Bases de données
Cours 4 : Le langage SQL pour ORACLE

Odile PAPINI

POLYTECH
Université d'Aix-Marseille
odile.papini@univ-amu.fr
http://odile-papini.pedaweb.univ-amu.fr/sources/BD.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 : J. Le Maitre : http ://lemaitre.univ-tln.fr/cours.htm
- Support de cours : C. Sabatier, Université de la Méditerranée.

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 Oracle

SQL : Quelques repères historiques

- Oracle est SGBD qui utilise SQL
- PL/SQL dit (L4G) langage procédural
- nombreux programmes utilitaires :
  - SQL*PLUS SQL interactif
  - SQL*FORMS : saisie et visualisation des données avec des formulaires
  - SQL*REPORTWRITER : rapports imprimés
  - WebDB : interface avec le Web

SQL comme LDD

Identificateurs :

- lettre suivie par : lettre ou chiffre ou _ ou ♯ ou $
- chaîne de caractères entre guillemets
- maximum 30 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

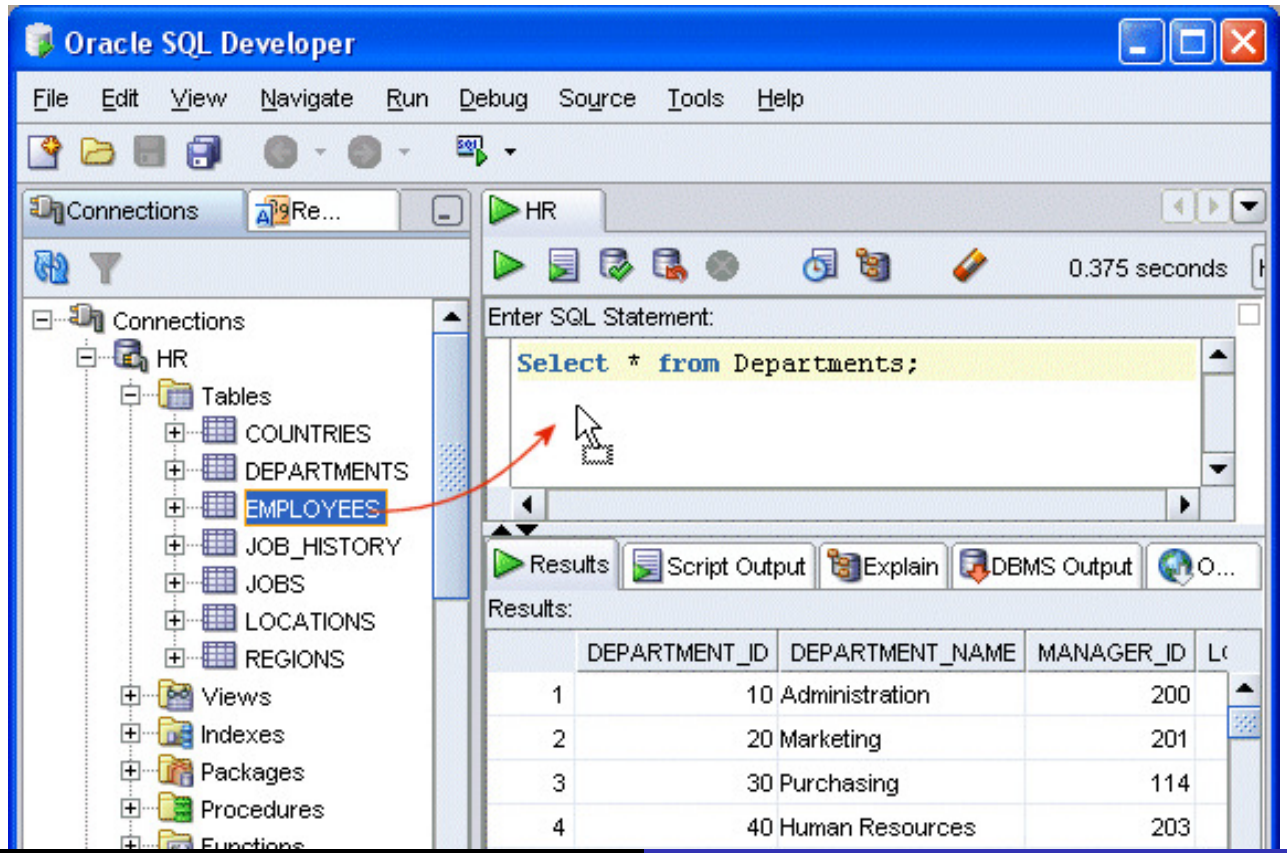
SQL comme LDD

Tables :

- relations d'un schéma relationnel stockées sous tables
- table : formée de lignes et de colonnes
- SQL2 : nom d'une table précédé du nom d'un schéma
- ORACLE :
  - 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 :



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 SQL2 :

- types pour les chaînes de caractères
- types numériques
- types temporels (date, heure, $\cdots$)
- SQL2 n'a pas de type pour les données volumineuses (images , sons)
- SQL2 ne permet pas à l'utilisateur de créer ses propres types

SQL comme LDD

SQL2 : 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 2000 octets
- VARCHAR(*taille_max*)
  - chaînes de caractères de longueur variable
  - taille comprise entre 1 et 4000 octets
- constantes
  - chaînes de caractères entre guilllemets

SQL comme LDD

ORACLE : Types pour les chaînes de caractères

- CHAR(*taille*) ou NCHAR(*taille*)
- VARCHAR(*taille_max*)
  MAIS de préférence
- VARCHAR2(*taille_max*) ou NVARCHAR2(*taille_max*)
- constantes
  - chaînes de caractères entre côtes

SQL comme LDD

ORACLE : Types pour les chaînes de caractères

exemples

- NCHAR(5) : chaînes de 5 caractères
- VARCHAR2(20) : chaînes de 20 caractères au plus
- 'Administration', 'Marketing'

SQL comme LDD

SQL2 : Types numériques

- types numériques pour les entiers :
  - SMALLINT pour 2 octets
  - INTEGER pour 4 octets
- types numériques pour les décimaux à virgule flottante :
  - REAL
  - DOUBLE PRECISION ou FLOAT
- types numériques pour les décimaux à virgule fixe :
  - DECIMAL(*nb_chiffres, nb_décimales*)
  - NUMERIC(*nb_chiffres, nb_décimales*)
- constantes
  - exemples : 43.8, -13, 5.3E-6

SQL comme LDD

ORACLE : Types numériques

exemples

- ORACLE accepte tous les types numériques de SQL2 mais il les traduit dans ses propres types
- NUMBER : nombre en virgule flottante avec jusqu'à 38 chiffres significatifs
- NUMBER(*nb_chiffres, nb_décimales*) : nombre en virgule fixe

SQL comme LDD

SQL2 : Types temporels

- DATE
  - pour les dates
- TIME
  - pour les heures, minutes et secondes
- TIMESTAMP
  - pour un moment précis : date et heure, minutes et secondes (précision jusqu' à la microseconde)

SQL comme LDD

ORACLE : Types temporels

- le type DATE remplace DATE et TIME de SQL2
- DATE correspond à une date avec une précision jusquà la seconde
- constantes
  - exemples : '1/05/2007' ou '1 MAY 2007'

SQL comme LDD

SQL2 : Type boolén

- BIT
  - pour enregistrer la valeur d'un bit
  - exemples : BIT(1), BIT(4)
- pas supporté par ORACLE

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 NUMBER(4)
  NOM_AV VARCHAR2(20)
  CAP NUMBER(4)
  LOC VARCHAR2(15)

- pilotes(no_PIL, NOM_PIL, VILLE)

  no_PIL NUMBER(4)
  NOM_PIL VARCHAR2(20)
  VILLE VARCHAR2(15)

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

  no_VOL VARCHAR2(5)
  no_AV NUMBER(4)
  no_PIL NUMBER(4)
  V_d VARCHAR2(15) H_d DATE
  V_a VARCHAR2(15) H_a DATE

SQL comme LDD

ORACLE : Types pour objets larges

LOB : large objet formé par :

- valeur du LOB : une grande donnée (jusqu'à 4 Go)
- index du LOB : structure d'accès
- le type localisateur du LOB pointeur vers l'endroit où il est stocké

Types pour objets larges

- CLOB ou NCLOB : pour le stockage de grandes chaînes de caractères
- BLOB : pour le stockage de grandes chaînes d'octets
- BFILE : pour le stockage de données binaires dans un fichier extérieur à la base

SQL comme LDD

ORACLE : Autres types

Les types chaînes d'octets

- RAW(taille) : 2000 octets max
- LONG RAW : 2 Go max
  - conversion automatique d'une chaîne d'octets en une chaîne de caractères représentant un nombre hexadécimal, et inversement

SQL comme LDD

ORACLE : Autres types

Le type adresse de ligne

- ROWID : une valeur de type est un nombre en base 64 dont les chiffres sont représentés par : A-Z, a-z, 0-9, +, /, il s'écrit : *OOOOOOFFFBBBBBBLLL*
  - *OOOOOO* : numéro de l'objet qui contient la ligne (6 chiffres en base 64)
  - *FFF* : numéro du fichier dans la BD (3 chiffres en base 64, le premier chiffre a le numéro 1 (AAB))
  - *BBBBBB* : numéro du bloc dans le fichier (6 chiffres en base 64)
  - *LLL* : numéro du ligne dans le bloc (3 chiffres en base 64), le premier chiffre a le numéro 0 (AAA))

SQL comme LDD

ORACLE : 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
- pour les types LOB
  - l'absence de valeur est l'absence de localisateur
  - un LOB vide n'est pas une 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 : :=
    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(<u>no_AV</u>, nom_AV, CAP, LOC)

```
CREATE TABLE avions (
    no_AV NUMBER(4)
        CONSTRAINT Cle_P_avions PRIMARY KEY,
    NOM_AV VARCHAR2(20),
    CAP NUMBER(4)
        CONSTRAINT Dom_CAP_avions CHECK (CAP ≥ 4),
    LOC VARCHAR2(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(no_VOL, ♯no_AV, ♯no_PIL, V_d, V_a, H_d, H_a)

```
CREATE TABLE vols (
    no_VOL VARCHAR2(5)
        CONSTRAINT Cle_P_vols PRIMARY KEY,
    no_AV NUMBER(4)
        CONSTRAINT Ref_no_AV_vols REFERENCES avions,
    no_PIL NUMBER(4)
        CONSTRAINT Ref_no_PIL_vols REFERENCES pilotes,
    V_d VARCHAR2(15) NOT NULL,
    V_a VARCHAR2(15) NOT NULL,
    H_d DATE,
    H_a DATE,
    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, ORACLE :

- 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 NUMBER(8)

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 | $\cdots$

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 nom_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**
  - afficher les dates de départ et d'arrivée de chaque vol en décalant les dates
    SELECT no_VOL, D_d + 1/24 D_d, D_a + 1/24 D_a
    FROM vols ;
  - pour connaître la durée en heures de tous les vols
    SELECT no_VOL, 24 *(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 (24 *(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 ou 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 (<>| > | < | <= | >= |) ALL
liste_de_expression ;

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

- pour connaître tous les vols dont le départ est à plus de 5 jours et dont la durée est moins de 5 heures
  SELECT no_VOL, D_d, 24*(D_a - D_d) Durée
  FROM vols
  WHERE D_d > ALL (sysdate +5, D_a -5/24);

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
INDEFINI	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 ($expr_1 \cdots expr_N$)
- expr op ALL ($expr_1 \cdots expr_N$)

sont vraies ssi expr a une valeur et si 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

Traitement de l'absence de valeur
Donner une valeur à l'absence de valeur

NVL ($expr_1$, $expr_2$) = $expr_1$ si elle définie, $expr_2$ sinon

$expr_1$ et $expr_2$ doivent être de même type

- pour qu'une capacité d'avion inconnue soit considérée comme nulle

```
SELECT no_VOL, NVL(CAP,0)
FROM avions;
```

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]) minimum des valeurs
  SUM ([DISTINCT | ALL]) somme 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(24 *(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 24*(D_a - D_d) DUR_VOL, COUNT(*) NBR_VOL
FROM vols
GROUP BY 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 différent 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

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' );
```

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') ;
```

Les vues

Les vues

- une vue est une table virtuelle résultat d'une requête
- rôle d'une vue
  - réduire la complexité syntaxique des requêtes
  - définir les schémas externes.
  - définir des contraintes d'intégrité.
  - définir un niveau additionnel de sécurité en restreignant l'accès à un sous ensemble de lignes et/ ou de colonnes.

Les vues (exemple)

vols						
No_VOL	No_AV	No_PIL	V_d	V_a	H_d	H_a
it200	100	1	nice	paris	7	8
it201	100	2	paris	toulouse	11	12
it202	101	1	paris	nice	12	13
it203	102	3	paris	toulouse	9	11
it204	104	3	nice	paris	17	18

Création d'une vue ligne

ligne	
V_d	V_a
nice	paris
paris	toulouse
paris	nice

Les vues : création

Les vues

- création d'une vue de schéma externe

  CREATE [OR REPLACE] [FORCE | NO FORCE]
  VIEW nom_ de_table [(liste de nom_de_colonne)]
  AS requête [WITH CHECK OPTION | WITH READ ONLY] ;

  - création de la vue correspondant aux vols qui partent de Paris

    CREATE VIEW vols_d_Paris
    AS SELECT no_VOL, V_a, H_d, H_a
    FROM vols
    WHERE V_d = 'Paris' ;

Les vues : interrogation

- interroger une vue
  - interrogation de la vue correspondant aux vols qui partent de Paris

    SELECT * FROM vols_d_Paris ;

- supprimer une vue

  DROP VIEW nom_ de_vue ;

  - suppression de la vue correspondant aux vols qui partent de Paris

    DROP VIEW vols_d_Paris ;

Requêtes avec vues

Exemples

- pour connaître les pilotes qui assurent le plus grand nombre de vols

```
CREATE VIEW nbre_ de_vols_par_pil
AS SELECT no_PIL, count(*) AS NBR_VOLS
FROM vols V
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL ;

SELECT no_PIL FROM nbre_ de_vols_par_pil
WHERE NBR_VOLS =
      (SELECT max( NBR_VOLS)
      FROM nbre_ de_vols_par_pil) ;
```

Les vues : mise à jour

- opérations sur les vues

  INSERT
  UPDATE
  DELETE

- restrictions : Ces instructions ne s'appliquent pas aux vues qui contiennent :
  - une jointure
  - un opérateur ensembliste : UNION, INTERSECT, MINUS
  - une clause GROUP BY, CONNECT BY, ORDER BY ou START WITH
  - la clause DISTINCT, une expression ou une pseudo-colonne dans la liste de sélection des colonnes.

Requêtes avec vues

- création de la vue pour la personne qui définit les vols

```
CREATE VIEW def_vols
AS SELECT no_VOL, V_d, D_d, V_a, D_a
FROM vols
WHERE no_VOL IS NULL AND no_PIL IS NULL ;
```

- définir un nouveau vol

```
INSERT INTO def_vols VALUES
('V999', 'Marseille', to_date('01/05/07 10 :30', 'DD/MM/RR
HH :MI'), 'Paris', to_date('01/05/07 10 :30', 'DD/MM/RR
HH :MI')) ;
```

Requêtes avec vues

- supprimer un vol non affecté

```
DELETE FROM def_vols
WHERE no_VOL = 'V998' ;
```

- modifier un vol non affecté

```
UPDATE def_vols
SET D_d= D_d + 1 / 24, D_a= D_a + 1 / 24 WHERE
no_VOL = 'V998' ;
```

- connaître les vols non affectés

```
SELECT * FROM def_vols ;
```

Requêtes avec vues

- création de la vue pour la personne qui définit les vols

```
CREATE VIEW def_vols
AS SELECT no_VOL, V_d, D_d, V_a, D_a
FROM vols
WHERE no_VOL IS NULL AND no_PIL IS NULL ;
```

- modifier un vol non affecté

```
UPDATE def_vols
SET D_d= D_d + 1 / 24, D_a= D_a + 1 / 24
WHERE no_VOL = 'V998' ;
```

- connaître les vols non affectés

```
SELECT * FROM def_vols ;
```

Requêtes avec vues

- création de la vue pour la personne qui affecte un avion et un pilote à un vol

```
CREATE VIEW affect_vols
AS SELECT no_VOL, no_AV, no_PIL
FROM vols ;
```

- affecter un avion et un pilote à un nouveau vol

```
UPDATE affect_vols
SET no_AV = 101, no_PIL = 5050
WHERE no_VOL = 'V999'
AND no_AV IS NUL AND no_PIL IS NULL ;
```

- affecter un nouvel avion à un vol

```
UPDATE affect_vols
SET no_AV = 202
WHERE no_VOL = 'V999' ;
```

Requêtes avec vues

- création de la vue pour la personne qui affecte un avion et un pilote à un vol

```
CREATE VIEW affect_vols
AS SELECT no_VOL, no_AV, no_PIL
FROM vols ;
```

- permuter l'affectation des pilotes de 2 vols

```
UPDATE affect_vols A1
SET no_PIL =
    (SELECT no_PIL
    FROM affect_vols A2
    WHERE
    (A1.no_VOL = 'V100' AND A2.no_VOL = 'V200')
    OR
    (A1.no_VOL = 'V200' AND A2.no_VOL = 'V100'))
    WHERE no_VOL = 'V100' OR no_VOL = 'V200' ;
```

Les vues : contrôle de mise à jour

- création d'une vue de vérification : contrôle de l'insertion ou de la modificaion de ligne

```
CREATE VIEW nom_de_vue
AS requête
WITH CHECK OPTION ;
```

- vérification des contraintes de domaine (interdiction des valeurs inconnues)

```
CREATE VIEW a_avions
AS SELECT * FROM avions
WHERE
no_AV > 0
AND CAP > 1
AND NOM_AV IN ('Airbus', 'Boeing', 'Caravelle')
WITH CHECK OPTION ;
```

Requêtes avec vues

- vérification des contraintes de domaine (autorisation des valeurs inconnues)

```
CREATE VIEW aa_avions
AS SELECT * FROM avions
WHERE
no_AV > 0
AND (CAP IS NULL OR CAP > 1)
AND (NOM_AV IS NULL OR IN ('Airbus', 'Boeing',
'Caravelle'))
WITH CHECK OPTION;
```

Requêtes avec vues

Contraintes de référence

1) valider l'insertion dans la table référenant
2) valider la suppression dans la table référencée

règle d'adéquation : les insertions et les suppressions se font toujours au travers des vues

Requêtes avec vues

Exemple : expression de clés étrangères de la relation vols

- validation des insertions dans vols

```
CREATE VIEW a_avions
AS SELECT * FROM vols
WHERE
no_AV > 0
AND (no_PIL IS NULL OR no_PIL IN(SELECT no_PIL FROM
pilotes))
AND (NOM_AV IS NULL OR IN (SELECT NOM_AV FROM
avions))
WITH CHECK OPTION;
```

Requêtes avec vues

Exemple : expression de clés étrangères de la relation vols

- validation des suppressions dans avions et dans pilotes

```
CREATE VIEW d_avions
AS SELECT * FROM avions A
WHERE NOT EXISTS ( SELECT * FROM vols V WHERE
A.no_AV = V.no_AV) ;

CREATE VIEW d_pilotes
AS SELECT * FROM pilotes P
WHERE NOT EXISTS ( SELECT * FROM vols V WHERE
P.no_PIL = V.no_PIL) ;
```

Requêtes avec vues

Exemple : ajout des contraintes de domaine de 2 façons

- agrégation

```
CREATE VIEW ad_avions
AS SELECT * FROM avions A
WHERE no_AV > 0
AND CAP > 1
AND NOM_AV IN ('Airbus', 'Boeing', 'Caravelle')
AND NOT EXISTS ( SELECT * FROM vols V WHERE
A.no_AV = V.no_AV)
WITH CHECK OPTION;
```

Requêtes avec vues

Exemple : ajout des contraintes de domaine de 2 façons

- modulaire

```
CREATE VIEW a_avions
AS SELECT * FROM avions
WHERE no_AV > 0
AND CAP > 1
AND NOM_AV IN ('Airbus', 'Boeing', 'Caravelle')
AS SELECT * FROM pilotes P
WITH CHECK OPTION;
CREATE VIEW ad_avions
AS SELECT * FROM a_avions
WHERE NOT EXISTS ( SELECT * FROM vols V WHERE
A.no_AV = V.no_AV);
```

Requêtes avec vues

Expression de contraintes générales

Exemple : empêcher l'affectation d'un même avion à deux vols différents dont les tranches horaires se chevauchent

```
CREATE VIEW a_vols
AS SELECT * FROM vols V1
WHERE NOT EXISTS (
    SELECT * FROM vols V2
    WHERE V1.no_AV = V2.no_AV
    AND NVL(V2.D_a, V2.D_d) >= NVL(V1.D_d, V1.D_a)
    AND NVL(V1.D_a, V1.D_d) >= NVL(V2.D_d, V2.D_a))
    WITH CHECK OPTION;
```

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 diverses pour requêtes SQL

Fonctions numériques (3)

- ROUND(nombre [position]) arrondi à la position spécifiée. Un entier positif (resp. négatif) indique une position après (resp. avant) la virgule. Par défaut il y a arrondi à l'unité.

- TRUNC(nombre [position]) troncature à la position spécifiée (voir ROUND). Par défaut il y a troncature à l'unité.

Fonctions diverses pour requêtes SQL

Conversion en chaîne de caractères (1)

- TO_CHAR(nombre[format[nls_param]])
  - format est une chaîne de caractères formée des éléments suivants :
    - 9 un chiffre quelconque
    - 0 un chiffre ou un zéro si absence de chiffres
    - $ symbole monétaire américain
    - B remplace le nombre 0 par des blancs
    - MI signe du nombre post-fixé s'il est négatif, un blanc post-fixé sinon.
    - S signe du nombre
    - PR nombre mis entre "<" et ">" s'il est négatif, nombre mis entre 2 blancs sinon.

Fonctions diverses pour requêtes SQL

Conversion en chaîne de caractères (2)

- et des éléments suivants :
  - D caractère qui sépare la partie entière de la partie fractionnaire
  - G symbole de séparation de groupes de chiffres
  - C symbole monétaire international
  - L symbole monétaire local
  - . , caractères affichés tels quels
  - V9...9 multiplie le nombre par 10n o n est le nombre de 9 après V.
  - EEEE écriture scientifique du nombre
  - RM rm écriture en chiffres (entre 0 et 3999)
  - FM écriture scientifique du nombre

Fonctions diverses pour requêtes SQL

Conversion en chaîne de caractères (3)

- TO_CHAR(nombre[format[nls_param]])
  - nls_param est une chaîne de caractères formée à partir des expressions suivantes :
    - NLS_NUMERIC_CHARACTERS = "dg"
    - LS_CURRENCY = "symbole_monétaire_local"
    - NLS_ISO_CURRENCY = territoire

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères : recherche de sous chaîne

- INSTR(chaîne, sous_chaîne [position [n]])
- INSTRB(chaîne, sous_chaîne [position [n]])

  à partir d'une position (par défaut 1), exprimée en nombre de caractères ou d'octets, relative au début de la chaîne si position est positif, ou relative à sa fin si position est négatif, recherche la position de la nième (par défaut 1ère) occurrence de la sous chaîne, retourne 0 lorsque la recherche échoue.

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères : extraction de sous chaîne

- SUBSTR(chaîne, position [longueur])
- SUBSTRB(chaîne, position [longueur])

  à partir d'une position, extrait une sous chaîne de longueur donnée (par défaut jusqu'à la fin de la chaîne).

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères : remplacement ou suppression

- REPLACE(chaîne,sous_chaîne [sous_chaîne_de_remplacement]) remplace ou supprime toutes les occurrences d'une sous chaîne.
- LTRIM(chaîne [ensemble_caractères_à_supprimer])
- RTRIM(chaîne [ensemble_caractères_à_supprimer]) supprime à gauche (ou à droite) de la chaîne les occurrences des caractères à supprimer (par défaut 1 blanc).

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères : répétition

- LPAD(chaîne,longueur [chaîne_répétée])
- RPAD(chaîne,longueur [chaîne_répétée])

répétition, à gauche (ou à droite) de la chaîne, d'une autre chaîne (par défaut 1 blanc) jusqu'à obtenir la longueur spécifiée.

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères : minuscules, majuscules

- LOWER(chaîne) met en minuscules la chaîne.
- UPPER(chaîne) met en majuscules la chaîne.
- INITCAP(chaîne) met en majuscules les premières lettres de chaque mot.

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères

- LENGTH(chaîne) longueur de la chaîne en caractères
- LENGTHB(chaîne) longueur de la chaîne en octets
- CHR(code [USING NCHAR_CS])

  retourne le caractère du jeu de caractères de base ou du jeu de caractères local, qui correspond au code.
- ASCII(chaîne)

  retourne le code décimal du premier caractère.

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères : conversion de type

- TO_NUMBER(chaîne) retourne le nombre correspondant
- HEXTORAW(chaîne_hexa) retourne la chaîne d'octets correspondants
- CHARTOROWID(chaîne) retourne le ROWID correspondant
- TRANSLATE(chaîne USING CHAR_CS | NCHAR_CS)

  conversion d'un type de chaîne de caractères basé sur le jeu de caractères de base dans un type de chaîne de caractères basé sur le jeu de caractères national, et inversement.

Fonctions diverses pour requêtes SQL

Fonctions sur les chaîne de caractères : traduction (le type est inchangé)

- TRANSLATE(chaîne,liste_source, liste_destination)

  traduit la chaîne en remplaant chacun de ses caractères figurant la liste source par son correspondant dans la liste destination.
- CONVERT(chaîne[jeu_de_carac_dest[jeu_de_carac_source]])

  traduction d'un jeu de caractères dans un autre.
- SOUNDEX(chaîne)

  retourne la chaîne phonétique.

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : recherche

- SYSDATE retourne la date courante
- LAST_DAY(date) retourne la dernière date du mois qui contient la date passée en argument.
- NEXT_DAY(date, nom_jour_ds_semaine)
  retourne la première date postérieure à la date passée en argument et qui correspond au jour de la semaine passé en argument.

- Pour connaître la date du prochain lundi.
  - SELECT NEXT_DAY(SYSDATE,'Lundi') ProchainLundi FROM Dual ;

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : calcul sur les mois

- ADD_MONTHS(date, nbre_mois)

  retourne la date passée en argument augmentée d'un nombre de mois

- MONTHS_BETWEEN(date1, date2)

  retourne le nombre de mois qui séparent les 2 dates, avec éventuellement une partie fractionnaire correspondant à une partie de 31 jours.

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : calcul sur les mois

- SELECT
  month_between('16/03/99','01/02/99') NBmois1,
  to_date('16/03/99')-to_date('01/02/99') NBjours1
  month_between('16/04/99','01/03/99') NBmois2,
  to_date('16/04/99')-to_date('01/03/99') NBjours2 From
  Dual ;

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : conversion de date

- NEW_TIME(date,timezone1,timezone2) retourne la date passée en argument convertie par changement de zone horaire.
  - AST,ADT Atlantique standard ou décalé
  - BST,BDT Bering standard ou décalé
  - CST,CDT Central standard ou décalé
  - EST,EDT Oriental standard ou décalé
  - GMT Greenwich
  - HST,HDT Hawaii-Alaska standard ou décalé
  - MST,MDT Mountain standard ou décalé
  - NST Terre-Neuve standard
  - PST,PDT Pacifique standard ou décalé
  - YST,YDT Yukon standard ou décalé

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : conversion de date

- TO_CHAR(date[format[nls_langue]]) conversion d'une date en une chaîne de caractères.
- TO_DATE(chaîne[format[nls_langue]]) conversion d'une chaîne de caractères en date.
  - format est une chaîne de caractères constituée de mots clés
  - nls_langue est une chaîne de la forme :
    'NLS_DATE_LANGUAGE = langue'
    langue : french, american, arabic, german ... (46 langues supportées)

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : conversion de date : formats (1)

- SCC, CC siècle avec et sans signe.
- SYEAR, YEAR année en lettres avec et sans signe.
- SYYYY, YYYY année sur 4 chiffres avec et sans signe.
- Y,YYY année sur 4 chiffres avec virgule.
- RRRR année sur 4 chiffres ou 2 chiffres avec correction année 2000.
- YYY année sur 3 chiffres (les 3 derniers).
- YY année sur 2 chiffres (les 2 derniers).
- RR année sur 2 chiffres avec correction année 2000.
- Y le dernier chiffre de l'année.

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : conversion de date : formats (2)

- BC, AD indication BC (Before Christ) ou AD (Ano Domini).
- Q N du trimestre de l'année (1-4).
- MONTH mois en toutes lettres sur 9 caractères.
- MON abréviation du mois sur 3 lettres.
- MM N du mois dans l'année.
- WW N de la semaine dans l'année (1-53).
- IW N de la semaine ISO dans l'année (1-52 ou 1-53).
- W N de la semaine dans le mois.
- DAY jour en toutes lettres sur 9 caractères.
- DY abréviation du jour (2 ou 3 lettres).

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : conversion de date : formats (3)

- DDD N du jour dans l'année.
- DD N du jour dans le mois.
- D N du jour dans la semaine.
- J jour du calendrier Julien.
- AM, PM indication AM ou PM.
- HH, HH12 heure sur 12 heures.
- HH24 heure sur 24 heures.
- MI minutes.
- SS secondes par minute (0-59).
- SSSSS secondes par jour (0-86399).
- "chaîne" la chaîne est reproduite telle quelle

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : formats de conversion

- les caractères – / , . ; : sont reproduits tels quels.
- utilisation des majuscules et des nimuscules précise le format de sortie.
  - exemple : 'MONTH' donnera 'JUIN' alors que 'Month' donnera 'Juin'.
- suffixes :
  - TH ajout du suffixe ordinal
  - SP nombre en toutes lettres
  - SPTH, THSP nombre en toutes lettres avec ajout du suffixe ordinal

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : formats de conversion

- modificateurs :
  - FM suppression de blancs et de zéros
  - FX obligation de respecter exactement le format, les blancs sont significatifs
- afficher la date courante sous différentes formes :

```
SELECT
    to_char(sysdate,'DAY DD MONTH YYYY') fmt1,
    to_char(sysdate,'FM DAY DD MONTH YYYY') fmt2,
    to_char(sysdate,'DD/MM/YYYY HH24 :MI :SS') fmt3
FROM Dual;
```

Fonctions diverses pour requêtes SQL

Fonctions sur les dates : troncature et arrondi d'une date

- TRUNC(date[format])
  retourne une date tronquée selon le format spécifié

- ROUND(date[format])
  retourne une date arrondie selon le format spécifié

Fonctions diverses pour requêtes SQL

troncature et arrondi d'une date : format (1)

- SCC, CC siècle (arrondi au milieu du siècle)
- SYEAR, YEAR année (arrondi au 1er juillet)
- SYYYY, YYYY
- YYY, YY, Y
- IYYY, IYY, IY, I année ISO (arrondi au 1er juillet)
- Q trimestre (arrondi au 16ème jour du 2ème mois du trimestre)
- MONTH, MON, MM, RMmois (arrondi au 16ème jour)

Fonctions diverses pour requêtes SQL

troncature et arrondi d'une date : format (2)

- WW semaine (7 jours) définie à partir du 1er janvier (arrondi au 5ème jour)
- IW semaine ISO (arrondi au vendredi)
- DAY, DY, D premier jour de la semaine, dépend du pays : lundi/France, dimanche/US, (arrondi au 5ème jour)
- DDD, DD, J jour, par défaut lorsque le format est absent (arrondi à la demi-journée)
- HH, HH12, HH24 heure (arrondi à la demi-heure)
- MI minute (arrondi à 30 secondes)

Fonctions diverses pour requêtes SQL

troncature et arrondi d'une date : exemple

- Le premier jour de la première semaine de l'année :

```
SELECT
    to_char(trunc(to_date('06/01/99'),'WW'),'FM Day DD'),
    to_char(trunc(to_date('06/01/99'),'IW'),'FM Day DD')
    from Dual ;
```

SQL : langage de contrôle de données (LCD)

Sécurité des données

- confidentialité

  gestion des rôles et des utilisateurs
  attribution de privilèges aux rôles et aux utilisateurs
  définition de filtres (protection de donées confidentielles, contrôle d'intégrité)

- pérennité

  gestion des transactions

- intégrité

  gestion des transactions

SQL : langage de contrôle de données (LCD)

- transaction : séquence d'opérations manipulant des données
- vérifient les propriétés suivantes :
  - atomicité
  - cohérence
  - indépendance
  - permanence

contrôle des transactions :

- COMMIT : valide la transaction en cours
- ROLLBACK : annule la transaction en cours

Gestion des utilisateurs et des privilèges

- création de rôle

  CREATE ROLE nom-de-rôle [IDENTIFIED BY mot-de passe] ;
- ajout, modification, suppression de mot de passe

  ALTER ROLE nom-de-rôle [IDENTIFIED BY mot-de passe] ;
- suppression de rôle

  DROP ROLE nom-de-rôle ;

Gestion des utilisateurs et des privilèges

- création d'utilisateur

  CREATE USER nom-d'utilisateur [IDENTIFIED BY mot-de passe] ;
- ajout, modification, suppression de mot de passe

  ALTER USER nom-d'utilisateur [IDENTIFIED BY mot-de passe] ;
- suppression de rôle

  DROP USER nom-d'utilisateur ;

Gestion des utilisateurs et des privilèges

- attribution de privilèges

  GRANT systeme-privileges | ALL [privileges]
  TO liste-roles-utilisateurs | PUBLIC
  [WITH ADMIN OPTION] ;

- systeme-privileges :

  CREATE ROLE
  CREATE SEQUENCE
  CREATE SESSION
  CREATE SYNONYM
  CREATE PUBLIC SYNONYM
  CREATE TABLE
  CREATE USER
  CREATE VIEW

Gestion des utilisateurs et des privilèges

- attribution de privilèges sur des objets oracle

  GRANT liste-droits
  ON nom-composant
  TO liste-roles-utilisateurs
  [WITH GRANT OPTION] ;

- liste-droits :

  SELECT
  INSERT
  UPDATE
  DELETE
  ALTER
  REFERENCES
  ALL [PRIVILEGES]

Gestion des utilisateurs et des privilèges

- suppression de privilèges

  REVOKE liste-systeme-privileges
  FROM liste-roles-utilisateurs

- suppression de privilèges sur des objets oracle

  REVOKE liste-privileges
  ON nom-composant
  FROM liste-roles-utilisateurs

Gestion des utilisateurs et des privilèges

- attribution de rôles

  GRANT liste-roles
  TO liste-roles-utilisateurs
  [WITH ADMIN OPTION] ;

- suppression de rôles

  REVOKE liste-roles
  FROM liste-roles-utilisateurs