Langage de Requête

- Langage de Définition de Données (LDD)
- Langage de Manipulation de Données (LMD)
- Niveaux :
  - formels : algèbre relationnelle, calcul relationnel de tuples, calcul relationnel de domaines, approches logiques)
  - Orientés utilisateur : SQL (norme), QUEL, Query By Example (QBE)
  - Couplé avec les langages de programmation (Embedded SQL / C, Fortran, ...)

SQL

- Fonctionnalités :
  - Définition et Manipulation de données au format relationnel
  - Contrôle des données (intégrité)
  - Contrôle du multi-utilisateurs (transaction)
  - Gestion de la représentation physique
- Pouvoir d'expression
  - Algèbre relationnelle
  - Fonctions (minimum, maximum, moyenne, somme, nombre)
  - Tri
- Philosophie
  - Non-procédural (en théorie)
  - Mélange de l'algèbre relationnelle et du calcul relationnel de tuples
  - Requête = suite d'opération de l'algèbre [+ Fonction] [+ Tri]

LDD : Définition et contrôle

- Schéma des données (relation = TABLE)
- Schéma des vues (VIEW)
- Spécification des contraintes d'intégrité
- Spécification des droits utilisateur
- Validation (COMMIT) / invalidation (ABORT) d'une session de travail
- Spécification du placement physique (non normalisé car SGBD dépendant)

Domaines de base

- Entier : INTEGER
- Décimal : DECIMAL (m,n)
- Réel flottant : FLOAT
- Chaîne de caractère : CHAR (longueur)
- Date : DATE
- + des domaines spécifiques à chaque SGBD (non normalisé)

Valeur nulle (null value)

- Valeur d'un attribut inconnue
  - Au moment de l'insertion, et sera (peut être) fournie ultérieurement
  - Innapplicable : ne sera jamais fournie (exemple : nom de jeune fille pour un garçon)
- Conséquences
  - Ambiguïtés sur les manipulations : sélection, jointure
  - Introduction d'une valeur : **NULL** (exemple : `DEGRE = NULL`)

➜ ATTENTION : valeur nulle ne veut pas dire valeur égale à 0 !

Gestion d'un schéma

- Création simple

  Exemple

```
CREATE TABLE VIN (
        NUM_VIN          INTEGER,
        CRU              CHAR(20),
        MILLESIME        INTEGER);
```

- Modification

  Exemple

```
ALTER TABLE VIN ADD COLUMN DEGRE INTEGER
```

- Suppression (différent de vider la relation de ses n-uplets)

  Exemple

```
DROP TABLE VIN
```

Contrainte d'intégrité

Définition Contrainte d'intégrité

Règle définissant la cohérence de données de la base

- SQL 1
  - non nullité des valeurs d'attribut
  - unicité de la valeur d'un attribut (ou groupe d'attributs)
  - valeur par défaut pour un attribut
  - contrainte de domaine
  - clé primaire
  - intégrité référentielle "minimale"

Exemple

```
CREATE TABLE VINS (
        NUM_VIN     INTEGER UNIQUE NOT NULL,
        CRU         CHAR (20),
        MILLESIME   INTEGER,
        DEGRE       INTEGER BETWEEN 5 AND 15);
```

LMD “manipulation” / LMD “recherche”

- Mise à jour : insertion, modification, suppression
  - Un seul n-uplet
  - Plusieurs n-uplets
- Recherche
  - Mono-relation
  - Multi-relations
  - Avec fonction-agrégat

Schéma EXEMPLE : Base COOPERATIVE

VINS (V) (NUM_VIN, CRU, MILLESIME)

VITICULTEURS (VT) (NUM_VITICULTEUR, NOM, PRENOM, VILLE)

PRODUCTIONS (P) (VIN, VITICULTEUR)

BUVEURS (B) (NUM_BUVEUR, NOM, PRENOM, VILLE)

COMMANDES (C) (NUM_COMMANDE, DATE, VIN, QUANTITE, BUVEUR)

EXPEDITION (E) (COMMANDE, DATE, QUANTITE)

LMD-Manipulation : Création/Insertion (Exemples)

Création

```
CREATE TABLE BORDEAUX
(NUM_VIN INTEGER, MILLESIME INTEGER, DEGRE INTEGER);
```

Insertion

- Un n-uplet

```
INSERT INTO VINS VALUES (100, 'Juranon', 1979, 12);
INSERT INTO VINS (NUM_VIN, CRU) VALUES (200, 'Gamay');
```

- Un ensemble de n-uplets
  - Par fichier (COPY FROM)
  - Par requête

```
INSERT INTO BORDEAUX
  SELECT NUM_VIN, MILLESIME, DEGRE
  FROM    VINS
  WHERE  CRU= 'Bordeaux'
```

Suppression

Exemples

- "Tous les n-uplets de VINS"

```
DELETE FROM VINS;
```

- "Le vin de numéro 150"

```
DELETE FROM VINS WHERE NUM_VIN = 150;
```

- "Les vins de degré < 9 ou > 12"

```
DELETE FROM VINS WHERE DEGRE < 9 OR DEGRE > 12;
```

- "Les commandes passées par Dupond"

```
DELETE FROM COMMANDES
        WHERE BUVEUR IN ( SELECT NUM_BUVEUR
                          FROM BUVEURS
                          WHERE NOM ='Dupond');
```

Modifications

Exemples

- “Mettre à 'Bordeaux' la ville du viticulteur 150”

```
UPDATE VITICULTEURS
SET    VILLE = 'Bordeaux'
WHERE  NUM_VITICULTEUR = 150;
```

- “Augmenter de 10% le degré des Gamay”

```
UPDATE VINS
SET    DEGRE = DEGRE * 1.1
WHERE  CRU = 'Gamay';
```

- “Augmenter de 10 les quantités commandées par 'Dupond”'

```
UPDATE COMMANDES
SET    QUANTITE = QUANTITE + 10
WHERE  BUVEUR IN (SELECT NUM_BUVEUR
                  FROM   BUVEURS
                  WHERE  NOM = 'Dupond' ;
```

Syntaxe générale de recherche

Définition Syntaxe générale recherche

```
SELECT <liste des attributs>
FROM   <liste des relations>
WHERE  <liste des critères>;
```

- **SELECT** : opérateur de projection
- **FROM** : opérateur de produit cartésien
- **WHERE** : opérateur de sélection

Union / Différence : approche algébrique

Requête Exemple : "Donner les vins de cru Chablis"

```
SELECT NUM_VIN, MILLESIME, DEGRE
FROM   VINS
WHERE  CRU = 'Chablis';
```

Exemples

- Requête : "Donner tous les vins"

```
SELECT *
FROM VINS ;
```

- Requête : "Donner les crus des vins de millésime 1976 et de degré 12, triés par ordre croissant"

```
SELECT CRU
FROM VINS
WHERE MILLESIME = 1976 AND DEGRE = 12
ORDER BY CRU ;
```

- Requête : "Donner la liste de tous les crus ($\rightarrow$ sans double)"

```
SELECT DISTINCT CRU
FROM VINS ;
```

Recherche sur un intervalle de valeurs

Exemples

- Requête : "Vins de degré compris entre 8 et 12"

```
SELECT *
FROM VINS
WHERE DEGRE >= 8 AND DEGRE <= 12;

SELECT *
FROM VINS
WHERE DEGRE BETWEEN 8 AND 12;

SELECT *
FROM VINS
WHERE DEGRE IN (8,9,10,11,12);
```

Recherche sur chaîne de caractères

Exemple

- Requête : “Vins dont le nom du cru commence par 'B' ou 'b'”

```
SELECT *
FROM VINS
WHERE CRU LIKE 'B%' OR 'b%'
```

Expression des jointures

Définition JOINTURE

Jointure = sélection (WHERE) sur produit cartésien (FROM)

EXEMPLE

- "Donner les numéros et noms des viticulteurs produisant du 'Muscadet"'

  Sélection :
  - CRU = 'Muscadet'
  - Numéros de viticulteur identiques dans **VITICULTEURS** et **PRODUCTIONS**
  - Numéros de vin identiques dans **VINS** et **PRODUCTIONS**

SELECT VT.NUM_VITICULTEUR, VT.NOM
FROM VITICULTEURS VT, VINS V, PRODUCTIONS P
WHERE VT.NUM_VITICULTEUR = P.VITICULTEUR **AND**
P.VIN = V.NUM_VIN **AND**
V.CRU = 'Muscadet' ;

Blocs imbriqués

Jointure de manière procédurale : Exemple

```
SELECT NUM_VITICULTEUR, NOM
FROM   VITICULTEURS
WHERE  NUM_VITICULTEUR IN
                    SELECT VITICULTEUR
                    FROM   PRODUCTIONS
                    WHERE  VIN IN
                           SELECT NUM_VIN
                           FROM   VINS
                           WHERE  CRU = 'Muscadet';
```

Auto-Jointure

Définition Auto-Jointure

Jointure d'une relation avec elle-même ($\rightarrow$ synonymes)

Exemple

- "Donner les couples de buveurs habitant la même ville"

```
SELECT B1.NUM_BUVEUR, B1.VILLE, B2.NUM_BUVEUR
FROM   BUVEURS B1, BUVEURS B2
WHERE  B1.VILLE = B2.VILLE AND
       B1.NUM_BUVEUR > B2.NUM_BUVEUR;
```

Opérateurs ensemblistes (Exemple)

Définition Opérateurs ensemblistes

Opérateurs binaires avec deux relations de même schéma en entrée (élimination automatique des doubles en sortie)

- **UNION**

```
  (SELECT NUM_VITICULTEUR FROM VITICULTEURS)
UNION
  (SELECT NUM_BUVEUR FROM BUVEURS)
```

- **INTERSECT**

```
 (SELECT NUM_VITICULTEUR FROM VITICULTEURS)
INTERSECT
 (SELECT NUM_BUVEUR FROM BUVEURS)
```

- **EXCEPT**

```
   (SELECT NUM_VITICULTEUR FROM VITICULTEURS)
 EXCEPT
   (SELECT NUM_BUVEUR FROM BUVEURS)
```

Fonctions prédéfinies

Principe Fonctions prédéfinies

- COUNT, SUM, AVG, MIN, MAX
- Elles s'appliquent à l'ensemble des valeurs d'**UNE** colonne d'une relation et produit une valeur unique

Cas particulier : COUNT(*) : nombre de n-uplets

Exemple

- "Donner le nombre de n-uplets de **VINS**"

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

Fonctions prédéfinies : Positionnement dans le **SELECT**

Exemples

- "Donner la moyenne des degrés de tous les vins"

```
SELECT AVG(DEGRE)
FROM VINS;
```

- "Donner la quantité totale commandée par le buveur de nom "Grosbuveur""

```
SELECT SUM(QUANTITE)
FROM COMMANDES C, BUVEURS B
WHERE B.NOM = 'Grosbuveur' AND
      B.NUM_BUVEUR = C.BUVEUR;
```

Fonctions prédéfinies : Positionnement dans le **WHERE**

Exemples

- "Donner les vins dont le degré est supérieur à la moyenne des degrés de tous les vins"

```
SELECT  *
FROM    VINS
WHERE   DEGRE > (SELECT AVG(DEGRE)
                 FROM   VINS);
```

- "Donner les numéros de commande où la quantité commandée a été totalement expédiée"

```
SELECT NUM_COMMANDE
FROM COMMANDES C
WHERE C.QUANTITE = (SELECT SUM(E.QUANTITE)
                    FROM EXPEDITIONS E
                    WHERE E.COMMANDE = C.NUM_COMMANDE);
```

Agrégats

Fonctions de calcul :

- Somme (SUM), moyenne (AVG)
- Minimum (MIN), maximum (MAX)
- Comptage (COUNT)

1. "Moyenne des degrés des VINS disponibles" $\Rightarrow$ 1 relation, 1 attribut, 1 n-uplet
2. "Moyenne des degrés des VINS par CRU" : ?

➜ Partition horizontale d'une relation

Exemple

VINS	NUMERO_VIN	CRU	MILLESIME	DEGRE	QUANTITE
	3	Chablis	1977	10,9	100
	5	Volnay	1977	10,8	400
	6	Chablis	1987	11,9	250
	8	Medoc	1985	11,2	200
	7	Volnay	1986	11,2	300

- "Moyenne des degrés des vins disponibles"
  - Résultat : 1 relation, 1 attribut, 1 n-uplet
  - R = AVG (**VINS**, DEGRE)

R	AVG
	11,2

Insuffisance des fonctions

VINS	NUMERO_VIN	CRU	MILLESIME	DEGRE	QUANTITE
	3	Chablis	1977	10,9	100
	5	Volnay	1977	10,8	400
	6	Chablis	1987	11,9	250
	8	Medoc	1985	11,2	200
	7	Volnay	1986	11,2	300

- Somme des quantités par cru
  - Résultat : 1 relation, 2 attributs, 1 ou plusieurs n-uplet(s)

VINS	CRU	QUANTITE
	Chablis	350
	Volnay	700
	Medoc	200

  - R = SUM (**VINS**, CRU, QUANTITE)

Partition d'une relation - **GROUP BY**

Principe Group By

- Partition horizontale d'une relation, selon les valeurs d'un attribut (ou d'un groupe d'attributs), spécifié dans la clause **GROUP BY**
- La relation est (logiquement) fragmentée en groupe de n-uplets, où tous les n-uplets de chaque groupe ont la même valeur pour l'attribut (ou le groupe d'attributs) de la partition

➜ Application possible d'une fonction à chaque groupe

➜ Restriction sur les groupes en fonction de critères (Clause **HAVING**)

➜ Fonctions sur les groupes

Exemples

- "Donner, pour chaque cru, la moyenne des degrés des vins de ce cru"

SELECT CRU, AVG(DEGRE)
FROM VINS
GROUP BY CRU ;

- Idem "avec tri par degré moyen croissant"

```
SELECT   CRU, AVG(DEGRE)
FROM     VINS
GROUP BY CRU
ORDER BY 2 DESC;
```

Restriction sur les groupes

Restriction

- Clause **WHERE** : sur les n-uplets d'une relation
- Clause **HAVING** : sur les groupes obtenus par un **GROUP BY**

Exemple

- Exemple : "Donner les numéros et les noms des buveurs ayant commandé plus d'une fois, ainsi que la moyennes des quantités commandées

```
SELECT B.NUM_BUVEUR, B.NOM, AVG(C.QUANTITE)
FROM BUVEURS B, COMMANDES C
WHERE B.NUM_BUVEUR = C.BUVEUR
GROUP BY B.NUM_BUVEUR, B.NOM
HAVING COUNT(*) > 1;
```

Requête erronée

```
SELECT CRU, NUM_VIN, AVG(DEGRE)
FROM VINS
GROUP BY CRU;
```

➜ **Résultats "attendus" :**

CRU	NUM_VIN	AVG(DEGRE)
Bordeaux	{1, 3, 6, 10}	10.0
Chablis	{5, 7}	11.0
Jurançon	{2, 8, 11}	13.0

➜ **Requête non valide en SQL**

- NUM_VIN est multi-valué par rapport à CRU
- La relation ne respecte pas l'atomicité des attributs

Requête erronée

```
SELECT CRU, NUM_VIN, AVG(DEGRE)
FROM VINS
GROUP BY CRU;
```

➜ **Résultats "attendus" :**

CRU	NUM_VIN	AVG(DEGRE)
Bordeaux	{1, 3, 6, 10}	10.0
Chablis	{5, 7}	11.0
Jurançon	{2, 8, 11}	13.0

➜ **Requête non valide en SQL**

- NUM_VIN est multi-valué par rapport à CRU
- La relation ne respecte pas l'atomicité des attributs

Questions quantifiées (**ALL**, **ANY**, EXISTS)

Définition ALL

Teste si la valeur d'un attribut satisfait un critère de comparaison avec **TOUS** les résultats d'une sous-question

Définition ANY

Teste si la valeur d'un attribut satisfait un critère de comparaison avec **AU MOINS** un résultat d'une sous-question

EXEMPLE

- "Donner les noms et numéros des buveurs qui ont le plus commandé"

```
SELECT B.NUM_BUVEURS, B.NOM
FROM  BUVEURS B, COMMANDES C
WHERE  B.NUM_BUVEUR = C.BUVEUR AND
       C.QUANTITE >= ALL ( SELECT QUANTITE
                           FROM   COMMANDES);
```

Prédicat d'existence (**EXISTS**)

Définition EXISTS

Teste si la réponse à une sous-question est vide

EXEMPLE

- "Viticulteurs ayant produit au moins un vin (quel que soit ce vin)"

```
SELECT VT.*
FROM   VITICULTEURS VT
WHERE  EXISTS ( SELECT P.*
                FROM   PRODUCTIONS P
                WHERE  VT.NUM_VITICULTEUR =
                          P.VITICULTEUR);
```

Division via EXISTS

- Quels sont les viticulteurs ayant produit TOUS les vins ?

➜ Paraphrase : Un viticulteur est sélectionné s'il n'existe aucun vin qui n'ait pas été produit par ce producteur (double négation)

```
SELECT VT.*
FROM VITICULTEURS VT
WHERE NOT EXISTS (
        SELECT V.*
        FROM VINS V
        WHERE NOT EXISTS (
              SELECT P.*
              FROM PRODUCTIONS P
              WHERE P.VITICULTEUR = VT.NUM_VITICULTEUR
               AND P.VIN = V.NUM_VIN;
```

Division via EXISTS

- Quels sont les viticulteurs ayant produit TOUS les vins ?

➜ Paraphrase : Un viticulteur est sélectionné s'il n'existe aucun vin qui n'ait pas été produit par ce producteur (double négation)

```
SELECT VT.*
FROM VITICULTEURS VT
WHERE NOT EXISTS (
        SELECT V.*
        FROM VINS V
        WHERE NOT EXISTS (
            SELECT P.*
            FROM PRODUCTIONS P
            WHERE P.VITICULTEUR = VT.NUM_VITICULTEUR
             AND P.VIN = V.NUM_VIN ;
```

Division via EXISTS

- Quels sont les viticulteurs ayant produit TOUS les vins ?

➜ Paraphrase : Un viticulteur est sélectionné s'il n'existe aucun vin qui n'ait pas été produit par ce producteur (double négation)

```
SELECT VT.*
FROM VITICULTEURS VT
WHERE NOT EXISTS (
        SELECT V.*
        FROM VINS V
        WHERE NOT EXISTS (
            SELECT P.*
            FROM PRODUCTIONS P
            WHERE P.VITICULTEUR = VT.NUM_VITICULTEUR
             AND P.VIN = V.NUM_VIN ;
```

Division via EXISTS

- Quels sont les viticulteurs ayant produit TOUS les vins ?

➜ Paraphrase : Un viticulteur est sélectionné s'il n'existe aucun vin qui n'ait pas été produit par ce producteur (double négation)

```
SELECT VT.*
FROM VITICULTEURS VT
WHERE NOT EXISTS (
        SELECT V.*
        FROM VINS V
        WHERE NOT EXISTS (
              SELECT P.*
              FROM PRODUCTIONS P
              WHERE P.VITICULTEUR = VT.NUM_VITICULTEUR
               AND P.VIN = V.NUM_VIN;
```

Résumé conditions de recherche

- Sélection de n-uplets (**WHERE**),
  sélection de groupes (**HAVING**)
- Conjonction/Disjonction/Négation de triplet (Attribut Comparateur, Valeur)
- Expression évaluée à Vrai / Faux / Inconnu
- Condition élémentaire (prédicat)
  - Comparaison : $=$, $<$, $<=$, ... entre attributs ou attribut / valeur
  - **BETWEEN**, **LIKE**, **IN**, **IS NULL**, **EXISTS**, **ANY**, **ALL**

Résumé Requête de recherche SQL

- (6) **SELECT** <attributs et/ou expressions sur attributs> A_j
- (1) **FROM** <relations> R_i
- (2) **WHERE** <conditions sur les n-uplets> C_i
- (3) **GROUP BY** <attributs> A_k
- (4) **HAVING** <conditions sur les groupes> C_j
- (5) **ORDER BY** <attributs> A_l

Interprétation :

- (1) Produit Cartésien des relations R_i
- (2) Sélection des n-uplets de (1) vérifiant C_i
- (3) Partition de l'ensemble obtenu suivant les valeurs A_k
- (4) Sélection des groupes de (3) vérifiant C_j
- (5) Tri des groupes obtenus en (4) suivant les valeurs A_l
- (6) π de (5)sur A_j, et fonctions sur les groupes (s'il y en a)

Exemple complet

- Donnez par ordre croissant le cru et le millésime des vins commandés par les buveurs parisiens, ainsi que la somme des quantités commandées, uniquement si cette quantité est strictement supérieure à 100 litres

```
SELECT    V.CRU, V.MILLESIME, SUM(C.QUANTITE)
FROM      BUVEURS B, COMMANDES C, VINS V
WHERE     B.NUM_BUVEUR = C.BUVEUR AND
             C.VIN = V.NUM_VIN AND
             B.VILLE = 'Paris'
GROUP BY  V.CRU, V.MILLESIME
HAVING    SUM(C.QUANTITE) > 100
ORDER BY  V.CRU, V.MILLESIME;
```