Introduction aux bases de données
Cours 4 : Le langage MySQL

Odile PAPINI

ESIL
Université de la méditerranée
Odile.Papini@esil.univmed.fr
http://odile.papini.perso.esil.univmed.fr/sources/BDMAT.html

Plan du cours

Bibliographie

Livres :

- G. Gardarin : Bases de données objet et relationnel. Eyrolles ed. 1999.
- C. J. Date : Introduction aux bases de données. (8ième edition). Vuibert ed. 2004.
- H. Garcia-Molina, J. D. Ullman, J. Widow : Database systems, the complete book. Prentice Hall ed. 2002.

Supports de cours :

- Support de cours : H. Etievant : http ://mysql.developpez.com/cours/
- Manuel de référence MySQL : http ://dev.mysql.com/doc/

SQL : Introduction

SQL : Structured Query Langage

langage de gestion de bases de données relationnelles pour

- définir les données (LDD)
- intérroger la base de données (Langage de requêtes)
- manipuler les données (LMD)
- contrôler l'accès aux données (LCD)

SQL : Introduction

SQL : Quelques repères historiques

- 1974 SEQUEL (Structured English Query Language)
  ancêtre de SQL
- 1979 premier SGBD basé sur SQL par Relational Software Inc.
  (rebaptisé Oracle)
- 1986 SQL1 1ière norme ISO
- 1989 ajout des contraintes d'intégrité de base
  (clé primaire et clé étrangère)
- 1992 SQL2 2ième norme extension de SQL1
  (nouveaux types et nouveaux opérateurs
- 1999 SQL3 extension de SQL2
  (introduction des types orientés objet)

SQL et MySQL

MySQL

MySQL : logiciel libre (SGBD relationnel)

http ://www.mysql.com

manuel d'utilisation :
http ://dev.mysql.com/doc/refman/5.0/fr/index.html

SQL comme LDD

Identificateurs :

- lettre suivie par : lettre ou chiffre ou _ ou $
- chaîne de caractères entre guillemets simples ' ' ou doubles " "
- maximum 64 caractères
- différent d'un mot clé
  ASSERT, ASSIGN, AUDIT, COMMENT, DATE, DECIMAL, DEFINITION, FILE, FORMAT, INDEX, LIST, MODE, OPTION, PARTITION, PRIVILEGE, PUBLIC, SELECT, SESSION, SET, TABLE

pas de distinction entre majuscules et minuscules : WINDOW
distinction entre majuscules et minuscules : LINUX

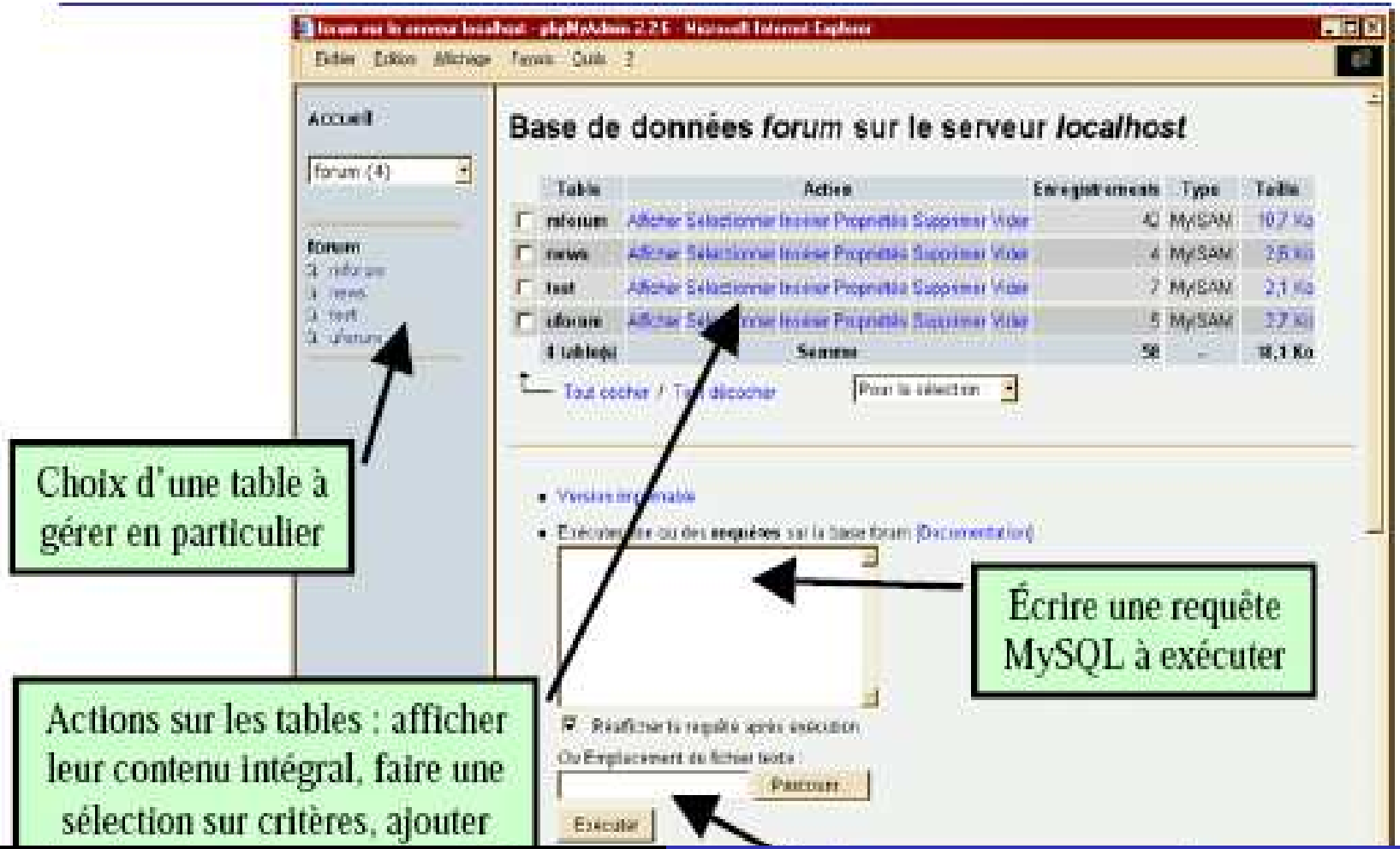
SQL comme LDD

Tables :

- relations d'un schéma relationnel stockées sous tables
- table : formée de lignes et de colonnes
- MySQL :
  - nom de schéma remplacé par le nom d'utilisateur qui a crée la table
  - par défaut le schéma est le nom de l'utilisateur connecté

SQL comme LDD

Tables : avec l'outil web phpMyAdmin



SQL comme LDD

Tables : colonnes

!

- toutes les données d'une colonne sont du même type
- identificateur unique pour les colonnes d'une même table
- 2 colonnes dans 2 tables différentes peuvent avoir le même nom
- nom complet d'une colonne comprend le nom complet de la table à laquelle elle appartient
- exemple : DEPARTEMENTS.DEPARTEMENT_ID ou HR.DEPARTEMENTS.DEPARTEMENT_ID

SQL comme LDD

Types de données de MySQL :

- types pour les chaînes de caractères
- types numériques
- types temporels (date, heure, $\cdots$)
- type pour les données volumineuses (images , sons)
- MySQL ne permet pas à l'utilisateur de créer ses propres types

SQL comme LDD

Types pour les chaînes de caractères

- CHAR(*taille*)
  - chaînes de caractères de longueur fixe
  - codage en longueur fixe : remplissage de blancs
  - taille comprise entre 1 et 255 caractèress
- VARCHAR(*taille_max*)
  - chaînes de caractères de longueur variable
- TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
  - chaînes de longueur max : 255, 65535 , 16777215, 4294967295 caractères

SQL comme LDD

MySQL : Types pour les chaînes de caractères

exemples

- CHAR(5) : chaînes de 5 caractères
- VARCHAR(20) : chaînes de 20 caractères au plus
- 'Administration', 'Marketing', "Secretariat"

SQL comme LDD

MySQL : Types numériques

- types numériques pour les entiers :
  - TINYINT pour 1 octets
  - SMALLINT pour 2 octets
  - MEDIUMINT pour 3 octets
  - INTEGER ZEROFILL pour 4 octets
  - BIGINT pour 8 octets
- types numériques pour les décimaux à virgule flottante :
  - REAL
  - DOUBLE PRECISION ou
- types numériques pour les réels à virgule fixe :
  - FLOAT(*nb_chiffres max, nb_décimales*) ZEROFILL
  - NUMERIC(*nb_chiffres max, nb_décimales*)
  - DECIMAL(*nb_chiffres max, nb_décimales*)
- ZEROFILL : non signé

SQL comme LDD

MySQL : Types temporels

- DATE : date format anglophone AAAA-MM-JJ
- DATETIME : date et heure format anglophone AAAA-MM-JJ HH :MM :SS
- TIMESTAMP : date et heure sans séparateur AAAAMMJJHHMMSS
- TIMESTAMP(M) : affiche les M (M pair) premiers caractères de AAAAMMJJHHMMSS
- TIME : heure au format HH :MM :SS
- YEAR : année au format AAAA

SQL comme LDD

MySQL : Type énuméré, type ensemble

- ENUM('valeur1', $\cdots$, 'valeur n') type énuméré

  un attribut de ce type ne peut prendre qu'une valeur parmi un ensemble explicitement spécifié

- SET('valeur1', $\cdots$ 'valeur n') type ensemble ,

  un attribut de ce type peut prendre plusieurs valeurs parmi l'ensemble explicitement spécifié

Base de donnée relationnelle : Manipulation : exemple (1)

exemple du cours précédent : BD vols-réservations

Voici 3 schémas de relations :

- avions(No_AV, NOM_AV, CAP, LOC)
- pilotes(No_PIL, NOM_PIL, VILLE)
- vols(No_VOL, No_AV, No_PIL, V_d, V_a, H_d, H_a)

Base de donnée relationnelle : Manipulation : exemple (2)

et voici les 3 tables correspondantes :

avions			
No_AV	NOM_AV	CAP	LOC
100	airbus	300	nice
101	airbus	300	paris
102	carav	200	toulouse

pilotes		
No_PIL	NOM_PIL	VILLE
1	laurent	nice
2	sophie	paris
3	claude	grenoble

vols						
No_VOL	No_AV	No_PIL	V_d	V_a	H_d	H_a
it100	100	1	nice	paris	7	8
it101	100	2	paris	toulouse	11	12
it102	101	1	paris	nice	12	13
it103	102	3	grenoble	toulouse	9	11
it104	101	3	toulouse	grenoble	17	18

SQL comme LDD : Exemple

- avions(no_AV, NOM_AV, CAP, LOC)

  no_AV INTEGER
  NOM_AV VARCHAR(20)
  CAP SMALLINT
  LOC VARCHAR(15)

- pilotes(no_PIL, NOM_PIL, VILLE)

  no_PIL INTEGER
  NOM_PIL VARCHAR(20)
  VILLE VARCHAR(15)

- vols(no_VOL, no_AV, no_PIL, V_d, V_a, H_d, H_a)

  no_VOL VARCHAR(5)
  no_AV INTEGER
  no_PIL INTEGER
  V_d VARCHAR(15) H_d DATETIME
  V_a VARCHAR(15) H_a DATETIME

Base de donnée relationnelle : exemple (2)

autre exemple

schéma de la relation :

- article(Id, Titre, Texte, Auteur, Parution, Rubrique)

Id INTEGER
Titre VARCHAR(80)
Texte TEXT
Auteur VARCHAR(15)
Parution DATE
Rubrique ENUM('politique', 'international', 'economie', 'culture', 'sport')

SQL comme LDD

MySQL : Absence de valeur

NULL : représente l'absence de valeur pour tous les types de données. Ce n'est pas une valeur

- pour les types chaîne de caractères :
  - la chaîne vide ' ' représente aussi l'absence de valeur
  - pour les types chaîne de caractères (taille fixe ou variable) une chaîne remplie de blancs n'est pas équivalente à la chaîne vide
- pour les types numériques
  - le nombre 0 ne représente pas l'absence de valeur

SQL comme LDD

création de table

CREATE TABLE nom_de_table (liste de définition_de_colonne,
[liste de contrainte_de_table]) ;

définition_de_colonne : :=

nom_de_colonne

(nom_de_domaine ou type)

[liste de contrainte_de_colonne]

[DEFAULT valeur_par_défaut]

SQL comme LDD

création de table : contrainte de colonne

contrainte_de_colonne : :=

[CONSTRAINT nom]
type_de_contrainte_de_colonne

type_de_contrainte_de_colonne : :=

AUTO_INCREMENT ou
PRIMARY KEY ou
NOT NULL ou
UNIQUE ou
CHECK(condition_sur_valeur) ou
REFERENCES nom_de_table(nom_de_colonne)

SQL comme LDD

création de table : contrainte de table

contrainte_de_table : :=

[CONSTRAINT nom]
type_de_contrainte_de_table

type_de_contrainte_de_table : :=

PRIMARY KEY (liste de nom_de_colonne) ou
NOT NULL (liste de nom_de_colonne) ou
UNIQUE (liste de nom_de_colonne) ou
CHECK (condition_sur_ligne) ou
FOREIGN KEY liste de nom_de_colonne REFERENCES
nom_de_de_table (liste de nom_de_colonne)

SQL comme LDD

Exemple : Création de la table avions à partir du schéma :

avions(no_AV, nom_AV, CAP, LOC)

```
CREATE TABLE avions (
    no_AV INTEGER
        CONSTRAINT Cle_P_avions PRIMARY KEY,
    NOM_AV VARCHAR(20),
    CAP SMALLINT
        CONSTRAINT Dom_CAP_avions CHECK (CAP ≥ 4),
    LOC VARCHAR(15) );
```

SQL comme LDD : Exemple : Création de la table vols

Création de la table vols à partir du schéma :

vols(<u>no_VOL</u>, no_AV, no_PIL, V_d, V_a, H_d, H_a)

- Contraintes de colonnes ?
- Contraintes de table ?

SQL comme LDD : Exemple : Création de la table vols

vols(<u>no_VOL</u>, no_AV, no_PIL, V_d, V_a, H_d, H_a)

```
CREATE TABLE vols (
    no_VOL VARCHAR(5)
        CONSTRAINT Cle_P_vols PRIMARY KEY,
    no_AV INTEGER
        CONSTRAINT Ref_no_AV_vols REFERENCES avions,
    no_PIL INTEGER
        CONSTRAINT Ref_no_PIL_vols REFERENCES pilotes,
    V_d VARCHAR(15) NOT NULL,
    V_a VARCHAR(15) NOT NULL,
    H_d DATETIME,
    H_a DATETIME,
    CONSTRAINT C1_vols CHECK (v_d ≠ v_a),
    CONSTRAINT C2_vols CHECK (h_d < h_a) );
```

SQL comme LDD

suppression de table

DROP TABLE nom ;

Quand une table est supprimée :

- efface tous les index qui y sont attachés quelque soit le propriétaire
- efface tous les privilèges qui y sont attachés
- MAIS les vues et les synomymes se référant à cette table ne sont pas supprimés

SQL comme LDD

modification de table

ALTER TABLE nom_de_table modification_de_table ;

modification_de_table : :=

ADD COLUMN définition_de_colonne

ADD CONSTRAINT contrainte_de_table

DROP COLUMN nom_de_colonne

DROP CONSTRAINT nom_de_contrainte

SQL comme LDD

Exemple : Ajout d'un colonne à la table vols de schéma :

vols(no_VOL, no_AV, no_PIL, V_d, V_a, H_d, H_a)

ALTER TABLE vols ADD COLUM COUT_VOL FLOAT

le schéma devient :

vols(no_VOL, no_AV, no_PIL, V_d, V_a, H_d, H_a, COUT_VOL)

SQL comme LDD

insertion de lignes dans une table

INSERT INTO nom_de_table [liste_de_colonnes] VALUES liste_de_valeurs ;

ou

INSERT INTO nom_de_table [liste_de_colonnes] requête ;

SQL comme LDD : Exemple

- ajouter un avion dans la table avions en respectant l'ordre des colonnes

```
INSERT INTO avions
VALUES (100, 'Airbus', 200, 'Paris');
```

- ajouter un avion dans la table avions sans connaître l'ordre

```
INSERT INTO avions (no_AV, CAP, LOC, NOM_AV)
VALUES (101, 200, 'Paris', 'Airbus');
```

- ajouter un avion dans la table avions dont la localisation est INDEFINI

```
INSERT INTO avions (no_AV, NOM_AV, CAP)
VALUES (102, 'Airbus', 200);
```

ou

```
INSERT INTO avions
VALUES (102, 'Airbus', 200, NULL);
```

SQL comme LDD

suppression de lignes d'une table

DELETE [FROM] nom_de_table [WHERE condition] ;

Exemples :

- vider la table avions

  DELETE FROM avions ;

- supprimer de la table avions tous les avions dont la capacité est inférieur à 100

  DELETE FROM avions
  WHERE CAP $<$ 100 ;

SQL comme LDD

modification de lignes dans une table

UPDATE nom_de_table SET liste_expression_colonne
[WHERE condition] ;

expression_colonne : :=

nom_de _colonne = expression ou

nom_de_colonne = requête

Exemple :

modifier la capacité de l'avion numéro 100

UPDATE avions SET CAP = 300

WHERE no_AV = 100 ;

SQL comme Langage de Requêtes

interrogation

requête : := SELECT [DISTINCT] projection

FROM liste de (nom_de_table [[AS] nom]) | (requête AS nom)

WHERE condition

[GROUP BY liste de nom_de_colonne]

[HAVING condition]

[ORDER BY liste de ((nom_de_colonne | rang_de_colonne) (ASC | DESC)] ;

requête : := requête (UNION | INTERSECT | EXCEPT) requête

requête : := (requête)

SQL comme Langage de Requêtes

projection : :=

* | nom_de_table | liste de (terme_de_projection[[AS] nom])

terme_de_projection : :=

expression | agrégation

expression : :=

valeur | nom_de_colonne | expression_arithmétique | ···

agrégation : :=

COUNT(*)

opérateur_d'agrégation([DISTINCT] expression)

opérateur_d'agrégation : :=

COUNT | SUM | AVG | MAX | MIN

SQL comme Langage de Requêtes

condition : :=

condition_élémentaire

NOT condition | condition(AND | OR) condition | (condition)

condition_élémentaire : :=

reconnaissance_de_modèle | test_de_valeur_nulle | comparaison |

appartenance_à_un_intervalle | appartenance_à_un_ensemble | existence

reconnaissance_de_modèle : :=

expression [NOT] LIKE _de_chaîne

test_de_valeur_nulle : :=

nom_de_valeur IS [NOT] NULL

SQL comme Langage de Requêtes

comparaison : :=

expression (= | <>| > | < | <= | >= |) expression |
expression (= | <>) (| SOME | ALL) requête
expression (> | < | <= | >=) (| SOME | ALL)
requête_mono_colonne

appartenance_à_un_intervalle : :=

expression BETWEEN expression AND expression

appartenance_à_un_ensemble : :=

expression (IN | NOTIN) (requête) |
(liste de expression) (IN | NOTIN) (requête)

ensemble_de_valeurs : := (liste de valeur) | requête_mono_colonne

existence : := EXISTS (requête)

SQL comme Langage de Requêtes

Sélection de lignes

- **toutes les lignes et toutes les colonnes**
  SELECT * FROM nom_de_table ;
  - pour connaître toutes les caractéristiques des avions stockés dans la table
    SELECT * FROM avions ;

- **toutes les lignes mais seulement certaines colonnes**
  SELECT liste de nom_de_colonne FROM nom_de_table ;
  - pour connaître les numéros de tous les vols
    SELECT no_VOL FROM vols ;

SQL comme Langage de Requêtes

Sélection de lignes

- **suppression des lignes dupliquées**
  SELECT DISTINCT liste de nom_de_colonne FROM nom_de_table ;
  - pour connaître les numéros des pilotes qui conduisent au moins un avion
    SELECT DISTINCT no_PIL FROM vols ;

colonnes calculées

SELECT expression [AS alias] FROM nom_de_table ;

- afficher une augmentation de 5% du coût de chaque vol
  SELECT no_VOL, '5%' "%",
  COUT_VOL*0.05 AUGM, COUT_VOL*1.05 "Nouveau coût"
  FROM vols ;

SQL comme Langage de Requêtes : Exemple de calcul

- **sur les chaînes de caractères**
  - afficher les trajets assurés par les vols sous la forme :
    Ville de départ $-->$ Ville d'arrivée
    SELECT no_VOL,
    V_d || $'-->'$ || V_a TRAJET
    FROM vols ;

- **sur les dates**
  - pour connaître la durée en heures de tous les vols
    SELECT no_VOL, TMEDIFF(D_a, D_d) durée
    FROM vols ;

SQL comme Langage de Requêtes

Recherche par comparaison

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE expression ;

- pour connaître tous les avions qui ont une capacité > 200 places

```
SELECT no_AV
FROM avions
WHERE CAP > 200 ;
```

- pour connaître tous les pilotes qui effectuent des vols qui durent plus d'une heure

```
SELECT DISTINCT no_PIL
FROM vols
WHERE (TIMEDIFF(D_a , D_d) )> 1 ;
```

SQL comme Langage de Requêtes

Recherche par ressemblance

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE expression [NOT] LIKE motif [caractères spéciaux] ;

- caractère spéciaux :
  - % : remplace 0, 1 ou plusieurs caractères
  - _ : remplace 1 caractère
- caractère d'échappement :
  - permet de traiter les caractère spéciaux comme de simples caractères
  - il n'y a pas de caractère d'échappement prédéfini

SQL comme Langage de Requêtes

Recherche par ressemblance : exemples

- pour connaître la capacité de tous les avions Boeing

```
SELECT no_AV, NOM_AV, CAP
FROM avions
WHERE NOM_AV LIKE 'Boeing%' ;
```

SQL comme Langage de Requêtes

Recherche avec condition conjonctive

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE condition AND condition ;

- pour connaître tous les avions qui sont à Marseille et dont la capacité est de 300 places
  SELECT no_AV
  FROM avions
  WHERE LOC = 'Marseille' AND CAP = 300 ;

SQL comme Langage de Requêtes

Recherche avec condition disjonctive

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE condition OR condition ;

- pour connaître tous les vols qui utilisent les avions 100 et 101
  SELECT no_VOL
  FROM vols
  WHERE no_AV = 100 OR no_AV = 101 ;

SQL comme Langage de Requêtes

Recherche avec condition négative

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE NOT condition ;

- pour connaître tous les pilotes qui n'habitent pas à Marseille
  SELECT no_PIL
  FROM pilotes
  WHERE NOT VILLE = 'Marseille' ;

SQL comme Langage de Requêtes

Recherche avec un intervalle

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE expression BETWEEN expression AND expression ;

- pour connaître tous les avions qui ont une capacité entre 200 et 300 places

```
SELECT no_AV
FROM avions
WHERE CAP BETWEEN 200 AND 300 ;
```

SQL comme Langage de Requêtes

Recherche avec une liste

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE expression [NOT] IN liste de expression ;

- pour connaître tous les pilotes qui habitent soit à Marseille soit à Nice

```
SELECT no_PIL
FROM pilotes
WHERE VILLE IN ('Marseille', 'Nice') ;
```

SQL comme Langage de Requêtes

Recherche avec une liste

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE expression (<>| > | < | <= | >= |) ANY
liste_de_expression ;

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE expression (<>| > | < | <= | >= |) SOME
liste_de_expression ;

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

- **sur les expressions numériques**
  - un calcul numérique ou de dates exprimé avec les opérateurs $+$, $-$, $*$, / n'a pas de valeur lorsqu'au moins une des composantes n'a pas de valeur

```
SELECT no_AV, 2*CAP/3 AS CAP_RED
FROM avions
WHERE no_AV = 320 ;
```

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

- **sur les chaînes de caractères**
  - un calcul de chaînes exprimé avec l'opérateur || n'a pas de valeur lorsque toutes ses composantes n'ont pas de valeur
  - la chaîne vide et l'absence de valeur sont confondues

```
SELECT no_CL, NOM_RUE
CL || '' || VILLE_CL AS ADR_CL
ou no_CL, NOM_RUE_CL || NULL || VILLE_CL AS ADR_CL
FROM clients
WHERE no_CL = 1035;
```

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

- **sur les comparaisons**
  - toute comparaison exprimée avec les opérateurs =, <>, >, <, <=, >=, LIKE qui comporte une expression qui n'a pas de valeur prend la valeur logique INDEFINIE
  - les comparaisons ignorent les lignes où il y a absence de valeur

```
SELECT * FROM pilotes
WHERE NAISS_PIL <> 1960 AND VILLE <> 'Paris' ;
```

  - comparaisons indéfinies :

```
SELECT *
FROM avions
WHERE NULL = NULL OR '' = '' OR '' LIKE '%'
OR 'A' LIKE '' OR 'A' NOT LIKE '' ;
```

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

table de vérité pour le connecteur $\wedge$

$\wedge$	VRAI	FAUX	INDEFINI
VRAI	VRAI	FAUX	INDEFINI
FAUX	FAUX	FAUX	FAUX
INCONNIE	INDEFINI	FAUX	INDEFINI

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

table de vérité pour le connecteur $\vee$

$\vee$	VRAI	FAUX	INDEFINI
VRAI	VRAI	VRAI	VRAI
FAUX	VRAI	FAUX	INDEFINI
INDEFINI	VRAI	INDEFINI	INDEFINI

table de vérité pour le connecteur $\neg$

	$\neg$
VRAI	FAUX
FAUX	VRAI
INDEFINI	INDEFINI

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

- **équivalences disjonctives**
  - expr NOT BETWEEN expr_1 AND expr_2
    $\Leftrightarrow$ expr $<$ expr_1 OR expr $>$ expr_2
  - expr IN ($\text{expr}_1 \cdots \text{expr}_N$)
    $\Leftrightarrow$ expr $=$ expr_1 OR $\cdots$ OR expr $=$ expr_N
  - expr op ANY ($\text{expr}_1 \cdots \text{expr}_N$)
    $\Leftrightarrow$ expr op expr_1 OR $\cdots$ OR expr op expr_N

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

Les expressions suivantes :

- expr NOT BETWEEN expr_1 AND expr_2
- expr IN ($\text{expr}_1 \cdots \text{expr}_N$)
- expr op ANY ($\text{expr}_1 \cdots \text{expr}_N$)

sont vraies ssi expr a une valeur et si au moins une des expressions expr_1, expr_N a une valeur qui satisfait les comparaisons

```
SELECT NUM_PIL FROM pilotes
WHERE VILLE IN ('Marseille', 'Nice', '') ;
```

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

- **équivalences conjonctives**
  - expr BETWEEN expr_1 AND expr_2
    $\Leftrightarrow$ expr >= expr_1 AND expr <= expr_2
  - expr NOT IN ($\text{expr}_1 \cdots \text{expr}_N$)
    $\Leftrightarrow$ expr <> expr_1 AND $\cdots$ AND expr <> expr_N
  - expr op ALL ($\text{expr}_1 \cdots \text{expr}_N$)
    $\Leftrightarrow$ expr op expr_1 AND $\cdots$ AND expr op expr_N

SQL comme Langage de Requêtes

Traitement de l'absence de valeur

Les expressions suivantes :

- expr BETWEEN expr_1 AND expr_2
- expr NOT IN ($\text{expr}_1 \cdots \text{expr}_N$)
- expr op ALL ($\text{expr}_1 \cdots \text{expr}_N$)

sont vraies ssi expr a une valeur et si au moins toutes les expressions expr_1, expr_N ont une valeur qui satisfait la comparaison

```
SELECT NUM_PIL FROM pilotes
WHERE VILLE NOT IN ('Marseille', 'Nice', '');
```

SQL comme Langage de Requêtes

Traitement de l'absence de valeur
recherche de l'absence de valeur

SELECT liste de nom_de_colonne FROM nom_de_table
WHERE expression IS [NOT] NULL ;

- pour connaître tous les vols auxquels on n'a pas encore affecté d'avions

```
SELECT no_VOL
FROM vols
WHERE no_AV IS NULL ;
```

SQL comme Langage de Requêtes

Ordonner les réponses

SELECT liste de nom_de_colonne
FROM nom_de_table
[WHERE expression]
ORDER BY { expression | position } [ASC | DESC]
[{ expression | position } [ASC | DESC]] ;

SQL comme Langage de Requêtes

Ordonner les réponses

- pour connaître les horaires des vols triés par ordre croissant des dates et heures de départ

```
SELECT no_VOL, DATE_d, DATE_a
FROM vols
ORDER BY DATE_d ;
```

SQL comme Langage de Requêtes

Les fonctions de groupe

- les fonctions de groupe calculent les résultats à partir d'une collection de valeurs.

  COUNT (*) comptage des lignes
  COUNT ([DISTINCT | ALL]) comptage des valeurs
  MAX ([DISTINCT | ALL]) maximum des valeurs
  MIN ([DISTINCT | ALL]) maximum des valeurs
  SUM ([DISTINCT | ALL]) minimum des valeurs
  AVG ([DISTINCT | ALL]) moyenne des valeurs
  STDDEV ([DISTINCT | ALL]) écart-type des valeurs
  VARIANCE ([DISTINCT | ALL]) variance des valeurs

SQL comme Langage de Requêtes

Les fonctions de groupe

- pour connaître le nombre d'avions

```
SELECT COUNT(*) NBR_AV
FROM avions ;
```

- pour connaître le nombre d'heures de vols du pilote 4020

```
SELECT SUM(TIMEDIFF(D_a,D_d)) NBR_H
FROM vols
WHERE no_PIL = 4020 ;
```

SQL comme Langage de Requêtes

Les regroupements de lignes

- les fonctions de groupe calculent les résultats à partir d'une collection de valeurs.

  SELECT liste_d'expressions1
  FROM nom_de_table
  GROUP BY liste_d'expressions2 ;

  les expressions de liste_d'expressions1 doivent être des expressions formées uniquement :
  - d'expressions de liste_d'expressions2
  - de fonctions de groupe
  - de constantes littérales

SQL comme Langage de Requêtes

Les regroupements de lignes

- pour connaître le nombre d'avions affectés à chaque ville d'affectation d'un avion

```
SELECT LOC, COUNT(*) NBR_AV
FROM avions
GROUP BY LOC ;
```

SQL comme Langage de Requêtes

Les regroupements de lignes

avions triés sur LOC	
no_AV	LOC
820	Ajaccio
715	Ajaccio
...	...
720	Marseille
225	Marseille
456	Marseille
...	...
531	Toulouse

$\Rightarrow$

table résultat	
LOC	NBR_AV
Ajaccio	2
...	...
Marseille	3
...	...
Toulouse	1

SQL comme Langage de Requêtes

Les regroupements de lignes

- pour connaître le nombre de vols qui ont la même durée

```
SELECT TIMEDIFF(D_a,D_d) DUR_VOL, COUNT(*)
NBR_VOL
FROM vols
GROUP BY TIMEDIFF(D_a, D_d);
```

SQL comme Langage de Requêtes

Les regroupements de lignes

- regroupement de lignes sélectionnées

```
SELECT liste_d'expressions1
FROM nom_de_table
WHERE condition
GROUP BY liste_d'expressions2 ;
```

- pour connaître le nombre d'avions différents utilisés par chaque pilote assurant un vol

```
SELECT LOC, no_PIL, COUNT(DISTINCT no_AV) NBR_AV
FROM vols
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL ;
```

SQL comme Langage de Requêtes

Conditions sur l'ensemble des lignes

SELECT liste_d'expressions
FROM nom_de_table
[WHERE condition]
HAVING condition_sur_lignes ;

les expressions de liste_d'expressions et condition_sur_lignes doivent être formées uniquement :

- de fonctions de groupe
- de constantes littérales

SQL comme Langage de Requêtes

Conditions sur l'ensemble des lignes

- pour savoir si le pilote 4010 assure tous les vols avec un avion diff ;érent à chaque fois

```
SELECT 'OUI' REP
FROM vols
WHERE no_PIL = 4010
HAVING COUNT(*) = COUNT(DISTINCT no_AV);
```

SQL comme Langage de Requêtes

Conditions sur l'ensemble des lignes

SELECT liste_d'expressions
FROM nom_de_table
[WHERE condition]
GROUP BY liste_d'expressions2
HAVING condition_sur_lignes ;

les expressions de liste_d'expressions et condition_sur_lignes doivent être formées uniquement :

- d'expressions de liste_d'expressions2
- de fonctions de groupe
- de constantes littérales

SQL comme Langage de Requêtes

Conditions sur l'ensemble des lignes

- pour connaître les pilotes qui conduisent au moins deux avions différents

```
SELECT no_PIL
FROM vols
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL
HAVING COUNT(DISTINCT no_AV) >= 2 ;
```

SQL comme Langage de Requêtes

Opérateurs ensemblistes

```
SELECT liste_d'expressions1
FROM nom_de_table
[ WHERE condition ]
[ GROUP BY liste_d'expressions2]
UNION | UNION ALL | INTERSECT | MINUS
SELECT liste_d'expressions3
FROM nom_de_table
[ WHERE condition ]
[ GROUP BY liste_d'expressions4] ;
```

SQL comme Langage de Requêtes

Opérateurs ensemblistes

- pour connaître les villes qui sont soit des villes de départ soit des villes d'arrivées d'un vol

```
SELECT V_d VILL
FROM vols
WHERE V_d IS NOT NULL
UNION
SELECT V_a
FROM vols
WHERE V_a IS NOT NULL ;
```

SQL comme Langage de Requêtes

Opérateurs ensemblistes

- pour connaître le nombre de vols assurés par chaque pilote

```
SELECT no_PIL, COUNT(*) NBR_VOL
FROM vols
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL
UNION ALL
(SELECT no_PIL, 0
FROM pilotes
MINUS
SELECT no_PIL, 0
FROM vols) ;
```

SQL comme Langage de Requêtes

Produit cartésien

SELECT liste_d'expressions
FROM liste_de(nom_de_table [alias])
[WHERE condition] ;

- pour connaître le coût de chaque classe du vol V900 lorsqu'on les applique au vol V100

SELECT Classe, COEF_PRIX * COUT_VOL COUT
FROM defclasses D, vols V
WHERE D.no_VOL = 'V900' AND V.no_VOL = 'V100' :

SQL comme Langage de Requêtes

Opérateur de jointure naturelle

```
SELECT liste_d'expressions
FROM liste_de(nom_de_table [ alias ])
WHERE expr comp expr [AND | OR expr comp expre ] ;
```

ou

```
SELECT liste_d'expressions
FROM
        nom_de_table [ alias ]
        INNER JOIN
        nom_de_table [ alias ]
        ON expr comp expr [ AND | OR expr comp expre ] ;
```

SQL comme Langage de Requêtes

Opérateur de jointure naturelle

- pour connaître le nombre de places de chaque vol qui a été affecté à un avion

```
SELECT no_VOL, CAP
FROM vols, avions
WHERE vols.no_AV = avions.no_AV ;
```

Opérateur de jointure naturelle : exemple (1)

table vols	
no_VOL	no_AV
V101	560
V141	101
V169	101
V631	NULL
V801	240

table avions	
no_AV	CAP
101	350
240	NULL
560	250

equi-jointure sur no_AV

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL

Opérateur de jointure naturelle : exemple (2)

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL

projection sur no_VOL, CAP

vols.no_VOL	avions.CAP
V101	250
V141	350
V169	350
V801	NULL

SQL comme Langage de Requêtes

Opérateur de semi-jointure externe

```
SELECT liste_d'expressions
FROM nom_de_table1, nom_de_table2
WHERE expr_table1 comp nom_table2.col(+)
[ AND expr_table1 comp nom_table2.col(+) ]
```

ou

```
SELECT liste_d'expressions
FROM
      nom_de_table1 [ alias ]
      LEFT JOIN | RIGHT JOIN
      nom_de_table2 [ alias ]
      ON expr comp expr [ AND | OR expr comp expr ] ;
```

SQL comme Langage de Requêtes

Opérateur de semi-jointure externe

- pour connaître le nombre de places de chaque vol (même lorsqu'aucun avion n'est affecté au vol)

```
SELECT no_VOL, CAP
FROM vols V LEFT JOIN avions A
ON V.no_AV = A.no_AV ;
```

Opérateur de semi-jointure externe : exemple (1)

table vols	
no_VOL	no_AV
V101	560
V141	101
V169	101
V631	NULL
V801	240

table avions	
no_AV	CAP
101	350
240	NULL
560	250

equi-jointure sur no_AV

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL

Opérateur de semi-jointure externe : exemple (2)

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL

equi-jointure externe

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240
V631	NULL

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL
NULL	NULL

Opérateur de semi-jointure externe : exemple (2)

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240
V631	NULL

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL
NULL	NULL

projection sur no_VOL, CAP

vols.no_VOL	avions.CAP
V101	250
V141	350
V169	350
V801	NULL
V631	NULL

Fonctions diverses pour requêtes SQL

Fonctions numériques (1)

- SIGN(nombre) signe du nombre spécifié
- ABS(nombre) valeur absolue
- ACOS(nombre) arc cosinus
- ASIN(nombre) arc sinus
- ATAN(nombre) arc tangente
- COS(nombre) cosinus
- SIN(nombre) sinus
- TAN(nombre) tangente
- COSH(nombre) cosinus hyperbolique
- SINH(nombre) sinus hyperbolique
- TANH(nombre) tangente hyperbolique

Fonctions diverses pour requêtes SQL

Fonctions numériques (2)

- EXP(puissance) e élevé à la puissance
- LN(nombre) logarithme naturel
- LOG(base,nombre) logarithme en base quelconque
- SQRT(nombre) racine carrée
- POWER(nombre,puissance) puissance d'un nombre
- MOD(dividende,diviseur) modulo (reste de la division)
- CEIL(nombre) le plus petit entier plus grand que le nombre passé en argument
- FLOOR(nombre) le plus grand entier plus petit ou égal au nombre passé en argument

Fonctions de chaines pour requêtes SQL

Fonctions de chaînes

Fonction	Description
TRIM(x)	Supprime les espaces de début et de fin de chaîne.
LOWER(x)	Converti en minuscules.
UPPER(x)	Converti en majuscules.
LONGUEUR(x)	Retourne la taille de la chaîne.
LOCATE(x,y)	Retourne la position de la dernière occurrence de x dans y. Retourne 0 si x n'est pas trouvé dans y.
CONCAT(x,y,...)	Concatène ses arguments.
SUBSTRING(s,i,n)	Retourne les n derniers caractères de s en commençant à partir de la position i.
SOUNDEX(x)	Retourne une représentation phonétique de x.

```
SELECT UPPER(nom)
FROM clients
WHERE SOUNDEX(nom) = SOUNDEX('Dupond')
```

On affiche en majuscules le nom de tous les clients dont le nom ressemble à 'Dupond'

Fonctions de dates pour requêtes SQL

Fonctions de dates et heures

Fonction	Description
NOW()	Retourne la date et heure du jour.
TO_DAYS(x)	Conversion de la date X en nombre de jours depuis le 1er janvier 1970.
DAYOFWEEK(x)	Retourne le jour de la semaine de la date x sous la forme d'un index qui commence à 1 (1=dimanche, 2=lundi...)
DAYOFMONTH(x)	Retourne le jour du mois (entre 1 et 31).
DAYOFYEAR(x)	Retourne le jour de l'année (entre 1 et 366).
SECOND(x), MINUTE(x), HOUR(x), MONTH(x), YEAR(x), WEEK(x)	Retournent respectivement les secondes, minutes, heures, mois, année et semaine de la date.

```
SELECT titre
FROM article
WHERE (TO_DAYS(NOW()) – TO_DAYS(parution)) < 30
```

Cet exemple affiche le titre des articles parus il y a moins de 30 jours.

Fonctions diverses pour requêtes SQL

Sous-requêtes

imbrication de sous-requêtes dans la clause WHERE

```
SELECT  projection
FROM nom_de_table
WHERE condition
      (SELECT  projection
      FROM nom_de_table
      WHERE condition, ···) ;
```

Fonctions diverses pour requêtes SQL

Sous-requêtes : donnant une seule ligne

```
SELECT  projection
FROM nom_de_table
WHERE expr op
      (SELECT  projection
      FROM nom_de_table
      WHERE condition, ··· );
```

op $\in \{ =, <>, <, <=, >, >= \}$

Fonctions diverses pour requêtes SQL

Sous-requêtes : donnant une seule ligne

- pour connaître les vols qui utilisent le même avion que celui utilisé par le vol V101

```
SELECT no_VOL
FROM vols
WHERE no_Av = (SELECT no_Av
      FROM vols
      WHERE no_VOL = 'V101' );

      FROM vols );
```

Fonctions diverses pour requêtes SQL

Sous-requêtes donnant au plus une ligne

SELECT projection
FROM nom_de_table
WHERE $(\text{expr}_1, \cdots \text{expr}_n)$ op
    (SELECT $(\text{expr}_1, \cdots \text{expr}_n)$
    FROM nom_de_table
    WHERE condition, $\cdots$) ;

op $\in \{=, <>\}$

Fonctions diverses pour requêtes SQL

Sous-requêtes donnant au plus une ligne

- pour connaître le vols qui assure le même trajet que celui du vol V101 mais 2 jours plus tard

```
SELECT no_VOL
FROM vols
WHERE (V_d, V_a, D_d, D_a) =
(SELECT (V_d, V_a, D_d+2, D_a+2)
      FROM vols
      WHERE no_VOL = 'V101' );
```

Fonctions diverses pour requêtes SQL

Sous-requêtes donnant 0, 1 ou plusieurs lignes

```
SELECT  projection
FROM nom_de_table
WHERE expr NOT IN
      (SELECT  (projection
      FROM ··· );
```

```
SELECT  projection
FROM nom_de_table
WHERE expr op ANY | ALL
      (SELECT  (projection
      FROM ··· );
      op ∈ { = , <>, < , <= , > , >= }
```

Fonctions diverses pour requêtes SQL

Sous-requêtes donnant 0, 1 ou plusieurs lignes

- pour connaître le vols qui sont assurés par un pilote qui habite Paris

```
SELECT no_VOL
FROM vols
WHERE no_PIL IN
(SELECT no_PIL
      FROM pilotes
      WHERE VILLE = 'Paris' );
```

Fonctions diverses pour requêtes SQL

Sous-requêtes donnant 0, 1 ou plusieurs lignes

- pour connaître les pilotes qui n'assurent aucun vol

```
SELECT no_PIL
FROM pilotes
WHERE no_PIL NOT IN
(SELECT no_PIL
      FROM vols
      WHERE no_PIL IS NOT NULL );
```

Fonctions diverses pour requêtes SQL

Sous-requêtes d'existence

```
SELECT  projection
FROM nom_de_table [Alias]
WHERE [NOT] EXISTS
      (SELECT  (projection
      FROM ··· ) ;
```

Fonctions diverses pour requêtes SQL

Sous-requêtes d'existence

- pour connaître les avions qui sont conduits par au moins un pilote de Marseille

```
SELECT DISTINCT no_AV
FROM vols
WHERE EXISTS
(SELECT *
      FROM pilotes
      WHERE no_PIL = vols.no_PIL
      AND VILLE = 'Marseille');
```