et procédures

La programmation de transaction dans un langage hôte nécessite le travail tuple à tuple. Pour cela, SQL propose la notion de curseur. Un **curseur** permet de repérer un tuple dans la base de données à l'aide d'une commande SELECT et d'un balayage séquentiel des tuples résultats. Il s'agit du moyen offert par SQL1 à la fois pour balayer séquentiellement les tuples résultats d'une question, et pour repérer un tuple dans la base. Un curseur est déclaré par la commande suivante :

DECLARE <NOM DE CURSEUR> **CURSOR**
FOR <COMMANDE DE RECHERCHE>
[**ORDER BY** {<SPECIFICATION DE COLONNE> [{**ASC** | **DESC**}]}+]

L'utilisation d'un curseur nécessite son ouverture par la commande :

OPEN <NOM DE CURSEUR>.

Cette commande provoque l'exécution logique de la question et le positionnement du curseur sur le premier tuple résultat. Il est ensuite possible de lire successivement chaque tuple résultat et de faire avancer le curseur par la commande :

FETCH <NOM DE CURSEUR> **INTO** <NOM DE VARIABLE>$^{+}$

La fin d'utilisation d'un curseur se signale au système par la commande :

CLOSE <NOM DE CURSEUR>.

SQL1 permet de composer des **modules**, eux-mêmes composés de **procédures** constituées d'une commande SQL, pouvant inclure des curseurs. Une procédure peut posséder des paramètres d'appel ou de retour. Elle peut être exécutée depuis un langage hôte ou plus généralement par une commande d'appel du type :

EXEC <NOM DE PROCEDURE> [(<PARAMETRES>$^{+}$)].

En guise d'illustration, nous avons représenté figure VII.17 un module définissant trois procédures pour ouvrir, lire et fermer les bons vins (de qualité A). Nous touchons là à l'intégration de SQL dans un langage de programmation tel PASCAL, C ou FORTRAN, que nous détaillerons dans le paragraphe suivant. Les curseurs ont un rôle essentiel pour réaliser une telle intégration ; il permettent d'effectuer des commandes de mises à jour tuple à tuple.

```
MODULE EXEMPLE
DECLARE BONVIN CURSOR
 FOR SELECT NV
 FROM VINS
 WHERE QUALITE = "A" ;
PROCEDURE OPENVIN ;
 OPEN BONVIN ;
PROCEDURE NEXTVIN(NV) NV INTEGER ;
 FETCH BONVIN INTO NV ;
PROCEDURE CLOSEVIN ;
 CLOSE BONVIN ;
```

Figure VII.17 — Exemple de module

5.3 Intégration de SQL et des langages de programmation

L'intégration de SQL à un langage procédural tel que C, COBOL, FORTRAN, PASCAL ou PL1 pose différents problèmes :

1. L'exploitation tuple à tuple des résultats des requêtes nécessite l'usage de curseurs. La commande `FETCH` est alors utilisée dans une boucle pour lire un à un les tuples résultats. Des commandes de mises à jour avec l'option de recherche `WHERE CURRENT OF` <nom de curseur> peuvent être employées pour les mises à jour.

2. Les variables du langage de programmation doivent être passées au SGBD par l'intermédiaire des requêtes SQL. Toute constante peut ainsi être remplacée par une variable de programmation qui doit être déclarée dans une section de déclaration de variables SQL. Un résultat peut être assigné à une variable de programme déclarée à SQL en liste résultat de l'ordre `FETCH`.

La figure VII.18 donne un exemple de programme PASCAL utilisant SQL. Ce programme ajoute un abus à la base pour les buveurs de Lyon ayant participé à une réception le 10-08-96. Ceux-ci sont déterminés par interrogation de l'utilisateur qui doit répondre O (oui) suite à l'affichage du nom du buveur si celui-ci a participé à la réception. Il doit alors préciser la quantité bue ce jour par le buveur. Le vin bu est déterminé par la constante `numvin`.

```
PROGRAM EXEMPLE
REPONSE STRING ;
RECORD TYPE BUVEUR
 NB INTEGER ;
 NOM STRING ;
 PRENOM STRING ;
END RECORD ;
EXEC SQL DECLARE SECTION END EXEC ;
 CONS NUMVIN = 100 ;
 VAR B BUVEUR ;
 VAR QUANTITE INTEGER ;
 VAR CODE SQLCODE ;
EXEC SQL END DECLARE SECTION END EXEC ;
EXEC SQL DECLARE CURSOR PARTICIPANT FOR
 SELECT NB, NOM, PRENOM
 FROM BUVEURS
 WHERE ADRESSE LIKE "%LYON%"
END EXEC ;
BEGIN
  EXEC SQL OPEN PARTICIPANT END EXEC ;
  WHILE CODE ≠ 100 DO
      BEGIN
       EXEC SQL FETCH PARTICIPANT INTO B END EXEC ;
       PRINT "NOM :",  B.NOM, "PRENOM", B.PRENOM ;
       PRINT "CE BUVEUR A-T-IL PARTICIPE ?"
       READ REPONSE ;
       IF REPONSE = "O" THEN
          BEGIN
             PRINT "QUANTITE ?" ;
             READ QUANTITE ;
             EXEC SQL INSERT INTO ABUS
             (B.NB, NUMVIN, "10-08-96", QUANTITE) END EXEC ;
          END ;
      END ;
  EXEC SQL CLOSE PARTICIPANT END EXEC ;
END.
```

Figure VII.18 — Exemple de programme intégrant des commandes SQL

En conclusion, on notera la lourdeur des programmes intégrant SQL à un langage de programmation. Ceci résulte tout d'abord de la différence de point de vue entre le modèle relationnel qui supporte des requêtes ensemblistes et les langages classiques qui sont avant tout séquentiels. Ces derniers permettent seulement d'écrire des programmes traitant un tuple (record) à la fois. D'où la nécessité de boucles complexes. La complexité résulte aussi de l'existence de deux systèmes de définition de types différents, dans notre exemple celui de PASCAL (RECORD TYPE) et celui de SQL (CREATE TABLE). D'où la nécessité de

doubles déclarations. Tout ces problèmes sont souvent mieux résolus au niveau des langages de 4e génération, qui malheureusement ne sont pas standardisés.

6. SQL2 : LE STANDARD DE 1992

SQL2 [ISO92] est une extension de SQL1 devenu le standard d'accès aux bases de données relationnelles depuis 1992. Compte tenu de la complexité de SQL2 qui se veut très complet, trois niveaux sont distingués :

1. **SQL2 entrée** est avant tout une correction de la norme SQL1 de 1989 et un complément avec les commandes manquantes indispensables.

2. **SQL2 intermédiaire** apporte les compléments relationnels indispensables au support complet du modèle et de l'algèbre ainsi que des types de données plus variés.

3. **SQL2 complet** apporte des types de données encore plus variés et quelques compléments non indispensables.

Dans la suite, nous détaillons successivement ces trois niveaux supplémentaires de SQL. Ils sont en principe des extensions compatibles de SQL1.

6.1 Le niveau entrée

SQL2 entrée propose une standardisation des codes réponses en ajoutant une variable retour des commandes appelées `SQLSTATE`. En effet, avec SQL1 un seul code réponse est retourné dans une variable de type `SQLCODE`. Trois valeurs sont spécifiées : 0 pour exécution correcte, +100 pour absence de données et une valeur négative - n pour toute erreur, la valeur étant spécifiée par le concepteur du SGBD. Cela rend les programmes non portables. Afin d'augmenter la portabilité, un code retour `SQLSTATE` est ajouté (`SQLCODE` est gardé afin d'assurer la compatibilité avec SQL1). Le code `SQLSTATE` est composé de deux caractères spécifiant la classe d'erreur (par exemple 22 pour exception erreurs sur les données) et de trois caractères précisant la sous-classe (par exemple, 021 pour caractère invalide). Les codes classes et sous-classes sont partiellement standardisés.

Avec SQL2, il est possible de renommer des colonnes résultats. Cette fonctionnalité est très utile, par exemple lors du calcul d'un agrégat. Avec SQL1, la colonne de la table résultat prend souvent un nom dépendant du système. La clause `AS` de SQL2 permet de résoudre ce problème. Par exemple, une question calculant la moyenne des degrés par crus pourra maintenant spécifier un nom de colonne résultat `MOYENNE` comme suit :

```
SELECT      CRU, AVG(DEGRE) AS MOYENNE
FROM        VINS
GROUP BY    CRU
```

Un autre ajout proposé à SQL1 au niveau de SQL2 entrée est la possibilité d'utiliser les mots clés de SQL comme des noms de table, d'attributs, etc. Pour ce faire, il suffit d'inclure ces mots clés entre double cotes. Par exemple, on pourra créer une table de nom `SELECT` par la commande :

```
CREATE TABLE "SELECT" (QUESTION CHAR(100)).
```

En résumé, les extensions de SQL1 proposées au niveau de SQL2 entrée sont donc des corrections et clarifications nécessaires au langage. SQL2 entrée est donc le nouveau standard minimal que devrait à terme fournir les constructeurs de SGBD. Il permettra une meilleure portabilité des programmes. L'interface avec C est d'ailleurs aussi précisée au niveau de SQL2 entrée.

6.2 Le niveau intermédiaire

6.2.1 Les extensions des types de données

SQL2 intermédiaire offre un meilleur support du temps. Trois nouveaux types de données DATE, TIME et TIMESTAMP sont introduits. Le type DATE est spécifié pour un attribut contenant une année, un mois et un jour (par exemple 1992/08/21). Le type TIME correspond à un groupe heures, minutes et secondes (par exemple, 09 :17 :23). Le type TIMESTAMP correspond à une concaténation d'une date et d'une heure (DATE concaténé à TIME). SQL2 offre également un type de données permettant de spécifier un intervalle de temps (INTERVAL) avec une précision en mois-année ou en seconde-minute-heure-jour.

Les opérations arithmétiques d'addition et soustraction sur les dates-heures et intervalles sont supportées avec SQL2. Il est aussi possible de multiplier ou diviser des intervalles par des valeurs numériques. Par exemple, en supposant l'attribut DATE de la table ABUS défini comme une date, il est possible de retrouver tous les abus commis dans un intervalle de temps de N mois (N est de type intervalle) par rapport à la date du 21-08-96 par la question suivante :

```
SELECT      *
FROM        ABUS
WHERE       DATE - 21/08/1996 < N
```

SQL2 admet également la création de domaines par l'utilisateur. Cela permet en particulier l'introduction de types de données par l'utilisateur au sein du modèle avec un meilleur contrôle de l'intégrité de domaine ; ces types restent en SQL2 purement des sous-ensembles des types prédéfinis ; aucune opération spécifique ne peut leur être associée. Par exemple, la création d'un domaine MONNAIE s'effectuera par la commande :

```
CREATE DOMAINE MONNAIE IS DECIMAL(5,2)
DEFAULT (-1)
CHECK (VALUE = -1 OR VALUE > 0)
NOT NULL
```

Ce domaine pourra être utilisé lors d'une création de table. Il s'agit essentiellement d'une macro-définition permettant d'inclure les contrôles d'intégrité, par exemple dans une table FACTURE :

```
CREATE TABLE FACTURE (NOM CHAR(5), MONTANT MONNAIE)
```

D'autres extensions sont proposées pour un meilleur support de types de données variés. On citera en particulier :

- le support de multiples alphabets et ensembles de caractères (CHARACTER SET) ;
- le support de différents ordres des lettres (COLLATE) ;

- la possibilité de concaténer des chaînes de caractères (||) et d' extraire des sous-chaînes (`SUBSTRING`) ;

- les facilités de conversion de types de données (`CAST`) ;

- les chaînes de caractères de longueur variables (`CHAR VARYING`) et la possibilité d'extraire la longueur d'une chaîne (fonction `LENGTH`).

6.2.2 Un meilleur support de l'intégrité

L'intégrité référentielle est supportée en SQL1, mais toute tentative de violation entraîne le rejet de la mise à jour. Au contraire, SQL2 intermédiaire permet de spécifier certaines actions correctives en cas de violation d'intégrité lors d'une suppression d'un tuple référencé, selon les options :

- cascader les suppressions `(ON DELETE CASCADE)`,

- rendre nul l'attribut référençant (`ON DELETE SET NULL`).

L'option doit être précisée lors de la définition de la contrainte référentielle dans la table référençante.

Plus généralement, les contraintes d'intégrité peuvent être nommées lors de leurs définitions. La validation des contraintes d'intégrité peut être immédiate à la fin de chaque opération ou différée en fin de transaction. Ceci est indiqué par une clause :

`SET CONSTRAINTS` <nom de contrainte>$^{+}$ **{`DEFERRED` | `IMMEDIATE`}**

Le nom de contrainte `ALL` indique que toutes les contraintes sont concernées.

Différents niveaux de contrôle de transactions sont aussi possibles. Une clause

`SET TRANSACTION` <mode>

est introduite, permettant de préciser le niveau d'isolation désiré (0 = aucune pour lecture non protégée, 1 = écriture protégée, lecture non protégée, 2 = écriture et lecture protégées, 3 = écriture et lecture exclusives), le mode d'accès (lecture seule `READ ONLY`, ou lecture-écriture `READ-WRITE`), et le nombre de diagnostics possibles.

6.2.3 Une intégration étendue de l'algèbre relationnelle

L'algèbre relationnelle offre les opérations ensemblistes d'union, intersection et différence de relations. SQL1 ne supporte que l'union. SQL2 généralise les expressions de `SELECT` en introduisant intersection (`INTERSECT`) et différence (`EXCEPT`). Des expressions parenthésées de `SELECT` sont même possibles. De plus, l'union est généralisée à l'union externe, avec complément des schémas des relations au même schéma par ajout de valeurs nulles aux tuples, cela préalablement à l'union proprement dite.

Afin d'illustrer ces possibilités, imaginons une table `NONBUVEURS (NB, NOM, PRENOM, ADRESSE)`, complémentaire de la table `BUVEURS(NB, NOM, PRENOM, ADRESSE,`

TYPE). La requête suivante construit une table contenant buveurs et non buveurs avec un type nul, à l'exception des gros buveurs :

```
SELECT      *
FROM        NONBUVEURS
OUTER UNION
(SELECT     *
 FROM       BUVEURS
EXCEPT
 SELECT     *
 FROM       BUVEURS
 WHERE      TYPE = "GROS")
```

Les jointures externes sont utiles pour retenir lors d'une jointure les tuples d'une table n'ayant pas de correspondant dans l'autre table, avec des valeurs nulles associées. On distingue ainsi des jointures externes droite, gauche et complète selon que l'on retient les tuples sans correspondant des deux tables ou seulement d'une. Rappelons aussi qu'une jointure est dite naturelle si elle porte sur des attributs de même nom. SQL2 offre la possibilité de spécifier de telles jointures au niveau de la clause FROM, selon la syntaxe suivante :

```
FROM  <NOM DE TABLE> [NATURAL] [{LEFT | RIGHT}]
    JOIN <NOM DE TABLE>
    [ON (<SPECIFICATION DE COLONNE>+=<SPECIFICATION DE COLONNE>+)]
```

On peut par exemple retrouver la somme des quantités bues par chaque buveur, y compris les buveurs n'ayant pas bu par la requête suivante :

```
SELECT      NB, NOM, SUM(QUANTITE)
FROM        BUVEURS NATURAL LEFT JOIN ABUS
GROUP BY    NB
```

6.2.4 La possibilité de modifier les schémas

SQL2 permet de modifier un schéma de table à l'aide de la commande :

ALTER TABLE <nom de table> <altération>

Différents types d'altérations sont possibles :

- ajout d'une colonne (ADD COLUMN)
- modification de la définition d'une colonne (ALTER COLUMN)
- suppression d'une colonne (DROP COLUMN)
- ajout d'une contrainte (ADD CONSTRAINT)
- suppression d'une contrainte (DROP CONSTRAINT).

Par exemple, l'ajout de l'attribut REGION à la table VINS pourra s'effectuer par la commande :

```
ALTER TABLE VINS ADD COLUMN REGION CHAR VARYING.
```

6.2.5 L'exécution immédiate des commandes SQL

Avec SQL1, toute commande SQL est compilée puis exécutée depuis un programme d'application. Les processus de compilation et d'exécution sont séparés et non contrôlés. Au contraire, SQL2 supporte une option dynamique incluant les possibilités d'exécution différée ou immédiate, cette dernière étant utile par exemple en interactif. Des commandes `PREPARE` <commande SQL> et `EXECUTE [IMMEDIATE]` <commande SQL> sont introduites afin de permettre l'exécution immédiate (`EXECUTE IMMEDIATE`) ou la compilation puis l'exécution de multiples fois à partir du résultat de la compilation (`PREPARE` suivi de plusieurs `EXECUTE`). Tous ces aspects de mode de fonctionnement sont ainsi intégrés au langage.

6.3 Le niveau complet

SQL2 offre un niveau complet (FULL SQL2) qui comporte en plus les fonctionnalités suivantes :

- type de données chaînes de bits (`BIT`) pour supporter des objets complexes tels des images ;
- extension du support des dates et temps, notamment avec la possibilité de définir des zones d'heures, des intervalles de temps d'échelles variées, etc. ;
- expressions de requêtes `SELECT` étendues avec correspondances de colonnes possibles ; par exemple, lors d'une union deux colonnes de nom différents peuvent être transformées en une seule colonne ;
- support de tables temporaires privées détruites en fin de transaction ;
- possibilité de `SELECT` en argument d'un `FROM` afin de construire une table temporaire à l'intérieur d'une requête SQL ;
- maintenance de vues concrètes facilitant l'interrogation de vues peu modifiées, notamment de vues avec agrégats ;
- support de contraintes d'intégrité multitables à l'aide de sous-questions intégrables dans la clause `CHECK` de déclaration de contraintes.

Ces multiples extensions et quelques autres, par exemple pour généraliser la maintenance automatique des contraintes référentielles lors des mises à jour, font de SQL2 complet un langage plutôt complexe pour manipuler des bases de données relationnelles. La spécification de SQL2 comporte 522 pages de syntaxe.

7. CONCLUSION

Le standard SQL2 est adopté depuis 1992 par les organismes de standardisation internationaux (ISO). Une large extension appelée SQL3 est en cours de finition et devrait être adoptée en 1999. SQL3 intègre les fonctionnalités nouvelles non purement relationnelles. En particulier, sont traitées au niveau de SQL3 les fonctionnalités orientées objet, le support de

questions récursives et le support de règles déclenchées par des événements bases de données (en anglais, *triggers*). Les fonctionnalités orientées objet sont offertes sous forme de constructeurs de types abstraits de données permettant la définition d'objets complexes par l'utilisateur. Les questions récursives permettent d'intégrer l'opérateur de fermeture transitive à SQL. Une syntaxe est proposée pour définir des *triggers*. Ces fonctionnalités seront étudiées dans les chapitres qui suivent. Plus spécifiquement, SQL3 se veut le langage des SGBD objet-relationnel ; à ce titre, nous l'étudierons donc dans la troisième partie de cet ouvrage.

En résumé, SQL est un langage standardisé de programmation de bases de données. Bien qu'à l'origine issu du modèle relationnel, SQL est aujourd'hui un langage qui couvre un spectre plus large. Les standards SQL1, SQL2 et SQL3 traitent de l'ensemble des aspects des bases de données, dont la plupart seront étudiés dans la suite. SQL sert aussi de langage d'échange de données entre SGBD. Cela explique donc son très grand succès, qui se traduit par le fait que tous les SGBD offrent une variante de SQL. La normalisation du langage assure une bonne portabilité des programmes bases de données. Elle a été possible grâce au soutient des grands constructeurs, tels IBM, DEC, ORACLE et SYBASE.

Certains critiquent le manque de rigueur et de cohérence de SQL [Date84]. Il est vrai que l'ensemble peut paraître parfois hétéroclite. La syntaxe est souvent lourde. Quoi qu'il en soit, SQL est et reste la référence. Il est le langage de nombreux systèmes commercialisés tels Oracle, DB2, Sybase, Informix et SQL Server.

8. BIBLIOGRAPHIE

[Boyce75] Boyce R., Chamberlin D.D., King W.F., Hammer M., « Specifying Queries as Relational Expressions », *Comm. de l'ACM*, Vol. 18, N° 11, novembre 1975.

Une présentation du language SQUARE qui permet d'exprimer en anglais simplifié des expressions de l'algèbre relationnelle. SQUARE est à l'origine du langage SQL.

[Chamberlin76] Chamberlin D.D., Astrahan M.M., Eswaran K.P., Griffiths P., Lorie R.A, et.al., « SEQUEL 2 : A Unified Approach to Data Definition, Manipulation and Control », *IBM Journal of Research and Development*, Vol. 20, N° 6, novembre 1976.

Cet article décrit la deuxième version du langage SEQUEL, le langage du fameux système R, le premier SGBD relationnel prototypé à IBM San-José de 1974 à 1980. Pendant cette période, l'université de Berkeley réalisait INGRES. SEQUEL 2 étend SEQUEL 1 avec des constructions dérivées de QUEL — le langage de INGRES — et permet de paraphraser en anglais les expressions de l'algèbre. Il introduit aussi les commandes de description et contrôle de données et constitue en cela un langage unifié. C'est en tout cas un langage assez proche de SQL1.

[Date84] Date C.J., « A Critique of the SQL Database Language », *ACM SIGMOD Record*, Vol. 14, N° 3, novembre 1984.

Chris Date, qui vient de quitter IBM en cette fin de 1984, critique la cohérence du langage SQL et démontre quelques insuffisances.

[Date89] Date C.J., *A Guide to the SQL Standard*, 2e édition, Addison-Wesley, Reading, Mass., 1989.

Une présentation didactique du standard SQL, avec beaucoup d'exemples.

[IBM82] IBM Corporation, « SQL/Data System Terminal Users Guide », *IBM Form Number SH24-5017-1, 1982.*

La présentation du langage du système SQL/DS d'IBM, disponible sur DOS/VSE. Il s'agit de la première implémentation commercialisée du langage SQL.

[IBM87] IBM Corporation, « Systems Application Architecture (SAA) : Common Programming Interface, Database Reference », *IBM Form Number SC26-4348-0, 1987*.

La définition du standard de convergence des systèmes IBM supportant SQL, dans le cadre de l'architecture unifiée SAA. En principe, tous les systèmes IBM réalisés dans des centres différents (DB2, SQL/DS, SQL AS/400, SQL OS2) ont convergé ou convergeront vers un même langage défini dans ce document, très proche de la norme SQL1.

[ISO89] International Organization for Standardization, « Information Processing Systems - Database Language SQL with Integrity Enhancement », *International Standard ISO/IEC JTC1 9075* : 1989(E), 2e édition, avril 1989.

Ce document de 120 pages présente la norme SQL1 : concepts de base, éléments communs, langage de définition de schémas, définition de modules de requêtes, langage de manipulation. Toutes la grammaire de SQL1 est décrite en BNF. Ce document résulte de la fusion du standard de 1986 et des extensions pour l'intégrité de 1989.

[ISO92] International Organization for Standardization, « Database Language SQL », *International Standard ISO/IEC JTC1/SC21* Doc. 9075 N5739, 1992.

Ce document de 522 pages présente la norme SQL2 : définitions, notations et conventions, concepts, expressions de scalaires, expressions de questions, prédicats, langage de définition et manipulation de schémas, langage de manipulation de données, langage de contrôle, SQL dynamique, SQL intégré à un langage, codes réponses, etc. Les niveaux entrée et intermédiaire sont clairement distingués. L'ensemble constitue un document très complet qui doit être accepté par l'ISO en 1992. L'approbation de cette nouvelle norme demande un vote positif de 75% des corps représentatifs de l'ISO dans les différents pays habilités.

[Melton96] Melton J., « An SQL3 Snapshot », *Proc. Int. Conf. On Data Engineering*, IEEE Ed., pp. 666-672, 1996.

Ce bref article donne un aperçu du langage SQL3 tel qu'il était en 1996. SQL3 est la nouvelle version de SQL pour les systèmes objet-relationnel. Nous étudierons la version actuelle de SQL3 plus loin dans cet ouvrage. J. Melton était à cette époque le responsable de la normalisation de SQL.

[Shaw90] Shaw Ph., « Database Language Standards : Past, Present, and Future », *Lecture Notes in Computer Science*, N° 466, Database Systems of the 90s, A. Blaser Ed., Springer Verlag, novembre 1990.

Cet article fait le point sur les efforts de standardisation de SQL. Il résume les développements passés en matière de standardisation des bases de données et introduit les propositions SQL2 et SQL3. Les niveaux de SQL2 sont particulièrement développés et illustrés par des exemples. Phil Shaw était à cette époque le responsable de la normalisation de SQL.

[X/Open92] X/Open Group, « Structured Query Language (SQL) » Common Application Environment *CAE Specification C201*, septembre 1992.

Ce document est une présentation du langage SQL2 élaborée par l'X/OPEN Group.

[Zook77] Zook W. et. al., « INGRES Reference Manual », Dept. of EECS, University of California, Berkeley, CA, 1977.

Ce document décrit les interfaces externes de la première version d'INGRES et plus particulièrement le langage QUEL.

[Zloof77] Zloof M., « Query-by-Example : A Data Base Language », *IBM Systems Journal*, Vol. 16, N° 4, 1977, pp. 324-343.

Cet article présente QBE, le langage par grille proposé par Zloof, alors chercheur à IBM. Ce langage bidimensionnel est aujourd'hui opérationnel en sur-couche de DB2 et aussi comme interface externe du système Paradox de Borland. Zloof discute aussi des extensions bureautiques possibles, par exemple pour gérer le courrier (OBE).