

4 Le langage SQL

- *Structured Query Language*
- Norme établie pour SGBD relationnel
- Partie LDD (DDL)
 - Conceptuel : CREATE SCHEMA, TABLE,...
 - Externe : CREATE VIEW, GRANT,...
 - Interne : CREATE INDEX, CLUSTER,...
- Partie LMD (DML)
 - SELECT, INSERT, DELETE, UPDATE

Quelques liens

- Standard SQL:
http://www.jcc.com/SQLPages/jccs_sql.htm
- <http://www.cssinfo.com/ncitsgate.html>
- Tutoriel et liens:
<http://www.contrib.andrew.cmu.edu/~shadow/sql.html>
- Syntaxe SQL2 :
<http://www.contrib.andrew.cmu.edu/~shadow/sql/sql2bnf.aug92.txt>
- Syntaxe SQL:1999 : <http://www.mkp.com/sql99>
- <http://www.sql-zone.com/>

Origine

- IBM Research (San Jose)
 - développement du prototype System/R
 - (Astrahan et al., 1976)
- SQUARE
 - algèbre relationnelle ~ anglais
 - (Boyce, Chamberlin, King & Hammer, 1975)
- SEQUEL
 - *Structured English QUERy Language*
 - (Chamberlin, Astrahan, Eswaran, Chamberlin, Griffiths & Lorie, 1976)

Standard ANSI/ISO

- SQL86
 - version préliminaire
- SQL89 (SQL, SQL1)
 - niveau minimal supporté
- SQL92 (SQL2)
 - support accru de l'intégrité
 - trois niveaux : entrée (SQL1^+), intermédiaire, complet
 - de plus en plus supporté
- SQL:1999 (SQL3)
 - extensions objet (UDT), TRIGGER, ROLE, SQL/PSM, ...
 - support très variable

4.1 Spécification du schéma relationnel avec SQL (LDD SQL)

■ Niveau conceptuel

- Schéma des tables (TABLE)
 - CREATE TABLE.
- Domaines (DOMAIN)
 - SQL2 intermédiaire : CREATE DOMAIN
- Contraintes d'intégrité
 - PRIMARY KEY, FOREIGN KEY, UNIQUE KEY, CHECK, ASSERTION, TRIGGER

Spécification du schéma relationnel avec SQL (suite)

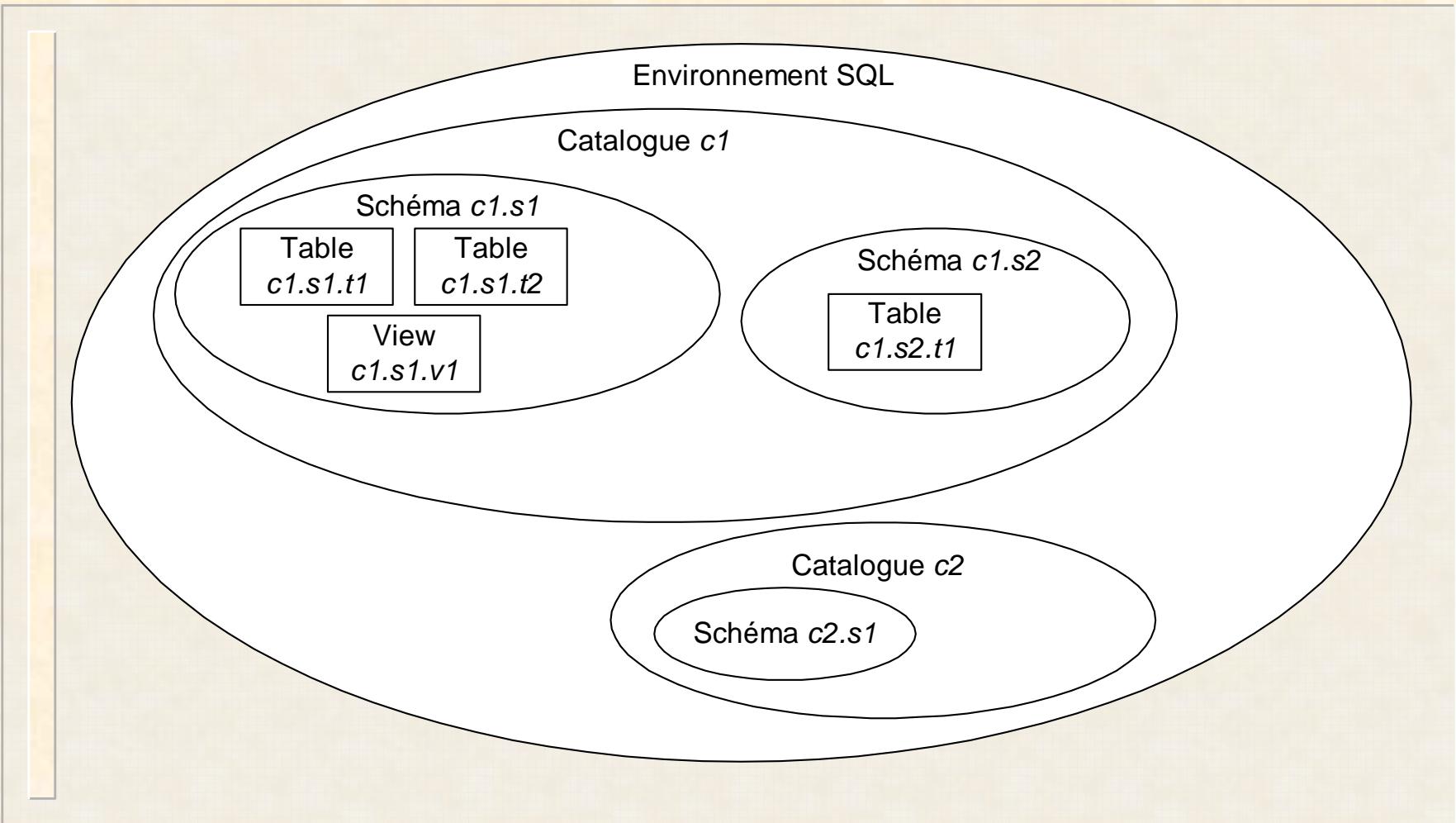
■ Niveau externe

- Vues (VIEW)
 - définie à partir d'autres tables (CREATE VIEW). Elle apparaît à l'utilisateur comme une table normale alors qu'elle est en réalité dérivée à partir d'autres tables normales ou virtuelles.
- Privilèges d'accès
 - GRANT

■ Niveau interne

- non standardisé (e.g. CREATE INDEX)

4.1.1 Environnement, catalogue, schéma et utilisateur SQL



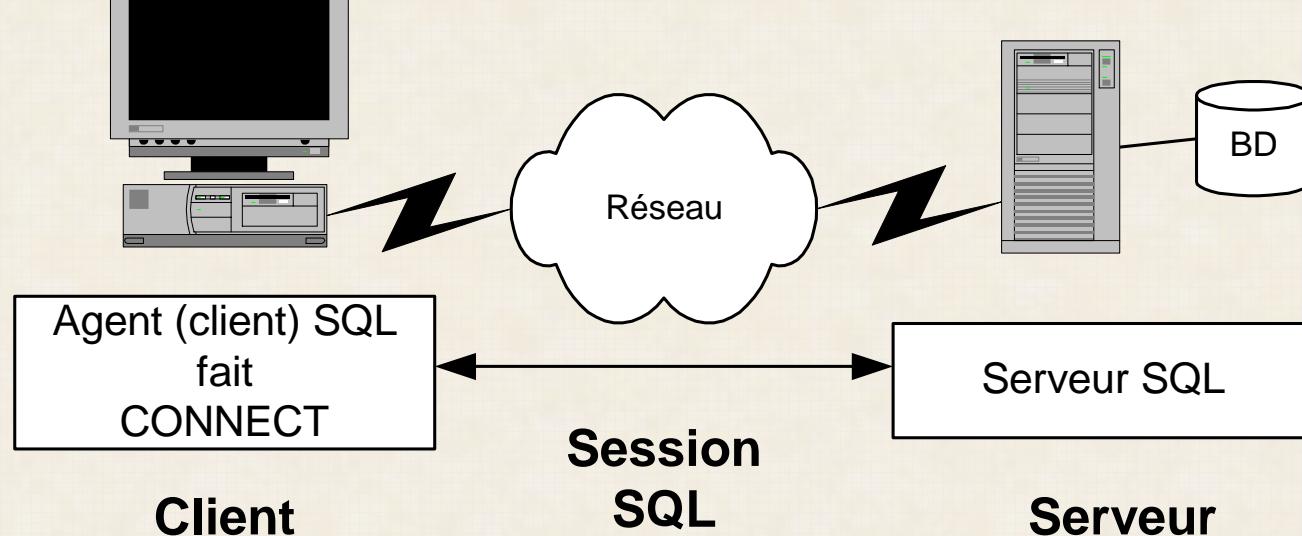
Utilisateur (user)

- Identificateur d 'utilisateur
 - *authorizationID*
 - non standardisé
- Mécanisme d 'authentification
 - e.g. mot de passe
- Utilisateur possède privilèges

Dialecte Oracle

- DATABASE ~ catalogue
 - une instance Oracle monte une DATABASE à la fois
- Nom du SCHEMA
 - = *authorizationID* du propriétaire
- CLUSTER Oracle
 - ≠ cluster de l'environnement SQL2
 - = méthode d'organisation de données par grappe

Agent (client), serveur, session SQL



- CONNECT spécifique
 - identification/authentification de l'utilisateur (*authorizationId/mot de passe*)
 - identification du serveur SQL
 - SCHEMA, TIME ZONE, CHARACTER SET
 - valeurs de défaut

Syntaxe des identificateurs

- Régulier
 - Max. 128 caractères
 - lettres, chiffres, souligné (_)
 - Débute par une lettre
 - Pas de mot réservé
 - SELECT, CREATE, TABLE, ORDER, ...
 - Insensible à la casse (conversion en majuscules)
- Délimité
 - CREATE TABLE "TABLE" ...

4.1.2 Creation du schema d'une table en SQL (CREATE TABLE)

■ Forme simple

```
CREATE TABLE Client  
  (noClient      INTEGER,  
   nomClient     VARCHAR(15),  
   noT l phone    VARCHAR(15)  
 )
```

■ Transmise   l'interpr te du LDD

- v rification
- cr ation de la table
 - sch ma stock  dans dictionnaire de donn es
 - allocation des structures physiques
 - clause non standardis e pour organisation primaire

Syntaxe générale du CREATE TABLE

```
CREATE TABLE nomDeLaTable  
  (spécificationDeColonne,  
   [ ,spécificationDeColonne]...  
   [ ,spécificationDeContrainte]... )
```

■ Syntaxe de *spécificationDeColonne*

```
nomColonne [ type|domaine ] [ DEFAULT valeurDeDéfaut ]  
[ NULL | NOT NULL ] [ UNIQUE | PRIMARY KEY ]  
[ REFERENCES nomTable[ listeColonnes ] ]  
[ [ CONSTRAINT nomContrainte ] CHECK ( conditionSQL ) ]
```

■ Syntaxe de *spécificationDeContrainte*

```
[ CONSTRAINT nomContrainte ]  
{ PRIMARY KEY listeColonnes |  
  FOREIGN KEY listeColonnes REFERENCES nomTable[ listeColonnes ]  
  [ MATCH { PARTIAL|FULL } ]  
  [ ON DELETE { NO ACTION|CASCADE|SET NULL|SET DEFAULT } ]  
  [ ON UPDATE { NO ACTION|CASCADE|SET NULL|SET DEFAULT } ] |  
  CHECK ( conditionSQL )  
}  
[ [ NOT ] DEFERRABLE INITIALLY { DEFERRED|IMMEDIATE } ]
```

Exemple *VentesPleinDeFoin* (script Oracle)

```
CREATE TABLE Client
  (noClient      INTEGER          NOT NULL,
   nomClient     VARCHAR(20)      NOT NULL,
   noTéléphone   VARCHAR(15)      NOT NULL,
   PRIMARY KEY    (noClient)
  )

CREATE TABLE Article
  (noArticle      INTEGER          NOT NULL,
   description    VARCHAR(20),
   prixUnitaire   DECIMAL(10,2)    NOT NULL,
   quantitéEnStock INTEGER         DEFAULT 0 NOT NULL
     CHECK (quantitéEnStock >= 0),
   PRIMARY KEY    (noArticle)
  )

CREATE TABLE Commande
  (noCommande     INTEGER          NOT NULL,
   dateCommande   DATE            NOT NULL,
   noClient       INTEGER          NOT NULL,
   PRIMARY KEY    (noCommande),
   FOREIGN KEY    (noClient) REFERENCES Client
  )

CREATE TABLE LigneCommande
  (noCommande     INTEGER          NOT NULL,
   noArticle      INTEGER          NOT NULL,
   quantité       INTEGER          NOT NULL
     CHECK (quantité > 0),
   PRIMARY KEY    (noCommande, noArticle),
   FOREIGN KEY    (noCommande) REFERENCES Commande,
   FOREIGN KEY    (noArticle) REFERENCES Article
  )

CREATE TABLE Livraison
  (noLivraison    INTEGER          NOT NULL,
   dateLivraison  DATE            NOT NULL,
   PRIMARY KEY    (noLivraison)
  )

CREATE TABLE DétailLivraison
  (noLivraison    INTEGER          NOT NULL,
   noCommande     INTEGER          NOT NULL,
   noArticle      INTEGER          NOT NULL,
   quantitéLivrée INTEGER         NOT NULL
     CHECK (quantitéLivrée > 0),
   PRIMARY KEY    (noLivraison, noCommande, noArticle),
   FOREIGN KEY    (noLivraison) REFERENCES Livraison,
```

4.1.3 Types SQL

■ Numérique exact

- INTEGER (ou INT)
 - Entier (précision non standardisée)
 - Exemples : 2, 3, 459
- SMALLINT
 - Petit entier (précision non standardisée)
 - Exemples : 2, 3, 459
- NUMERIC(p, c) (ou DECIMAL(p, c) ou DEC(p, c))
 - Nombre décimal avec p chiffres significatifs (excluant le point) et c chiffres après le point
 - Exemples : 2.5, 456.342, 6

Types SQL (suite)

■ *Numérique approximatif*

- REAL
 - Point flottant (précision non standardisée)
 - Exemples : 3.27E-4, 24E5
- DOUBLE PRECISION
 - Point flottant à double précision (non standardisée)
 - Exemples : 3.27265378426E-4, 24E12
- FLOAT(n)
 - Point flottant
 - précision minimale est de n chiffres pour la mantisse
 - Exemples : 3.27E-4, 24E5

Types SQL (suite)

- *Chaîne de caractères* (VARYING et NATIONAL : SQL2 intermédiaire)
 - CHARACTER(n) (ou CHAR(n))
 - Chaîne de caractère de taille fixe égale à n
 - Exemples : 'G. Lemoyne-Allaire', 'Paul L"Heureux'
 - CHARACTER VARYING (n) (ou VARCHAR(n))
 - Taille variable (max de n caractères)
 - Mécanismes d 'internationalisation
 - Ensemble de caractères par défaut
 - NATIONAL CHARACTER(n)
 - Ensemble de caractères alternatif spécifique à l'implémentation
 - NATIONAL CHARACTER VARYING(n)
 - Taille variable
 - Crédit d 'ensembles de caractères alternatifs
 - CREATE CHARACTER SET
 - COLLATION : relation d 'ordre des caractères

Types SQL (suite)

- *Date et temps* (SQL2 intermédiaire; précision p : SQL2 complet)
 - DATE
 - année (quatre chiffres), mois (2 chiffres) et jour (2 chiffres)
 - Exemple : DATE '1998-08-25'
 - TIME[(p)]
 - heure (2 chiffres), minutes (2 chiffres), secondes (2 + p chiffres)
 - Exemple : TIME '14:04:32.25'
 - TIMESTAMP[(p)]
 - DATE + TIME
 - Exemple : TIMESTAMP '1998-08-25 14:04:32.25'
 - INTERVAL
 - Représente un intervalle de temps
 - Exemple : INTERVAL '2' DAY (intervalle de deux jours)

Types SQL (suite)

- *Booléen* (SQL2 complet)
 - BIT (n)
 - Vecteur de n bits.
 - Exemples : B'00100110', X'9F'
 - BIT VARYING (n)
 - taille variable (max = n)
- *Données de grande taille* (LOB SQL:1999)
 - BINARY LARGE OBJECT (n) (BLOB(n))
 - n : taille en octets (ex: 1024, 5K, 3M, 2G)
 - Exemple : X '52CF4 ' (hexadecimal)
 - CHARACTER LARGE OBJECT (n) (CLOB(n))
 - NATIONAL CHARACTER LARGE OBJECT (n) (NCLOB(n))

UDT (*User Defined Type*)

SQL:1999

- Voir chapitre 17

Dialecte Oracle

- **NUMBER($p,[c]$)**
 - numérique exact; p entre 1 et 38
 - c doit être entre -84 et +127 (défaut, $c=0$)
 - valeur négative signifie un arrondissement.
- **VARCHAR2(n) : $n \leq 4000$**
- **RAW(n)**
 - Binaire de taille n octets ($n \leq 2000$).
- **LONG(n)**
 - Chaîne de caractères de taille variable ($n \leq 2G$)
 - Maximum une colonne LONG par table
- **LONG RAW(n)**
 - Binaire de taille variable ($n \leq 2G$).
 - Maximum une colonne de type LONG RAW par table

Dialecte Oracle (suite)

- ROWID : identifiant de ligne composé de
 - identificateur de fichier
 - identificateur de bloc relatif au fichier
 - identificateur de ligne relatif au bloc
- UROWID
 - identificateur universel de ligne (à partir de la version 8.1).
 - distingue index primaire (ORGANIZATION INDEX)
- Conversions implicites

Type SQL2	Type Oracle
CHARACTER (<i>n</i>), CHAR (<i>n</i>)	CHAR (<i>n</i>)
NUMERIC (<i>p,s</i>), DECIMAL (<i>p,s</i>), DEC (<i>p,s</i>)	NUMBER (<i>p,s</i>)
INTEGER, INT, SMALLINT	NUMBER (38)
FLOAT (<i>p</i>)	FLOAT (<i>p</i>)
REAL	FLOAT (63)
DOUBLE PRECISION	FLOAT (126)
VARCHAR(<i>n</i>), CHARACTER VARYING(<i>n</i>)	VARCHAR2 (<i>n</i>)

Dialecte Oracle (suite)

- DATE
 - ~TIMESTAMP SQL2
- Mécanisme d 'internationalisation
 - Paramètre de configuration NLS_LANG
 - CHARACTER SET
 - DATE_FORMAT
 - ...
- ALTER SESSION
 - pour modifier

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY'
```
- LOB : taille max 4G
- BFILE : fichier externe

4.1.4 Suppression d'une table (DROP TABLE)

```
DROP TABLE nomTable [ RESTRICT | CASCADE ]
```

■ RESTRICT

- rejet si élément dépendant existe
 - ex: FOREIGN KEY

■ CASCADE

- supprime élément dépendant

4.1.5 Modification du schéma de table (ALTER TABLE)

■ Syntaxe

```
ALTER TABLE nomTable
  {ADD COLUMN spécificationColonne |
  DROP COLUMN nomColonne [RESTRICT|CASCADE] |
  ADD spécificationContrainte|
  DROP nomContrainte [RESTRICT|CASCADE] |
  ALTER nomColonne SET DEFAULT valeurDéfaut |
  ALTER nomColonne DROP DEFAULT}
```

```
ALTER TABLE Client
ADD COLUMN age INTEGER CHECK(age >0 )
```

4.1.6 Le dictionnaire de données SQL (INFORMATION_SCHEMA)

- Normalisé en SQL2
- BD relationnelle
 - contient les méta-données d'un CATALOG
- DEFINITION_SCHEMA
 - tables
- INFORMATION_SCHEMA
 - VIEWS sur les tables du
DEFINITION_SCHEMA

Exemples de VIEWS du INFORMATION_SCHEMA

- SCHEMATA
 - les SCHEMA créés par CURRENT_USER
- DOMAINS
 - les DOMAIN accessibles par CURRENT_USER ou PUBLIC
- TABLES
 - les noms des tables accessibles par CURRENT_USER ou PUBLIC
- VIEWS
 - les vues accessibles par CURRENT_USER ou PUBLIC
- COLUMNS
 - les colonnes des TABLE accessibles par CURRENT_USER ou PUBLIC
- TABLE_CONSTRAINTS
 - contraintes des TABLE créées par CURRENT_USER
- CHECK_CONSTRAINTS
 - contraintes CHECK des TABLE créées par CURRENT_USER
- ASSERTIONS
 - ASSERTION créées par CURRENT_USER
- TABLE_PRIVILEGES
 - privilèges accordés par CURRENT_USER, à CURRENT_USER, ou à PUBLIC

Dictionnaire de données Oracle avec SQL*plus

```
SQL> CREATE TABLE Client  
  2  (noCLIENT    INTEGER,  
  3   nomClient    VARCHAR(15),  
  4   noTéléphone  VARCHAR(15))  
  5 /
```

Table créée.

```
SQL> SELECT Table_Name  
  2  FROM    USER_TABLES  
  3 /
```

TABLE_NAME

CLIENT

```
SQL> SELECT Column_Name, Data_Type  
  2  FROM    USER_TAB_COLUMNS  
  3  WHERE   Table_Name = 'CLIENT'  
  4 /
```

COLUMN_NAME	DATA_TYPE
-----	-----
-----	-----
NOCLIENT	NUMBER
NOMCLIENT	VARCHAR2
NOTÉLÉPHONE	VARCHAR2

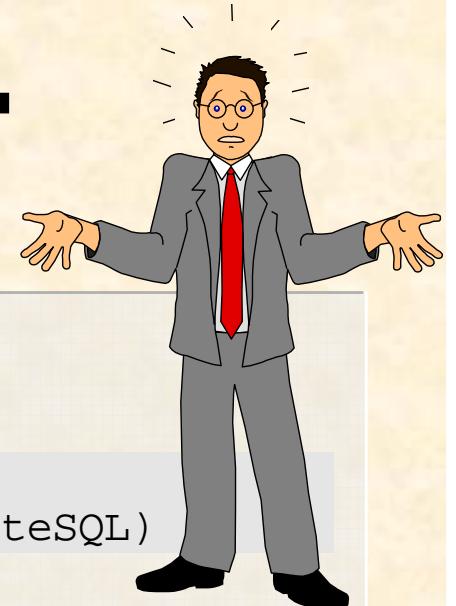
Recherche d 'une table du dictionnaire de données

```
SQL> SELECT Table_Name  
  2  FROM DICTIONARY  
  3  WHERE Table_Name like '%TABLE%'  
  4  /
```

TABLE_NAME
ALL_ALL_TABLES
ALL_NESTED_TABLES
ALL_OBJECT_TABLES
ALL_PART_TABLES
ALL_TABLES
ALL_UPDATABLE_COLUMNS
USER_ALL_TABLES
USER_NESTED_TABLES
USER_OBJECT_TABLES
USER_PART_TABLES
USER_QUEUE_TABLES
USER_TABLES
USER_TABLESPACES
USER_UPDATABLE_COLUMNS
TABLE_PRIVILEGES

15 ligne(s) sélectionnée(s).

4.2 Requêtes SQL (SELECT)



■ Syntaxe de *requêteSQL*

```
selectSQL |  
(requêteSQL) {UNION|INTERSECT|EXCEPT} (requêteSQL)
```

■ Syntaxe du *selectSQL*

```
SELECT      {[ALL|DISTINCT] expression [AS nomColonne]  
           [,expression [AS nomColonne]]...} | *  
FROM        table [AS nomTable [(nomColonne[,nomColonne])]]]  
           [,table [AS nomTable [(nomColonne[,nomColonne])]]]...]  
[WHERE      conditionSQL]  
[GROUP BY   nomColonne [,nomColonne]...  
[HAVING     conditionSQL]  
[ORDER BY   nomColonne [ASC|DESC] [,nomColonne[ASC|DESC]]...]  
...
```

4.2.1 Projection d'une table et la clause DISTINCT

- Produire les *noClient* et *dateCommande* de toutes les *Commandes*

```
SELECT      noClient, dateCommande  
FROM        Commande
```

noClient	dateCommande
10	01/06/2000
20	02/06/2000
10	02/06/2000
10	05/07/2000
30	09/07/2000
20	09/07/2000
40	15/07/2000
40	15/07/2000

Multi-ensemble !

```
SELECT      ALL noClient, dateCommande  
FROM        Commande
```

Clause DISTINCT

- Produire les *noClient* et *dateCommande* de toutes les *Commandes*

```
SELECT      DISTINCT noClient, dateCommande  
FROM        Commande
```

noClient	dateCommande
10	01/06/2000
20	02/06/2000
10	02/06/2000
10	05/07/2000
30	09/07/2000
20	09/07/2000
40	15/07/2000

$$\pi_{noClient, dateCommande} (Commande)$$

4.2.2 Sélection sur une table (WHERE)

- Sélectionner les *Articles* dont le prix est inférieur à \$20.00 et le numéro est supérieur à 30

```
SELECT      *
FROM        Article
WHERE       prixUnitaire < 20 AND noArticle > 30
```

noArticle	description	prixUnitaire
60	Erable argenté	15.99
70	Herbe à puce	10.99
95	Génévrier	15.99

$\sigma_{prixUnitaire < 20.00 \text{ ET } noArticle > 30} (Article)$

Syntaxe de *conditionSQL*

```
{conditionSimple  
  (conditionSQL) |  
  NOT(conditionSQL) |  
  conditionSQL AND conditionSQL |  
  conditionSQL OR conditionSQL}
```

■ Syntaxe (incomplète) de la *conditionSimple* :

```
{expression {=|<|>|<=|>=|>>} expression |  
  expression BETWEEN expression AND expression |  
  expression {IS NULL | IS NOT NULL} |  
  expression {IN | NOT IN} listeConstantes |  
  expression {LIKE | NOT LIKE} patron}
```

ConditionSQL - BETWEEN

- Sélectionner les *Commandes* du mois de juin de l'année 2000

```
SELECT      *
FROM        Commande
WHERE       dateCommande BETWEEN '01/06/2000' AND '30/06/2000'
```

```
SELECT      *
FROM        Commande
WHERE       dateCommande >= '01/06/2000' AND
            dateCommande <= '30/06/2000'
```

ConditionSQL - IN

- Sélectionner les *Commandes* du *Client* dont le *noClient* est 10 ou 40 ou 80

```
SELECT      *
FROM        Commande
WHERE       noClient IN (10, 40, 80)
```

```
SELECT      *
FROM        Commande
WHERE       noClient = 10 OR noClient = 40 OR noClient = 80
```

ConditionSQL - LIKE

- Sélectionner les *Clients* dont le *nomClient* contient le mot *Le*

```
SELECT *
FROM Client
WHERE nomClient LIKE '%Le%'
```

- 2ième lettre du *nomClient* = *o* et dernière lettre est un *k*

```
SELECT *
FROM Client
WHERE nomClient LIKE '_o%k'
```

ConditionSQL - IS NOT NULL

- Sélectionner les *Articles* dont la description n'est pas une valeur nulle

```
SELECT      *
FROM        Article
WHERE       description IS NOT NULL
```

4.2.3 Sélection-projection sur une table

- Produire les *noClient* et *dateCommande* des *Commandes* dont la date est supérieure au *05/07/2000*

```
SELECT      noClient , dateCommande  
FROM        Commande  
WHERE       dateCommande > '05/07/2000'
```

noClient	dateCommande
30	09/07/2000
20	09/07/2000
40	15/07/2000
40	15/07/2000

Laboratoire
Créer le schéma de la BD
PleinDeFoin :
SchemaVentesPleinDeFoin.sql
Exercices 1 a), b) , c) , n), o)

4.2.4 Produit cartésien avec SELECT-FROM

- Produire toutes les combinaisons possibles de lignes de *Client* et de *Commande*...

```
SELECT      *
FROM        Client , Commande
```

Client × *Commande*

4.2.5 Jointure naturelle avec SELECT-FROM-WHERE

- Produire les informations au sujet des *Clients* et de leurs *Commandes*

```
SELECT      Client.noClient, nomClient, noTéléphone, noCommande,  
            dateCommande  
FROM        Client, Commande  
WHERE       Client.noClient = Commande.noClient
```

Client.noClient	nomClient	noTéléphone	noCommande	dateCommande
10	Luc Sansom	(999)999-9999	1	01/06/2000
10	Luc Sansom	(999)999-9999	3	02/06/2000
10	Luc Sansom	(999)999-9999	4	05/07/2000
20	Dollar Tremblay	(888)888-8888	2	02/06/2000
20	Dollar Tremblay	(888)888-8888	6	09/07/2000
30	Lin Bô	(777)777-7777	5	09/07/2000
40	Jean Leconte	(666)666-6666	7	15/07/2000
40	Jean Leconte	(666)666-6666	8	15/07/2000

$\pi_{Client.noClient, nomClient, noTéléphone, noCommande, dateCommande}$

$(\sigma_{Client.noClient = Commande.noClient} (Client \times Commande))$

4.2.6 Jointure avec JOIN (SQL2 intermédiaire)

- Produire les informations au sujet des *Clients* et de leurs *Commandes*

```
SELECT      *
FROM        Client NATURAL JOIN Commande {SQL2}
```

Client  *Commande*

```
Client NATURAL JOIN Commande {Illégal!}
```

- Jointure- θ (si noms de colonnes de jointure sont différents)

```
SELECT      *
FROM        Client JOIN Commande ON
                    Client.noClient = Commande.numéroCLient {SQL2}
```

4.2.7

Jointure de plusieurs tables

- Sélectionner les *nomClient* des *Clients* qui ont commandé au moins un plant d'herbe à puce

```
SELECT      nomClient
FROM        Client, Commande, LigneCommande, Article
WHERE       description = 'Herbe à puce' AND
           Client.noClient = Commande.noClient AND
           Commande.noCommande = LigneCommande.noCommande AND
           LigneCommande.noArticle = Article.noArticle
```

$$\pi_{nomClient} (\sigma_{description = "Herbe à puce"} (Client \downarrow Commande \downarrow LigneCommande \downarrow Article))$$

4.2.8 Formulations équivalentes, performance et indépendance des données

- ~Algèbre relationnelle

```
SELECT      nomClient
FROM        Client, Commande, LigneCommande, Article
WHERE       description = 'Herbe à puce' AND
           Client.noClient = Commande.noClient AND
           Commande.noCommande = LigneCommande.noCommande AND
           LigneCommande.noArticle = Article.noArticle
```

- Ordre quelconque des tables du FROM
 - la plupart du temps...
- AND commutatif...
- Processus d 'évaluation de requête

4.2.9 Définition d'un alias (clause AS)

- ~ renommer (ρ)

```
SELECT      Client.noClient, nomClient, noTéléphone, noCommande,  
            dateCommande  
FROM        Client, Commande  
WHERE       Client.noClient = Commande.noClient
```

```
SELECT      Cl.noClient, nomClient, noTéléphone, noCommande,  
            dateCommande  
FROM        Client AS Cl, Commande AS Co  
WHERE       Cl.noClient = Co.noClient
```

Laboratoire
Exercices 1 d) e) f) h)

Auto-jointure

- Quels sont les *Clients* qui ont le même numéro de téléphone?

```
SELECT      Client.noClient, Client2.noClient  
FROM        Client, Client AS Client2  
WHERE       Client.noTéléphone = Client2.noTéléphone
```

$$\pi_{Client.noClient, Client2.noClient, } (\sigma_{Client.noTéléphone = Client2.noTéléphone} (Client \times \rho_{Client2} (Client)))$$

```
SELECT      noClient, noClient2  
FROM        Client NATURAL JOIN {SQL2}  
Client AS Client2(noClient2, nomClient2, noTéléphone)
```

$$\pi_{noClient, noClient2} (Client \downarrow \rho_{Client2(noClient2, nomClient2, noTéléphone)} (Client))$$

4.2.10 Jointure externe (OUTER JOIN)

- Produire les informations au sujet des *Clients* et de leurs *Commandes* incluant les informations sur les *Clients* qui n'ont pas placé de *Commande*

Client = \downarrow *Commande*

```
SELECT      *
FROM        Client NATURAL LEFT OUTER JOIN Commande {SQL2}
```

- Oracle
 - « + » après colonne pour inclure la valeur NULL

```
SELECT      *
FROM        Client, Commande
WHERE      Client.noClient = Commande.noClient(+)
```

4.2.11 Opérations ensemblistes (UNION, INTERSECT, EXCEPT)

- Produire les noms et numéros de téléphone des *Employés* qui sont aussi des *Clients* de la pépinière

Table <i>Client</i>		
noClient	nomClient	noTéléphone
10	Luc Sansom	(999)999-9999
20	Dollard Tremblay	(888)888-8888
30	Lin Bô	(777)777-7777
40	Jean Leconte	(666)666-6666
50	Hafedh Alaoui	(555)555-5555
60	Marie Leconte	(666)666-6666
70	Simon Lecoq	(444)444-4419
80	Dollard Tremblay	(333)333-3333

Table <i>Employé</i>		
codeEmployé	nomEmployé	noTéléphone
CASD1	Dollard Tremblay	(888)888-8888
PIOY1	Yan Piochuneshot	911
LAFH1	Yvan Lafleur	(111)111-1111
HASC1	Jean Leconte	(666)666-6666

```
( SELECT nomClient as nomPersonne, noTéléphone
  FROM Client)
INTERSECT
( SELECT nomEmployé as nomPersonne, noTéléphone
  FROM Employé)
```

nomPersonne	noTéléphone
Dollard Tremblay	(888)888-8888
Jean Leconte	(666)666-6666

Laboratoire
Exercices 1 g) i) j)

4.2.12 Expressions générales sur les colonnes

- La liste des *noArticle* avec le *prixUnitaire* avant et après inclusion de la taxe de 15%

```
SELECT noArticle, prixUnitaire, prixUnitaire*1.15 AS prixPlusTaxe  
FROM Article
```

noArticle	prixUnitaire	prixPlusTaxe
10	10.99	12.64
20	12.99	14.94
40	25.99	29.89
50	22.99	26.44
60	15.99	18.39
70	10.99	12.64
80	26.99	31.04
81	25.99	29.89
90	25.99	29.89
95	15.99	18.39

4.2.12 Expressions (suite)

- Produire le détail de chacun des *Articles* commandés la *Commande #1* incluant le prix total avant et après la taxe de 15% pour chacun des *Articles* commandés

```
SELECT L.noArticle, quantité, prixUnitaire, prixUnitaire*quantité AS total,  
      prixUnitaire*quantité*1.15 AS totalPlusTaxe  
  FROM LigneCommande AS L, Article AS A  
 WHERE L.noArticle = A.noArticle AND  
       L.noCommande = 1
```

noArticle	quantité	prixUnitaire	total	totalPlusTaxe
10	10	10.99	109.90	126.38
70	5	10.99	54.95	63.19
90	1	25.99	25.99	29.89

Expression sur colonne du WHERE

- Les *Articles* dont le *prixUnitaire* incluant la taxe de 15% est inférieur à \$16.00

```
SELECT noArticle, prixUnitaire, prixUnitaire*1.15 AS prixPlusTaxe  
FROM Article  
WHERE prixUnitaire*1.15 < 16
```

noArticle	prixUnitaire	prixPlusTaxe
10	10.99	12.64
20	12.99	14.94
70	10.99	12.64

Opérateurs

Symbol	Signification
+	Somme
-	Différence
*	Produit
/	Division
	Concaténation de chaîne (SQL2)

- Conversions automatiques entre types compatibles

Pseudo-colonnes

- Les *Commandes* de la journée

```
SELECT *  
FROM Commande  
WHERE dateCommande = CURRENT_DATE
```

- CURRENT TIME
- CURRENT TIMESTAMP
 - SYSDATE Oracle
- CURRENT_USER (ou USER)
- SESSION_USER

Priorité en ordre décroissant

+ , - (unaire)
* , /
+ , - ,
= , != , < , > , <= , >= , IS NULL , LIKE , BETWEEN , IN, SIMILAR
NOT
AND
OR

Sélection par un CASE

- Produire la quantité qui a été livrée pour l'*Article #50* de la *Commande #4*

Table <i>DétailLivraison</i>			
noLivraison	noCommande	noArticle	quantitéLivrée
100	1	10	7
100	1	70	5
101	1	10	3
102	2	40	2
102	2	95	1
100	3	20	1
103	1	90	1
104	4	40	1
105	5	70	2

```
SELECT
    CASE
        WHEN SUM(quantitéLivrée) IS NULL THEN 0
        ELSE SUM(quantitéLivrée)
    END AS quantitéTotaleLivrée
FROM    DétailLivraison
WHERE   noArticle = 50 AND noCommande = 4
```

quantitéTotaleLivrée
0

Quelques fonctions SQL2

- POSITION(patron IN chaîne)
- CHARACTER_LENGTH(chaîne)
- OCTET_LENGTH (chaîne)
- BIT_LENGTH(chaîne)
- EXTRACT(champ FROM dateOuTime)
- SUBSTRING (chaîne FROM indiceDébut FOR nombreCaractères)
- UPPER | LOWER (chaîne)
- TRIM ([LEADING|TRAILING|BOTH] caractère FROM chaîne)
- CAST(expression AS type)
- ...
- Voir documentation du SGBD

Expressions de DATE Oracle

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
-----
02-02-05

SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MM-YYYY HH24:MI:SS';

Session altered.

SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
-----
05-02-2002 09:08:26

SQL> SELECT TO_DATE('05/02/2000', 'DD/MM/YYYY') FROM DUAL;

TO_DATE('05/02/2000'
-----
05-02-2000 00:00:00

SQL> SELECT TO_CHAR(SYSDATE, 'DD/MM/YYYY') FROM DUAL;

TO_CHAR(SY
-----
22/01/2002
```

Opérations sur DATE

```
SQL> SELECT SYSDATE + INTERVAL '1' DAY FROM DUAL;  
  
SYSDATE+INTERVAL'1'  
-----  
23-01-2002 16:02:18  
  
SQL> SELECT SYSDATE - INTERVAL '1' DAY FROM DUAL;  
  
SYSDATE-INTERVAL'1'  
-----  
21-01-2002 16:02:18  
  
SQL> SELECT SYSDATE + 1 FROM DUAL;  
  
SYSDATE+1  
-----  
23-01-2002 16:02:18  
  
SQL> SELECT SYSDATE + 1/24 FROM DUAL;  
  
SYSDATE+1/24  
-----  
22-01-2002 17:02:18  
  
SQL> SELECT SYSDATE + INTERVAL '30' SECOND FROM DUAL;  
  
SYSDATE+INTERVAL' 30  
-----  
22-01-2002 16:02:48
```

4.2.13 Expressions et conditions sur les valeurs nulles (NULL)

- Arithmétique
 - opérande NULL => NULL
- Comparaison (>, <, ...)
 - opérande NULL => UNKNOWN

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

NULL pour les nuls ?

- Si x est NULL

```
SELECT *
FROM T
WHERE x = 0 OR x <> 0
```

- UNKNOWN OR UNKNOWN = UNKNOWN
 - pas dans le résultat !

4.2.14 Fonctions de groupe

- Le nombre d'*Articles* différents à vendre ainsi que le *prixUnitaire* moyen des *Articles*

```
SELECT COUNT(*) AS nombreArticles,  
       AVG(prixUnitaire) AS prixMoyen  
FROM Article
```

nombreArticles	prixMoyen
10	19.49

suite

```
SELECT      Count(DISTINCT prixUnitaire) AS nombrePrix  
FROM        Article
```

nombrePrix

6

```
SELECT      Count(prixUnitaire) AS nombrePrixNotNull  
FROM        Article
```

nombrePrixNotNull

10

4.2.15 Partition d'une table avec la clause GROUP BY

- Produire le nombre de *Commandes* passées par chacun des *Clients* qui ont passé au moins une *Commande*

```
SELECT      noClient, COUNT( * ) AS nombreCommandes  
FROM        Commande  
GROUP BY    noClient
```

Table <i>Commande</i>		
noCommande	dateCommande	noClient
1	01/06/2000	10
3	02/06/2000	10
4	05/07/2000	10
2	02/06/2000	20
6	09/07/2000	20
5	09/07/2000	30
7	15/07/2000	40
8	15/07/2000	40

noClient	nombreCommandes
10	3
20	2
30	1
40	2

Pour chacune des *LigneCommande* pour lesquelles au moins une *Livraison* a été effectuée, produire le *noCommande* et *noArticle*, la quantité totale livrée et le nombre de *Livraisons* effectuées

```
SELECT      noCommande, noArticle, SUM(quantitéLivrée) AS totalLivré,
            COUNT(*) AS nombreLivraisons
FROM        DétailLivraison
GROUP BY    noCommande, noArticle
```

Table <i>DétailLivraison</i>			
noLivraison	noCommande	noArticle	quantitéLivrée
100	1	10	7
101	1	10	3
100	1	70	5
102	2	40	2
102	2	95	1
100	3	20	1
103	1	90	1
104	4	40	1
105	5	70	2

noCommande	noArticle	totalLivré	nombreLivraisons
1	10	10	2
1	70	5	1
1	90	1	1
2	40	2	1
2	95	1	1
3	20	1	1
4	40	1	1
5	70	2	1

Calcul de plusieurs agrégats à la fois avec CUBE et ROLLUP SQL:1999

4.2.16 Clause HAVING

- Produire le nombre de *Commandes* passées par chacun des *Clients* qui ont passé deux *Commandes* ou plus

```
SELECT      noClient , COUNT( * ) AS nombreCommandes  
FROM        Commande  
GROUP BY    noClient  
HAVING     COUNT( * ) >= 2
```

Table <i>Commande</i>		
noCommande	dateCommande	noClient
1	01/06/2000	10
3	02/06/2000	10
4	05/07/2000	10
2	02/06/2000	20
6	09/07/2000	20
5	09/07/2000	30
7	15/07/2000	40
8	15/07/2000	40

noClient	nombreCommandes
10	3
20	2
30	1
40	2

Produire le nombre de *Commandes* passées par chacun des *Clients* qui ont passé deux *Commandes* ou plus après le 02/06/2000

```

SELECT      noClient, COUNT( * ) AS nombreCommandes
FROM        Commande
WHERE       dateCommande > '02/06/2000'
GROUP BY    noClient
HAVING     COUNT( * ) >= 2
  
```

Table <i>Commande</i>		
noCommande	dateCommande	noClient
1	01/06/2000	10
3	02/06/2000	10
4	05/07/2000	10
2	02/06/2000	20
6	09/07/2000	20
5	09/07/2000	30
7	15/07/2000	40
8	15/07/2000	40

noClient	nombreCommandes
10	1
20	1
30	1
40	2

4.2.17 Tri du résultat (ORDER BY)

- Les *Clients* en ordre alphabétique du nom

```
SELECT      *
FROM        Client
ORDER BY    nomClient
```

```
SELECT      *
FROM        Client
ORDER BY    nomClient DESC, noTéléphone ASC
```

Laboratoire
Exercices 1 q) s) t) v)

4.2.18 SELECT imbriqué

4.2.18.1 Opération élément de (IN)

- Les *Clients* qui ont passé au moins une *Commande*

```
SELECT      *
FROM        Client
WHERE       noClient IN
            (SELECT  noClient
             FROM    Commande)
```

Client ▷ *Commande*

```
SELECT      DISTINCT Client.noClient, nomClient, noTéléphone
FROM        Client, Commande
WHERE       Client.noClient = Commande.noClient
```

Ligne à plusieurs colonnes

- Chercher les *LigneCommandes* pour lesquelles au moins une *Livraison* a été effectuée

```
SELECT      *
FROM        LigneCommande
WHERE       (noCommande,noArticle) IN
           (SELECT      noCommande, noArticle
            FROM        DétailLivraison)
```

4.2.18.2 SELECT imbriqué qui retourne une ligne

- Sélectionner les *Commandes* du *Client* Hugh Paycheck

```
SELECT      *
FROM        Commande
WHERE       noClient =
            (SELECT      noClient
             FROM        Client
             WHERE       nomClient = 'Hugh Paycheck' )
```

- Exception si plusieurs lignes retournées par SELECT imbriqué

4.2.18.3 SELECT imbriqué corrélé

- Produire les informations au sujet des *Clients* qui ont passé au moins une *Commande*

```
SELECT      *
FROM        Client
WHERE       0 <
           (SELECT      COUNT( * )
            FROM       Commande
            WHERE      noClient = Client.noClient)
```

Référence à une
colonne non locale

POUR chaque ligne de *Client*

Exécuter le SELECT suivant :

```
(SELECT      COUNT( * )
         FROM       Commande
         WHERE      noClient = Client.noClient)
```

SI le compte retourné > 0

Placer la ligne de *Client* dans la table du résultat à retourner

FIN SI

FIN POUR

4.2.18.4 Test d'ensemble vide (EXISTS)

- Produire les informations au sujet des *Clients* qui ont passé au moins une *Commande*

```
SELECT      *
FROM        Client
WHERE       EXISTS
           ( SELECT      *
             FROM        Commande
             WHERE       noClient = Client.noClient )
```

4.2.18.5 Test de double (UNIQUE)

- Vérifier s'il y a plus d'un *Client* qui porte le même nom
NOT UNIQUE
(SELECT nomClient FROM Client)

- *Clients* qui ont passé au moins deux *Commandes*

```
SELECT      *
FROM        Client
WHERE      NOT UNIQUE
          (SELECT      noClient
           FROM        Commande
           WHERE      noClient = Client.noClient)
```

4.2.18.6 Quantificateurs (ALL, SOME/ANY)

- *Commandes* passées après la dernière *Livraison* (date ultérieure)

```
SELECT * FROM Commande
WHERE dateCommande > ALL
  (SELECT dateLivraison
   FROM Livraison)
```

- *Commandes* passées après au moins une des *Livraisons*

```
SELECT * FROM Commande
WHERE dateCommande > ANY
  (SELECT dateLivraison
   FROM Livraison)
```

4.2.18.7 Test d'inclusion entre deux tables et division

- $T_1 \subseteq T_2$

NOT EXISTS (T_1 EXCEPT T_2)

Quelles sont les *noCommande* des *Commandes* qui incluent **tous** les *Articles* dont le *prixUnitaire* est \$10.99

noArticle
10
70



noCommande	noArticle
1	10
1	70
1	90
2	40
2	95
3	20
4	40
4	50
5	70
5	10
5	20
6	10
6	40
7	50
7	95

```

SELECT      noCommande
FROM        Commande
WHERE       NOT EXISTS
    ( (SELECT      noArticle
      FROM        Article
      WHERE       prixUnitaire = 10.99
    )
     EXCEPT
    (SELECT      noArticle
      FROM        LigneCommande
      WHERE       noCommande = Commande.noCommande
    )
  )
)

```

noCommande
1
5

Lab
Exercices 1 e) f) j) k) p) u)
m)

4.2.18.8 SELECT imbriqué dans le FROM

- Produire les *noClient* et *dateCommande* des *Commandes* dont la *dateCommande* est supérieure au *05/07/2000*

```
SELECT noClient, dateCommande  
FROM  
  (SELECT * {SQL 2}  
   FROM Commande  
   WHERE dateCommande > '05/07/2000'  
  )
```

Laboratoire
Exercices 2 a) b)

4.2.19 Récursivité en SQL:1999

```
WITH RECURSIVE Chemin(x,y) AS
    (SELECT x,y FROM Arc)
UNION
    (SELECT Chemin1.x, Chemin2.y
     FROM Chemin AS Chemin1, Chemin AS Chemin2
      WHERE Chemin1.y = Chemin2.x)
SELECT * FROM Chemin
```

Table <i>Arc</i>	
x	y
1	3
2	3
3	4
3	5
5	6

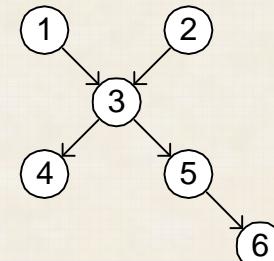


Table <i>Chemin</i>	
x	y
1	3
2	3
3	4
3	5
5	6
1	4
1	5
1	6
2	4
2	5
2	6
3	6

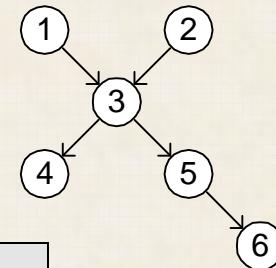
CONNECT BY Oracle

■ Chemins à partir de 1

```
SQL> SELECT y  
2  FROM Arc  
3  START WITH x=1 CONNECT BY PRIOR y = x  
4  /
```

Y

3
4
5
6



4.3 Opérations de mise à jour des tables en SQL

- Insert**

- Delete**

- Update**

4.3.1 Insertion dans une table (INSERT)

- Insérer une nouvelle ligne dans la table *Client*

```
INSERT INTO Client  
VALUES (100, 'G. Lemoine-Allaire', '911')
```

- Changer l'ordre de défaut

```
INSERT INTO Client(nomClient, noClient, noTéléphone)  
VALUES ('G. Lemoine-Allaire', 100, '911')
```

Insertion d 'une partie des colonnes

```
CREATE TABLE Article
  (noArticle          INTEGER      NOT NULL,
   description        VARCHAR(20),
   prixUnitaire       DECIMAL(10,2) NOT NULL,
   quantitéEnStock    INTEGER      DEFAULT 0 NOT NULL ,
   PRIMARY KEY (noArticle)
  )
```

```
INSERT INTO Article(noArticle, prixUnitaire)
VALUES (30, 5.99)
```

```
INSERT INTO Article(noArticle, description, prixUnitaire, quantitéEnStock)
VALUES (30, NULL, 5.99, 0)
```

Insertion à partir d 'un SELECT

- Produire les lignes de *DétailLivraison* pour la *Livraison* #106 à partir des *LigneCommandes* de la *Commande* #7

```
INSERT INTO DétailLivraison
    SELECT      106, noCommande, noArticle, quantité
    FROM        LigneCommande
    WHERE       noCommande = 7
```

4.3.2 Suppression de lignes (DELETE)

- Supprimer toutes les lignes de la table *Client*

```
DELETE FROM Client
```

- Supprimer le *Client* #70 de la table *Client*

```
DELETE      FROM Client  
WHERE      noClient = 70
```

- Supprimer les *Clients* qui n'ont pas passé de *Commande*

```
DELETE FROM Client  
WHERE noClient NOT IN  
(SELECT      DISTINCT noClient  
FROM        Commande )
```

4.3.3 Modification de lignes (UPDATE)

- Changer le *noTéléphone* du *Client* #10 pour (222)222-2222

```
UPDATE      Client
SET         noTéléphone = '(222)222-2222'
WHERE       noClient = 10
```

- Augmenter tous les *prixUnitaires* des *Articles* de 10%

```
UPDATE      Article
SET         prixUnitaire = prixUnitaire * 1.1
```

- Modification de plusieurs colonnes à la fois

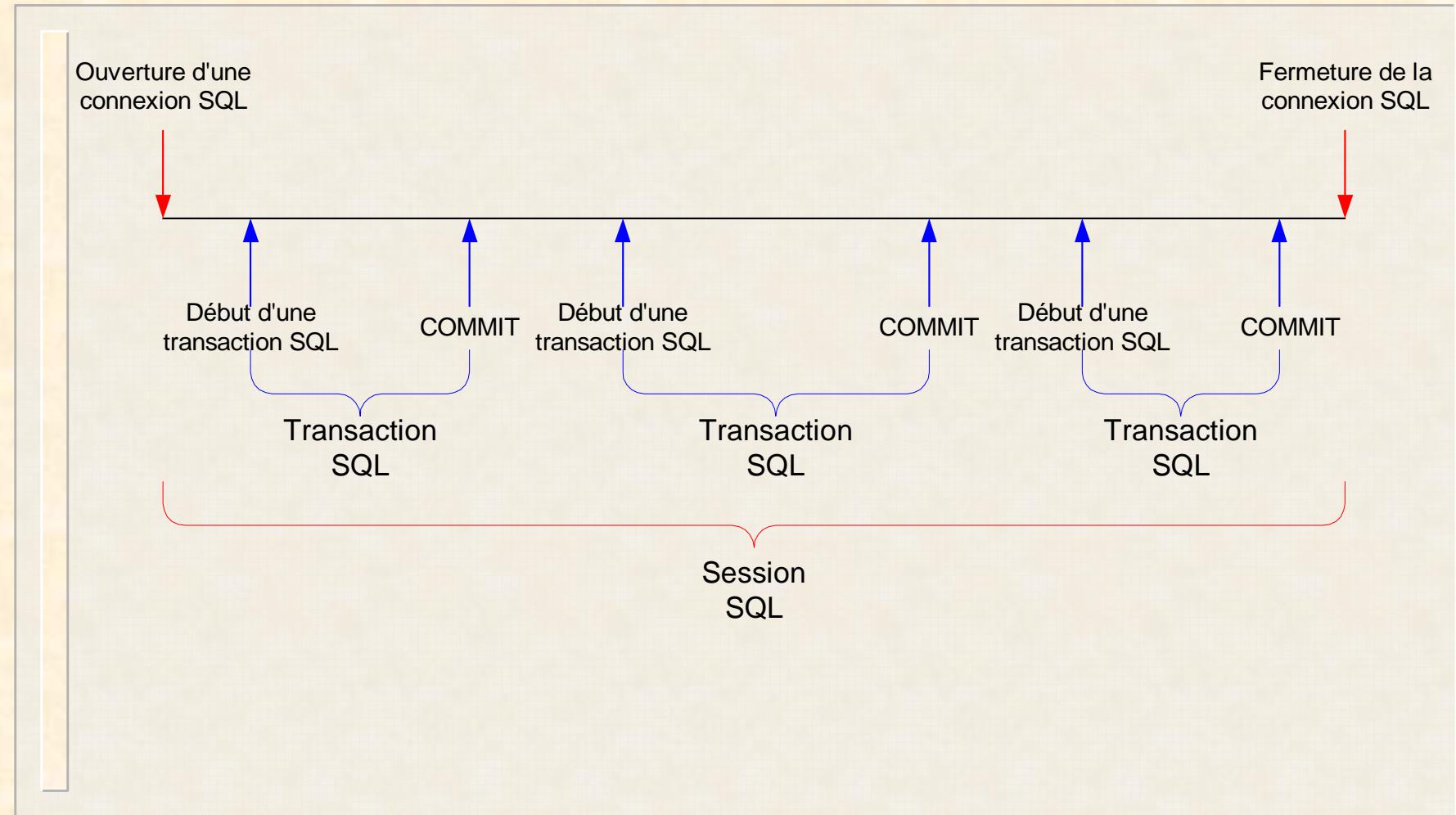
```
UPDATE      Article
SET         prixUnitaire = 12.99, quantitéEnStock = 5
WHERE       noArticle = 10
```

Laboratoire
Exercices 1 x) y) z)

4.3.4 Gestion des transactions en SQL

- COMMIT WORK
 - confirme la transaction en cours
- ROLLBACK WORK
 - annule la transaction en cours
- Début de transaction implicite
 - début de session
 - fin de la précédente
- Commande LDD provoque un COMMIT

Session et transaction



4.4 Niveau externe du schéma en SQL

- Gestion de la sécurité
 - GRANT
- Tables virtuelles
 - VIEWS

4.4.1 Sécurité en SQL (GRANT)

4.4.1.1 Identification et authentification

- Identification des utilisateurs
 - *authorizationID*
 - PUBLIC : tous les utilisateurs
- Authentification
 - mot de passe, ou ...
- Oracle
 - Utilisateurs administrateurs créés à l'installation
 - SYS, SYSTEM, ...
 - Pour créer d'autres utilisateurs
 - CREATE USER *authorizationID* ...

4.4.1.2 Privilèges

```
GRANT listePrivilèges ON objet TO listeAuthorizationIDs  
[WITH GRANT OPTION]
```

■ *privilege* :

```
SELECT |  
DELETE |  
INSERT [ listeColonnes ] |  
UPDATE [ listeColonnes ] |  
REFERENCES listeColonnes |  
USAGE
```

■ *objet* :

```
[ TABLE ] nomTable |  
DOMAIN nomDomaine |  
CHARACTER SET nomCharacterSet  
COLLATION nomCollation  
TRANSLATION nomTranslation
```

Exemples

```
GRANT SELECT ON Commande      TO commisLivraison  
GRANT SELECT ON LigneCommande TO commisLivraison
```

```
GRANT SELECT, DELETE, INSERT, UPDATE ON Commande  
      TO commisAchat  
GRANT SELECT, DELETE, INSERT, UPDATE ON LigneCommande  
      TO commisAchat
```

```
GRANT SELECT ON Article TO PUBLIC
```

```
GRANT UPDATE(quantitéEnStock) ON Article TO commisLivraison
```

Privilèges (suite)

- Commandes LDD
 - propriétaire du schéma
- Création d 'une VIEW sur T
 - SELECT sur T
- FOREIGN KEY sur T
 - privilège REFERENCES sur T
- SQL:1999
 - ROLE = ensemble de privilèges, Oracle
 - nouveaux privilèges
 - TRIGGER ON TABLE *nomTable*
 - EXECUTE ON PROCEDURE/FUNCTION *nomProcOuFunc*
 - UNDER ON TYPE *nomType*

4.4.1.3 Suppression de privilèges

```
REVOKE [ GRANT OPTION FOR ] listePrivilèges ON objet  
FROM listeIdUtilisateurs [ RESTRICT | CASCADE ]
```

PARAMÉTRAGE DE LA SÉCURITÉ À L'INSTALLATION DU SGBD

- Isoler le SGBD sur une machine à part
- Sécuriser le système d'exploitation
- Support d'installation sûr !
- Éliminer les services non essentiels
- Éliminer les utilisateurs pré-définis non essentiels
- Configurer les autres utilisateurs
- Configurer les mécanismes d'audit

ÉTABLISSEMENT DE L'UTILISATEUR D'UNE SESSION SQL

- Agent SQL établit connexion avec serveur SQL
 - débute une session SQL
- Contexte de session maintenu par serveur
 - identificateur d'utilisateur courant SQL et/ou de ROLE
 - pseudo-colonne CURRENT_USER (USER avec Oracle) et CURRENT ROLE
 - nom de schéma
 - nom de catalogue
 - zone de temps
 - ensemble de caractères
- Établissement de l'utilisateur SQL courant
 - à la connexion, par un mécanisme non normalisé
 - paramètres de l'opération SQL CONNECT
 - appel à une routine SQL avec privilèges de la routine
 - créateur de la routine
 - gestion par pile lors d'une cascade d'appels
- Authentification *proxy* d'Oracle pour sécurité de bout en bout multitiers
 - session légère de utilisateur *x* sur connexion pour utilisateur *y*

MÉCANISMES D'AUTHENTIFICATION

- Non normalisé en SQL
- Mot de passe
- Authentification biométrique (e.g. par empreinte digitale, reconnaissance de visages, ...)
- Authentification par le système d'exploitation
- Certificat digital
- Authentification par la couche réseau
 - e.g. *Secure Sockets Layer* (SSL)
- Authentification centrale par un serveur d'authentification externe (*Single Sign-On SSO*)
 - e.g. serveur LDAP

MENACES A LA SECURITE

- Vol de mot de passe
- Menaces sur réseau
 - interception de données
 - *sniffer* qui intercepte les paquets en transit
 - modification de données
 - usurpation d'identité
 - ...

Mécanismes avancés

- *Cryptographie symétrique (clé privée)*
 - $\text{Décrypter}(\text{cléSecrète}, \text{Encrypter}(\text{cléSecrète}, \text{message})) = \text{message}$
 - norme *Data Encryption Standard (DES)*
- *Cryptographie asymétrique (clé publique)*
 - $\text{Décrypter}(\text{cléPrivée}, \text{Encrypter}(\text{cléPublique}, \text{message})) = \text{message}$
 - système RSA (*Rivest, Shamir et Adleman*)
 - difficile de factoriser un grand nombre
 - $\text{Décrypter}(\text{cléPublique}, \text{Encrypter}(\text{cléPrivée}, \text{message})) = \text{message}$
 - employée dans *signatures digitales* et *certificats numériques*

Oracle Advanced Security (OAS)

- Plusieurs mécanismes d'authentification
- Encryptage des données
 - PL/SQL DBMS_CRYPTO
- Identification/authentification unique
 - service d'annuaire *Oracle Internet Directory* (OID)
- *Oracle Virtual Private Database*
 - règles de sécurité complexes par API PL/SQL
- *Oracle Label Security*
 - niveaux de sécurité assigné à des lignes individuelles et aux utilisateurs
 - e.g. public, secret, top secret, ...

4.4.2 Table virtuelle (VIEW)

Table Article			
noArticle	description	prixUnitaire	quantitéEnStock
10	Cèdre en boule	10.99	10
20	Sapin	12.99	10
40	Epinette bleue	25.99	10
50	Chêne	22.99	10
60	Erable argenté	15.99	10
70	Herbe à puce	10.99	10
80	Poirier	26.99	10
81	Catalpa	25.99	10
90	Pommier	25.99	10
95	Génévrier	15.99	10

```
CREATE VIEW ArticlePrixModique AS
    SELECT      noArticle, description, prixUnitaire
    FROM        Article
    WHERE       prixUnitaire < 15
```

```
SELECT      *
FROM        ArticlePrixModique
```

VIEW ArticlePrixModique		
noArticle	description	prixUnitaire
10	Cèdre en boule	10.99
20	Sapin	12.99
70	Herbe à puce	10.99

4.4.2.1 Implémentation des tables virtuelles

Résolution des vues par modification de requête

```
CREATE VIEW ArticlePrixModique AS  
    SELECT      noArticle, description, prixUnitaire  
    FROM        Article  
    WHERE       prixUnitaire < 15
```

```
SELECT      *  
FROM        ArticlePrixModique
```



```
SELECT *  
FROM (  
    SELECT      noArticle, description, prixUnitaire  
    FROM        Article  
    WHERE       prixUnitaire < 15)
```

Résolution des vues par matérialisation

- Table stockée
- Redondance
- Maintenance de la cohérence
- Meilleure performance du SELECT
- Moins bonne performance des mises à jour
- Entrepôts de données

4.4.2.2 Mise à jour de tables virtuelles

■ SQL2

- une seule table
- sans DISTINCT
- colonnes simples
- pas de SELECT imbriqué

```
SELECT nomColonne, [nomColonne] FROM T WHERE conditionSQL
```

■ SQL:1999

- spécification très complexe

Exemple de mise à jour par modification de requête

VIEW ArticlePrixModique		
noArticle	description	prixUnitaire
10	Cèdre en boule	10.99
20	Sapin	12.99
70	Herbe à puce	10.99

```
DELETE      FROM ArticlePrixModique  
WHERE       noArticle = 20
```

```
DELETE      FROM Article  
WHERE       noArticle = 20 AND prixUnitaire < 15
```

Table Article			
noArticle	description	prixUnitaire	quantitéEnStock
10	Cèdre en boule	10.99	10
20	Sapin	12.99	10
40	Epinette bleue	25.99	10
50	Chêne	22.99	10
60	Erable argenté	15.99	10
70	Herbe à puce	10.99	10
80	Poirier	26.99	10
81	Catalpa	25.99	10
90	Pommier	25.99	10
95	Génévrier	15.99	10

4.4.2.3 Problèmes de mise à jour d'une table virtuelle

```
INSERT INTO ArticlePrixModique VALUES(200, 'Viagra', 50.99)
```

```
INSERT INTO Article VALUES(200, 'Viagra', 50.99, 0)
```

Table Article			
noArticle	description	prixUnitaire	quantitéEnStock
10	Cèdre en boule	10.99	10
20	Sapin	12.99	10
40	Epinette bleue	25.99	10
50	Chêne	22.99	10
60	Erable argenté	15.99	10
70	Herbe à puce	10.99	10
80	Poirier	26.99	10
81	Catalpa	25.99	10
90	Pommier	25.99	10
95	Génévrier	15.99	10
200	Viagra	50.99	0

VIEW ArticlePrixModique		
noArticle	description	prixUnitaire
10	Cèdre en boule	10.99
20	Sapin	12.99
70	Herbe à puce	10.99

- Sémantique incohérente...

Rejet de mise à jour incohérente avec WITH CHECK OPTION

```
CREATE VIEW ArticlePrixModique AS
    SELECT      noArticle, description, prixUnitaire
    FROM        Article
    WHERE       prixUnitaire < 15
    WITH CHECK OPTION
```

```
INSERT INTO ArticlePrixModique VALUES(200, 'Viagra', 50.99)
{Insertion rejetée}
```

```
UPDATE ArticlePrixModique
SET prixUnitaire = 20.99
WHERE noArticle = 10 {Modification rejetée}
```

4.4.2.4 Hiérarchie de tables virtuelles

```
CREATE VIEW ArticlePrixModique AS
    SELECT      noArticle, description, prixUnitaire
    FROM        Article
    WHERE       prixUnitaire < 15

CREATE VIEW ArticlePrixMoyen AS
    SELECT      noArticle, description, prixUnitaire
    FROM        ArticlePrixModique
    WHERE       prixUnitaire > 12
    WITH CASCADED CHECK OPTION

    INSERT INTO ArticlePrixMoyen VALUES(200, 'Viagra', 50.99)
                                {Insertion rejetée}
```

```
CREATE VIEW ArticlePrixMoyen AS
    SELECT      noArticle, description, prixUnitaire
    FROM        ArticlePrixModique
    WHERE       prixUnitaire > 12
    WITH LOCAL CHECK OPTION

    INSERT INTO ArticlePrixMoyen VALUES(200, 'Viagra', 50.99)
                                {Insertion acceptée}
```

4.4.2.5 Renommer les colonnes d'une VIEW

```
CREATE VIEW TotalCommande (noCommande, totalCommande, totalPlusTaxe) AS  
    SELECT noCommande, SUM(quantité*prixUnitaire),  
          SUM(prixUnitaire*quantité*1.15)  
     FROM      LigneCommande AS L, Article AS A  
    WHERE      L.noArticle = A.noArticle  
    GROUP BY  noCommande
```

N.B. Modification interdite

VIEW <i>TotalCommande</i>		
noCommande	totalCommande	totalPlusTaxe
1	190.84	219.47
2	99.95	114.94
3	12.99	14.94
4	48.98	56.33
5	152.87	175.80
6	190.84	219.47
7	54.97	63.22
8	38.97	44.82

4.4.2.6 Indépendance logique des données et encapsulation par les tables virtuelles

- Catalogue (noArticle, *description*, *prixUnitaire*)
- Inventaire (noArticle, *quantitéEnStock*)

```
CREATE VIEW Article (noArticle, description, prixUnitaire, quantitéEnStock)AS
    SELECT      C.noArticle, description, prixUnitaire, quantitéEnStock
    FROM        Catalogue AS C, Inventaire AS I
    WHERE       C.noArticle = I.noArticle
```

4.4.2.7 Sécurité par les tables virtuelles

```
CREATE VIEW User_Tables AS  
    SELECT *  
    FROM Tables  
    WHERE Table_Owner = CURRENT_USER
```

```
GRANT SELECT ON User_Tables TO PUBLIC
```

4.5 Schéma interne

- Non standardisé
 - organisation primaire de la table
 - organisations secondaires (INDEX)

```
CREATE INDEX indexNoComNoArtDétLiv  
ON DétailLivraison (noCommande, noArticle)
```