IUT Paul Sabatier – Département SeRéCom

Aide Mémoire SQL

Jean-Marie Pécatte

Janvier 2013

SOMMAIRE

I. Présentation

Le langage SQL (Structured Query Language) est un langage de communication entre les applications et le Système de Gestion de Bases de Données (SGBD). Il offre des commandes permettant de :

- DEFINIR LE SCHEMA DE LA BASE : créer une relation (**CREATE**), supprimer une relation (**DROP**), ou modifier la structure d'une relation (**ALTER**). Des commandes similaires existent pour manipuler la notion de sous-schéma (vue).
  => les commandes du langage de définition de données (DDL, Data Definition Language).
- MANIPULER LES DONNEES : ajouter des données (**INSERT**), supprimer des données (**DELETE**), ou modifier des données (**UPDATE**), mais aussi rechercher et sélectionner un sous-ensemble de données (**SELECT**).
  => les commandes du langage de manipulation de données (DML, Data Manipulation Language).
- CONTROLER LA BASE DE DONNEES : ajouter les droits d'accès (**GRANT** et **REVOKE**), gérer les transactions (**COMMIT**, **ROLLBACK**),…
  => les commandes du langage de contrôle de données (DCL, Data Control Language).

C'est le langage le plus employé dans les SGBD ; quasiment tous, relationnels ou non, utilisent SQL. Mais de nombreux autres logiciels (tableurs, logiciels de statistiques, ...) possèdent aujourd'hui une interface SQL permettant un accès simplifié aux données.

C'est un langage non-procédural : on spécifie seulement les données que l'on cherche, sans avoir à décrire et à programmer la méthode de recherche, ce qui entraîne un gain de temps dans le développement.

C'est un langage en pleine évolution déjà normalisé à 4 reprises :

- SQL86, SQL89, SQL92 (ou SQL2), SQL99 (SQL3)
- La majorité des SGBD supportent une grande partie de la norme SQL2, voire quelques éléments de la norme SQL3.

> *Notes* :
> - 1) les exemples de ce document seront illustrés avec une base de données concernant les informations administratives relatives au baccalauréat et au lycée des étudiants s'inscrivant à l'université ; le schéma est le suivant :
>
> ETUD (codetud, nom, prenom, datenais, bac, note, *codelycee*)
> LYCEE (codelycee, nom, ville)
>
> - 2) la version SQL présentée est celle de la norme SQL2. Certains compléments où certaines limites des SGBD seront signalés par le sigle ***!SGBD!***

II. Le Langage de Définition de Données

II.1. Les relations (le schéma de la BD)

II.1.1. Création d'une relation

La syntaxe de la commande SQL pour créer le schéma d'une relation est la suivante :

CREATE TABLE *<ident_relation>* **(** *<définition> { ,* *<définition> }* **)**

Lors de la création d'une relation, on définit les attributs et, éventuellement, les contraintes associées.

- **Définition d'attribut**

Pour chaque attribut, on doit préciser son type et s'il est à valeur facultative, NULL (par défaut), ou à valeur obligatoire (NOT NULL).

<définition> ::= *<ident_attribut> <type> [* **NULL** / **NOT NULL***]*

La liste des types de données possibles pour un attribut est la suivante :

	Type	**Explication**
chaîne de caractères ASCII (sur 1 octet)	CHAR (n)	Chaîne de caractères de longueur fixe *n*
	VARCHAR (n)	Chaîne de caractères de longueur variable avec maximum *n* caractères
Chaîne de caractères UNICODE (sur 2 octets)	NCHAR (n)	Chaîne de caractères de longueur fixe *n,* fondée sur un codage spécifique
	NCHAR VARYING (n)	Chaîne de caractères de longueur variable, fondée sur un Codage spécifique, avec maximum *n* caractères
Numériques	INT	Entier long
	SMALLINT	Entier court
	FLOAT(p,s)	Valeur réelle approchée
	REAL(p,s)	Valeur réelle approchée (petit réel par rapport à Float)
	DOUBLE(p,s)	Valeur réelle approchée (grand réel par rapport à Float)
	NUMERIC(p,s) ou DECIMAL(p,s)	Valeur réelle exacte avec une précision de *p* chiffres dont *s* à droite du point décimal ($1 \leq p \leq 38$, défaut p=38, $s \in [-84,127]$)
Temporels	DATE	Date comprise entre le 1er janvier de l'an 1 et 31 décembre de l'an 9999 – Format standard AAAA-MM-JJ
	TIME	Temps sur 24 heures Format standard HH:MM:SS.nnn
	TIMESTAMP	Combiné date-temps
	INTERVAL	Durée
Chaînes de bits	BIT(n)	Chaîne de bits de taille fixe n octets
	BIT VARYING (n)	Chaîne de bits de taille variable maximum n octets

Exemple :
prenom char(10) ***NOT NULL***
⇒ *prenom* est un attribut à valeur obligatoire de type chaîne de 10 caractères.

➢ **Valeur par défaut**

Il est possible de préciser une valeur par défaut pour un ou plusieurs attributs.

> Exemple : la valeur par défaut de l'attribut est 'S'
> *bac **VARCHAR**(3) **DEFAULT** 'S'* .

➢ **Définition de contraintes**

• Clé de la relation :
<*définition*> ::= **CONSTRAINT** <*ident_contrainte*> **PRIMARY KEY (**<*liste d'attributs*>**)**

> Exemple :
> ***CONSTRAINT** cle_etud **PRIMARY KEY** (codetud)*
> ⇒ la contrainte *cle_etud* définit l'attribut *codetud* comme la clé de la relation.

• Domaine de validité :
<*définition*> ::= **CONSTRAINT** <*ident_contrainte*> **CHECK** <condition>
la syntaxe d'une condition est la même que celle d'une clause WHERE

> Exemple :
> ***CONSTRAINT** note_ok **CHECK** (note> = 0 **AND** note < = 20)*
> ⇒ la contrainte *note_ok* définit le domaine de validité de l'attribut *note.*

• Contrainte d'intégrité référentielle :
<*définition*> ::= **CONSTRAINT** <*ident_contrainte*> **FOREIGN KEY (**<*attribut local*>**)**
REFERENCES <*relation de référence*>**(**<*attribut de référence*>**)**
[<règles de gestion>]

la valeur de l'*attribut local* (appelée aussi clé étrangère) n'est acceptable que si elle appartient à l'ensemble des valeurs de l'*attribut de référence*.

> Exemple : l'attribut *codelycee* de ETUD doit forcément faire référence à une valeur de l'attribut *codelycee* de LYCEE
> ***CONSTRAINT** exist_lycee **FOREIGN KEY** (codelycee)*
> ***REFERENCES** Lycee (codelycee)*

il est possible de rajouter des règles de gestion de l'intégrité lors de mises à jour des données.

Règle de gestion	*Explication*
ON DELETE NO ACTION	Ne rien faire en cas de suppression
ON UPDATE NO ACTION	Ne rien faire en cas de modification
ON DELETE CASCADE	Suppression en cascade : la suppression d'un tuple et donc de la clé primaire entraîne la suppression dans la relation fille de tous les tuples ayant pour valeur de la clé étrangère celle de la clé primaire.
ON UPDATE CASCADE	Modification en cascade : la modification de la clé primaire est répercutée automatiquement dans toutes les clefs étrangères de la relation fille.
ON DELETE SET NULL	Déférencement en cas de suppression : lors de la suppression de la clé primaire, toutes les valeurs des clés étrangères possédant cette valeur sont automatiquement mise à NULL.
ON UPDATE SET NULL	Déférencement en cas de mise à jour : lors de la modification de la clé primaire, toutes les valeurs des clés étrangères possédant cette valeur sont automatiquement mise à NULL.
ON DELETE SET DEFAULT	Reprise de la valeur par défaut de la clé étrangère lors de la suppression de la clé primaire.
ON UPDATE SET DEFAULT	Reprise de la valeur par défaut de la clé étrangère lors de la modification de la clé primaire.

Exemple : la suppression d'un lycée dans la relation LYCEE entraînera automatiquement la suppression des tous les étudiants de la relation ETUD provenant de ce lycée.

***CONSTRAINT** exist_lycee **FOREIGN KEY** (codelycee)*
REFERENCES** Lycee (codelycee) **ON DELETE CASCADE

***!SGBD! :** la gestion des contraintes d'intégrité est complexe ; les SGBD les gèrent plus ou moins bien. Par exemple MySql ne gère pas de base ce type de contrainte ; pour les prendre en charge, il faut une version de MySql qui intègre InnoDb et il faut créer les tables de type InnoDb.*

• Attribut à valeur unique :

<*définition*> :: = **CONSTRAINT** <*ident_contrainte*> **UNIQUE** <*liste_d'attributs*>

➢ **Création de la base exemple**

Création de la relation *lycée* :

```
CREATE TABLE lycee
( codelycee     INT       NOT NULL,
  nom           VARCHAR (40)   NOT NULL,
  ville         VARCHAR (30)   NULL,
 CONSTRAINT cle_lycee PRIMARY KEY (codelycee)
)
```

Création de la relation *étud* :

```
CREATE TABLE etud
( codetud       INT       NOT NULL,
  nom           VARCHAR (20)   NOT NULL,
  prenom        VARCHAR (20)   NOT NULL,
  datenais      DATE           NULL,
  bac           VARCHAR (3)    NULL,
  note          NUMERIC(4,2)   NULL,
  codelycee     INT       NULL,
 CONSTRAINT  cle_etud PRIMARY KEY (codetud),
 CONSTRAINT  note_ok CHECK (note>= 0 AND note <= 20),
 CONSTRAINT  exist_lycee FOREIGN KEY (codelycee) REFERENCES Lycee (codelycee)
)
```

II.1.2. Création d'une relation à partir d'une requête

Il existe une deuxième syntaxe de la commande **CREATE TABLE** permettant de créer une relation à partir de relations existantes :

CREATE TABLE <*ident_relation*> *[* **(** <*ident_attribut*> *{* **,** <*ident_attribut*> *}* **)** *]*
AS <*requête_SELECT*>

Le schéma de la relation est composé des attributs de la projection de la requête SELECT avec la même définition. Les tuples de la nouvelle relation sont une copie de ceux résultant de la requête SELECT.

Exemple : création de la relation EtudBac(nom, prénom, bac) à partir de la relation Etud

```
CREATE TABLE EtudBac AS SELECT nom, prénom, bac FROM Etud
```

Attention : ne pas confondre la création d'une table à partir d'une requête et la création d'une vue.

II.1.3. Création d'une relation temporaire

La syntaxe de la commande SQL est similaire à celle de la création d'une relation permanente à l'exception du mot clé TEMPORARY. Toute relation temporaire est détruite automatiquement à la fin de la transaction ou de la session :

CREATE [LOCAL / GLOBAL] TEMPORARY TABLE <*ident_relation*>
(<*définition*> *{* **,** <*définition*> *}* **)** *[* **ON COMMIT** *{* **DELETE** | **PRESERVE** *}* **ROWS** *]*

Avec l'option LOCAL, la relation n'est visible que dans la transaction alors qu'avec le mot clé GLOBAL, la relation est visible aux autres transactions et utilisateurs.

L'option ON COMMIT permet de choisir de garder (PRESERVE) ou supprimer (DELETE) les données à la fin de la transaction.

***!SGBD!** : la gestion des tables temporaires est assez différente suivant les SGBD. MySql ne gère pas les tables temporaires. Postgresql ne gère que des tables LOCAL, Oracle que des GLOBAL. Postgresql ne gère pas l'option ON COMMIT.*

II.1.4. Suppression d'une relation

Cette commande permet de supprimer la relation avec toutes ses données :

DROP TABLE <*ident_relation*>

Exemple : suppression de la relation etud : ***DROP TABLE** etud*

II.1.5. Modification d'une relation :

- ajout d'un attribut :
  ALTER TABLE <*ident_relation*> **ADD COLUMN** <*def_attribut*>
- modification d'un attribut :
  ALTER TABLE <*ident_relation*> **ALTER COLUMN** <*attribut*>
  { **SET DEFAULT** <valeur> | **DROP DEFAULT** | **DROP DOMAIN** }
- suppression d'un attribut :
  ALTER TABLE <*ident_relation*> **DROP COLUMN** <*ident_attribut*>
- ajout d'une contrainte :
  ALTER TABLE <*ident_relation*> **ADD CONSTRAINT** <*def_contrainte*>
- suppression d'une contrainte :
  ALTER TABLE <*ident_relation*> **DROP CONSTRAINT** <*ident_contrainte*>

!SGBD! *: pour certains SGBD (comme Oracle), seules les modifications qui n'amènent pas de perte d'information sont autorisées. De plus Oracle permet de modifier le type d'un attribut : il faut alors utiliser la commande ALTER TABLE <ident_relation> MODIFY <def_attribut>. MySql le permet aussi mais avec la commande CHANGE.*

II.2. Les vues

II.2.1. Création et suppression

Une vue est une relation virtuelle :
- son schéma est composé d'attributs de relation réelles
- ses données sont celles des relations réelles ayant servi à sa création.

Quand on manipule les données d'une vue, on travaille en réalité sur les données des relations réelles.

- Création :

  CREATE VIEW <*ident_vue*> *[***(**<*liste de noms d'attributs*>**)***]*
  AS <*requête*> *[***WITH CHECK OPTION***]*

> Exemple :
> ***CREATE VIEW*** *Etud_S*
> ***AS*** ***SELECT*** *nom, codetud, prenom*
> ***FROM*** *etud*
> ***WHERE*** *bac = 'S'*
> *à travers la vue Etud_S, on ne voit que le nom, le prenom et le codetud des étudiants de la relation etud qui ont un bac S.*

> Exemple :
> ***CREATE VIEW*** *Stat_Lycee (nom_lycee, nbre_etud, codelycee)*
> ***AS*** ***SELECT*** *lycee.nom,* ***COUNT*** *(codetud), lycee.codelycee*
> ***FROM*** *lycee* ***INNER JOI****N etud* ***USING*** *(codelycee)*
> ***GROUP BY*** *lycee.nom , lycee.codelycee*
> *crée la vue Stat_Lycee de schéma (nom lycée, nombre étudiants, code du lycée) et de données (liste des lycées avec le nombre d'étudiants).*

- suppression :

  DROP VIEW <*ident_vue*>

- modification : il n'existe pas de commande pour modifier le schéma d'une vue ; il faut supprimer la vue et la créer à nouveau avec sa nouvelle définition.

II.2.2. Manipulations de données

- sélection : une vue peut être utilisée comme n'importe quelle relation dans une expression de sélection.

> Exemple : liste des étudiants des lycées dont le nombre d'étudiants est < 10
> ***SELECT*** *nom*
> ***FROM*** *stat_lycee, etud*
> ***WHERE*** *etud.codelycee = stat_lycee.codelycee*
> ***AND*** *nbre_etud < 10*

- insertion, suppression, modification

La contrepartie d'une vue par rapport à une relation est que les données d'une vue ne sont pas toujours modifiables.

C'est pourquoi la norme SQL2 indique que les données d'une vue sont modifiables uniquement si la vue a été créée à partir d'une requête SELECT contenant :
- une seule relation ou une seule vue (elle-même modifiable)
- ni jointure (sous forme standard ou relationnelle), ni intersection, ni union, ni différence
- ni la clause DISTINCT, ni une expression de calcul dans la clause SELECT
- ni la clause GROUP BY, ni HAVING
- aucune requête imbriquée (sous forme procédurale) faisant référence à la relation initiale.

Si cette règle est vérifiée alors il est possible de faire un INSERT, UPDATE ou DELETE directement sur la vue.

> Exemple :
> *aucune opération n'est possible sur la vue STAT_LYCEE*

Remarque : le **INSERT** n'est possible que si la vue respecte les règles précédentes et si tous les attributs **NOT NULL** de la relation sont aussi présents dans la vue.

> Exemple :
> *- Il y a 3 attributs **NOT NULL** dans la relation **ETUD** qui font tous partie de la vue Etud_S, donc l'insertion est possible dans Etud_S*
> ***INSERT INTO** Etud_S **VALUES** ('Dupond', 1034, 'Jean')*
>
> - la vue n'a pas de données, donc l'étudiant Dupont est rajouté dans la relation ETUD, sans date de naissance, sans bac, sans codelycee et sans note.

- Problème de mise à jour *:*

Sur l'exemple précédent, l'étudiant DUPONT n'ayant pas de valeur pour l'attribut bac, il n'apparaîtra pas à travers la vue Etud_S.

Pour éviter ce problème et donc interdire les mises à jour à travers une vue qui donnent des tuples qui ne sont plus visibles à travers cette vue, il faut rajouter l'option WITH CHECK OPTION lors de la création de la vue.

> Exemple :
> ```
> CREATE VIEW Etud_S_bis
> AS      SELECT nom, codetud, prenom
>         FROM etud
>         WHERE bac = 'S'
> WITH CHECK OPTION
> ```

Il n'est pas possible de rajouter (INSERT) des étudiants à travers la vue Etud_S_bis, car ces étudiants n'ayant pas de valeur pour l'attribut Bac, il n'apparaîtraient pas à travers cette vue.

***! SGBD !** : cette notion de vue est assez complexe à mettre en œuvre ; dans certains SGBD cette notion n'existe pas (comme dans MySql) ou bien elle n'est que partiellement implémentée (vues en lecture uniquement pour Postgresql). D'autres, comme Oracle, ont au contraire essayé de rendre plus souple leur utilisation en réduisant les règles fixées par SQL2 pour les vues modifiables.*

II.3. Les index

Un index sert à accélérer les recherches. Toutefois lors de la mise à jour de données dans la relation portant des index, il est nécessaire de mettre à jour aussi les index ; la création d'index ralentit le temps de mise à jour des informations. Il faut donc les utiliser avec parcimonie.

II.3.1. Création

Il n'y a pas de syntaxe standard pour la gestion des index mais la définition d'un index reprend souvent la syntaxe de l'exemple ci-dessous :

> Exemple : création d'un index sur le nom des étudiants pour accélérer les recherches.
> ```
> CREATE INDEX EtudNom ON Etud(nom)
> ```

II.3.2. Suppression

> Exemple : suppresion de l'index EtudNom
> ```
> DROP INDEX EtudNom
> ```

III. Le Langage de Manipulation de Données

III.1. Le langage de mise à jour des données

III.1.1. Ajout d'un tuple dans une relation

La commande INSERT permet de rajouter **un** tuple dans **une** relation.

• Première forme :

INSERT INTO <*ident_relation*> **VALUES (** <*liste de valeurs*>**)**

⇒ obligation de donner une valeur pour chaque attribut de la relation et dans l'ordre de la définition.

> Exemple :
> ***INSERT INTO*** *Lycee* ***VALUES (****101, 'Borde Basse', 'Castres'****) est correct*
> ***INSERT INTO*** *Lycee* ***VALUES (****'Borde Basse',101, 'Castres'****) est incorrect*
> ***INSERT INTO*** *Lycee* ***VALUES (****102,'Soult',* ***NULL)*** *est correct*
> ⇒ Il faut utiliser la valeur NULL si on ne veut pas préciser de valeur pour un attribut à valeur facultative.

• Deuxième forme :

INSERT INTO <*ident_relation*> **(**<*liste d'attributs*>**) VALUES (**<*liste de valeurs*>**)**

⇒ on ne précise les valeurs que pour les attributs de la liste et dans l'ordre de cette liste.

> Exemple :
> ***INSERT INTO*** *lycee* ***(****codelycee, nom****) VALUES (****101, 'Borde Basse'****)***
> ***INSERT INTO*** *lycee* ***(****ville, codelycee, nom****) VALUES (****'Castres', 101, 'Borde Basse'****)***

Remarque : si un attribut est défini à valeur obligatoire, il doit forcément faire partie de la liste.

INSERT INTO *Lycee* ***(****codelycee, ville****)*** est incorrect car il manque l'attribut *nom* qui est défini **NOT NULL**

III.1.2. Suppression de tuples

La commande DELETE supprime tous les tuples de la relation qui vérifient la condition (éventuelle).

DELETE FROM <*ident_relation*>
[**WHERE** <*condition*>*]*

> Exemple :
> ***DELETE FROM*** *Etud*
> ***WHERE*** *note < 12* ***AND*** *bac = 'ES'*
> ⇒ Supprime de la relation *Etud* tous les étudiants qui ont un bac ES et une note < 12.

III.1.3. Modification de tuples

La commande UPDATE permet de modifier les valeurs d'un ou plusieurs attributs d'**une** relation :

UPDATE <*ident_relation*>
SET <*ident_attribut*> = <*expression*> { **,** <*ident_attribut*> = <*expression*> }
*[***WHERE** <*condition*>*]*

Cette commande modifie tous les tuples qui vérifient la condition (éventuelle).

> Exemple : ***UPDATE*** *lycee*
> ***SET*** *ville = 'Mazamet'*
> ***WHERE*** *nom = 'Soult'*

Il est possible d'avoir comme expression une requête SELECT qui doit renvoyer une valeur unique ; cette requête peut-être corrélée avec la relation sur laquelle porte la modification.

Exemple : soit la relation EtudNbBac(Bac, NbEtudiants) qui donne pour chaque Bac le nombre d'étudiants ; la requête suivante permet de mettre à jour cette relation à partir de Etud

```
UPDATE EtudNbBac
SET NbEtudiants = (SELECT count(codeetud) FROM Etud
        WHERE Etud.Bac=EtudNbBac.Bac )
```

III.1.4. Copie de tuples entre deux relations

INSERT INTO <*ident_relation*> *[* **(**<*liste d'attributs*>**)** *]* <*requête SELECT*>

Cette syntaxe de la commande INSERT permet de rajouter (copier) des données dans une relation à partir de données existantes dans d'autres relations.

Exemple : copier dans la relation EtudBac tous les étudiants de Etud qui ont un bac S

```
INSERT INTO EtudBac SELECT nom, prénom, bac FROM Etud WHERE Bac='S'
```

III.2. Le langage d'interrogation de données

Syntaxe générale

SELECT *<liste d'attributs>*
FROM *<liste de relations>*
[**WHERE** *<condition>]*
[**GROUP BY** *<liste d'attributs>]*
[**HAVING** *<condition>]*
[**ORDER BY** *<liste d'attributs>]*

III.2.1. les opérateurs de l'algèbre relationnelle

- **Projection**

SELECT *<liste d'attributs>*
FROM *<ident de la relation>*

Exemple : Liste des nom des lycées
SELECT *nom*
FROM *Lycée*

Remarque : par défaut, SQL garde les éventuels doublons ; il faut rajouter le mot clé **DISTINCT** pour les supprimer.

Exemple : ***SELECT DISTINCT*** *bac*
FROM *etud* *les différents bacs n'apparaissent qu'une fois*

- **Restriction**

SELECT *
FROM *<ident_relation>*
WHERE *<condition>*

L'étoile '*' représente l'ensemble des attributs du schéma relationnel

Exemple : Liste de toutes les informations relatives aux étudiants qui ont eu entre 12 et 14 au Bac
SELECT *
FROM *etud*
WHERE *(note> 12 AND note < 14)*

- **Sélection**

La sélection est la combinaison d'une restriction et d'une projection.

Exemple : Liste des noms des étudiants qui ont eu un Bac S
SELECT *nom*
FROM *etud*
WHERE *bac = 'S'*

- **Union, Intersection, Différence,**

SELECT *<liste d'attributs>*
FROM *<liste de relations>*
WHERE *<condition>*
UNION | **INTERSECT** | **EXCEPT** [**ALL**]
SELECT *<liste d'attributs>*
FROM *<liste de relations>*
WHERE *<condition>*

Remarque : la liste d'attributs doit être identique ou compatible du point de vue des types dans les deux sélections.

Exemple : Liste des noms des étudiants qui ont eu un bac ES ou un bac S

```
SELECT nom FROM Etud WHERE bac = 'ES'
UNION
SELECT nom FROM Etud WHERE bac = 'S'
```

Par défaut les opérateurs ensemblistes suppriment les doublons, le mot clé **ALL** permet de les garder si nécessaire.

!SDBD! *: certains SGBD ne respectent pas le nom standardisé ; par exemple, pour Oracle, la différence s'écrit MINUS. D'autres comme MySql n'implémentent aucun de ces 3 opérateurs.*

➢ **Produit cartésien**

SELECT *
FROM <*liste de relations*>

Exemple : Produit cartésien des relations etud et lycée

```
SELECT *
FROM etud, lycee
```

Exemple : Produit cartésien des relations etud et lycée

```
SELECT *
FROM etud CROSS JOIN lycee
```

➢ **Jointure**

• syntaxe standard de la jointure interne

Exemple : Liste des noms des étudiants qui ont suivi leurs études dans un lycée de Castres

```
SELECT etud.nom
FROM etud INNER JOIN lycee ON etud.codelycee = lycee.codelycee
WHERE ville = 'Castres'
```

Exemple : forme simplifiée dans le cas où l'attribut à le même nom dans les 2 relations

```
SELECT etud.nom
FROM etud INNER JOIN lycee USING (codelycee)
WHERE ville = 'Castres'
```

Exemple : forme "naturelle" : on ne précise pas l'attribut de jointure, les attributs de même nom dans les 2 relations sont utilisés comme condition de jointure

```
SELECT nom
FROM etud NATURAL INNER JOIN lycee
WHERE ville = 'Castres'
```

!!! Attention !!! cette syntaxe présente un danger si plusieurs attributs des 2 relations ont le même nom. Pour cette exemple la jointure se fait sur le couple (codelycee, nom) ce qui implique que l'étudiant ait le même nom que le lycée dont il provient !!!
La forme naturelle de la jointure n'est pas conseillée.

• la forme procédurale (plusieurs requêtes emboîtées)

Exemple : Liste des noms des étudiants qui ont suivi leurs études dans un lycée de Castres

```
SELECT nom
FROM etud
WHERE codelycee IN ( SELECT codelycee
                     FROM lycee
                     WHERE ville = 'Castres')
```

• la forme relationnelle (pour les SGBD ne supportant pas la norme ANSI) la condition de jointure est exprimée comme une condition de restriction

Exemple : Liste des noms des étudiants qui ont suivi leurs études dans un lycée de Castres

```
SELECT etud.nom
FROM etud, lycee
WHERE  (etud.codelycee = lycee.codelycee) AND (ville = 'Castres')
```

Remarque : pour lever l'ambiguïté éventuelle d'un nom attribut, on utilise :
- soit le nom de la relation (cf exemple ci-dessus)
- soit des variables de désignation

Exemple : il y a deux attributs *nom* et deux attributs *codelycee* (lycée et etud)

```
SELECT X.nom
FROM etud X INNER JOIN lycee Y ON X.codelycee = Y. codelycee
WHERE ville = 'Castres'
```

Exemple :

```
SELECT X.nom
FROM etud X, lycee Y
WHERE (X.codelycee = Y. codelycee) AND (ville = 'Castres')
```

• La jointure externe

Ce type de jointure permet de garder toutes les données d'une (des 2) relations même si elles ne vérifient pas la condition de jointure.

Exemple : Liste de tous les noms des étudiants avec le nom du lycée connu plus les noms des étudiants dont le lycée est inconnu plus les noms des lycées pour lesquels il n'y aurait pas d'étudiants : jointure externe (OUTER JOIN)

```
SELECT etud.nom,  lycee.nom
FROM etud FULL OUTER JOIN lycee ON etud.codelycee = lycee.codelycee
WHERE ville = 'Castres'
```

ou encore :

```
SELECT etud.nom,  lycee.nom
FROM etud FULL OUTER JOIN lycee USING (codelycee)
WHERE ville = 'Castres'
```

Exemple : Liste de tous les noms des étudiants avec le nom du lycée (connu ou pas) jointure externe gauche (LEFT JOIN)

```
SELECT etud.nom, lycee.nom
FROM etud LEFT OUTER JOIN lycee ON etud.codelycee = lycee.codelycee
```

Exemple : jointure externe droite (RIGHT JOIN)

```
SELECT etud.nom, lycee.nom
FROM lycee RIGHT OUTER JOIN etud ON etud.codelycee = lycee.codelycee
```

• L'auto-jointure

Ce type de jointure permet réaliser une jointure en utilisant plusieurs fois la même relation.

Exemple : Liste des noms des étudiants qui ont une note supérieure à celle de l'étudiant Dupont

```
SELECT X.nom
FROM etud X INNER JOIN etud Y ON X.note > Y.note
WHERE Y.nom = "Dupont"
```

Remarque : la condition de jointure n'est pas forcément une égalité

• requêtes emboîtées corrélées

Il est possible d'écrire une jointure avec des requêtes emboîtées (cf. forme procédurale) mais en utilisant un lien entre les différentes requêtes.

Exemple :Liste des noms des étudiants qui ont une note supérieure ou égale à la moyenne de leur lycée

```
SELECT nom, note
FROM etud X
WHERE note >= ( SELECT AVG(note)
                FROM etud Y
                WHERE X.codelycee=Y.codelycee)
```

III.2.2. la clause WHERE

• Comparaison à une valeur : <, >, > =, < =, =, < >

• Comparaison à une fourchette : **BETWEEN**, **NOT BETWEEN**

Exemple :

```
SELECT nom
FROM etud
WHERE note BETWEEN 12 AND 14
```

• Appartenance à un ensemble de valeurs : **IN**, **NOT IN**

Exemple :

```
SELECT nom
FROM etud
WHERE bac IN ('S', 'ES')
```

• Comparaison à un ensemble de valeurs : **ANY**, **ALL**

- à au moins un élément de l'ensemble

  WHERE <*attribut*> <*operateur*> **ANY** <*ensemble*>

- à tous les éléments de l'ensemble

  WHERE <*attribut*> <*operateur*> **ALL**<*ensemble*>

Exemple :

```
WHERE note > ALL (SELECT note FROM Etud WHERE bac = 'ES')
                 doit être supérieure à toutes les notes de l'ensemble
```

• Comparaison d'une chaîne de caractères à un filtre : **LIKE**, **NOT LIKE**

Le filtre est défini à l'aide de deux caractères spéciaux : '%' n'importe quelle séquence de caractères
'-' n'importe quel caractère

Exemple :
Liste des étudiants dont le nom commence par la lettre A

```
SELECT nom
FROM etud
WHERE nom LIKE 'A%'
```

Liste des étudiants dont le nom comporte 4 lettres et commence par la lettre B

```
SELECT nom
FROM etud
WHERE nom LIKE 'B---'
```

• Test sur les valeurs indéterminées : **IS NULL**, **IS NOT NULL**

Exemple :
Liste des noms des étudiants pour lesquels on ne connaît pas la note au bac

```
SELECT nom
FROM etud
WHERE note IS NULL
```

III.2.3. Structure conditionnelle

Il existe une structure conditionnelle en SQL permettant un branchement multiple sur différentes expressions booléennes (structure comparable aux structures conditionnelles multiples des langages de programmation). Il existe deux syntaxes possibles.

<struct_conditionnelle> ::= **CASE** <*expression*> **WHEN** <*valeur1*> **THEN** *expr1*
[**WHEN** <*valeur*$_n$> **THEN** *expr*$_n$ *]*
[**ELSE** <*expr_defaut*> *]*
END

<struct_conditionnelle> ::= **CASE WHEN** <*condition1*> **THEN** *expr1*
[**WHEN** <*condition*$_n$> **THEN** *expr*$_n$ *]*
[**ELSE** <*expr_defaut*> *]*
END

Exemple :*la mention au bac*

```
SELECT nom, prenom, bac, CASE
                            WHEN note >=16 THEN 'TB'
                            WHEN note >=14  THEN 'B'
                            WHEN note >=12 THEN 'AB'
                            ELSE  'Passable'
                         END AS mention
FROM etud
```

Exemple :
Lister les étudiants dans l'ordre : Bac S, Bac ES, Bac STT (impossible avec order by)

```
SELECT nom, prenom, bac, CASE bac
                            WHEN 'S' THEN 1
                            WHEN 'ES' THEN 2
                            ELSE  3
                         END AS codebac
FROM etud
ORDER BY codebac
```

III.2.4. Les fonctions

➢ **Fonctions numériques**

abs(n**)**	Valeur absolue de n
ceil(n**)**	Entier supérieur ou égal à n
mod(m,n**)**	Reste de la division entière de m par n
power(m,n**)**	m élevé à la puissance n
round(m,n**)**	m arrondi à n décimales
sqrt(n**)**	Racine carré de n

➢ **Fonctions sur les chaînes de caractères**

||	Opérateur de concaténation
initcap(s**)**	la première lettre de s est mise en majuscules
upper(s**)**	Conversion de s en majuscules
lower(s**)**	Conversion de s en minuscules
trim(s**)**	Suppression des espaces
character_length(s**)**	Longueur de la chaîne s
substring(s **from** r **for** l**)**	Extrait une sous-chaîne de s à partir du rang r et de longueur l

➢ **Fonctions sur les dates** (spécifiques Oracle)

months_between(d1,d2)	Nombre de mois entre les 2 dates d1 et d2
Sysdate	Date et heure système
to_char(d, format**)**	Convertit une date en chaîne de caractères
to_date(s, format**)**	Convertit une chaîne de caractères en date

Exemple : afficher la date système
SELECT sysdate **FROM** dual *(dual est une relation utilitaire)*

Le format de date par défaut du système ORACLE est DD-MON-YY (par exemple 03-OCT-99). Si l'on veut manipuler des dates sous un autre format, il est alors nécessaire d'utiliser les fonctions **to_char** ou **to_date.**

Exemple : afficher l'heure système
SELECT to_char(sysdate,'HH:MM') **FROM** dual

Exemple : afficher le jour de la semaine
SELECT to_char(sysdate,'DAY') **FROM** dual

Exemple : modifier la date de naissance de Dupont au 01-Avril-1948
UPDATE etud **SET** datenais = **to_date**('01-Avril-1948','DD-MONTH-YYYY')
WHERE nom='Dupont'

III.2.5. le tri

Une relation est considérée comme un ensemble de données ; il n'y a donc pas de notion d'ordre. L'ordre des tuples peut être différent pour deux exécutions d'une même requête. La commande suivante permet de définir un ordre :

SELECT <*liste d'attributs*>
FROM <*liste de relations*> *[***WHERE** <*condition*>*]*
ORDER BY <*nom attribut*> | <*position*> *[***ASC** | **DESC***]*

ASC -> par ordre croissant (par défaut) **DESC ->** par ordre décroissant

Exemple : Liste des étudiants par ordre alphabétique
SELECT *nom*
FROM *etud*
ORDER BY *nom*

III.2.6. les groupes

Il est possible de partitionner une relation en groupes de manière à ce que chaque tuple d'un groupe ait la même valeur pour un attribut particulier.

SELECT <*liste d'attributs*>
FROM <*liste de relations*>
*[***WHERE** <*condition*>*]*
GROUP BY <*liste d'attributs*>
*[***HAVING** <*condition*>*]*

Exemple :
SELECT *bac*
FROM *etud*
GROUP BY *bac*
La relation etud est partitionnée en groupes : chaque groupe correspond à l'ensemble des étudiants ayant le même bac. Il y a donc autant de groupes que de bacs différents.

Remarque : si on partitionne une relation à l'aide d'un GROUP BY sur une liste d'attributs, alors on ne peut faire la projection dans le SELECT qu'au maximum sur un sous-ensemble de cette liste d'attributs (non compris les fonctions d'agrégats bien sur).

➢ **Fonctions d'agrégats**

Ces fonctions permettent de faire des calculs sur l'ensemble des valeurs d'un attribut d'un groupe :

COUNT	:	nombre d'occurrences
SUM	:	somme des valeurs
AVG	:	moyenne des valeurs
MAX	:	la plus grande valeur
MIN	:	la plus petite valeur
STDDEV	:	l'écart type
VARIANCE	:	la variance

Exemple :
SELECT COUNT (nom), bac
FROM etud
GROUP BY bac
pour chaque bac, le nombre d'élèves qui l'ont obtenu.

Remarque : si on n'emploie pas l'opérateur GROUP BY, la relation est considérée comme un seul groupe :

Exemple :
SELECT MIN *(note),* ***MAX*** *(note),* ***AVG*** *(note)*
FROM *etud*
La note minimum, la note maximum, la moyenne des notes au bac, tous bacs confondus

➢ **Condition sur les groupes**

Il est possible de ne conserver que certains groupes, ceux qui vérifient la condition du HAVING.

Exemple : Liste des bacs où moins de 10 étudiants ont été reçus.
SELECT *bac*
FROM *etud*
GROUP BY *bac*
HAVING COUNT *(bac) <10*

IV. Le Langage de Contrôle de Données

IV.1. Les droits d'accès et privilèges

• Celui qui crée une relation possède tous les privilèges sur cet objet. Il peut donner des droits d'accès à d'autres utilisateurs à l'aide de la commande :

GRANT <*privilèges*> **ON** <*relation*> **TO** <*utilisateur*> *[***WITH GRANT OPTION***]*

Les privilèges les plus fréquents sont les suivants :

INSERT	:	droit de rajouter des tuples
DELETE	:	droit de supprimer des tuples
UPDATE	:	droit de modifier des tuples
SELECT	:	droit de sélectionner des données
ALTER	:	droit de modifier la structure d'une relation
ALL	:	tous les privilèges précédents
REFERENCES	:	droit de faire références aux attributs de la relation lors de la définition d'une clé étrangère

Remarque : l'option **WITH GRANT OPTION** autorise celui qui reçoit le privilège à le transmettre à un autre utilisateur .

Exemple : ***GRANT*** *select, update* ***ON*** *etud* ***TO*** *Dupont*
cette commande est exécutée par le créateur de la relation etud (Durand) *et donne le droit à l'utilisateur Dupont de sélectionner et de modifier les données de la relation etud.*

!SGBD! *Dans certains SGBD comme Oracle, pour les privilèges INSERT et UPDATE, il est possible de préciser la liste des attributs autorisés.*

Exemple :
GRANT update(nom, prenom) **ON** etud **TO** Dupont
cette commande donne le droit à l'utilisateur Dupont de modifier les données de la relation etud mais uniquement les nom et prénom.

• Pour accéder à une relation dont on n'est pas le propriétaire mais pour laquelle on a reçu un privilège, il faut faire précéder le nom de la relation par le nom du propriétaire.

Exemple : ***SELECT*** *** ***FROM*** *Durand.etud*
accès à la relation etud dont Durand est le propriétaire.

• Pour supprimer un privilège accordé précédemment :
REVOKE <*privileges*> **ON** <*relation*> **FROM** <*utilisateur*>

Remarque : un nom utilisateur particulier PUBLIC correspond à tous les utilisateurs du SGDB.

Exemple :
GRANT *select* ***ON*** *lycee* ***TO PUBLIC***
autorise tous les utilisateurs du SGBD à sélectionner des données de la relation lycee.

IV.2. Les transactions

Une transaction est définie comme une unité logique de travail : un ensemble de changements dans une base de données sur une ou plusieurs tables.

Une transaction débute lors de la connexion à la base de données ou bien juste après la fin de la transaction précédente.

A la fin d'une transaction, on peut valider les modifications à l'aide de la commande **COMMIT** ou bien annuler toutes les modifications à l'aide de la commande **ROLLBACK**.

L'interface PSQL du SGBD PostgreSQL

Cette interface permet d'exécuter des **commandes SQL** en mode **interactif**.

• **Exécution de l'interface** :
psql <nom_base>
(PostgreSQL prend comme nom d'utilisateur celui donné lors de la connexion au système).
p*sql – U <utilisateur> <nom_base>*

• **Sortie de l'interface** : *q*

• **Saisie d'une commande SQL**

La saisie se fait directement après 'nom_base => '. Cette saisie peut se faire sur plusieurs lignes.

```
Nom_base => select nom
Nom_base => from etud
Nom_base => where nom='Dupont'
```

Remarque : L'interface PSQL gère un historique des précédentes commandes.

• **Exécution d'une commande :** *g ou bien terminer la saisie par un* **;**

• **Modification d'une commande :** *utiliser l'historique ou \e pour la modifier avec un éditeur de texte*

• **De l'aide sur une commande SQL** : *h <commande_SQL>*

• **De l'aide sur l'interface PSQL** : *?*

• **Sauvegarde d'une commande dans un fichier** : *\w <nom_fichier>*

• **Exécution d'un fichier de commandes** : *i <nomfichier>*

• **Sauvegarde des résultats** :
pour commencer l'enregistrement dans un fichier des résultats :
\o <nom_fichier>
à partir de ce moment les résultats de toutes les commandes SQL iront dans le fichier
pour arrêter l'enregistrement : \o

• **Schéma d'une Base de données :** Toutes les informations manipulées par le SGBD PostgreSQL pour sa propre gestion, le sont sous forme de relation.

`Select * from pg_tables` *affiche la liste des relations de la base de données*
ou encore
`\d` *idem sans les relations systèmes*
ou bien
`\dt` *pour n'avoir que les relations*

La commande *\d nom_relation* donne le schéma d'une relation

`\d toto` *affiche le schéma de la relation* `toto`

L'interface SQL*PLUS du SGBD ORACLE

Cette interface permet d'exécuter des **commandes SQL** en mode **interactif**.

• **Exécution de l'interface** : *sqlplus*
Enter user-name : / ↵
Ne pas saisir de nom d'usager (Oracle prend celui donné lors de la connexion au système).

• **Sortie de l'interface** : *quit*

• **Saisie d'une commande SQL**

La saisie se fait directement après 'SQL> '. Cette saisie peut se faire sur plusieurs lignes.

```
SQL> select nom
  2  from etud
  3  where nom='Dupont'
  4
SQL>
```
← *La saisie se **termine** par une **ligne vide***

Remarque : Le tampon de SQPLUS ne peut contenir qu'une **seule** commande SQL. La saisie d'une nouvelle commande fait perdre la précédente.

• **Exécution d'une commande :** *RUN*

• **Modification d'une commande avec 'emacs'** : *EDIT*

• **De l'aide sur une commande SQL ou SQL*PLUS** : *HELP commande*

• **Sauvegarde d'une commande dans un fichier** : *SAVE nom_fichier*
Le suffixe .SQL est ajouté au nom du fichier afin de l'identifier comme un fichier de requête(s) SQL.

• **Charger une commande à partir d'un fichier** : *GET nom_fichier*
La commande GET permet de charger dans le tampon un fichier de requêtes SQL.

• **Exécution d'un fichier de commandes** : *START nom_fichier*
La commande START permet d'exécuter un fichier de requêtes. Cette commande peut être paramétrée (&1 ---> &9).

• **Sauvegarde des résultats** : *SPOOL nom_fichier | SPOOL OFF*
Toutes les informations qui sont affichées à l'écran (après avoir tapé la commande SPOOL) seront stockées dans le fichier spécifié. Ce processus s'arrête lorsqu'on tape la commande SPOOL OFF.

• **Schéma d'une Base de données :** Toutes les informations, manipulées par le SGBD ORACLE pour sa propre gestion, le sont sous forme de relation. Ainsi la relation USER_CATALOG donne la liste des relations appartenant à l'utilisateur et la relation USER_CONSTRAINTS donne la liste des contraintes définies par l'utilisateur sur ses relations.

```
select * from user_catalog
```
affiche la liste des relations de la base de données

La commande *DESC nom_relation* donne le schéma d'une relation

```
desc toto
```
affiche le schéma de la relation `toto`

MEMENTO

Sélections :

SELECT <*liste d'attributs*>
FROM <*liste de relations*>
*[***WHERE** <*condition*>*]*
*[***GROUP BY** <*attribut*>*]*
*[***HAVING** <*condition*>*]*
*[***ORDER BY** <*liste d'attributs*>*]*

Relations :

création1 : **CREATE TABLE** <*ident_relation*> **(** <*définition*> *{* **,** <*définition*> *}* **)**

- définition d'attributs :
  <*définition*> ::= <*ident_attribut*> <*type*> *[* **NULL / NOT NULL***]*
- définition de contraintes :
  clé de la relation :
  <*définition*> ::= **CONSTRAINT** <*ident_contrainte*> **PRIMARY KEY (**<*liste d'attributs*>**)**
  domaine de validité :
  <*définition*> ::= **CONSTRAINT** <*ident_contrainte*> **CHECK** <condition>
  contrainte d'intégrité référentielle :
  <*définition*> ::= **CONSTRAINT** <*ident_contrainte*> **FOREIGN KEY (**<*attribut local*>**)**
  REFERENCES <*relation de référence*>**(**<*attribut de référence*>**)**
  attribut à valeur unique :
  <*définition*> :: = **CONSTRAINT** <*ident_contrainte*> **UNIQUE (**<*liste_d'attributs*>**)**

création2 : **CREATE TABLE** <*ident_relation*> **AS** <*requête SELECT*>

suppression : **DROP TABLE** <*ident_relation*>

modification :

- ajout d'un attribut : **ALTER TABLE** <*ident_relation*> **ADD COLUMN** <*def_attribut*>
- suppression d'un attribut :**ALTER TABLE** <*ident_relation*> **DROP COLUMN** <*attribut*>
- modification d'un attribut : **ALTER TABLE** <*ident_relation*> **ALTER COLUMN**
  SET DEFAULT <valeur> **| DROP DEFAULT | DROP DOMAIN**
- ajout d'une contrainte : **ALTER TABLE** <*ident_relation*> **ADD CONSTRAINT** <*def_contrainte*>
- suppression d'une contrainte : **ALTER TABLE** <*ident_relation*>
  DROP CONSTRAINT <*ident_contrainte*>

Mises-à-jour :

ajouter un tuple : **INSERT INTO** <*ident_relation*> *[***(**<*liste d'attribut*>**)***]* **VALUES (**<*liste de valeurs*>**)**
supprimer des tuples : **DELETE FROM** <*ident_relation*> *[* **WHERE** <*condition*>*]*
modifier des tuples : **UPDATE** <*ident_relation*>
SET <*ident_attribut*> = <*expression*> *{* **,** <*ident_attribut*> = <*expression*> *}*
*[***WHERE** <*condition*>*]*
copier des tuples : **INSERT INTO** <*ident_relation*> <*requête SELECT*>

Privilèges :

accorder : **GRANT** <*privileges*> **ON** <*relation*> **TO** <*utilisateur*> *[***WITH GRANT OPTION***]***
retirer : **REVOKE** <*privileges*> **ON** <*relation*> **FROM** <*utilisateur*>

Vues :

création : **CREATE VIEW** <*ident_vue*> *[***(**<*liste d'attributs*>**)***]* **AS** <*requête*> *[***WITH CHECK OPTION***]***
suppression : **DROP VIEW** <*ident_vue*>