Chapitre 7
Le langage SQL

- le langage SQL est une norme de l'ANSI (SQL-92) et de l'ISO (SQL2) pour les SGBD relationnels ;
- il y a une nouvelle norme SQL:1999 prend en compte les extensions objets ainsi que les donnée géographiques, temps réels, séries temporelles, multimedia, OLAP, données et routines externe.
- il existe 4 niveaux de conformité dans la norme SQL-92: 1 Entry, 2 Intermediate, and Full ;
- les fabricants de SGBD essaient de se conformer à cette norme, mais ils le font chacun à des niveaux différents de conformité ;
- la version Oracle 8 est conforme au niveau Entry et contient quelques éléments des niveaux Intermediate et Full ;
- dans le cours, nous utilisons Oracle ; il est possible que des requêtes SQL exécutant sur d'autres SGBD (comme Microsoft Access) ne puisse être exécutées sous Oracle, et vice-versa.
- SQL est principalement un langage de type déclaratif.
  - On spécifie ce que l'on veut obtenir ou faire et c'est la machine qui décide comment elle doit l'exécuter.
  - Ce n'est pas un vrai langage de programmation

7.1 Structure du langage

7.1.1 SQL déclaratif

- DDL (*Data Definition Language*). Le DDL réunit les commandes SQL permettant de créer (CREATE), de modifier (ALTER). ou de supprimer (DROP) les objets de la base.
- DML (*Data Manipulation Language*). Le DML réunit les commandes permettant d'ajouter (INSERT), de modifier (UPDATE), de supprimer (DELETE) ou d'extraire (SELECT) des données.
- DCL (*Data Control Language*). Le DCL réunit les ordres SQL permettant de définir les privilèges afférents aux utilisateurs (GRANT, REVOKE).
- TCL (Transaction Control Language) réunit les commandes SQL permettant de contrôler la bonne exécution des transactions précédentes à l'aide des commandes (SET TRANSACTION, COMMIT, ROLLBACK).

7.1.2 SQL procédural

- PSM (*Persistent Stored Module*). Le PSM concerne les fonctions, procédures, méthodes et déclencheurs (triggers) en tant qu'objets de la base (donc stokés dans la base et par conséquent persistants).
- CLI (Call Level Interface). Il s'agit des API destinées à piloter des objets encapsulant des données dans des langages hôtes par l'intermédiaire, la plupart du temps, d'un middleware (BDE, dbExpress de Borland, OBDC, ADO, OleDB de Microsoft, JDBC …).
- Embedded SQL. Ce module gère le lancement d'ordre SQL depuis un langage hôte et la récupération des données dans un programme via l'utilisation de CURSOR.

7.2 Langage de définition des données

7.2.1 Les noms des identificateurs dans les commandes SQL

- Un identificateur d'objet (table, colonne, contrainte, vue...) doit avoir les caractéristiques suivantes :
  - ne pas dépasser 128 caractères
  - commencer par une lettre
  - comprendre uniquement les caractères suivants ['A' .. 'Z'] U ['a' .. 'z'] U ['0' .. '1'] U ['_']
  - un identificateur ne peut pas être un mot réservé de SQL sauf à être utilisé avec des guillemets
  - être insensible à la casse

VALABLE	**INTERDIT**
T_CLIENT	01_INFORMATIQUE
xyz	_XyZ_
SELECTION	SELECT
"SELECT"	MOTDEPLUSDE128CARACTERES...
CLI_NUM	CLI#
NUM_CLI	#CLI

7.2.2 Table

- une table est un ensemble de tuples ;
- on utilise aussi *relation* comme synonyme de table,
  et *ligne* ou *enregistrement* comme synonymes de tuple ;
- tous les tuples d'une table ont le même format ;
  ce format est défini par un ensemble d'attributs ;
- on peut représenter graphiquement une table par une matrice ou les colonnes sont les attributs ; la première ligne comporte les noms des attributs, et les autres lignes représentent les tuples ; l'ordre d'énumération des tuples ou des attributs n'a pas d'importance ;
- la définition en SQL d'une table comporte les éléments suivants :
  - son nom
  - ses attributs
  - ses contraintes d'intégrité ; il y a trois types de contraintes d'intégrité :
    *clé primaire ;
    *clé unique ;
    *clé étrangère.

7.2.3 Types en SQL

- La norme ANSI SQL définit des types de données. Il appartient aux fabricant de bases de données (comme Oracle, IBM, Microsoft) de suivre cette norme. Le tableau ci-dessous donne les types ANSI SQL et les types spécifiques à Oracle. Par souci de portabilité, il est préférable d'utiliser les type ANSI supportés par Oracle ; ils sont en caractères gras dans la colonne ANSI SQL.

Type de données ANSI SQL	Type correspondant spécifique à Oracle
CHARACTER (n), **CHAR(n)**	CHAR(n)
NUMERIC(p, s) , DECIMAL (p, s), DEC (p, a)	NUMBER (p, s)
INTEGER, INT, SMALLINT	NUMBER (p)
FLOAT (p), REAL, DOUBLE PRECISION	FLOAT(p) (max 126 bits)
CHARACTER VARYING(n), **VARCHAR(n)**	VARCHAR2(n)
DATE	DATE
TIME	DATE
TIMESTAMP	DATE

- `CHAR(n)`
  - représente une chaîne de caractères de longueur fixe n ;
  - une chaîne de caractères est mise entre des apostrophes simples (') ;
  - pour inclure un ' dans une chaîne de caractères, on utilise deux '.
    exemple : La chaîne `'abc12'` est une valeur du type `CHAR(5)`.
    La chaîne `'ab''12'` contient un `'` au milieu.

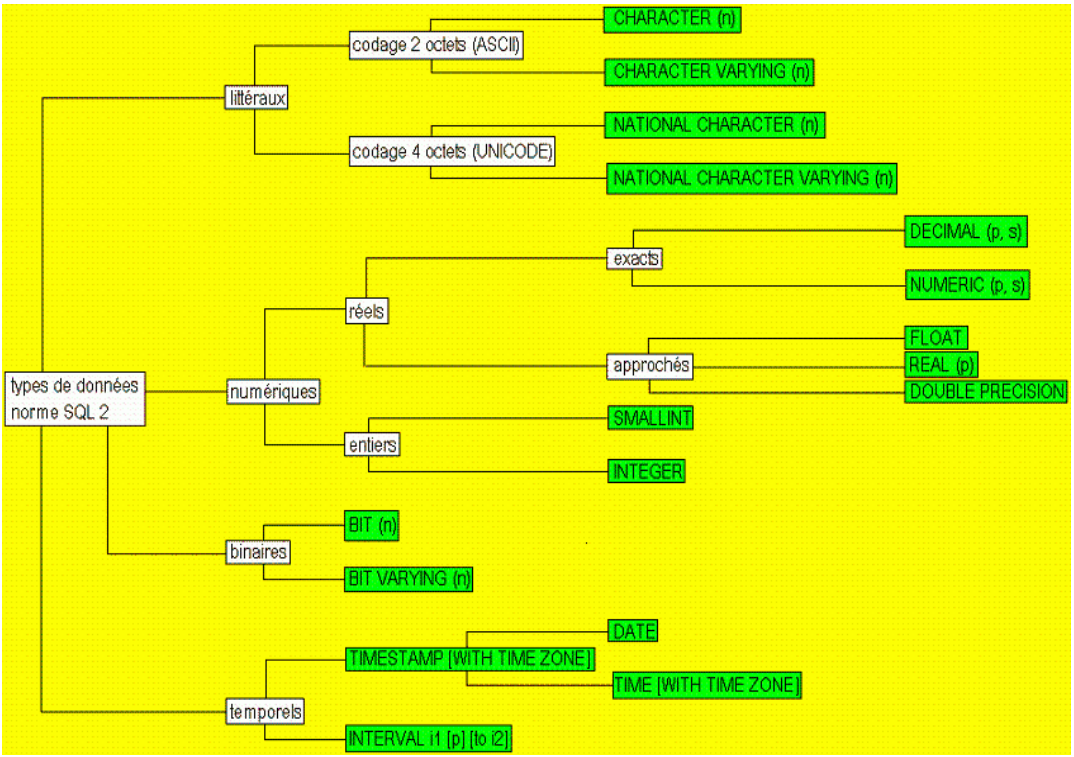
- `NUMERIC(p,s)`
  - p indique le nombre total de chiffres stockés pour un nombre ; la valeur de p doit être entre 1 et 38 ;
  - s>0 indique le nombre total de chiffres après la virgule ;
  - s<0 indique un arrondissement du nombre de s chiffre avant la virgule ;
    exemple :
    - `NUMERIC(5,2)` peut contenir une valeur comportant 5 chiffres, dont 3 avant la virgule (soit 5-2) et 2 chiffres après la virgule ; exemple de valeur : 123,45
    - `NUMERIC(2,5)` peut contenir une valeur comportant de 2 chiffres ; comme p < s dans ce cas, on a seulement des chiffres après la virgule ; les 3 premiers (soit 5-2) ont comme valeur 0 ; exemple de valeur : 0,00012
    - `NUMERIC(5,-2)` peut contenir une valeur comportant de 7 chiffres avant la virgule (soit 5 - -2), mais seulement les 5 premiers chiffres sont stockés, les 2 derniers sont toujours 0 ; il n'y aucun chiffre après la virgule ; exemple de valeur : 1234500 ;
    lorsqu'on stocke une valeur dans la base de données, elle est toujours arrondie à deux chiffres avant la virgule ;
    exemple 1234550 est stockée comme 1234600 et 1234549 est stocké comme 1234500.
    - `NUMERIC(2,-5)` peut contenir une valeur comportant de 7 chiffres avant la virgule (soit 2 - -5), mais seulement les 2 premiers chiffres sont stockés, les 5 derniers sont toujours 0 ; il n'y aucun chiffre après la virgule ; exemple de valeur : 1200000 ;

• REAL
  - permet de stocker un nombre en virgule flottant (c-a-d une valeur représentée par une mantisse et un exposant)

• VARCHAR(n)
  - permet de stocker une chaîne de caractères de longueur maximale n ;
  - par rapport à CHAR(n), permet de mieux gérer l'espace disque si les chaînes de caractères ne sont pas toujours de longueur n ;

• DATE ET HEURE
- le type DATE de la norme SQL2 comprend seulement une date en format `YYYY-MM-DD` ;
- le type TIME de la norme SQL2 comprend seulement une heure en format `HH:MM:SS` ;
- le type TIMESTAMP de la norme SQL2 comprend la date et l'heure
- en format `YYYY-MM-DD HH:MM:SS.ffffff`, où ffffff représente une fraction de seconde ;
- le type DATE d'Oracle est presque équivalent au type TIMESTAMP de SQL2 ; il comprend la date et l'heure `YYYY-MM-DD HH:MM:SS`, mais sans fraction de seconde ; la valeur d'une date varie entre le 1er janvier 4712 avant J-C et 31 décembre 4712 après J-C.
- la date sous Oracle est affichée selon le format donné par le paramètre global NLS_DATE_FORMAT ;
- il existe plusieurs fonctions pour manipuler des dates en Oracle ; **exemple** : `to_date('29-02-2000','DD-MM-YYYY')` retourne la valeur de la date 29 février 2000 selon le type DATE d'Oracle.

• Vue générale des types en SQL



7.2.4 Définition des tables

Notation utilisée pour décrire la syntaxe du langage SQL

- `MOT_CLÉ` : mot réservé du langage SQL ;
- <symbole terminal> : peut être remplacé par un identificateur ou une constante (nombre, chaîne de caractère, etc) ;
- «symbole non terminal» : doit être remplacé par sa définition ;
- ::= : définition d'un symbole non terminal ;
- "{" et "}" : équivalent des parenthèses en mathématiques ;
- + : une ou plusieurs occurrences ;
- * : zéro ou plusieurs occurrences ;
- [élément optionnel]
- | : choix entre plusieurs options

7.2.4.1 Syntaxe générale

```
«creation-table» ::=
  CREATE TABLE < nom-table > (
      «liste-attributs»
      [ , «liste-contraintes» ]
    )
```

7.2.4.2 Définition des attributs

```
«liste-attributs» ::=
  «attributs» {, «attributs» }*
«attribut» ::=
  < nom-attribut > < type > [DEFAULT < expression > ]
  [NOT NULL] [CHECK ( < condition > )]
```

• la valeur par défaut est utilisée si la valeur de l'attribut n'est pas spécifiée lors de la création d'un tuple ;
• la condition doit être vérifiée lors de la création ou de la mise à jour d'un tuple ;
• `NOT NULL` : la valeur de l'attribut ne peut contenir la valeur spéciale `NULL` ;
• exemples de condition
 -`< nom-attribut > { = | > | >= | … } « expression »`
 -`< nom-attribut >[IN] (« liste-valeurs »)`
 -`« condition » { [AND] | [OR] } « condition »`
 -`NOT « condition »`
 - plusieurs autres (voir manuel Oracle).

7.2.4.3 Définition des contraintes

```
«liste-contraintes» ::=
  « contrainte » {, « contrainte » }*
«contrainte» ::=
  «cle-primaire» | «cle-unique» | «cle-etrangere»
```

7.2.4.3.1 Clé primaire

```
«cle-primaire» ::=
  CONSTRAINT <nom-contrainte> PRIMARY KEY
    («liste-noms-attribut»)
«liste-noms-attribut»» ::=
  <nom-attribut> {, <nom-attribut> }*
```

• `<nom-attribut>` sont ceux définis dans la table
• il ne peut y avoir deux tuples avec les mêmes valeurs pour les attributs de la clé primaire ;
• on peut définit une seule clé primaire pour une table ;
• la valeur d'un attribut d'une clé primaire ne peut être NULL dans un tuple.

7.2.4.3.2 Clé unique

```
«cle-unique» ::=
  CONSTRAINT <nom-contrainte> UNIQUE ( «liste-noms-attribut» )
«liste-noms-attribut» ::=
  <nom-attribut> {, <nom-attribut> }*
```

- `<nom-attribut>` sont ceux définis dans la table
- il ne peut y avoir deux tuples dans la table avec les mêmes valeurs pour les attributs de la clé unique ;
- on peut définir plusieurs clés uniques pour une table ;
- un attribut d'une clé unique peut être `NULL`, toutefois, la combinaison de tous les attributs non `NULL` doit être unique.

7.2.4.3.3 Clé étrangère

```
«cle-étrangère» ::=
  CONSTRAINT < nom-contrainte >
  FOREIGN KEY («liste-attributs»)
  REFERENCES < nom-table-referencee >
  [ («liste-nom-attributs») ]
  [ON DELETE CASCADE]
«liste-noms-attribut»» ::=
  <nom-attribut> {, <nom-attribut> }*
```

- `<nom-attribut>` sont ceux définis dans la table
- pour chaque tuple de la table dont les attributs de clé étrangère sont tous différents de `[NULL]`, il doit exister un tuple dans `< nom-table-référencée >` avec la même valeur pour `«liste-attributs»` ;
- `ON DELETE CASCADE` : si un tuple dans `< nom-table-référencée >` est supprimé, tous les tuples de la table qui le référence sont aussi supprimés.
  Il y a plusieurs autres triggers possibles prévu par le langage SQL
  [ON DELETE SET NULL],
  [ON DELETE SET DEFAULT],
  [ON UPDATE CASCADE]
  …
- Les clauses [ON DELETE …] et [ON UPDATE …] servent à maintenir l'intégrité référentielle

•

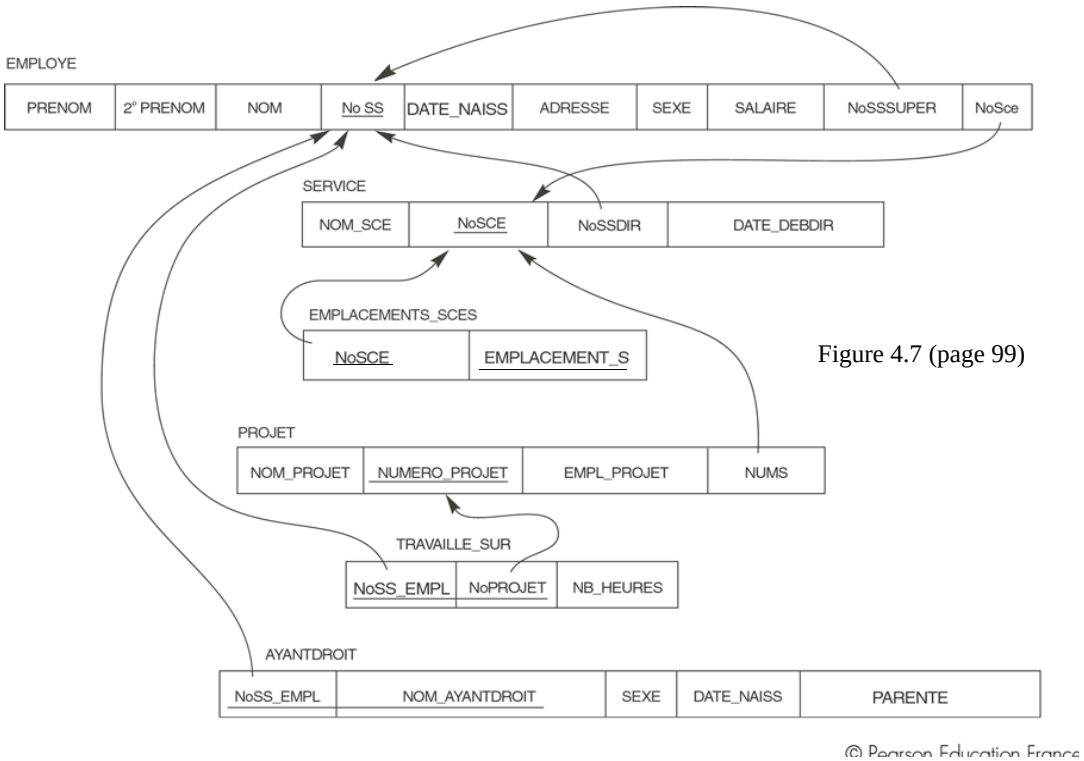


Figure 4.7 (page 99)


```
CREATE TABLE employe
(prenom         VARCHAR(15)     NOT NULL,
 prenom2        CHAR,
 nom            VARCHAR(15)     NOT NULL
                CHECK(UPPER(SUBSTR(nom,1,1)) = SUBSTR(nom,1,1)),
 noSs           CHAR(13)        NOT NULL
                CHECK (noSs >= 1000000000000) ,
 dateNaiss      DATE,
 adresse        VARCHAR(40),
 sexe           CHAR    CHECK (sexe IN ('H', 'F')),
 salaire        DECIMAL(10,2),
 noSssuper      CHAR(13),
 noSce          INT             NOT NULL,
 CONSTRAINT cleEmploye          PRIMARY KEY (noSs)
 -- pour traitement en lot
 -- CONSTRAINT refEmployeNoSssuper
                FOREIGN KEY (noSssuper)
                REFERENCES employe
 -- pour éviter la recursion infinie
 -- CONSTRAINT refEmployeServiceNoSce
                FOREIGN KEY (noSce)
                REFERENCES service
 );
```

```
CREATE TABLE service
(nomSce         VARCHAR(15)     NOT NULL,
 noSce          INT             NOT NULL,
 noSsdir        CHAR(13)        NOT NULL,
 date_debdir    DATE,
 CONSTRAINT cleService
                PRIMARY KEY (noSce),
 CONSTRAINT cleCandidateService
                UNIQUE (nomSce),
 CONSTRAINT refServiceEmployeNoSs
                FOREIGN KEY (noSsdir)
                REFERENCES employe(noSs)
                ON DELETE CASCADE
 );
```



```
CREATE TABLE emplacementsSces
(noSce          INT             NOT NULL,
 emplacements   VARCHAR(15)     NOT NULL,
 CONSTRAINT cleEmplacementsSces
                PRIMARY KEY (noSce, emplacements),
 CONSTRAINT refEmpl_ScesServ_NoSce
                FOREIGN KEY (noSce)
                REFERENCES service
                ON DELETE SET NULL
 );
```

```
CREATE TABLE projet
(nomProjet      VARCHAR(15)     NOT NULL,
 numeroProjet   INT NOT NULL,
 emplProjet     VARCHAR(15),
 nums                   INT     NOT NULL,
 CONSTRAINT cleProjet
                PRIMARY KEY (numeroProjet),
 CONSTRAINT cleCandidateProjet
                UNIQUE (nomProjet),
 CONSTRAINT refProjetServiceNoSce
                FOREIGN KEY (nums)
                REFERENCES service(noSce)
 );
```



```
CREATE TABLE travailleSur
(noSsEmpl       CHAR(13)        NOT NULL,
 noProjet       INT             NOT NULL,
 nbHeures       DECIMAL(3,1)    NOT NULL,
 CONSTRAINT cleTrav_Sur
                PRIMARY KEY (noSsEmpl, noProjet),
 CONSTRAINT refTrav_SurEmployeNoSs
                FOREIGN KEY (noSsEmpl)
                REFERENCES employe(noSs),
 CONSTRAINT refTravSurProjNum_Proj
                FOREIGN KEY (noProjet)
                REFERENCES projet(numeroProjet)
 );
```

```
CREATE TABLE ayantDroit
(noSsEmpl       CHAR(13)        NOT NULL,
 nomAyantDroit  VARCHAR(15)     NOT NULL,
 sexe           CHAR            CHECK (sexe IN ('H', 'F')),
 dateNaiss      DATE,
 parente        VARCHAR(8),
 CONSTRAINT cleAyantDroit
                PRIMARY KEY (noSsEmpl, nomAyantDroit),
 CONSTRAINT refAyantDroitEmpl_NoSs
                FOREIGN KEY (noSsEmpl)
                REFERENCES employe(noSs)
 );
```


7.2.5 Modification des tables

```
ALTER TABLE < nom-table >  {
  «ajout-attribut» |
  «modification-attribut» |
  «suppression-attribut» |
  «ajout-contrainte» |
  «suppression-contrainte» }
```

7.2.5.1 Ajout d'attributs

```
«ajout-attribut» ::= ADD ( «liste-attributs» )
```

- ajoute les attributs de la liste à la table ;
- pour chaque ligne les attributs ont la valeurs `null` ou la nouvelle valeur par defaut s'il y a lieu ;
- si la table contient déjà des lignes, on ne peut ajouter un attribut avec la contrainte `not null` évidemment ;

```
ALTER TABLE employe ADD (
 plusHautDiplomz        VARCHAR(3)      DEFAULT 'B. Sc.',
 idRAMQ                 CHAR(12)        DEFAULT 'AAAA99999999' ) ;
```


7.2.5.2 Modification d'attributs

```
«modification-attribut»::= MODIFY ( «liste-attributs» )
```

- modifie le type, la valeur par défaut ou l'option `NULL` or `NOT NULL` des attributs de la liste ;
- on spécifie seulement les parties à modifier ;
- pour modifier le type, la valeur de chaque attribut doit être `NULL` pour tous les tuples de la table ;
- pour spécifier `NOT NULL`, il faut que l'attribut satisfasse déjà cette condition.
- Oracle accepte d'allonger la dimension d'une colonne sans que celle-ci soit vide

```
ALTER TABLE employe MODIFY (
 plusHautDiplome VARCHAR(5) ,
 idRAMQ  DEFAULT 'ZZZZ99999999' ) ;
```

7.2.5.3 Suppression d'attributs

«suppression-attribut» ::= DROP («liste-noms-attribut»)

• «liste-noms-attribut» ::= voir acetate 17 ou 18

• supprime les attributs ;

```
ALTER TABLE employe DROP (
                plusHautDiplome ,
                idRAMQ);
```

• Dans les versions récentes de SQL, la syntaxe est

«suppression-attribut» ::= DROP [COLUMN] <nom-attribut>

- pour notre version de oracle le mot cle COLUMN est obligatoire

```
ALTER TABLE employe DROP COLUMN plusHautDiplome ;
ALTER TABLE employe DROP COLUMN idRAMQ ;
```


7.2.5.4 Ajout de contraintes

«ajout-contrainte» ::= ADD CONSTRAINT «contrainte»

• ajoute une contrainte ;
• les tuples de la table doivent satisfaire la contrainte.

```
ALTER TABLE employe
 ADD CONSTRAINT refEmpl_Serv_NoSce      FOREIGN KEY (noSce)
                                        REFERENCES service ;


ALTER TABLE employe
 ADD CONSTRAINT refEmployeNoSsuper      FOREIGN KEY (noSsuper)
                                        REFERENCES employe ;
```

7.2.5.5 Suppression de contraintes

```
«suppression-contrainte» ::= DROP CONSTRAINT < nom-contrainte > [CASCADE]
```

• supprime la contrainte ;
• CASCADE : supprime aussi toutes les contraintes qui dépendent de la contrainte supprimée.

```
ALTER TABLE employe DROP CONSTRAINT refEmpl_Serv_NoSce ;
ALTER TABLE employe DROP CONSTRAINT refEmployeNoSssuper ;
```


7.2.6 Suppression des tables

```
DROP TABLE < nom-table >  [ CASCADE CONSTRAINTS ]
```

• supprime la table

```
DROP TABLE employe CASCADE CONSTRAINTS ;
DROP TABLE service CASCADE CONSTRAINTS ;
DROP TABLE emplacementsSces CASCADE CONSTRAINTS ;
DROP TABLE projet CASCADE CONSTRAINTS ;
DROP TABLE travailleSur CASCADE CONSTRAINTS ;
DROP TABLE ayantDroit CASCADE CONSTRAINTS ;
```

- Vous permet de tout effacer

```
select * from tab
```

- Vous permet de connaître tous les tables dans votre BD

7.3 Langage de manipulation des données

7.3.1 Insert

```
INSERT INTO < nom-table >
  [ ( «liste-noms-attribut» ) ]
  { VALUES ( «liste-expressions» ) | «select» }
```

• La variante du INSERT avec SELECT sera vu avec la commande SELECT

```
INSERT INTO employe VALUES
('Etienne', 'H', 'Borgue', '1371184245901', '1937-11-10', '45 avenue des
Settons, Les Ulis', 'H', 55000, null, 1 ) ;


INSERT INTO employe (
  prenom,  nom    ,  noSs                       , noSce)
  VALUES  (
 'Andre', 'Mayers', rpad ('123456789', 13, '0') , 5    ) ;


INSERT INTO service VALUES ('Enseignement', 10, '1234567890123', sysdate -
4 * 365) ;
```


7.3.2 Update

```
UPDATE < nom-table >
  SET { «liste-affectation» | «affectation-select» }
  [WHERE «condition»]

«liste-affectation» ::=
  «affectation» [,«affectation»*]

«affectation» ::=
  < nom-attribut > = «expression»

«affectation-select» ::=
( «liste-noms-attribut] ) = «select»
```

• «expression» peut être un énoncé select.
• encore une fois les variantes utilisant la commande SELECT seront vus plus tard

```
UPDATE employe
 SET salaire = salaire * 1.1
 WHERE salaire < 40000 ;


UPDATE employe
 SET salaire = salaire * 0.8915
 WHERE nom = 'Borgue' ;
```

7.3.3 Delete

```
DELETE FROM < nom-table >
  [WHERE «condition» ]
```

• si `WHERE` n'est pas spécifié, l'énoncé `DELETE` supprime tous les tuples.

```
DELETE FROM employe
 WHERE prenom = 'Andre';
```

```
DELETE FROM service
 WHERE to_number(to_char (dateDebDir, 'YYYY')) > 2000 ;
```


7.3.4 Select

• Le SELECT est la commande de base du SQL destinée à extraire des données d'une base ou calculer de nouvelles données à partir d'existantes

7.3.4.1 Syntaxe générale

```
«enonce-select-base»::=
  SELECT «liste-expressions-colonne»
    FROM «liste-expressions-table»
    [WHERE «condition-tuple»]
    [GROUP BY «liste-expressions-colonne»][HAVING «condition-groupe»}]
    [ORDER BY «liste-expressions-colonne»]
```

```
«enonce-select-compose» ::=
  «enonce-select-base»
  {UNION [ALL]| [INTERSECT]|[MINUS]}
  «enonce-select-compose»
```

Les colonnes à afficher

```
SELECT [DISTINCT] «liste-expressions-colonne»
```

```
«liste_expression-colonne» ::=
    «expression-colonne» [,«expression-colonne»*]
```

```
«expression-colonne» ::= «expression» [AS <nom colonne>]
```

- juste après le mot clef SELECT, on précise les colonnes qui doivent être présentées dans la réponse, ces colonnes «`expression`» peuvent être des attributs de tables existentes ou des expressions utilisant celles-ci (`voir “Expressions” on page 38`).
- L'opérateur AS
  sert à donner un nom à de nouvelles colonnes créées par la requête.
- L'opérateur || (double barre verticale) permet de concaténer des champs de type caractères.
  `SELECT prenom || ' ' || nom  AS NOM`
- L'opérateur * (étoile) récupère toutes les colonnes de la table précisée dans la clause FROM .
- L'opérateur DISTINCT (ou ALL par défaut)
  le mot clef DISTINCT permet d‘éliminer les doublons dans la réponse.
- `<attribut>` Lorsque les noms de colonne proviennent de plusieurs tables distinctes, on peut utiliser l’opérateur “.” et des <alias> pour distinguer ces tables comme `t1` et `t2` dans l’exemple suivant

```
SELECT  COUNT(DISTINCT t1.noSs)         AS "nb employés",
        COUNT(DISTINCT t2.emplacements) AS "nb emplacements"
FROM    employe t1, emplacementsSces t2
```


7.3.4.2 Expressions

- expression élémentaire : nom d'attribut, fonction, constante ;
- expression composée : opérateur appliqué sur expressions élémentaires ou composées ;
- alias : renommer une expression ou une relation ;
- `*, R.*` : retourne tous les attributs de la table
  -

La sélection des tables

```
FROM «liste-expressions-table»
```

Il s'agit de choisir ou construire les tables dans lesquelles seront extraites les données

```
«liste_expressions-table» ::=
    «expressions-table» [,«expressions-table»*]
«expressions-table» ::=
    {{[<schema>.]{<table> | <vue>}} | ( «enonce-select-base» ) } [alias]
```

- <schema> est le nom d'une base de donnée accessible, par exemple celle d'un de vos collègues.
- <vue> est le nom d'une table virtuelle, nous verrons ce concept plus tard.
- «enonce-select-base», en effet la table peut être construites, i.e. le résultat d'un autre énoncé select.

exemple :

```
SELECT MAX(COUCHAGE) AS MAX_COUCHAGE
FROM (SELECT SUM(CHB_COUCHAGE) AS COUCHAGE
      FROM T_CHAMBRE
      GROUP BY CHB_ETAGE)
```


Le choix des lignes

```
[WHERE «condition-tuple»]
```

```
«condition-tuple» ::=  <condition> | «jointure-externe»
«jointure-externe» ::=
  <table1> . <attribut> {{= <table2> . <attribut> (+)} |
                        {(+) = <table2> . <attribut>} }
```

- <condition> prédicat qui doit être vrai pour la ligne pour que les valeurs de ses attributs soient prises en compte.
- expression appliquée à un subselect (clause WHERE)
  - `<expr> in (<select>)` : vrai si <expr> est un élément de l'ensemble des tuples retournés par le select ;
  - `<expr> not in (<select>)` : vrai si <expr> n'appartient pas aux tuples retournés par le select ;
  - `<expr> >= any (<select>)` : vrai s'il existe un tuple t ∈ <select> tel que <expr> >= t ;
  - `<expr> >= all (<select>)` : vrai si pour tous les tuple t ∈ <select>, <expr> >= t ;
  - `exists (<select>)` : vrai si l'ensemble des tuples du select est non vide ;
  - `not exists (<select>)` : vrai si l'ensemble des tuples du select est vide.
- La jointure externe permet d'extraire toutes les lignes d'une table en correspondance ou non avec l'autre table à laquelle elle est jointe.

Le choix des lignes

WHERE exemple

```
SELECT t1.tata “tata”, t2.toto “toto”
FROM db1 t1, db2 t2
WHERE t1.no = t2.no (+);
```

tata	toto
asdf	
adf	sg
fedf	fgdg
uytgyu	

OUTER JOIN EXEMPLE

```
SQL> SELECT * FROM T1 ;
```

A	B	C
1	**2**	3
4	*5*	*4*
5	*4*	

```
SQL> SELECT * FROM T2 ;
```

A	B	D
1	1	7
1	**2**	8
1	1	9

```
SQL> SELECT * FROM  T1, T2 WHERE T1.A = T2.A  AND T1.B = T2.B ;
```

A	B	C	A	B	D
1	**2**	3	**1**	**2**	8

```
SQL> SELECT * FROM  T1, T2 WHERE T1.A = T2.A (+) AND T1.B = T2.B (+);
```

A	B	C	A	B	D
1	**2**	3	**1**	**2**	8
4	*5*	*4*			
5	*4*				

L’aggrégation de plusieurs lignes

[GROUP BY «liste-expressions-colonne»][HAVING «condition-groupe»]

- fonctions qui prennent compte de l'ensemble des tuples d'un select ;
  `count(«expr»)`,
  `sum(«expr»)`,
  `avg(«expr»)`,
  `min(«expr»)`,
  `max(«expr»)`, etc ;

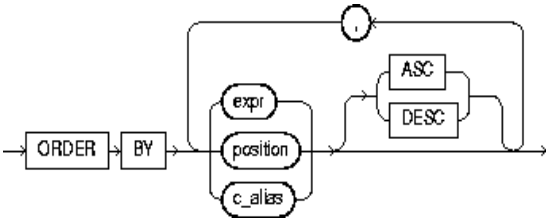
- `count`
  - `count(*)` : compte aussi les valeurs `NULL` ;
  - `count(attribut)` : compte seulement les valeurs non `NULL` ;
  - `count(distinct attribut)` : une valeur est comptée une seule fois, même si plusieurs tuples ont cette valeur ;

```
SELECT nom , COUNT (cours) "Nombre de cours"
 FROM etudiant
 GROUP BY nom ;
```

Quel est le nombre de cours par étudiant ?

```
Nom              Nombre de cours
-------------- -----------------
      André                   1
      Jacques                 4
      Lise                    4
```


L’ordre des lignes de la table affichée



- Cette clause permet de définir le tri des colonnes de la réponse, soit en donnant son expression, soit en précisant son n° d'ordre dans l'énumération qui suit le mot clef SELECT ou soit en donnant le nom littéral de la colonne.
- ASC spécifie l’ordre ascendant et DESC l’ordre descendant du tri. L'ordre est ascendant par défaut.
- Souvent, le fait de placer DISTINCT suffit, en général, à établir un tri puisque le moteur doit se livrer à une comparaison des lignes mais ce mécanisme n'est pas garanti car ce tri s’effectue dans un ordre non contrôlable qui peut varier d’un serveur à l’autre.

```
SELECT     CLI_NOM, CLI_PRENOM
FROM       T_CLIENT
ORDER BY   CLI_NOM, CLI_PRENOM
```

ou

```
SELECT     CLI_NOM, CLI_PRENOM
FROM       T_CLIENT
ORDER BY   1, 2
```

7.3.4.3 Sémantique

Le résultat d'un énoncé `SELECT` est égal au résultat des opérations suivantes. *Note: chaque SGBD utilise un algorithme propre pour exécuter un énoncé* `SELECT`*. Toutefois, le résultat est le même que celui donné par la procédure ci-dessous.*

1. évalue le produit cartésien des relations du `FROM` ;
1. sélectionne les tuples satisfaisant la clause `WHERE` ;
2. tuples regroupés selon la clause `GROUP BY` ;
3. sélectionne les groupes selon la condition `HAVING` ;
4. évalue les expressions du `SELECT` ;
5. élimine les doublons si clause `DISTINCT` ;
6. évalue l'union, l'intersection, ou la différence des selects (si nécessaire) {prochain cours};
7. trie les tuples selon la clause `ORDER BY`.

7.3.4.4 L'exemple de la bibliothèque

Voici le schéma relationnel.

1. editeur

idediteur	nom	pays

8. auteur

idauteur	nom

9. livre

idlivre	titre	idauteur	idediteur	dateAcquisition	prix

10.membre

idmembre	nom	telephone	limitePret

11.pret

idmembre	idlivre	datePret

12.reservation

idreservation	idmembre	idlivre	dateReservation

Voici les énoncés de création des tables.

```
-----------------------------------------
-- une ligne de commentaire commence par deux '--'
-- Exemple de la bibliotheque
-- Marc Frappier, Universite de Sherbrooke
-- 2001-01-08
-- devrait être fait avant celui sur les entreprises (André)
-----------------------------------------
  CREATE TABLE auteur (
  idAuteur        numeric(3) ,
  nom             varchar(12) NOT NULL,
  CONSTRAINT cleAuteur PRIMARY KEY (idAuteur)
  ) ;
```



```
  CREATE TABLE livre (
  idLivre         numeric(3) ,
  titre           varchar(20) NOT NULL
                  check(upper(substr(titre,1,1)) = substr(titre,1,1)),
  idAuteur        numeric(3) NOT NULL ,
  idEditeur       numeric(3) NOT NULL ,
  dateAcquisition date ,
  prix            numeric(7,2) ,
  CONSTRAINT cleLivre PRIMARY KEY (idLivre) ,
  CONSTRAINT cleCandidateTitreAuteur UNIQUE (titre,idAuteur) ,
  CONSTRAINT cleCandidateTitreEditeur UNIQUE (titre,idEditeur) ,
  CONSTRAINT refLivreAuteur FOREIGN KEY (idAuteur) REFERENCES auteur ,
  CONSTRAINT refLivreEditeur FOREIGN KEY (idEditeur) REFERENCES editeur
  ) ;
```

```
CREATE TABLE membre (
idMembre        numeric(3) ,
nom             varchar(10) NOT NULL ,
telephone       numeric(10) check(telephone >= 8190000000 and
                                  telephone <=8199999999) ,
limitePret      numeric(2) check(limitePret > 0 and limitePret <= 10) ,
CONSTRAINT cleMembre PRIMARY KEY (idMembre)
) ;
```



```
CREATE TABLE pret (
idMembre        numeric(3) ,
idLivre         numeric(3) ,
datePret        date NOT NULL ,
CONSTRAINT clePret PRIMARY KEY (idMembre,idLivre,datePret) ,
CONSTRAINT refPretMembre FOREIGN KEY (idMembre) REFERENCES membre ,
CONSTRAINT refPretLivre FOREIGN KEY (idLivre) REFERENCES livre
) ;
```

```
CREATE TABLE reservation (
idReservation   numeric(3) ,
idMembre        numeric(3) ,
idLivre         numeric(3) ,
dateReservation date NOT NULL ,
CONSTRAINT cleReservation PRIMARY KEY (idReservation) ,
CONSTRAINT cleCandidateReservation UNIQUE (idMembre,idLivre) ,
CONSTRAINT refReservationMembre FOREIGN KEY (idMembre) REFERENCES membre ,
CONSTRAINT refReservationLivre FOREIGN KEY (idLivre) REFERENCES livre
) ;
```


7.3.4.5 Quelques exemples

1. **Sélection de colonnes d'une table** :

   Afficher la liste des livres avec leur titre.

```
select  idlivre, titre
from    livre
```

```
  IDLIVRE TITRE
--------- --------------------
        1 Concep + arch des bd
        2 Fdmts des bds
        3 Synth et ex corr
        4 Intro aux bd
        5 Bd objd et relat
        6 Db Sys:Appl-Ori App
        7 Oracle:Compl Ref
        8 Princ of Db + KB I
        9 Princ of Db + KB II
       10 1 Course in Db Sys
       11 Db management sys
       12 Db Sys Concepts

12 rows selected.
```

2. **Sélection de lignes d'une table avec une condition élémentaire** :

   Afficher la liste des livres avec leur titre pour l'auteur idauteur = 3.

```
select  idlivre, titre
from    livre
where   idauteur = 3
```


3. **Sélection de lignes d'une table avec une condition composée** :

   Afficher la liste des livres avec leur titre pour les auteurs d'idauteur = 3 ou 10.

```
select  idlivre, titre
from    livre
where   idauteur = 3 or idauteur = 10
```

   ou bien

```
select  idlivre, titre
from    livre
where   idauteur in (3,10)
```

4. **Ordonnancement du résultat** :

   Afficher la liste des livres avec leur titre pour les livres des auteurs d'idauteur = 3 ou 10, triée en ordre croissant de idauteur et titre.

```
select  idlivre, titre
from    livre
where   idauteur in (3,10)
order by idauteur, titre
```


5. **Spécification de colonnes calculées** :

   Afficher la liste des livres avec leur titre et le prix incluant la TPS et la TVQ, pour les livres des auteurs d'idauteur = 3 ou 10, triée en ordre croissant de idauteur et titre.

```
select  idlivre, titre,
        TO_CHAR(prix*1.075*1.07, '$99,999.99') PrixTTC
from    livre
where   idauteur in (3,10)
order by idauteur, titre
```

6. **Sélection de lignes à partir d'expressions** :

   Afficher la liste des livres avec leur titre et le prix incluant la TPS et la TVQ, pour les livres des auteurs d'idauteur = 3 ou 10 et dont le prix TTC est <= 100 $, triée en ordre croissant de idauteur et titre.

```
select  idlivre, titre, prix*1.075*1.07 PrixTTC
from    livre
where   idauteur in (3,10) and
        prix*1.075*1.07 <= 100
order by idauteur, titre
```


7. **Fonction d'agrégation** : Afficher le nombre d'éditeurs dans la base de données.

```
select  count(idediteur) "nb editeurs"
from    editeur
```

8. **Fonction d'agrégation avec élimination des doublons** :

   Afficher le nombre d'éditeurs et le nombre d'auteurs dans la base de données.

```
select  count(distinct t1.idediteur) "nb editeurs",
        count(distinct t2.idauteur) "nb auteurs"
from    editeur t1, auteur t2
```


9. **Jointure de plusieurs tables** :

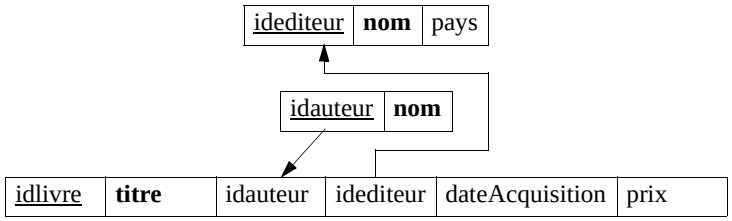
   Afficher la liste des livres avec leur titre, nom de l'auteur et nom de l'éditeur, pour les idauteur = 3 ou 10, triée en ordre croissant de idauteur et titre

```
select  t1.idlivre, t1.titre, t2.nom, t3.nom
from    livre t1, auteur t2, editeur t3
where   t1.idauteur = t2.idauteur and
        t1.idediteur = t3.idediteur and
        t1.idauteur in (3,10)
order by t1.idauteur, t1.titre
```

editeur

auteur

livre



10. **Calcul d'expressions de groupe** :

Afficher la liste des éditeurs avec le nombre de livres édités.

```
select  t1.idediteur, t1.nom, count(*) "nb livres"
from    editeur t1, livre t2
where   t1.idediteur = t2.idediteur
group by t1.idediteur, t1.nom
order by t1.idediteur
```

On note que si un éditeur n'a pas de livre édité, il n'apparaît pas dans le résultat du select. (voir #12)

11. **Sélection de groupes** :

Afficher la liste des éditeurs avec le nombre de livres édités, en sélectionnant les éditeurs qui ont édité 5 livres ou plus.

```
select  t1.idediteur, t1.nom, count(t2.idlivre) "nb livres"
from    editeur t1, livre t2
where   t1.idediteur = t2.idediteur
group by t1.idediteur, t1.nom
having  count(t2.idlivre) >= 5
order by t1.idediteur
```

12.**Jointure externe (outer join)** :

Afficher la liste des éditeurs avec le nombre de livres édités. Si un éditeur n'a aucun livre, afficher 0.

```
select  t1.idediteur, t1.nom, count(t2.idlivre) "nb livres"
from    editeur t1, livre t2
where   t1.idediteur = t2.idediteur (+)
group by t1.idediteur, t1.nom
order by t1.idediteur
```

Le (+) permet de faire un (right) outer join entre la table editeur et la table livre.

13.**Opérateur any** :

Afficher les auteurs qui ont au moins un éditeur en commun avec l'auteur dont idauteur = 15.

```
select  distinct t1.nom, t1.idauteur
from    auteur t1, livre t2
where   t1.idauteur = t2.idauteur and
        t2.idediteur =any(
    select  t3.idediteur
    from    editeur t3, livre t4
    where   t3.idediteur = t4.idediteur and
            t4.idauteur = 15) ;
```

1. editeur

<u>idediteur</u>	**nom**	pays

2. auteur

<u>idauteur</u>	**nom**

3. livre

<u>idlivre</u>	**titre**	idauteur	idediteur	dateAcquisition	prix

14.**Opérateur all** :

Afficher le (ou les) livre(s) dont le prix est le plus élevé (les éléments maximaux) pour chaque éditeur. (voir aussi #17)

```
select  t1.idediteur, t1.nom, t2.idlivre, t2.titre, t2.prix
from    editeur t1, livre t2
where   t1.idediteur = t2.idediteur and
        t2.prix >=all (
    select  t3.prix
    from    livre t3
    where   t3.idediteur = t2.idediteur)
order by t1.idediteur, t2.idlivre
```

Si plusieurs livres ont le prix le plus élevé pour un éditeur donné, chacun est affiché.

15.**Opérateur all** :

Afficher le livre le plus cher (le supremum) de chaque éditeur, s'il existe.

```
select  t1.idediteur, t1.nom, t2.idlivre, t2.titre, t2.prix
from    editeur t1, livre t2
where   t1.idediteur = t2.idediteur and
        t2.prix >all (
    select  t3.prix
    from    livre t3
    where   t3.idediteur = t2.idediteur and
            t3.idlivre != t2.idlivre)
order by t1.idediteur, t2.idlivre
```

16.

Opérateur exists :

Afficher les auteurs qui ont publié au moins un livre avec l'éditeur 8.

```
select  t1.idauteur, t1.nom
from    auteur t1
where   exists (
    select  *
    from    livre t2
    where   t2.idauteur = t1.idauteur and
            t2.idediteur = 8)
```

L'énoncé ci-dessous est équivalent.

```
select  distinct t1.idauteur, t1.nom
from    auteur t1, livre t2
where   t1.idauteur = t2.idauteur and
        t2.idediteur = 8
```

• DISTINCT parce qu'un auteur peut avoir publié plusieurs livres chez l'éditeur 8.

17.**Opérateur not exists** :

Afficher le (ou les) livre(s) dont le prix est le plus élevé (les éléments maximaux) pour chaque éditeur. (voir aussi #14)

```
select  t1.idediteur, t1.nom, t2.idlivre, t2.titre, t2.prix
from    editeur t1, livre t2
where   t1.idediteur = t2.idediteur and
        not exists (
    select  *
    from    livre t3
    where   t3.idediteur = t2.idediteur and
            t2.prix < t3.prix)
order by t1.idediteur, t2.idlivre
```

Si plusieurs livres ont le prix le plus élevé pour un éditeur donné, chacun est affiché.

18.**Opérateur not exists** :

Afficher le livre le plus cher (le supremum) de chaque éditeur, s'il existe.

```
select  t1.idediteur, t1.nom, t2.idlivre, t2.titre, t2.prix
from    editeur t1, livre t2
where   t1.idediteur = t2.idediteur and
        not exists (
    select  *
    from    livre t3
    where   t3.idediteur = t2.idediteur and
            not (t3.idlivre = t2.idlivre) and
            t2.prix <= t3.prix)
order by t1.idediteur, t2.idlivre
```


19.**Requête de type 'pour tous' (Quantification universelle)** :

Afficher les auteurs qui ont publié un livre avec chaque éditeur.

```
select  t1.idauteur, t1.nom
from    auteur t1
where   not exists (
    select *
    from    editeur t2
    where   not exists (
        select *
        from    livre t3
        where   t3.idauteur = t1.idauteur and
                t3.idediteur = t2.idediteur))
```

Solution alternative mais qui n’a pas une portée aussi générale.

```
SELECT auteur.nom
FROM auteur, livre
WHERE livre.idAuteur = auteur.idAuteur
GROUP BY auteur.nom
HAVING COUNT(distinct livre.idEditeur) = (SELECT count(*) FROM editeur) ;
```

20.**select imbriqué (clause from)** :

Afficher le (ou les) livre(s) ayant le plus grand nombre de prêts pour chaque éditeur (éléments maximaux).

```
select  t3.idediteur, t3.nom, t3.idlivre, t3.titre, t3.nbpret
from    (
    select t1.idediteur, t1.nom, t2.idlivre, t2.titre,
           count(t5.idlivre) nbpret
    from editeur t1, livre t2, pret t5
    where t1.idediteur = t2.idediteur (+) and
          t2.idlivre = t5.idlivre (+)
    group by t1.idediteur, t1.nom, t2.idlivre, t2.titre) t3
where   not exists (
    select *
    from livre t4, pret t6
    where t4.idediteur = t3.idediteur and
          t4.idlivre = t6.idlivre (+)
    group by t4.idlivre
    having count(t6.idlivre) > t3.nbpret)
order by t3.idediteur
```


7.3.4.6 Opérations ensemblistes

1. `UNION` : union de tous les tuples des subselects avec élimination des doublons ;
4. `UNION ALL` : union de tous les tuples des subselects sans élimination des doublons ;
5. `INTERSECT` : intersection avec élimination doublon ;
6. `MINUS` : différence, avec élimination doublon.

2. **Operation ensemblistes (union)** :

   Afficher la liste de tous les auteurs et de tous les editeurs en indiquant leur type ('auteur' ou 'editeur').

```
select  t1.nom nom_a_e, 'auteur' type
from    auteur t1
union
(select  t2.nom nom_a_e, 'editeur' type
from    editeur t2)
order by nom_a_e
```


7.4 Divers

7.4.1 Table virtuelle : vue

- une vue est construite à partir d'un select ;
- `CREATE VIEW <nom-vue> AS`
  <select> ;
- une vue peut-être utilisée dans un select ; la vue est évaluée au moment où le FROM du select est évalué ;
- elle peut-être utilisé dans un UPDATE si la vue réfère à une seule table, et qu'un tuple de la vue correspond à exactement un tuple de la table originale (clé (primaire ou unique) incluse)
  plus généralement une vue peut être mise à jour sans problème lorsqu'il n'y a qu'une seule façon de la mettre à jour, s'il y a plusieurs façon, certaine implémentation choisisse la plus probable alors que d'autres demandent à l'utilisateur de faire un choix (voir dans votre manuel page 186-187) ;
- elle permet de restreindre l'accès aux tables .
- Une vue est toujours supposée à jour ; si on modifie des tuples dans les tables de base sur lesquelles la vue est définie, il faut que la vue reflète automatiquement ces modifications.

Table virtuelle : vue

```
CREATE VIEW travailleSurView
AS SELECT prenom, nom, nomProjet, nbHeures
   FROM employe, projet, travailleSur
   WHERE noSs = noSsEmpl AND
         numeroProjet = noProjet ;

SELECT prenom, nom
FROM travailleSurView
WHERE nomProjet = 'ProduitX' ;


CREATE VIEW infoService (nomService, nbreEmploye, salaireTotal)
AS SELECT t1.nomSce, COUNT(t2.salaire), SUM(t2.salaire)
   FROM service t1, employe t2
   WHERE t1.noSce = t2.noSce (+)
   GROUP BY nomSce ;
```

• il est intéressant de noter que la jointure externe crée des nulls qui sont comptabilisés dans le COUNT sur oracle ; essayez par exemple COUNT(t1.nomSce) à la place de COUNT(t2.salaire)

```
UPDATE travailleSurView
SET prenom = 'Louise'
WHERE nom = 'Wattwiller' ;

SET prenom = 'Louise'
    *
ERROR at line 2:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```


Le problème #20 (page 71) à l'aide des views

```
CREATE VIEW t3 AS
select t1.idediteur, t1.nom, t2.idlivre, t2.titre,
          count(t5.idlivre) nbpret
    from editeur t1, livre t2, pret t5
    where t1.idediteur = t2.idediteur (+) and
          t2.idlivre = t5.idlivre (+)
    group by t1.idediteur, t1.nom, t2.idlivre, t2.titre ;


select  t3.idediteur, t3.nom, t3.idlivre, t3.titre, t3.nbpret
from    t3
where   not exists (
    select *
    from livre t4, pret t6
    where t4.idediteur = t3.idediteur and
          t4.idlivre = t6.idlivre (+)
    group by t4.idlivre
    having count(t6.idlivre) > t3.nbpret)
order by t3.idediteur ;
```

7.4.2 Contraintes d'intégrité (section 8.1 de la version française)

Les assertions sont des contraintes dont l'étendue dépasse les types de données, les colonnes et la table pour permettre de poser des règles de validation entre différentes colonnes de diverses tables ou vues à l'aide de prédicats.

```
«création-assertion» ::=
    CREATE ASSERTION <nom-contrainte>
        CHECK (<condition>) [«attribut-assertion»]
«attribut-assertion» ::= {INITIALLY DEFERRED | INITIALLY IMMEDIATE}
[[NOT] DEFFERABLE] | [[NOT] DEFFERABLE] [INITIALLY DEFERRED | INITIALLY
IMMEDIAT]
```

- <condition> comme dans un WHERE d'un select
- La contrainte suivante spécifie que le salaire d'un employé ne doit pas être supérieur au salaire du responsable du service dans lequel l'employé travaille.

```
CREATE ASSERTION contrainteSalaire CHECK
(NOT EXISTS (
               SELECT *
               FROM employe e, employe m, service d
               WHERE e.salaire > m.salaire AND
                     e.noSce = d.noSce AND
                     d.noSsDir = m.noSS )) ;
```

- pas disponible en Oracle
- suite sur la prochaine acétate

Contraintes d'intégrité (suite)

- Ce qui peut remplacer les assertions en oracle, ce sont les trigger

```
CREATE [OR REPLACE] TRIGGER <nom-trigger>
    {BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <nom-table>
    [FOR EACH ROW [WHEN (<condition>)]]
    <enonce PL/SQL>
```

- sera vu dans le cours IFT287

- Pour l'examen, vous ne devez que connaître l'existence de ce concept et son utilité, mais pas l'utilisation de ce concept.

7.4.3 Index (pas dans SQL)

```
CREATE [ UNIQUE ] INDEX <nom-index>
ON TABLE <nom-table> (<liste-nom-attributs>)
```

- UNIQUE est une spécification de contrainte qui signifie que les valeurs de la colonne ou les colonnes formant l'indexe sont uniques. Remarquez que ce n'est pas en général la bonne place pour spécifier ce genre de contrainte.

- Les index sont souvent créés automatiquement par le SGBD pour faciliter l'accès aux données.
  - Oracle crée automatiquement de tel index pour les clés primaires et les clés uniques
- La création et la modification d'index prend cependant un certain temps lors de l'ajout ou la modification des données prise en compte dans les index. Il y a donc un compromis à faire.

- Vous créez les index, c'est le SGBD qui gère son utilisation.

```
CREATE INDEX nomIdx
ON employe (nom, prenom);
```

```
DROP INDEX nomIdx ;
```


7.4.4 Schéma

```
«creation-schema» ::=
CREATE SCHEMA AUTHORIZATION <nom-schema> <objet-schema-liste>
«objet-schema» ::= [create_table_statement] |
                   [create_view_statement]  | [grant_statement]
```

- **schéma** : ensemble de tables, vues, domaines, séquences, procédures, synonymes, index, contraintes d'intégrité, et autres.

La commande suivante crée un schéma nommé blair pour l'usager blair et créée une table sox et une vue red_sox et accorde le privilège de faire des select à l'usager waites sur la vue red_sox.

```
CREATE SCHEMA AUTHORIZATION blair
CREATE TABLE sox
(color VARCHAR2(10) PRIMARY KEY, quantity NUMBER)
CREATE VIEW red_sox
AS SELECT color, quantity FROM sox WHERE color = 'RED'
GRANT select ON red_sox TO waites;
```

- Il existe une autre structure le catalogue (dictionnaire de données en Oracle): ensemble de schémas.

- Pour l'examen, vous ne devez que connaître l'existence de ce concept et son utilité, mais pas l'utilisation de ce concept.
-

7.4.5 Domaine

domaine-definition :: = `CREATE DOMAIN <nom-du-domaine> [AS] <type> [ DEFAULT <valeur> ] < liste de contrainte de domaine >;`

- **pas disponible en Oracle**

- exemple

```
CREATE DOMAIN sexeDomain CHAR CHECK (sexeDomain IN ('H', 'F'))
--
CREATE TABLE ayantDroit
(noSsEmpl       CHAR(13)        NOT NULL,
 nomAyantDroit  VARCHAR(15)     NOT NULL,
 sexe           sexeDomain,
 dateNaiss      DATE,
 parente        VARCHAR(8),
 CONSTRAINT cleAyantDroit
                PRIMARY KEY (noSsEmpl, nomAyantDroit),
 CONSTRAINT refAyantDroitEmpl_NoSs
                FOREIGN KEY (noSsEmpl)
                REFERENCES employe(noSs)
 );
```

- Pour l'examen, vous ne devez que connaître l'existence de ce concept et son utilité, mais pas l'utilisation de ce concept.

7.4.6 La valeur spéciale NULL

- on utilise `IS NULL` pour tester si une expression est `NULL` ;
- on utilise `IS NOT NULL` pour tester si une expression n'est pas `NULL` ;
- une fonction évaluée avec une valeur `NULL` retourne une valeur `NULL` (sauf la fonction NVL(«expression»,«valeur») qui retourne «valeur» si expression est `NULL`, sinon elle retourne la valeur de «expression») ;
- dans toutes les fonctions de groupe sauf `count(*)`, la valeur spéciale `NULL` est ignorée ;
- count(*) compte le nombre de tuple (doublons inclus), incluant les valeurs `NULL` ;
- une expression booléenne atomique utilisant une valeur `NULL` retourne la valeur inconnu ;
- une expression booléenne composée est définie selon les tables suivantes:

not	vrai	faux	inconnu
	faux	vrai	inconnu

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

- une clé primaire ne peut contenir de valeurs `NULL` pour les attributs d'un tuple ;
- une clé unique peut contenir des valeurs NULL pour un tuple. Toutefois, l'unicité doit être préservée pour les valeurs non NULL.

7.4.7 Oracle et la norme SQL2

Quelques différences entre la syntaxe SQL d'Oracle et la syntaxe SQL de la norme SQL2.

Oracle	SQL2
from R1 u, R2 v	R1AS u, R2 AS v
inexistant	CREATE DOMAIN
select * from R, S where R.B = S.C (+)}	select * from R left outer join S on R.B = S.C
select * from R, S where R.B (+) = S.C	select * from R right outer join S on R.B = S.C
select * from R, S where R.B(+) = S.C(+)	select * from R full outer join S on R.B = S.C

Remarque : Il existe quelques BD du domaine public qui sont très populaires.