SQL

1.1. Introduction

Le langage SQL (**S**tructured **Q**uery **L**anguage) est un standard (norme ISO) des bases de données relationnelles, qui utilise les concepts de :

- table (relation, à part qu'une table peut contenir des doublons)
- ligne (tuple)
- colonne (attribut)
- type de donnée (domaine)

Ce langage comprend différentes parties :

- **Langage de définition des données (DDL ou LDD) :** définition de schémas de relations, destruction de relation, création d'index, spécification des **contraintes d'intégrité**, etc.
- **Langage de manipulation de données (DML ou LMD) :** pour poser des questions, ajouter, modifier ou supprimer des tuples.

1.2. Langage de définition de données (LDD)

1.2.1. TYPES DE DONNEES (SQL-92)

Les principaux types des valeurs des attributs sont les suivants :

- chaînes de caractères : **CHAR**(n), **VARCHAR**(n)
- nombres : entiers (**INT**, **SMALLINT**), nombres à virgule de taille fixée (**NUMERIC**(3, 1) pour des nombres de 3 chiffres, dont 1 après la virgule), réels (**REAL**, **FLOAT**(n))
- date : **DATE** année (à 4 chiffres), mois, jour du mois
- horaire : **TIME** heures, minutes, secondes

1.2.2. DEFINITION DU SCHEMA D'UNE TABLE

CREATE TABLE nom_table(A1 D1, A2 D2, …, An Dn,
contrainte d'intégrité 1, …,
contrainte d'intégrité k)

Où A*i* est le i^{ème} attribut, D*i* son type. On peut indiquer les contraintes avec l'attribut, ou après. Les principales contraintes d'intégrité sont : **PRIMARY KEY** et **CHECK**. PRIMARY KEY(A1, …, An) indique que les attributs A1, …, An forment une clef primaire. CHECK(C) indique que la condition C doit être satisfaite par chaque ligne de la table. **NOT NULL**, interdit à un attribut de ne pas avoir de valeur nulle.

Exemples :

```
CREATE TABLE Agence (Nom_agence CHAR(15) NOT NULL, Ville_agence CHAR(30), Capital INTEGER, PRIMARY KEY (Nom_agence), CHECK (Capital >= 0))
CREATE TABLE Depot (Nom_client CHAR(20) NOT NULL, Num_compte CHAR(10) NOT NULL, PRIMARY KEY (Nom_client, Num_compte))
CREATE TABLE Client (Nom_client CHAR(20) NOT NULL, Rue_client CHAR(30), Ville_client CHAR(30), PRIMARY KEY (Nom_client))
CREATE TABLE Compte (Num_compte CHAR(10) NOT NULL, Nom_agence CHAR(15), Solde INTEGER, PRIMARY KEY (Num_compte), CHECK (Solde >= 0))
```

Il est implicite qu'une clef primaire est non nulle et unique. Même chose pour la condition donnée dans CHECK.

- ***La condition du CHECK peut être une requête***, par exemple pour la relation *Compte* :

CHECK(Nom_agence IN (SELECT Nom_agence FROM Agence))

Dans ce cas la condition doit être vérifiée non seulement lors d'ajouts ou de modifications dans la table *Compte*, mais également lors de modifications de la table *Agence* (suppression).

C'est là un exemple de **contrainte d'intégrité référentielle** (clef étrangère : cf. § 4.2).

- ***Il est possible de donner une valeur par défaut à un attribut***, par exemple pour l'attribut Anniversaire d'une table de clients : Anniversaire DATE **DEFAULT** DATE '0000-00-00'

1.2.3. INDEX ET VUES

Une vue est une table calculée par une requête (à partir des tables de la base). Seul le calcul (la définition de la vue) est stocké dans la base. La syntaxe est la suivante : **CREATE VIEW** nom_de_la_vue **AS** requête_SQL
On utilise des vues pour simplifier l'écriture de requêtes, mais également en administration, pour gérer l'accès aux informations.

Un index est une structure de données associée à un attribut d'une table : l'index permet de **trouver rapidement un enregistrement à partir de la valeur de cet attribut**. Dans l'index, les valeurs sont ordonnées. Par exemple, plutôt que parcourir tous les comptes en testant si l'agence est à Blois, on veut *accéder directement* aux comptes de Blois : c'est possible en indexant la table *Agence* sur l'attribut *Ville_agence*.
CREATE INDEX Ville **ON** Agence(Nom_ville)

1.2.4. MODIFICATION DES OBJETS DE LA BASE

La commande générale est **ALTER.** Et on indique le type d'objet à modifier (table, vue, index…)

- Pour ajouter l'attribut *A* de type *D* au schéma d'une table *T*, on écrira : **ALTER TABLE** *T* **ADD** *A D*

Lorsqu'on ajoute un attribut à une table existante, chaque ligne reçoit la valeur *Null* pour ce nouvel attribut.

- Pour supprimer l'attribut *A* du schéma d'une table *T*, on écrira : **ALTER TABLE** *T* **DROP** *A*

1.2.5. SUPPRESSION DES OBJETS DE LA BASE

La commande générale est **DROP.** Et on indique le type d'objet à modifier (table, vue, index…)

- Pour supprimer une table *T*, on écrira : **DROP TABLE** *T*

1.3. Les modifications de DONNÉES

Une fois le schéma de la base spécifié, on peut remplir les tables…

1.3.1. AJOUT DE DONNEES DANS UNE TABLE

INSERT INTO T VALUES (a1, a2, …, an) ajoute dans la table T la ligne de valeurs (a1, a2, …, an), les ai étant, dans l'ordre, les valeurs des attributs de T.

Exemple : *"Ajouter un compte de n°A-973 et de solde 2000 à Perryridge"*
INSERT INTO Compte VALUES ("Perryridge", "A-973", 2000)

1.3.2. SUPPRESSION DE DONNEES DANS UNE TABLE

DELETE FROM T WHERE C supprime de la table T les lignes qui satisfont la condition C (interrogations pour la description des conditions C possibles).

Exemple : *"Supprimer tous les prêts dont le montant se situe entre 10000 et 13000"*
DELETE FROM Pret WHERE Montant BETWEEN 10000 AND 13000

1.3.3. MISES A JOUR DES DONNEES D'UNE TABLE

UPDATE T SET A = V WHERE C affecte à l'attribut A de la table T la valeur V pour les lignes vérifiant C.

1.4. Contraintes d'intégrité

Les contraintes sont automatiquement vérifiées à chaque modification des données de la base. Si l'une des contraintes n'est pas satisfaite, la modification n'est pas effectuée :

1.4.1. LES DOMAINES

Exemples

CREATE DOMAIN Salaire_horaire NUMERIC(5,2) CONSTRAINT Test_valeur_salaire CHECK(VALUE >= 36)
CREATE DOMAIN Type_compte CHAR(10) CONSTRAINT Test_type_compte CHECK(VALUE IN ("Chèque", "Epargne"))

1.4.2. LES CLEFS (PRIMARY KEY, VUE PRECEDEMMENT)

1.4.3. L'INTEGRITE REFERENTIELLE

Lorsqu'une valeur d'une ligne d'une table T1 nécessite la présence d'une valeur d'une ligne dans une table T2. Par **exemple,** on ne souhaite pas qu'il y ait dans la table *Compte* un compte dans l'agence "Blois-Vienne" s'il n'existe pas dans la table *Agence* d'agence appelée "Blois-Vienne". On dit que l'attribut *Nom_agence* dans la table *Compte* est une "clef externe". On l'exprime de la façon suivante en SQL :
CREATE TABLE Compte

(Num_compte CHAR(10) NOT NULL,
Nom_agence CHAR(15),
Solde INTEGER,
PRIMARY KEY (Num_compte),
FOREIGN KEY (Nom_agence) **REFERENCES** Agence,
CHECK (Solde >= 0))

La clef externe doit référencer la clef primaire de l'autre table.

On peut aussi écrire directement :

Nom_agence CHAR(15) **REFERENCES** Agence

1.4.4. LES DEPENDANCES FONCTIONNELLES

La notion de dépendance fonctionnelle généralise celle de clef : un attribut A1 dépend fonctionnellement d'un attribut A2 si la valeur de A2 permet de connaître à coup sûr la valeur de A1.

Par **exemple**, soit le schéma de relation : INFO_PRET(Nom_agence, Num_prêt, Nom_client, Montant)

L'ensemble des dépendances fonctionnelles sera :

Num_prêt >> Montant et Num_prêt >> Nom_agence

Mais on ne souhaitera pas forcément que Num_prêt >> Nom_client, si on accepte des comptes joints.

1.5. Le langage d'INTERROGATION de bases de données SQL

SELECT < liste des colonnes de la table résultat >
FROM < liste des tables impliquées dans l'interrogation >
WHERE < condition de sélection des lignes >
Exemple SELECT Nom FROM cru WHERE Couleur = 'rouge'

1.5.1. SELECT

- On peut éliminer les doublons dans la réponse : SELECT **DISTINCT** Region FROM vins
- On peut indiquer qu'on veut tous les attributs : SELECT * FROM R
- On peut faire des calculs sur les valeurs des attributs avec +, -, *, / : SELECT solde *** 10** FROM Compte

1.5.2. WHERE

- Opérateurs booléens : AND, OR, NOT
- Opérateurs de comparaison : classiques, mais aussi BETWEEN, IN, LIKE, et la valeur NULL

1.5.3. FROM

- Représente un produit cartésien des tables impliquées.

1.6. De l'algèbre relationnelle à l'interrogation en SQL

Les correspondances sont les suivantes :

- la projection $\pi_{A1,A2,\ldots,An}(R)$: SELECT A1, A2, …, An FROM R
- la sélection $\sigma_c(R)$: SELECT * FROM R WHERE C
- le produit cartésien $R \times S$: SELECT R.*, S.* FROM R, S
- la jointure $R \ {}_{(Ai1\ \theta 1\ Bj1)\wedge(Ai2\ \theta 2\ Bj2)\wedge\ldots\wedge(Ain\ \theta n\ Bjn)}\triangleright\!\triangleleft\ S$:
  SELECT R.*, S.* FROM R, S
  WHERE R.Ai1 θ1 S.Bj1 and R.Ai2 θ2 S.Bj2 and…and R.Ain θn S.Bjn
- la jointure naturelle $R \triangleright\!\triangleleft S$: SELECT R.*, Bp, Bq, …, Bt FROM R, S
  WHERE R.Ai=S.Ai and R.Aj=S.Aj and…and R.Ak=S.Ak
  où Bp, Bq, …, Bt sont les attributs de S qui ne sont pas communs avec R,
  et Ai, Aj, …, Ak sont les attributs de jointure de R et S.
- l'union $R \cup S$: SELECT * FROM R UNION SELECT * FROM S
- l'intersection $R \cap S$: SELECT * FROM R INTERSECT SELECT * FROM S
- la différence R – S : SELECT * FROM R MINUS SELECT * FROM S (EXCEPT en SQL-2)

1.7. Possibilités supplémentaires pour l'interrogation

1.7.1. RENOMMAGE DE TABLE OU DE COLONNE

- pour donner un nom à une colonne du résultat :

  SELECT Région AS produit_en FROM vin

- pour un alias de table local :

  SELECT DISTINCT nom_client, T.num_compte
  FROM Depot AS R, Clients AS T, Compte AS S
  WHERE R.num_compte=S.num_compte and R.nom_client=T.nom_client

1.7.2. TRI DU RESULTAT

SELECT * FROM Compte ORDER BY solde DESC, num_compte ASC

1.8. Fonctions de calcul d'agrégat

Les fonctions d'agrégation agissent sur un ensemble de valeurs et retournent une valeur. Il y en a 5 :

- Moyenne : AVG
- Minimum : MIN
- Maximum : MAX
- Somme : SUM
- Comptage : COUNT

Exemple : *"Quel est le solde moyen des comptes de l'agence Perryridge ?"*

SELECT **AVG**(solde) FROM Compte WHERE nom_agence = "Perryridge"

Exemple : *"Combien y a t-il de clients ?"*

SELECT **COUNT**(*) FROM Client

On veut parfois appliquer une agrégation à des tuples regroupés selon un attribut : GROUP BY

Exemple : *"Quel est le solde moyen de chaque agence ?"*

SELECT nom_agence, AVG(solde) FROM Compte
GROUP BY nom_agence

Dans de tels cas de figure, il est parfois nécessaire de sélectionner selon une condition qui s'applique aux groupes et non pas aux lignes "individuelles" : HAVING

Exemple : *"Quelles sont les agences dont le solde moyen est supérieur à 8000?"*

SELECT nom_agence, AVG(solde) FROM Compte
GROUP BY nom_agence **HAVING** AVG(solde) > 8000

1.9. Sous requêtes

Les sous-requêtes sont utiles pour structurer une requête un peu complexe sous une forme plus "procédurale" qu'une jointure. Elles sont indispensables lorsque la condition met en jeu un résultat de calcul d'aggrégat.

- pour tester l'appartenance à un ensemble : "*Quels sont les clients qui ont à la fois un emprunt et un compte* ?"

SELECT nom_client FROM Emprunt WHERE nom_client
IN (SELECT nom_client FROM Depot)

On peut également trouver les clients qui ont à la fois un emprunt et un compte en utilisant EXIST :

SELECT nom_client FROM Emprunt WHERE **EXIST**
(SELECT * FROM Depot WHERE Depot.nom_client = Emprunt.nom_client)

- pour comparer 2 ensembles : par exemple, la phrase "plus grand qu'au moins un" se représente en SQL par **> ANY**. "Quels sont les noms des agences qui ont un capital supérieur à celui d'au moins une agence de Brooklyn ?" peut s'exprimer :

SELECT DISTINCT T.nom_agence FROM Agence AS T, Agence AS S
WHERE T.capital > S.capital AND S.ville_agence = "Brooklyn"

Mais également :

SELECT nom_agence FROM Agence
WHERE capital **> ANY** (SELECT capital FROM Agence WHERE ville_agence = "Brooklyn")
La sous-requête fournit la table des capitaux de toutes les agences situées à Brooklyn, la clause WHERE est donc vérifiée si le capital de l'agence est supérieur à au moins un élément de cette table.
On peut également utiliser ALL : "Quels sont les noms des agences qui ont un capital supérieur à ceux de toutes les agences de Brooklyn ?" s'exprimera :
SELECT nom_agence FROM Agence
WHERE capital **> ALL**
(SELECT capital FROM Agence WHERE ville_agence = "Brooklyn")
On peut tester également s'il y a des lignes dupliquées dans le résultat d'une sous-requête avec le mot UNIQUE.
Une sous-requête peut apparaître dans la partie FROM, auquel cas on lui donne un nom, utilisé dans la partie WHERE.
Enfin, les fonctions d'agrégation sont très utilisées **en combinaison avec les sous-requêtes** :
Exemple : *"Quelle est l'agence qui a le plus grand solde moyen ?"*
SELECT nom_agence FROM Compte GROUP BY nom_agence
HAVING AVG(solde) >= ALL
(SELECT AVG(solde) FROM Compte GROUP BY nom_agence)

1.10. Jointures

Une jointure représente un produit cartésien sur lequel on fait une sélection (partie WHERE).
La version SQL-92 offre également d'autres mécanismes pour joindre 2 relations, le résultat étant en général destiné à figurer dans la partie FROM d'une requête.

1.10.1. EXEMPLES

Exemple 1 (**jointure avec condition**): pour joindre les tables Prêt et Emprunt avec comme condition l'égalité des numéros de compte de prêt, on peut écrire :
SELECT Pret.*, Emprunt.* FROM Pret, Emprunt WHERE Pret.num_pret = Emprunt.num_pret
Exemple 1 avec mot-clef : SELECT * FROM Pret **INNER JOIN** Emprunt **ON** Pret.num_pret = Emprunt.num_pret
A partir des tables de l'annexe, cette expression désigne la table suivante :

Nom_agence	Num_pret	Montant	Nom_client	Num_pret
Downton	L-17	10000	Jones	L-17
Redwood	L-52	25000	Hayes	L-52
Downton	L-23	150000	Smith	L-23

Exemple 2 (**jointure naturelle**) pour joindre "naturellement" les tables Prêt et Emprunt avec comme condition l'égalité des numéros de compte de prêt on peut écrire :
SELECT Pret.*, Emprunt.Nom_client
FROM Pret, Emprunt
WHERE Pret.num_pret = Emprunt.num_pret
Exemple 2 avec mot clef : Pret **NATURAL INNER JOIN** Emprunt
Résultat :

Nom_agence	Num_pret	Montant	Nom_client
Downton	L-17	10000	Jones
Redwood	L-52	25000	Hayes
Downton	L-23	150000	Smith

Exemple 3 (**jointure externe**) pour joindre les tables Prêt et Emprunt avec comme condition l'égalité des numéros de compte de prêt, en gardant tous les éléments de Prêt :

SELECT Pret.*, Emprunt.*
FROM Pret, Emprunt
WHERE Pret.num_pret = Emprunt.num_prêt **(+)**

Exemple 3 avec mot clef : Pret **LEFT OUTER JOIN** Emprunt **ON** Pret.num_pret = Emprunt.num_pret

Résultat :

Nom_agence	Num_pret	Montant	Nom_client	Num_pret
Downton	L-17	10000	Jones	L-17
Redwood	L-52	25000	Hayes	L-52
Mianus	L-22	5000	*null*	*nul*l
Downton	L-23	150000	Smith	L-23

1.10.2. TYPES DE JOINTURES ET CONDITIONS

Toutes les variantes offertes par SQL-92 concernent un certain type de jointure et une certaine condition de jointure :

- la condition spécifie quelles lignes des 2 tables on retrouvera dans le résultat, et quelles colonnes;
- le type indique comment sont traitées les lignes des tables qui ne correspondent à aucune ligne de l'autre table : sont-elles ignorées, sont-elles acceptées (et complétées par des valeurs Null)…

1.11. La gestion des droits d'accès aux éléments de la base

SQL permet de gérer les droits d'accès des utilisateurs à chaque éléments de la base. Cette gestion s'appuie sur la notion de "privilèges", et l'affectation (ou le retrait) de ces privilèges. Les principaux privilèges sont :

- SELECT droit de lecture
- INSERT droit d'ajout
- DELETE droit de suppression
- UPDATE droit de modification de valeur
- REFERENCES droit de référence à une table (ou une vue) dans une contrainte d'intégrité

On écrit SELECT(A1, …, An) pour parler de droits sur les colonnes A1, …, An.

En principe le créateur d'un objet est le seul à posséder tous les privilèges sur cet objet. Mais (s'il en a le droit) il peut transmettre des privilèges sur l'objet à d'autres utilisateurs, avec la commande :

GRANT liste_privilèges **ON** liste_objets **TO** liste_utilisateurs
[**WITH GRANT OPTION**]

Il peut également retirer des privilèges :

REVOKE [**GRANT OPTION FOR**] liste_privilèges
ON liste_objets **FROM** liste_utilisateurs