V. LE LANGAGE SQL

1. Introduction

a) Généralités

SQL = *Structured Query Language*
Issu de SEQUEL (*Structured English as a Query Language*).
Standard ANSI en Octobre 1986 (proposition X3H2).

SQL permet la définition, la manipulation et le contrôle d'une base de données relationnelle, en mode interactif ou en mode intégré à partir d'un langage hôte (ex., Pro*C d'ORACLE).

b) Quelques implémentations de SQL

- SQLBASE (Gupta Technology) pour micro-ordinateurs sous MS-DOS.
- SQL/DS (IBM) pour les systèmes IBM sous DOS-VSE et VM/CMS.
- DB2 (IBM) pour les systèmes d'exploitation MVS et MVS-XA.
- INGRES (Relational Technology Inc) : un des premiers SGBD relationnels commercialisés. INGRES fonctionne sous les systèmes d'exploitation VMS, UNIX et MS-DOS.
- INFORMIX (Informix, Software Inc) pour UNIX.
- DBASE (Ashon Tate) sur PC.
- ACCESS (Microsoft) sur PC.
- ORACLE (Oracle Corporation) : SQL*Plus. Large gamme de machines : IBM, VAX, BULL, HP, micros. Proche de la norme. Environnement de développement très riche : SQL*Report, SQL*Calc, SQL*Forms, SQL*ReportWriter… Version 7 : contraintes de clé primaire et de clé étrangère.
- IDMS-SQL (CULLINET) : greffe d'une couche relationnelle sur un SGBD réseau.

c) Notations

- Les mots réservés du langage sont en majuscules ;
- les éléments terminaux sont représentés en minuscules ;
- les éléments non terminaux sont encadrés par <…> ;
- les parties optionnelles sont encadrées par […] ;
- les parties alternatives sont séparées par des barres verticales et encadrées par des accolades sous la forme {…|…|…} ;
- une répétition d'éléments (liste) est notée {…}°.

La syntaxe adoptée est celle de SQL*Plus d'ORACLE.

d) Base de données exemple

La base FABRICATION sera utilisée pour tous les exemples.

```
PIECE (NOP, DESIGNATION, COULEUR, POIDS)
SERVICE (NOS, INTITULE, LOCALISATION)
ORDRE (NOP, NOS, QUANTITE)
NOMENCLATURE (NOPA, NOPC, QUANTITE)
```

SERVICE

NOS	INTITULE	LOCALISATION
S1	DIFFUSION GP	Paris
S2	PROMOTION	Clermont-Fd
S3	DIFFUSION GR	Lyon
S4	PROMOTION	Moulins
S5	DIFFUSION GP	Clermont-Fd

NOMENCLATURE

NOPA	NOPC	QUANTITE
P6	P1	1
P6	P5	1
P7	P3	1
P7	P6	2

PIECE

NOP	DESIGNATION	COULEUR	POIDS
P1	Lavabo	ivoire	30
P2	Lavabo	bleu	30
P3	Baignoire	ivoire	50
P4	Baignoire	bleu	50
P5	Colonne	ivoire	10
P6	Lavabo luxe	ivoire	40
P7	Set luxe	ivoire	130

ORDRE

NOS	NOP	QUANTITE
S1	P1	10
S1	P2	40
S1	P3	20
S1	P4	15
S2	P1	10
S2	P3	20
S3	P2	15
S3	P4	10
S5	P3	15

Forme tabulaire de la base de données FABRICATION

2. Commandes de définition des données

a) Définition du schéma

Schéma = ensemble de tables et vues

```
CREATE SCHEMA AUTHORIZATION nom-schéma [ {<élément-schéma>}° ]

<élément-schéma> := { <définition-table> |
                      <définition-vue> |
                      <définition-privilège>}
```

Note : Le nom du schéma nom-schéma doit être le même que le nom d'utilisateur Oracle.

b) Définition des tables

```
CREATE TABLE <nom-table> ( {<élément-table>}° )

<nom-table>:= nom-table-simple | nom-schéma.nom-table-simple
<élément-table>:= <définition-colonne> | <contrainte-table>
<définition-colonne>:= nom-colonne <type> [<clause-défaut>]
                       [<contrainte-colonne>]
<clause-défaut>:= DEFAULT valeur
<type> définit le type des données parmi la liste suivante (non limitative) :
    NUMBER(n, m) (décimaux à n chiffres au total, m après la virgule)
    CHAR(n) (chaîne de longueur fixe)
    VARCHAR(n) (chaîne de longueur variable)
    DATE (généralement au format 'DD-MM-YY HH:MM:SSam')
```

Exemple pour la base FABRICATION :

```
CREATE SCHEMA AUTHORIZATION darmont
CREATE TABLE SERVICE (NOS CHAR(3), INTITULE CHAR(20), LOCALISATION
CHAR(15))
CREATE TABLE PIECE (NOP CHAR(3), DESIGNATION CHAR(25), COULEUR
CHAR(15) DEFAULT 'blanc', POIDS NUMBER (6,3))
…
;
```

c) Définition des contraintes d'intégrité

- Clés primaires
- Clés étrangères
- Contraintes de domaine

```
<contrainte-colonne>:=
[ [CONSTRAINT nom-contrainte] [NOT] NULL ] ]
[ [CONSTRAINT nom-contrainte] {UNIQUE | PRIMARY KEY} ]
[ [CONSTRAINT nom-contrainte] REFERENCES <nom-table> [({nom-
colonne}°)] [ON DELETE CASCADE] ]
[ [CONSTRAINT nom-contrainte] CHECK (<condition>) ]
```

```
<contrainte-table>:=
[ [CONSTRAINT nom-contrainte] {UNIQUE | PRIMARY KEY} ({nom-colonne}°)
]
[ [CONSTRAINT nom-contrainte] FOREIGN KEY ({nom-colonne}°) REFERENCES
<nom-table> [ ({nom-colonne}°)] [ON DELETE CASCADE] ]
[ [CONSTRAINT nom-contrainte] CHECK (<condition>) ]
```

Le nommage d'une contrainte par la clause `[CONSTRAINT nom-contrainte]` permet sa suppression ultérieure avec la commande `DROP CONSTRAINT`.

Les noms de colonne apparaissant dans la clause `REFERENCES` doivent être ceux d'une clé primaire ou doivent être déclarés avec la clause `UNIQUE | PRIMARY KEY`.

Les clauses `UNIQUE NOT NULL` et `PRIMARY KEY` sont équivalentes.

La contrainte `REFERENCES` définit une contrainte d'intégrité référentielle par rapport à une clé unique ou primaire. On ne peut insérer pour la clé étrangère que des valeurs déjà présentes pour la clé primaire. En l'absence d'option, toute tentative de suppression ou de modification d'une valeur de la clé primaire sera rejetée par le système si cette valeur apparaît au niveau de la clé étrangère correspondante. L'option `ON DELETE CASCADE` autorise la suppression d'une valeur de la clé primaire. Les valeurs de la clé étrangère correspondante sont alors supprimées (effet de cascade). Certains systèmes prévoient d'autres options.

Exemple d'utilisation pour la base FABRICATION :

```
CREATE TABLE SERVICE (NOS CHAR(5), …, PRIMARY KEY (NOS)) ;
CREATE TABLE PIECE (NOP CHAR(5), …, PRIMARY KEY (NOP)) ;
CREATE TABLE ORDRE (NOS CHAR(5), NOP CHAR(5),
    QUANTITE NUMBER(6, 3) CHECK (QUANTITE > 10),
    PRIMARY KEY (NOS, NOP),
    FOREIGN KEY (NOS) REFERENCES SERVICE (NOS),
    FOREIGN KEY (NOP) REFERENCES PIECE (NOP)) ;
CREATE TABLE NOMENCLATURE (NOPA CHAR(5), NOPC CHAR(5),
    QUANTITE NUMBER (6,3),
    CONSTRAINT cle_prim PRIMARY KEY (NOS, NOP),
    CONSTRAINT cle_etr1 FOREIGN KEY (NOPA) REFERENCES PIECE (NOP),
    CONSTRAINT cle_etr2 FOREIGN KEY (NOPC) REFERENCES PIECE (NOP)) ;
```

d) Définition des vues

Vue = table *virtuelle* calculée à partir des tables de base grâce à une requête

```
CREATE VIEW [nom-schéma.]nom-vue ({ nom-colonne }°)
AS <requête-pleine>
[WITH CHECK OPTION [CONSTRAINT nom-contrainte] ]
```

`nom-vue` est le nom de la vue.

`nom-colonne` est un nom de colonne d'une table existante.

`<requête-pleine>` est une requête d'interrogation.

`WITH CHECK OPTION` sert à imposer la vérification des mises à jour.

Exemple : Vue des numéros et désignations de pièces

```
CREATE VIEW nopdes
AS SELECT NOP, DESIGNATION FROM PIECE ;
```

e) Création des index

Index ⇒ accélération des accès aux données par la clé d'index (colonne) choisie.

```
CREATE [UNIQUE] INDEX [nom-schéma.]nom-index ON <nom-table>
( { nom-colonne [ASC | DESC ] }° )
```

`UNIQUE` ⇒ pas de double.

`ASC` et `DESC` ⇒ tri en ordre croissant ou décroissant (respectivement).

Exemple : index sur les numéros de pièce, sans double, en ordre croissant

```
CREATE UNIQUE INDEX pie_idx ON PIECE (NOP ASC) ;
```

f) Destructions et restructurations

Destruction de tables, de vues, d'index :

```
DROP TABLE <nom-table> [CASCADE CONSTRAINTS]
DROP VIEW [nom-schéma.]nom-vue
DROP INDEX [nom-schéma.]nom-index
```

Ex.
```
DROP TABLE NOMENCLATURE ;
DROP VIEW nopdes ;
DROP INDEX pie_idx ;
```

Ajout d'éléments dans une table :

```
ALTER TABLE <nom-table> ADD ( { <élément-table> }° )
```

Ex.
```
ALTER TABLE PIECE ADD (
    NOF CHAR(5),
    PRIX NUMBER(6, 2) CHECK (PRIX > 0),
    CONSTRAINT fab FOREIGN KEY (NOF) REFERENCES FABRIQUANT(NOF) ;
```

Modification d'éléments dans une table :

```
ALTER TABLE <nom-table> MODIFY ( { <définition-colonne> }° )
```

Ex.
```
ALTER TABLE PIECE MODIFY (DESIGNATION CHAR(100)) ;
```

Suppression de contraintes dans une table :

```
ALTER TABLE <nom-table> DROP { PRIMARY KEY | UNIQUE ({nom-colonne}°)
| [CONSTRAINT nom-contrainte] } [CASCADE]
```

Ex.
```
ALTER TABLE PIECE DROP PRIMARY KEY ;
ALTER TABLE NOMENCLATURE DROP CONSTRAINT cle_etr1 ;
```

g) Création et suppression de synonymes

Création et suppression d'alias pour les tables, vues, etc.

```
CREATE [PUBLIC] SYNONYM [nom-schéma.]synonyme FOR [nom-schéma.]objet
DROP [PUBLIC] SYNONYM [nom-schéma.]synonyme
```

Ex.
```
CREATE SYNONYM COMMANDE FOR ORDRE ;
DROP SYNONYM COMMANDE ;
```

h) Renommage

Changement de nom de tables, de vues, etc.

```
RENAME ancien-nom TO nouveau-nom
```

Ex.
```
RENAME ORDRE TO COMMANDE ;
```

3. Commandes de mise à jour des données

a) Insertion externe d'une ligne

```
INSERT INTO { <nom-table> | nom-vue} [{nom-colonne}°]
VALUES ( {<atome>}° )
<atome>:= {constante | variable-hôte | NULL }
```

La liste des atomes de la clause `VALUES` doit correspondre à celles des colonnes à insérer, tant en ce qui concerne le nombre que le type.

Ex.
```
INSERT INTO SERVICE VALUES ('S6', 'PROMOTION', 'Riom') ;
```

b) Insertion interne de plusieurs lignes

```
INSERT INTO { <nom-table> | nom-vue} [{nom-colonne}°]
<requête-pleine>
```

Les lignes à insérer proviennent d'une requête d'interrogation. Là encore la correspondance entre colonnes doit être assurée.

Ex.
```
INSERT INTO SERVICE2
    SELECT NOS, INTITULE, LOCALISATION FROM SERVICE ;
```

c) Mise à jour de colonnes

```
UPDATE {<nom-table> | nom-vue} SET { nom-colonne = {<expression> |
<requête-pleine>} }° [WHERE <condition>]
```

Ex.
```
UPDATE SERVICE SET LOCALISATION = 'MOULINS' WHERE NOS = 'S3' ;
UPDATE ORDRE SET QUANTITE = QUANTITE + 10 WHERE NOP = 'P1' ;
```

d) Suppression de lignes

```
DELETE FROM {<nom-table> | nom-vue} [WHERE <condition>]
```

En l'absence de clause `WHERE`, tous les tuples de la table sont supprimés.

Ex.
```
DELETE FROM PIECE WHERE COULEUR = 'ivoire' ;
```

4. Commandes d'interrogation

a) Syntaxe et notations

Sous-requête :

```
<sous-requête>:=
    SELECT [ {ALL | DISTINCT }] { * | <liste-sélection> }
    FROM { {nom-table | nom-vue} [alias-local] }°
    [ WHERE <condition> ]
    [ GROUP BY {<expression>}° [ HAVING <condition> ] ]
```

- `ALL` et `DISTINCT` permettent de conserver ou d'éliminer les lignes en double.
- `FROM` précise tous les objets (tables et/ou vues) à manipuler.
- `WHERE` précise la condition que doivent satisfaire les lignes à sélectionner.
- `GROUP BY` partitionne en groupes.
- `HAVING` sélectionne les groupes selon une condition.

```
<liste-sélection>:= { {nom-table | nom-vue | alias-local}.* |
{nom-table | nom-vue | alias-local}.nom-colonne | {<expression>
[en-tête]} }°
```

* est une notation compacte pour récupérer dans la liste de sélection toutes les colonnes des tables et vues de la clause `FROM`.

`<expression>` est une expression arithmétique dont les opérandes sont des noms de colonnes éventuellement préfixés par un nom de table, de vue ou d'alias.

Tous les noms de tables et de vues peuvent être préfixées par le nom du schéma.

Un *alias local* facilite la formulation. Il doit être déclaré au niveau de la clause `FROM`.

Le résultat d'une sous-requête est une table dont les colonnes sont définies sur la `liste-sélection`.

Requête pleine :

```
<requête-pleine>:= {<sous-requête> | (<requête-pleine>)}
<ope-ens> {<sous-requête> | (<requête-pleine>)}
<ope-ens>:= { INTERSECT | MINUS | UNION [ALL] }
```

Les tables manipulées doivent être compatibles en ce qui concerne le nombre et le type de leurs colonnes.

Requête :

```
<requête>:= <requête-pleine>
[ ORDER BY { {nom-colonne | numéro} [{ASC | DESC}] }°]
```

La clause `ORDER BY` permet de trier le résultat de la requête.

Ex. `SELECT * FROM PIECE ORDER BY DESIGNATION ;`
⇔ `SELECT * FROM PIECE ORDER BY 2 ;`

Condition :

Expression logique parenthésée faisant intervenir les opérateurs logiques `AND`, `OR`, `NOT`. Les termes booléens sont obtenus à partir de diverses formes prédicatives.

```
<condition>:= <forme-prédicative> { AND | OR } <forme-prédicative>
<forme-prédicative>:= [ NOT ] { <prédicat> | <condition> }

<prédicat>:= {
    | <expression> <ope-comp> {
        | <expression>
        | ( <requête-pleine> )
        | ANY ( <requête-pleine> )
        | ALL ( <requête-pleine> ) }
    | <expression> [ NOT ] BETWEEN <expression> AND <expression>
    | <expression> IS [ NOT ] NULL
    | <expression> [ NOT ] LIKE { constante | variable-hôte }
    | <expression> [ NOT ]  IN { | (<requête-pleine>)
        | {<expression>}° }
    | EXISTS (<requête-pleine>) …}
<ope-comp>:= {= | <> | > | >= | < | <=}
```

Six formes principales de prédicats :

1) prédicat de comparaison scalaire entre deux valeurs : la valeur d'une expression d'une part, et la valeur d'une expression ou le résultat d'une requête monoligne d'autre part ;
2) prédicat quantifié formé avec les quantificateurs `ANY`, `ALL` pour comparer une valeur à un ensemble de valeurs ;
3) prédicat de comparaison `BETWEEN` et `LIKE` ;
4) prédicat de test de valeur nulle (`NULL`) ;
5) prédicat de comparaison ensembliste `IN` ;
6) prédicat `EXISTS` pour tester l'existence de lignes dans la table résultat.

Expressions arithmétiques :

```
<expression>:= <opérande> { { + | - | * | / | } <opérande> }°
<opérande>:= [ {+ | - }] { nom-colonne | constante | variable-hôte |
<fonction> | (<expression>) }
```

Il existe de très nombreuses fonctions pour tous les types de données.

Fonctions d'agrégat :

Elles opèrent sur un ensemble de valeurs :

- `AVG` : moyenne des valeurs,
- `SUM` : somme des valeurs,
- `MIN` : valeur minimum,
- `MAX` :valeur maximum,
- `COUNT` : nombre de valeurs.

Ex. `SELECT AVG(POIDS) FROM PIECE ;`

b) Requêtes simples

Numéros des services ayant en commande la pièce P1 avec une quantité supérieure à 10, dans l'ordre croissant :

```
SELECT NOS FROM ORDRE
WHERE NOP = 'P1' AND QUANTITE >10
ORDER BY NOS ;
```

L'emballage d'une pièce a un poids égal à environ 20% du poids de la pièce. Poids total emballé pour les différentes pièces, dans l'ordre décroissant :

```
SELECT NOP, POIDS+0.2*POIDS FROM PIECE
ORDER BY 2 DESC ;
```

Nombre de services :

```
SELECT COUNT(*) FROM SERVICE ;
```

Nombre de services ayant des commandes :

```
SELECT COUNT (DISTINCT NOS) FROM ORDRE ;
```

Quantité moyenne commandée pour la pièce P3 :

```
SELECT AVG(QUANTITE) FROM ORDRE
WHERE NOP = 'P3' ;
```

c) Jointure interne

C'est un produit cartésien suivi d'une sélection relationnelle : `FROM + WHERE`.

Liste des pièces commandées par le service S1 avec leur libellé et leurs poids :

```
SELECT NOS, ORDRE.NOP, DESIGNATION, POIDS
FROM ORDRE PIECE
WHERE ORDRE.NOP = PIECE.NOP AND NOS = 'S1'
ORDER BY ORDRE.NOP ;
```

Pour alléger l'écriture, il est possible d'affecter un *alias local* à un nom de table.

```
SELECT NOS, R.NOP, DESIGNATION, POIDS
FROM ORDRE R, PIECE P
WHERE R.NOP = P.NOP AND NOS = 'S1'
ORDER BY R.NOP ;
```

Cette forme de spécification permet d'opérer sur une même table (autojointure) à condition d'utiliser des synonymes différents. Numéros des services qui ont commandé la pièce P1 et la pièce P3 en même quantité :

```
SELECT A1.NOS FROM ORDRE A1, ORDRE A2
WHERE A1.NOP = 'P1' AND A2.NOP = 'P3'
AND A1.NOS = A2.NOS AND A1.QUANTITE = A2.QUANTITE ;
```

Il est possible de formuler toute jointure et pas seulement des équijointures. Pour chaque pièce, numéro et désignation de toutes les pièces qui ont un poids supérieur :

```
SELECT P1.NOP, P2.NOP, P2.DESIGNATION
FROM PIECE P1, PIECE P2
WHERE P1.POIDS < P2.POIDS ;
```

d) Opérations ensemblistes

Numéros des pièces qui, soit ont un poids inférieur à 50, soit ont été commandées par le service S2 :

```
SELECT NOP FROM PIECE WHERE POIDS <50
UNION
SELECT NOP FROM ORDRE WHERE NOS = ‘S2’ ;
```

Note : Les doublons sont éliminés.

e) Les différents types de prédicats

Prédicats de comparaison scalaire :

Numéros des services qui ont commandé la pièce P3 avec une quantité inférieure à la quantité moyenne commandée pour cette pièce :

```
SELECT NOP, NOS, QUANTITE FROM ORDRE
WHERE NOP = 'P3' AND QUANTITE <
    (SELECT AVG(QUANTITE) FROM ORDRE
    WHERE NOP = 'P3') ;
```

La commande `SELECT` imbriquée peut concerner la même table que la commande `SELECT` d'appel et dépendre de la ligne d’appel (*corrélation*). Caractéristiques de chaque pièce ayant un poids inférieur à la moyenne des poids des pièces de leur couleur :

```
SELECT * FROM PIECE P
WHERE POIDS <
    (SELECT AVG(POIDS) FROM PIECE
    WHERE COULEUR = P.COULEUR)
ORDER BY NOP ;
```

Note : Pour chaque ligne de la question d'appel, la sous-question est réévaluée.

Prédicat `BETWEEN` :

Les prédicats `BETWEEN` et `NOT BETWEEN` servent à déterminer si une valeur numérique appartient ou n'appartient pas à un intervalle (bornes incluses). Numéros des pièces dont le poids est compris entre 50 et 100 :

```
SELECT NOP FROM PIECE
WHERE POIDS BETWEEN 50 AND 100 ;
```

Prédicat `IS NULL` :

Les prédicats `IS NULL` et `IS NOT NULL` servent à tester si une ligne d'une table présente ou ne présente pas une valeur nulle dans une colonne. Numéros des services qui n'ont pas valué leurs commandes et les numéros des pièces correspondantes :

```
SELECT NOS, NOP FROM ORDRE
WHERE QUANTITE IS NULL ;
```

Prédicat `LIKE` :

Les prédicats `LIKE` et `NOT LIKE` servent à déterminer si une forme particulière existe ou n'existe pas dans une chaîne de caractères. Le symbole % joue le rôle de joker.

- `X LIKE 'LYON'` prend la valeur vraie si la chaîne X est identique à LYON.
- `X LIKE '%LYON%'` prend la valeur vraie si la chaîne X contient LYON.
- `X LIKE 'LYON%'` prend la valeur vraie si la chaîne X débute par LYON.
- `X LIKE '%LYON'` prend la valeur vraie si la chaîne X se termine par LYON.

Numéros des pièces dont la désignation se termine par 'ON' et dont le poids est compris entre 50 et 100 :

```
SELECT NOP FROM PIECE
WHERE DESIGNATION LIKE '%ON' AND POIDS BETWEEN 50 AND 100 ;
```

Prédicat `IN` :

Les prédicats `IN` et `NOT IN` servent à déterminer si une valeur appartient ou n'appartient pas à un ensemble de valeurs du même type. Désignation des pièces de couleur ivoire, rouge ou blanc :

```
SELECT DESIGNATION FROM PIECE
WHERE COULEUR IN ('ivoire', 'rouge', 'blanc') ;
```

Il est possible de spécifier la *jointure* avec le prédicat `IN`. Numéro et désignation des pièces qui sont commandées par le service S1 :

```
SELECT NOP, DESIGNATION FROM PIECE
WHERE NOP IN
    (SELECT NOP FROM ORDRE
    WHERE NOS = 'S1')
ORDER BY NOP ;
```

Numéro et désignation des pièces qui sont commandées par un service DIFFUSION :

```
SELECT NOP, DESIGNATION FROM PIECE
WHERE NOP IN
    (SELECT NOP FROM ORDRE
    WHERE NOS IN
        (SELECT NOS FROM SERVICE
        WHERE INTITULE LIKE  'DIFFUSION%'))
ORDER BY NOP ;
```

Prédicats `ALL` et `ANY` :

Le prédicat `ALL` prend la valeur *vrai* si la requête pleine qui le suit ne fournit aucune valeur ou si le prédicat de comparaison associé prend la valeur *vrai* pour toutes les valeurs scalaires fournies par la requête pleine.

Prédicat `ANY` prend la valeur *vrai* si le prédicat de comparaison associé prend la valeur *vrai* pour au moins une des valeurs scalaires fournies par la requête pleine.

Si la requête pleine qui suit ces prédicats fournit uniquement des valeurs nulles, ils prennent la valeur ‘inconnue’.

Numéros des services ayant commandé au moins une pièce en quantité strictement supérieure à chacune des quantités de pièces commandées par le service S1 :

```
SELECT DISTINCT NOS FROM ORDRE
WHERE QUANTITE > ALL
    (SELECT QUANTITE FROM ORDRE WHERE NOS = 'S1') ;
```

Prédicat `EXISTS` :

Le prédicat `EXISTS` prend la valeur *faux* si la requête pleine qui le suit a comme résultat l'ensemble vide, *vrai* sinon.

Intitulés des services ayant commandé au moins une pièce.

```
SELECT INTITULE FROM SERVICE S
WHERE EXISTS
    (SELECT * FROM ORDRE C WHERE C.NOS = S.NOS) ;
```

Intitulés des services qui n'ont pas commandé de pièce.

```
SELECT INTITULE FROM SERVICE S
WHERE NOT EXISTS
    (SELECT * FROM ORDRE C WHERE C.NOS = S.NOS) ;
```

Expression du quantificateur universel à l'aide du prédicat `EXISTS` :

$\forall$ x (p) $\Leftrightarrow$ NOT (EXISTS x (NOT(p))

Ex. "Services tels que pour toute pièce il existe une commande" $\Leftrightarrow$ "Services tels qu'il n'existe pas une pièce telle qu'il n'existe pas une commande".

Intitulés des services ayant commandé toutes les pièces :

```
SELECT INTITULE FROM SERVICE S
WHERE NOT EXISTS
    (SELECT * FROM PIECE P
    WHERE NOT EXISTS
        (SELECT * FROM ORDRE C
        WHERE S.NOS = C.NOS AND C.NOP = P.NOP ) ) ;
```

Reformulation des prédicats `ALL` et `ANY` à l'aide du prédicat `EXISTS` :

```
x θ ANY (SELECT y FROM T WHERE p ) ⇔
EXISTS (SELECT * FROM T WHERE (p) AND (x θ T.y))

x θ ALL (SELECT y FROM T WHERE p ) ⇔
NOT EXISTS (SELECT * FROM T WHERE (p) AND NOT (x θ T.y))
```

Numéros des services ayant commandé au moins une pièce en quantité strictement supérieure à chacune des quantités de pièces commandées par le service S1 :

```
SELECT DISTINCT NOS FROM ORDRE CX
WHERE NOT EXISTS
    (SELECT * FROM ORDRE CY
    WHERE CY.NOS = 'S1' AND CX.QUANTITE <= CY.QUANTITE) ;
```

f) Jointure externe

Jointure externe entre la relation R et la relation S = conserver toutes les lignes de R, même celles qui n'ont pas de lignes en correspondance dans S.

Pour chaque service, numéro et intitulé, et pour ceux qui ont des commandes, la liste des pièces commandées :

```
SELECT S.NOS, INTITULE, NOP FROM SERVICE S, ORDRE R
WHERE S.NOS = R.NOS
UNION
SELECT NOS, INTITULE, '    ' FROM SERVICE
WHERE NOT EXISTS
    (SELECT * FROM ORDRE WHERE SERVICE.NOS =
    ORDRE.NOS) ;
```

Opérateur de jointure externe spécifique à SQL*Plus d'ORACLE :

```
SELECT S.NOS, INTITULE, NOP FROM SERVICE S, ORDRE R
WHERE S.NOS(+) = R.NOS ;
```

g) Groupement

1) Sélection des lignes en fonction de la clause `WHERE`.
2) Groupement des lignes restantes en groupes disjoints en fonction de leurs valeurs pour les colonnes spécifiées dans `GROUP BY` : toutes les lignes d'un groupe ont mêmes valeurs pour ces colonnes.
3) Sélection des groupes en fonction de la clause `HAVING`.
4) Relation résultat : autant de lignes qu'il y a de groupes sélectionnés.
5) Colonnes du résultat : soit des expressions faisant uniquement intervenir les colonnes de groupement, soit des fonctions d'agrégat appliquées aux lignes d'un groupe.

Quantités commandées pour les pièces P1, P2, P3 :

```
SELECT NOP, SUM(QUANTITE) FROM ORDRE
WHERE NOP IN ('P1', 'P2', 'P3')
GROUP BY NOP
ORDER BY NOP ;
```

Quantité moyenne commandée pour les pièces faisant l'objet de plus de 3 commandes :

```
SELECT NOP, AVG(QUANTITE) FROM ORDRE
GROUP BY NOP HAVING COUNT(*) > 3
ORDER BY NOP ;
```

h) Exploration de structures hiérarchiques

Clause `CONNECT BY` (spécifique à SQL*Plus d'ORACLE).

```
CONNECT BY <condition1> [ START WITH <condition2>]
```

`CONNECT BY` spécifie la liaison de parenté par le mot clé `PRIOR`.
`<condition1>` doit contenir un prédicat de la forme :

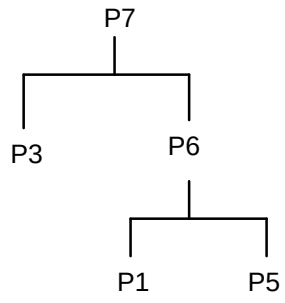
```
PRIOR <expression> <ope-comp> <expression>
```

ORACLE évalue d'abord la première expression pour la ligne du parent, puis la deuxième expression pour toutes les lignes de la table. Les fils correspondent aux lignes pour lesquelles le prédicat est vrai.

`START WITH <condition2>` sélectionne la ligne à utiliser pour commencer l'exploration (racine de l'arbre ou racine d'un sous-arbre).

NOMENCLATURE

NOPA	NOPC	QUANTITE
P6	P1	1
P6	P5	1
P7	P3	1
P7	P6	2
-	P7	-



Exemple de nomenclature pour la base FABRICATION

Pièces intervenant dans la composition de la pièce P7 :

```
SELECT NOPC FROM NOMENCLATURE
CONNECT BY PRIOR NOPC = NOPA
START WITH NOPC = 'P7' ;
```

1) Début traitement pour NOPC = 'P7'.
2) Utilisation de cette valeur comme référence pour explorer la colonne NOPA.
3) Pour chaque possibilité obtention d'une nouvelle référence de NOPC.
4) Itération du processus pour chaque cas.

Pièces intervenant dans la composition de la pièce P7, avec leur niveau dans la hiérarchie et la quantité correspondante :

```
SELECT LEVEL, NOPC, QUANTITE FROM NOMENCLATURE
CONNECT BY PRIOR NOPC = NOPA
START WITH NOPC = 'P7' ;
```

`LEVEL` : pseudo-colonne permettant la manipulation du niveau de chaque élément dans la hiérarchie, la racine ayant le niveau 1.

SQL25

NOPC
P7
P3
P6
P1
P5

SQL26

LEVEL	NOPC	QUANTITE
1	P7	-
2	P3	1
2	P6	2
3	P1	1
3	P5	1

Résultats des deux dernières requêtes

5. Sécurité et autorisation

a) Transmission de privilèges d'accès et de mise à jour

```
GRANT {<privilège>}° ON {nom-table | nom-vue}
TO [ {id-usager}° | PUBLIC ] [WITH GRANT OPTION]
```

`id-usager` est un identificateur d'usager.

`PUBLIC` désigne tous les usagers.

Privilèges possibles :

- `SELECT` : lecture seulement,
- `INSERT` : insertion de lignes,
- `UPDATE` : mise à jour de lignes,
- `DELETE` : suppression de lignes,
- `ALL` : tous les privilèges sur une table,
- `ALTER` : destruction de la table,
- `INDEX` : construction d'un index sur la table.

`ALL`, `ALTER` et `INDEX` ne sont utilisables qu'avec une table.

`WITH GRANT OPTION` : transmission intermédiaire des privilèges.

Ex. Tous les privilèges sur la table PIECE octroyés à l'utilisateur 'machin', avec droit pour 'machin' de transmettre à son tour les privilèges.

```
GRANT ALL ON PIECE TO machin WITH GRANT OPTION ;
```

b) Suppression de privilèges

```
REVOKE {<privilège>}° ON {nom-table | nom-vue}
FROM [{id-usager}° | PUBLIC]
```

Ex. `REVOKE ALL ON PIECE FROM machin ;`

6. Utilité et problèmes des vues

a) Problèmes de mise à jour

Une mise à jour des données à partir d'une vue peut être envisagée à partir du moment où les modifications satisfont la définition de la vue. Cependant, la répercussion de

ces modifications sur les tables primaires pose un certain nombre de problèmes et la plupart des systèmes imposent des conditions très restrictives.

Les principales restrictions sont les suivantes :

- le mot clé `DISTINCT` doit être absent ;
- la clause `FROM` doit faire référence à une seule table sur laquelle les opérations de mise à jour sont autorisées ;
- la clause `SELECT` doit faire référence directement aux colonnes de la table sous-jacente (les colonnes calculées sont prohibées) ;
- les clauses `GROUP BY` et `HAVING` sont interdites.

b) Intérêt des vues

1) Simplification de l'accès aux données en masquant les opérations de jointures

Ex : la requête "Quelles sont les pièces commandées avec une quantité >10" se formule comme suit à partir de la vue PIECE_COM :

```
SELECT NOP, DESIGNATION FROM PIECE_COM WHERE QUANTITE >10 ;

CREATE VIEW PIECE_COM (NOP, DESIGNATION, POIDS, NOS, QUANTITE)
AS SELECT P.NOP, DESIGNATION, POIDS, R.NOS, QUANTITE
FROM PIECE P, ORDRE R  WHERE P.NOP=R.NOP ;
```

2) Sauvegarde indirecte de requêtes complexes

3) Présentation de mêmes données sous différentes formes adaptées aux différents usagers en particulier pour les mises à jour.

4) Support de l'indépendance logique.

Ex : la table PIECE, suite à un remaniement est éclatée en deux tables :

```
PIECE1 (NOP, DESIGNATION, COULEUR)
PIECE2 (NOP, POIDS)
```

La création de la vue PIECE_COM doit être reformulée comme suit :

```
CREATE VIEW PIECE_COM (NOP, DESIGNATION, POIDS, NOS, QUANTITE)
AS SELECT P1.NOP, DESIGNATION, POIDS, R.NOS, QUANTITE
    FROM PIECE1 P1, PIECE2 P2, ORDRE R
    WHERE P1.NOP = P2.NOP AND P1.NOP = R.NOP ;
```

mais les requêtes qui utilisaient l'ancienne vue PIECE_COM n'ont pas à être remaniées !

5) Renforcement de la sécurité des données par masquage des lignes et des colonnes sensibles aux usagers non habilités.

7. Contrôle des transactions

Transaction = *unité de recouvrement* qui doit être soit validée, soit annulée (en cas d'incident) dans sa totalité.

Le début d'une transaction est implicitement défini :

- par la première commande SQL (début d'une application ou d'une session),
- par la fin d'une transaction précédente.

Fin implicite (automatique) d'une transaction :

- commande de définition de données : validation des opérations précédentes ;
- fin normale de l'application ou de la session : validation ;
- fin anormale de l'application ou de la session : annulation.

Fin explicite (provoquée par l'utilisateur) d'une transaction :

- commande `COMMIT` pour valider la transaction,
- commande `ROLLBACK` pour annuler la transaction.

Verrouillage d'une table :

```
LOCK TABLE nom-table IN [ SHARE | EXCLUSIVE ] MODE
```

`SHARE` : partage en lecture seulement.
`EXCLUSIVE` : tout partage en écriture est interdit.

Selon les systèmes d'autres modes sont disponibles. Les commandes `COMMIT` et `ROLLBACK` libèrent tous les verrous.

Initialisation du mode d'une transaction :

```
SET TRANSACTION [ READ ONLY | READ WRITE ]
```

La norme SQL 92 prévoit d'autres possibilités.

8. Catalogue du système

Le *catalogue* (ou *dictionnaire*) *système* contient sous forme relationnelle la définition de tous les objets créés par le système et les usagers.

Chaque usager peut accéder avec SQL (en mode consultation seulement) à la définition des objets qu'il a créés ou sur lesquels il a un privilège.

Les noms des tables systèmes sont spécifiques à chaque système. Voici quelques tables utiles gérées par le système ORACLE :

```
USER_CATALOG (TABLE_NAME, TABLE_TYPE)
USER_TAB_COLUMNS (TABLE_NAME, COLUMN_NAME, …)
USER_IND_COLUMNS (INDEX_NAME, TABLE_NAME, COLUMN_NAME, …)
ALL_TABLES (TABLE_NAME, OWNER, …)
```

Exemples d'utilisation :

Tables qui contiennent une colonne de nom INTITULE :

```
SELECT TABLE_NAME FROM USER_TAB_COLUMNS
WHERE COLUMN_NAME = 'INTITULE' ;
```

Colonnes de la table SERVICE :

```
SELECT COLUMN_NAME FROM USER_TAB_COLUMNS
WHERE TABLE_NAME = 'SERVICE' ;
```

Tables de l'utilisateur 'darmont' :

```
SELECT TABLE_NAME FROM ALL_TABLES
WHERE OWNER='DARMONT' ;
```

9. Exercices

I. Formuler avec le langage SQL les requêtes utilisées pour présenter l'algèbre relationnelle.

1) Désignation et couleur de toutes les pièces.

2) Désignation des pièces de couleur ivoire.

3) Intitulé des services ayant en commande la pièce P2.

4) Intitulé des services ayant en commande au moins une pièce de couleur ivoire.

5) Intitulé des services n'ayant pas en commande la pièce P1.

6) Numéro des services ayant en commande toutes les pièces.

7) Numéro des services ayant en commande au moins toutes les pièces commandées au service S2.

II. Sur la base FABRICATION, exprimer les requêtes suivantes.

1) Donner pour chaque service le poids de la pièce commandée de couleur bleue la plus pesante.

2) Donner le poids moyen des pièces commandées pour chacun des services "Promotion".

3) Donner les pièces de couleur bleue qui sont commandées par plus de trois services différents.

4) Donner le maximum parmi les totaux des pièces commandées par les différents services.

Correction des exercices

I.

1) SELECT DESIGNATION, COULEUR FROM PIECE ;
2) SELECT DESIGNATION FROM PIECE WHERE COULEUR=’ivoire’ ;
3) SELECT DISTINCT INTITULE FROM SERVICE S, ORDRE O WHERE S.NOS=O.NOS AND NOP=’P2’ ;
4) SELECT DISTINCT INTITULE FROM SERVICE S, ORDRE O, PIECE P WHERE S.NOS=O.NOS AND O.NOP=P.NOP AND COULEUR=’ivoire’ ;
5) SELECT INTITULE FROM SERVICE S WHERE NOT EXISTS (SELECT * FROM ORDRE O WHERE S.NOS=O.NOS AND NOP=’P1’) ;
6) SELECT NOS FROM SERVICE S WHERE NOT EXISTS (SELECT * FROM PIECE P WHERE NOT EXISTS (SELECT * FROM ORDRE O WHERE S.NOS=O.NOS AND O.NOP=P.NOP)) ;
7) SELECT NOS FROM SERVICE S WHERE NOT EXISTS (SELECT * FROM PIECE P, ORDRE O1 WHERE P.NOP=O1.NOP AND O1.NOS=’S2’ AND NOT EXISTS (SELECT * FROM ORDRE O2 WHERE S.NOS=O2.NOS AND O2.NOP=P.NOP)) ;

II.

1) SELECT INTITULE, MAX(POIDS) FROM SERVICE S, ORDRE O, PIECE P WHERE S.NOS=O.NOS AND O.NOP=P.NOP AND COULEUR='bleu' GROUP BY INTITULE ;
2) SELECT AVG(POIDS) FROM SERVICE S, ORDRE O, PIECE P WHERE S.NOS=O.NOS AND O.NOP=P.NOP AND INTITULE=’Promotion’ GROUP BY S.NOS ;
3) SELECT P.NOP FROM PIECE P WHERE 3 < (SELECT COUNT(DISTINCT NOS) FROM ORDRE O WHERE O.NOP=P.NOP) ;
4) SELECT MAX(SUM(QUANTITE)) FROM ORDRE GROUP BY NOS ;