Bases de Données - HMIN112M

Mastère Informatique Pour les Sciences - IPS

2018-2019

I. Mougenot & A-M. Chifolleau

Chapitre 3

Modèle Relationnel

Contents

3.1 Introduction

Proposé à partir de 1970 (travaux de E.F. CODD), le modèle relationnel de données connaît un grand succès pour les raisons suivantes :

— représentation simple des données à l'aide du concept unique de **relation** ;
— existence d'une démarche rigoureuse (normalisation) permettant la construction du schéma ;
— SGBD relationnels et langages d'interrogation (SQL) ayant fait leur preuve.

Dans ce chapitre, nous nous intéresserons aux concepts de ce modèle, et à la description des langages "théoriques" attachés au modèle relationnel à savoir les langage algébriques et prédicatifs.

3.2 Définitions

Le modèle relationnel n'utilise que le concept mathématique de **relation**. Une relation dans la théorie des ensembles est un sous-ensemble du **produit cartésien** de **domaines**.
Un **domaine** est un ensemble de valeurs.

Exemples de domaines :

— un ensemble d'entiers ;
— un ensemble de chaînes de caractères ;
— {rouge,vert,bleu}.

Le **produit cartésien** des domaines D1,D2,...,Dn est dénoté par :

$$D1 \times D2 \times .... \times Dn$$

et est vu comme l'ensemble des n-uplets ou tuples (v1,v2,...,vn) tels que vi $\in$ Di pour i=1,2,...,n.
Une **relation** sur les domaines D1,D2,...,Dn est alors un sous-ensemble du produit cartésien de ces domaines.
On appelle le nombre de domaines n l'**arité** de la relation et **n-uplets** (ou encore tuples) les

éléments de la relation. Le nombre de n-uplets d'une relation est appelé son **cardinal** (ou sa cardinalité).
Il est courant de représenter une relation sous forme d'une **table**, où les lignes correspondent aux n-uplets et les colonnes aux attributs.

Ordinateur	Nom	Constructeur	Prix	Année
	Emac G4	Apple	836	2004
	Latitude	Dell	1000	2003
	Armada	HP Compaq	500	2001
	Vaio	Sony	1300	2004

Figure 3.1 – Une extension possible de la relation Ordinateur

Remarques :

- un nom différent est attribué à chaque colonne de manière à les distinguer (notion de rôle),
- le couple (nom,domaine) est appelé **attribut** de la relation. L'ordre des attributs tout comme l'ordre des n-uplets demeure sans importance,
- on appelle **schéma de la relation** le nom de la relation suivi de la liste de ses attributs. Pour reprendre l'exemple de la relation Ordinateur, le schéma de la relation est ainsi : Ordinateur (Nom : caractère(10), Constructeur : caractère(10), Prix : numérique, Année entier, clé **Nom,Constructeur**)

La notion de **clé** (rencontrée dans le modèle E-A au travers de la notion d'identifiant) s'applique à la structure relationnelle. Ainsi les attributs Nom et Constructeur forment une clé pour la relation Ordinateur. En d'autres termes, deux tuples de la relation ne peuvent avoir la même valeur à la fois pour Nom et Constructeur.

Généralisation :

Soit $\mathcal{R}$ un schéma de relation défini sur un ensemble U d'attributs, et K un sous-ensemble de U. K est appelé clé si il n'existe pas de couples de n-uplets dans les relations R (réalisations de $\mathcal{R}$), ayant même valeurs d'attributs pour les attributs de K.

3.3 Schéma de base de données relationnel

Un schéma de base de données relationnel est constitué d'un ensemble de **schémas de relation** définis sur des ensembles d'attributs et soumis à un certain nombre de contraintes dites **contraintes d'intégrité** qui expriment la sémantique de la représentation.
Le schéma d'une relation est une organisation logique (on peut parler d'intention de la relation) qui donne l'interprétation de la relation. Une instanciation du schéma ou relation ou encore **instance de la relation** correspond au contenu à un instant donné (relation en extension).

Un schéma de relation est donc constitué :

- d'un nom,
- de la liste des attributs,
- de la liste des contraintes imposées.

Exemple de schéma relationnel

Societe(numSiret entier, nom char(15), ville char(20), pays char(15))
Salarie (numEmploye entier, nom char(20), prenom char(20), fonction char(20), numSociete entier)
Ordinateur (nomOrdi char(10), marqueOrdi char(15), type char(15), processeur char(15), numSociete entier)
AffecteA (numEmploye entier, nomOrdi char(15), marqueOrdi char(15))

3.4 Les contraintes

Les contraintes d'intégrité exprimables au niveau schéma peuvent être variées :
— contraintes de domaine,
— dépendances fonctionnelles,
— contraintes référentielles ou contraintes d'inclusion.

3.4.1 Contraintes de domaine

Elles peuvent revêtir plusieurs formes :
— définition extensive ou en intention du domaine d'un attribut avec pour exemples :
  — l'attribut nomOrdi du schéma de relation Ordinateur est contraint à être une chaîne de caractères de longueur 10,
  — l'attribut marqueOrdi du schéma de relation Ordinateur a ses valeurs dans l'ensemble {Dell, HP Compaq, Apple, Sony, IBM}.
  — présence obligatoire ou non d'une valeur pour un attribut à l'aide de la contrainte NULL ou NOT NULL.
— l'existence de la valeur d'un attribut peut être liée à la valeur d'un autre attribut :
  — par exemple, l'attribut Nommarital d'un schéma de relation Personne ne prend de valeur que si la valeur de l'attribut Sexe est "féminin".
— règles de calcul indiquant comment la valeur d'un attribut est déduite de la valeur d'un ou plusieurs attributs. Par exemple l'attribut date de naissance est contraint a avoir la même valeur pour l'année que les caractères 2 et 3 de l'attribut N_SS (numéro de sécurité sociale). L"attribut N_SS est contraint à son tour à être une chaîne de caractères de longueur 13.
— contraintes de dépendances entre attributs que nous allons détailler (DF ou dépendances fonctionnelles, DJ ou dépendances de jointure, DI ou dépendances d'inclusion). Les DJ et DI seront étudiées lors du processus de normalisation.

3.4.2 Dépendances fonctionnelles

Définition :

Soit $\mathcal{R}$(A1,A2,....An) un schéma de relation et G et D deux sous-ensembles de l'ensemble A1,A2,...,An. On dit que G détermine D ou que D dépend fonctionnellement de G et on note $G \rightarrow D$ si pour toute relation R de $\mathcal{R}$ acceptable, tous les n-uplets u,v de R qui ont même valeur dans G ont aussi même valeur dans D.

$$\boxed{\text{ssi } (\forall \text{ R instance de } \mathcal{R})\ (\forall\ u,v \in R \text{ tel que } u[G] = v[G] \Rightarrow u[D] = v[D])}$$

En d'autres termes, si deux n-uplets de R coïncident sur G alors ils coïncident sur D.
Remarque :
si DG est l'ensemble des attributs de $\mathcal{R}$, on ne peut avoir deux n-uplets u,v ∈ R instance de $\mathcal{R}$ coïncidant sur G car sinon ils seraient égaux. Or les relations sont des ensembles, un n-uplet ne figure qu'une seule fois (doublons interdits).
Exemple : A l'ensemble de phrases suivantes :
— une voiture est identifiée par un numéro d'immatriculation N_immat,
— une voiture a une couleur ;
— à une voiture correspond un type ;
— un type de voiture correspond une puissance.

On peut associer l'ensemble de DF suivant :
{N_imm → Type, N_imm → Couleur, Type → Puissance}
La notion de dépendance fonctionnelle permet de préciser la notion de surclé et de clé : K est surclé de $\mathcal{R}$ sur U ssi $K \rightarrow U$
Deux n-uplets distincts ne peuvent avoir la même surclé (sinon ils coïncident entièrement et R contient 2 n-uplets égaux). U est toujours une surclé (triviale)

Définition d'une clé de schéma de relation

Soit $\mathcal{R}$(A1,A2,....An), un schéma de relation et K un sous-ensemble de A1,A2,...,An, on dit que K est une clé pour $\mathcal{R}$ si :

— propriété 1 : K $\rightarrow$ A1,A2.....An,

— propriété 2 : pour tout K' inclus dans K si K' $\rightarrow$ A1,A2,....An alors K=K'.

La deuxième condition introduit la minimalité de l'ensemble d'attributs qui constitue une clé.

$$\boxed{\text{K est clé de } \mathcal{R} \text{ ssi } \neg\ \exists\ \text{A} \in \text{K tel que K-}\{\text{A}\} \rightarrow \text{U}}$$

Remarque : Si plusieurs clés sont "possibles", on en choisit une dite clé primaire, les autres sont appelées clés candidates.

3.5 Les langages d'interrogation "théoriques"

3.5.1 L'algèbre relationnelle

Les opérations de base sont de deux sortes :

1. les opérations ensemblistes,
2. les opérations spécifiques relationnelles.

Opérations ensemblistes

UNION Opération portant sur deux relations de même schéma Relation 1 et Relation 2 consistant à construire la relation 3 de même schéma ayant pour n-uplets ceux appartenant à R1 ou à R2 ou aux deux relations.

Notations possibles :

UNION (Relation 1, Relation 2)
Relation 1 $\bigcup$ Relation 2

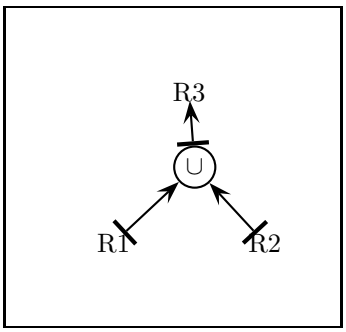

FIGURE 3.2 – Notation graphique de l'union

DIFFERENCE Opération portant sur les deux relations 1 et 2 de même schêma, le résultat est la relation 3 de même schéma et ayant pour n-uplets ceux appartenant à R1 et n'appartenant pas à R2.

Notations possibles :

DIFFERENCE (Relation 1, Relation 2)
Relation 1 − Relation 2

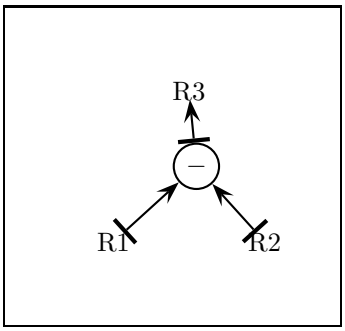

FIGURE 3.3 – Notation graphique de la différence

PRODUIT CARTESIEN Opération portant sur deux relations 1 et 2, consistant à construire une relation 3 ayant pour schéma la juxtaposition de ceux des relations opérandes et pour n-uplets toutes les combinaisons des n-uplets des relations 1 et 2.

Notations possibles :

Relation 1 × Relation 2

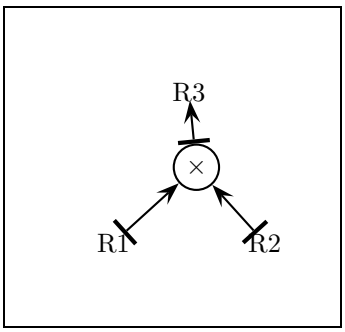

FIGURE 3.4 – Notation graphique du produit cartésien

Opérations spécifiques

SELECTION Opération sur une relation produisant une relation de même schéma mais ne comportant que les seuls n-uplets vérifiant la condition précisée en opérande.

Notations possibles :

SELECTION (Relation 1, condition) avec condition du type : Attribut opérateur Valeur (opérateurs possibles : <, >,=,<>, ...)
Relation 1 : condition
$\sigma_{condition}$(Relation 1)

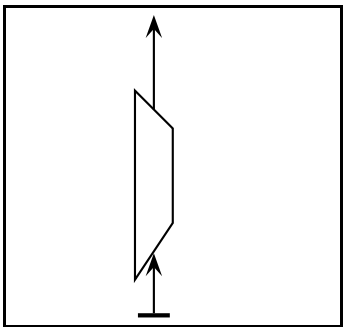

FIGURE 3.5 – Notation graphique de la sélection

PROJECTION Opération sur une relation consistant à composer une relation 2 en enlevant à la relation initiale tous les attributs non mentionnés en opérande et en éliminant les n-uplets qui constituent des "doublons".

Notations possibles :

PROJECTION (Relation 1, att i,....)
Relation 1 [atti,....]
Πatti,..(Relation 1)

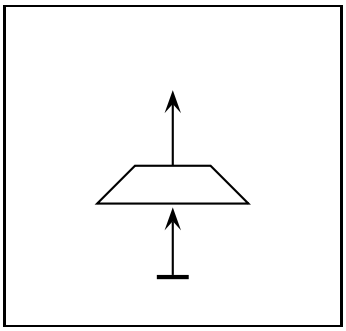

FIGURE 3.6 – Notation graphique de la projection

JOINTURE Opération consistant à rapprocher selon une condition les n-uplets de deux relations 1 et 2 afin de former une relation 3 qui contient l'ensemble de tous les n-uplets obtenus en concaténant un n-uplet de 1 et un n-uplet de 2 vérifiant la condition de rapprochement. (celle-ci est du type Attribut1 opérateur Attribut 2).

Notations possibles :

JOIN(Relation 1, Relation 2,Condition)
R1 ⋈ R2

Remarques :

Si l'opérateur noté de manière générique Θ est l'opérateur d'égalité (=), l'opération est appelée équijointure.
La jointure naturelle (notée ⋈), en d'autres termes la jointure pour laquelle la condition est l'égalité des attributs de même nom en ne conservant qu'une seule fois ces attributs dans la relation 3, sera le plus souvent utilisée.
une Θ-jointure désignera une jointure mettant en jeu un opérateur autre que l'égalité (<,>,<>,>=, . . .)

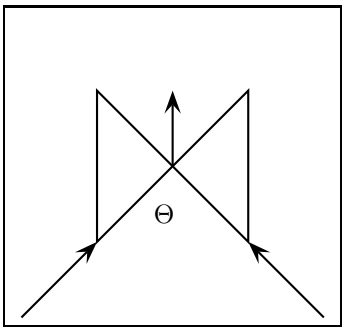

FIGURE 3.7 – Notation graphique de la jointure

Opérations complémentaires

INTERSECTION L'intersection peut être obtenue à partir de la différence
R1 ⋂ R2 = R1-(R1-R2) ou R2-(R2-R1)
Opération portant sur deux relations 1 et 2 de même schéma consistant à réaliser une relation 3 de même schéma ayant pour n-uplets ceux appartenant à la fois à 1 et 2.

Notations possibles :

INTERSECTION(Relation 1,Relation 2)
Relation 1 ⋂ Relation 2

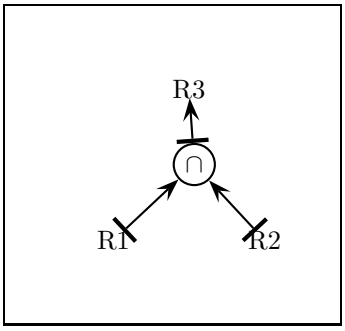

FIGURE 3.8 – Notation graphique de l'intersection

QUOTIENT La relation quotient d'une relation Relation 1 de schéma R1(A1,A2,...An) par une Relation2 de schéma R2(Ap+1,Ap+2,...An) est la relation R1 ÷ R2 de schéma Q (A1,A2,...Ap) constituée des p-uplets t tels que pour tout (n-p)-uplet u de R2, le n-uplet tu est dans R1
Remarque :
R1(X,Y) ÷ R2(Y) = R1[X] - ((R1[X] × R2 (Y)) - R1(X,Y)) [X]

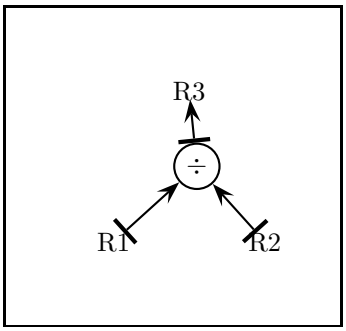

FIGURE 3.9 – Notation graphique du quotient

ANTIPROJECTION Opération portant sur une relation R(A1,A2) permettant d'obtenir les n-uplets dont la projection sur l'attribut A2 est associée à toutes les valeurs distinctes possibles des projections de la relation sur l'attribut A1. Elle est notée R(A1, A2)]A2[et définie par :
R(A1, A2)]A2[= R(A1, A2) ÷ R [A1]

SEMI-JOINTURE Opération portant sur deux relations notée Relation 1 $\ltimes$ Relation 2 qui consiste à projeter sur les attributs de R le résultat de la jointure naturelle Relation 1 $\bowtie$ Relation 2
Relation 1 $\ltimes$ Relation 2 = $\Pi_{attdeRelation1}$(Relation 1 $\bowtie$ Relation 2)

COMPLEMENT Le complément d'une relation R(X) notée ¬R(X) est un opérateur qui nécessite la connaissance des domaines associés aux attributs.
Soient les relations R(PIECE, FOURNISSEUR) et S(PIECE, PROJET). Les domaines associés aux divers attributs sont :

D1 pour PIECE = {écrou, boulon,vis}
D2 pour FOURNISSEUR = {pierre, paul, alice}

R	PIÈCE	FOURNISSEUR
	écrou	pierre
	écrou	paul
	boulon	alice

FIGURE 3.10 – Relation R en extension

¬R	PIÈCE	FOURNISSEUR
	écrou	alice
	boulon	pierre
	boulon	paul
	vis	alice
	vis	pierre
	vis	paul

FIGURE 3.11 – le complément de R

CONCLUSION L'ensemble des opérateurs union, différence, produit cartésien, sélection, projection permet d'engendrer tous les autres opérateurs, mis à part et le complément. Un langage possédant ces opérateurs ou pouvant les engendrer est dit **complet**.

3.6 Les Langages d'interrogation réels

Les langages d'interrogation commercialisés s'inspirent des langages algébriques ou des langages prédicatifs mais introduisent des fonctionnalités supplémentaires (tris, groupage-partitionnement, fonctions chaînes de caractère, fonction d'agrégation).

3.6.1 Langage SQL

Le langage SQL (Structured Query Language) est un dérivé de SEQUEL, lui-même dérivé de SQUARE, langage d'interrogation associé au prototype System R de chez IBM.
SQL est utilisé dans de nombreux SGBD commerciaux (relationnels) :

- DB2 (IBM), SQL/DS (gros ordinateurs sous systèmes d'exploitation MVS, VM/CMS)
- Postgres (Linux), Informix ;
- ORACLE, Sybase, MySQL (gros ordinateurs et micro-ordinateurs) ;
- Access, SQL Server (micro-ordinateurs) ;
- DB4, Paradox (micro-ordinateurs).

En SQL, la **table** désigne une collection de lignes (**ROW**) ou n-uplets. Une table peut être vue comme une généralisation de la notion de relation ; plusieurs lignes peuvent en effet y être identiques. On parle alors de doublons (voir options ALL/DISTINCT). De plus la colonne (**COLUMN**) désigne un attribut.

SQL en tant que langage de manipulation de données (LMD)

Nous allons illustrer les commandes SQL au travers de la base de données exemple présentée précédemment.

La commande SELECT En SQL, la commande SELECT appelée consultation ou question ou interrogation (query) est fondamentale. Elle n'a aucun rapport sémantique avec l'opération algébrique de sélection (σ).

Forme générale (partielle) :

```
SELECT [ALL | DISTINCT] { liste <colonne/expression> | * }
FROM liste [<proprietaire>.] { <table> | <vue> } [<alias >]
[WHERE <condition>]
[GROUP BY {liste expression } [HAVING condition]] ;
```

Sémantique : la commande visualise une table par sélection, projection et jointure d'autres tables, aux doublons près.

Exemples :

Visualisation de toutes les lignes et colonnes d'une table (notée Tab)
SELECT * FROM Tab ;
Illustration SELECT * FROM Ordinateur ;
Visualisation de toutes les lignes et d'une partie des colonnes d'une table
SELECT col1,col2 FROM Tab ;
Illustration SELECT nomOrdi, marqueOrdi from Ordinateur ;
Ceci n'est pas une projection dans la mesure où nous pouvons obtenir des n-uplets identiques !
Si d'aventure, nous souhaitons éliminer les n-uplets identiques, le mot SELECT doit être suivi de DISTINCT :
SELECT DISTINCT col1,col2 FROM Tab ;

Illustration $\Pi_{nomOrdi,marqueOrdi}$(Ordinateur)
SELECT DISTINCT nomOrdi, marqueOrdi from Ordinateur ;

Visualisation d'une partie des lignes d'une table
SELECT * FROM Tab WHERE condition ;
Cette opération est une sélection si la table de départ est une relation.
Illustration $\sigma_{type='portable'}$(Ordinateur)
SELECT DISTINCT * FROM Ordinateur WHERE type='portable' ;
Pour exprimer des conditions nous pouvons avoir recours aux opérateurs de comparaison, aux connecteurs logiques ou encore à certains prédicats. Nous détaillons ces opérateurs, connecteurs et prédicats dans un paragraphe ci-dessous.
Visualisation de l'ensemble (ou d'une partie) des données de plusieurs tables ou vues
Nous définissons au préalable deux tables notées tab1 et tab2 et ayant pour schémas tab1(col1,col2,col3) et tab2(col1,col4).
SELECT tab1.col1,col2,col4 FROM tab1,tab2 WHERE tab1.col1=tab2.col1 ;
Cette requête effectue une jointure entre tab1 et tab2, la colonne col1 est commune aux deux tables, le nom de la colonne pour être identifié est préfixé par le nom de sa table suivi d'un point.
Illustration $\sigma_{marqueOrdi='Dell'}$(AffecteA) $\bowtie$ (Salarie)
SELECT * FROM AffecteA, Salarie WHERE AffecteA.numEmploye = Salarie.numEmploye AND marqueOrdi='Dell' ;
Il est également possible d'effectuer des "auto-jointures" (jointure d'une table avec elle même ou encore utilisation différentielle de la même table) :
SELECT A.col1, B.col2 FROM tab1 A, tab1 B WHERE NOT (A.col1=B.col1) AND A.col2=B.col3 ; A et B sont des variables lignes ou alias.
Illustration Un exemple d'utilisation différentielle est donnée ci-dessous :
SELECT x.idarticle, x.qtestock, y.idarticle, y.qtestock from article x, article y
WHERE x.qtestock = 2*y.qtestock ;
Sous-requêtes imbriquées. La condition exprimée dans la clause WHERE peut porter sur une sous-requête ou requête imbriquée ; celle-ci doit être enfermée dans des parenthèses et ne doit avoir qu'une seule colonne ou expression dans sa clause SELECT, retourner une et une seule ligne excepté quand elle est précédée de IN,ALL,ANY.
Le prédicat EXISTS est le seul qui permet l'emploi de SELECT * dans une sous-requête.
Illustration
SELECT nom, salaire FROM employe
WHERE salaire > ALL (SELECT salaire FROM employe WHERE departement = 'production') ;

les opérations ensemblistes
SQL supporte les opérations d'union, d'intersection et de différence au travers des opérateurs UNION, INTERSECT et MINUS. SELECT col1 FROM tab1
{UNION|INTERSECT|MINUS}
SELECT col2 FROM tab2 ;
Les informations résultantes de chaque requête (ici col1 et col2) doivent être du même type.
Illustration
SELECT nom, prenom from Etudiant
UNION
SELECT nom, prenom from Enseignant ;

Opérateurs de comparaison, prédicats et connecteurs logiques
les opérateurs de comparaison =, <, >, <=,>=, <>. Ces opérateurs (Θ) peuvent être quantifiés. Soit S une expression dénotant un ensemble :
A Θ ANY(S) signifie $(\exists\ x)(S(x) \wedge A\ \Theta\ x)$
A Θ ALL(S) signifie $(\forall\ x)(S(x) \wedge A\ \Theta\ x)$
les prédicats

— BETWEEN teste l'appartenance d'une valeur à un intervalle donné (intervalle fermé). Par exemple, salaire BETWEEN 9000 AND 15000
— IN teste l'appartenance d'une valeur à une liste donnée : A [NOT] IN (S). Par exemple : lieu IN ('Versailles','Paris','Montpellier','Vierzon')
— LIKE recherche toute valeur d'une colonne de type chaîne de caractères contenant une chaîne de caractères donnée : A LIKE 'modèle'. SQL offre deux caractères génériques de substitution (le pourcentage pour la substitution de zéro à plusieurs caractères et le souligné (ou underscore) pour la substitution d'un et un seul caractère). Par exemple : lieu LIKE '_ri_' ou encore lieu LIKE '%er%'
— EXISTS teste la non vacuité d'un ensemble : EXISTS S
— NULL est un valeur non renseignée qui est par conséquente différente de zéro ou du blanc. SQL offre une comparaison sur les valeurs nulles à l'aide du prédicat IS [NOT] NULL

les opérateurs logiques : NOT, AND, OR

Les conditions élémentaires de sélection et de jointure vont pouvoir utiliser les opérateurs logiques de conjonction (AND), de disjonction (OR) et de négation (NOT)

Des extensions

les fonctions d'agrégat

Ces fonctions d'évaluation de tables (au travers d'expressions) vont permettre d'effectuer des calculs en regroupant toutes les lignes ensembles.

— COUNT([DISTINCT|ALL] expression)
— MIN([DISTINCT|ALL] expression)
— MAX([DISTINCT|ALL] expression)
— AVG([DISTINCT|ALL] expression)
— SUM([DISTINCT|ALL] expression)

Illustration

comptage du nombre de lignes :

```
SELECT COUNT(*)
FROM Ordinateur
WHERE marqueOrdi='Sanyo';
```

Autre Illustration

Moyenne et somme pour les colonnes de type numérique :
Minimum, Maximum pour les colonnes de type numérique ou chaîne de caractères

```
SELECT AVG(prix) AS Prix_Moyen)
FROM Ordinateur
```

le groupement au travers de la clause GROUP BY

Il peut s'avérer utile d'agréger les résultats non pas pour toute la table mais par groupes de lignes. L'opération de regroupement conditionnel s'effectue en outre par la clause : GROUP BY <liste colonne> HAVING <condition>. La clause HAVING effectue une sélection à l'image du WHERE mais de groupes de la partition et non plus des lignes de la table. Attention, chaque colonne ne participant pas au regroupement doit être calculée via une fonction d'évaluation (count, min, max, avg, sum).

Illustration

```
SELECT fonction, count(numEmploye)
FROM Salarie
GROUP BY fonction;
```

Autre Illustration

```
SELECT fonction
FROM Salarie
GROUP BY fonction HAVING COUNT(numEmploye)>=10;
```

les opérations de tri

SQL autorise le tri croissant (par défaut ASC) ou décroissant (DESC) sur une ou plusieurs colonnes. Les colonnes utilisées dans la clause ORDER BY doivent cependant faire partie de la clause SELECT

Illustration
SELECT nom, prenom, fonction, numSociete
FROM Salarie
ORDER BY nom, prenom ;

les opérateurs arithmétiques
Les opérateurs de multiplication et de division (∗ et /) sont prioritaires par rapport aux opérateurs d'addition et de soustraction (+ et -). Les parenthèses vont permettre de forcer les ordres de priorité si nécessaire.
les fonctions numériques
SQL offre un ensemble de fonctions traitant les données de type numérique qui peuvent être exprimées dans des expressions dans la clause SELECT ou WHERE :
— ABS(n) retourne la valeur absolue de n
— CEIL(n) retourne le plus petit entier supérieur ou égal à n
— COS(n) retourne le cosinus de n (n exprimé en radian)
— COSH(n) retourne le cosinus hyperbolique de n (n exprimé en radian)
— EXP(n) retourne la valeur exponentielle de n
— FLOOR(n) retourne la valeur entière de n
— LN(n) retourne le logarithme népérien de n (n¿0)
— LOG(m,n) retourne le logarithme à base de m, de n
— MOD(m,n) retourne le reste de la division entière de m par n
— POWER(m,n) retourne la valeur de m à la puissance n
— ROUND(n,|m] retourne la valeur de n arrondi à m postions à droite du point décimal
— SIGN(n) retourne -1 si n<0, 0 si n=0 et 1 si n>0
— SIN(n) retourne le sinus de n (n exprimé en radian)
— SINH(n) retourne le sinus hyperbolique de n (n exprimé en radian)
— SQRT(n) retourne la racine carrée de n
— TAN(n) retourne la tangente de n (n exprimé en radian)
— TANH(n) retourne la tangente hyperbolique de n (n exprimé en radian)
— TRUNC(n,m) retourne la valeur n tronquée à m positions décimales.
quelques fonctions de chaînes de caractères
L'opération de concaténation peut être réalisée sur les chaînes à l'aide de l'opérateur || (C1 || C2).
— INITCAP(char) met en majuscule la première lettre de chaque mot de la chaîne
— LOWER(char) met la chaîne de caractères char en minuscules
— UPPER(char) retourne la chaîne char avec toutes les lettres en majuscule
— LENGTH(char) retourne la longueur de la chaîne char en caractères
quelques fonctions de de date
Les opérateurs + et - peuvent être utilisés pour ajouter ou soustraire un nombre de jours à une date.
— SYSDATE retourne la date et l'heure courante du système
— ADD_MONTHS(d,n) ajoute n mois à la date d. Le résultat est une date
— MONTHS_BETWEEN(d1,d2) retourne le nombre de mois entre les dates d1 et d2.
— ROUND(D [, format]) retourne la longueur de la chaîne char en caractères

Forme complète de la commande SELECT

```
SELECT [ALL|DISTINCT] liste_de_selection
FROM liste_de_tables
[WHERE condition]
[[START WITH condition] CONNECT BY condition ]
[GROUP BY liste_d'expression [HAVING condition]]
[{UNION|UNION ALL|INTERSECT|MINUS} commande SELECT]
[ORDER BY {expr|position} [ASC|DESC] [,{expr|position} [ASC|DESC]]...
```

Quelques illustrations supplémentaires

Afficher des commerciaux qui se voient affecter un ou des ordinateurs de la marque Sanyo (expression au travers d'une jointure)
SELECT DISTINCT Salarie.numEmploye, nom, prenom FROM Salarie, AffecteA WHERE Salarie.numEmploye=AffecteA.numEmploye AND marqueOrdi='Sanyo' AND fonction='commercial';
Afficher des employés qui se voient affecter un ou des ordinateurs de la marque Sanyo (expression au travers d'un select imbriqué)
SELECT DISTINCT numEmploye, nom, prenom FROM Salarie WHERE numEmploye IN (SELECT numEmploye from AffecteA WHERE marqueOrdi='Sanyo') AND fonction='commercial';
Afficher des employés qui se voient affecter un ou des ordinateurs de la marque Sanyo (formulation à l'aide de l'opérateur d'existence EXISTS)
SELECT DISTINCT numEmploye, nom, prenom FROM Salarie WHERE EXISTS(SELECT * from AffecteA WHERE marqueOrdi='Sanyo' and Salarie.numEmploye=AffecteA.numEmploye) AND fonction='commercial';

SQL a un caractère prédicatif ou assertionnel lié à la clause SELECT. Cependant il a également une tendance impérative (algébrique) parce qu'il autorise l'imbrication des SELECT.

SQL en tant que Langage de Définition de données (LDD)

Le langage de définition de données permet de déclarer tous les objets décrivant la structure (c-à-d le schéma) de la base : tables, attributs, index... Il existe différentes variantes syntaxiques au niveau de la définition des données, nous utiliserons celle du SQL ORACLE.

Définition de table Définir une table revient à déclarer son nom et sa structure. La définition de chaque attribut consiste à en donner un nom, un type, une taille et éventuellement à lui apposer des contraintes d'intégrité (par exemple, dans l'exemple illustratif numpl porte une contrainte de non nullité **NOT NULL**).

```
CREATE TABLE <table>
(définition d'une colonne | définition d'une contrainte, ...)
[[SPACE <définition_espace_stockage>] [PCTFREE n]/ [CLUSTER <cluster> (liste <colonne>)]
/ AS <commande SELECT données provenant d'une requête>]

Colonne ::= nom type [DEFAULT expression] [contrainte de colonne] ...
```

Les types de données admis par ORACLE sont :
CHAR(n) chaîne de caractères de longueur fixe (n<=240)
VARCHAR(n) ou VARCHAR2(n) : chaîne de caractères de longueur variable n (n<=2000)
LONG chaîne de caractères de longueur pouvant aller jusqu'à 65535 caractères;
NUMBER(longueur,[précision]) nombre positif ou négatif sur n digits avec d digits à droite du point décimal
INTEGER entier
DATE données temporelles
ROWID chaîne hexadécimale représentant l'adresse unique d'une ligne de la table. Sa valeur est retournée par la pseudo-colonne de même nom.

Exemple de création de table

CREATE TABLE PILOTE (numpl NUMBER (6) NOT NULL, nompl CHAR (20), salaire NUMBER (8) DEFAULT 800, adresse CHAR (80));

Remarque

Si aucune zone de stockage (SPACE) n'est spécifiée, la création a lieu dans la zone privée de l'utilisateur. La table créée est vide. Il est possible de créer une table non vide à partir d'autres tables existant dans la base au travers de la <commande SELECT> (requête portant sur des tables existantes). Il n'est pas nécessaire de spécifier les attributs. La requête définie dans la clause AS détermine les n-uplets et éventuellement les attributs de la table créée. Le type et la taille des attributs seront hérités des attributs cités dans la requête. Le nombre d'attributs spécifiés explicitement doit être égal au nombre d'attributs cités dans la requête.

Exemple

CREATE TABLE pilote-province AS SELECT numpl, nompl WHERE adresse = 'Paris' ;

L'effet de cette commande est de créer une table pilote-province ayant deux attributs et les n-uplets vérifiant la condition adresse 'Paris'.

Notion de valeurs nulles Une valeur nulle (NULL) indique un 'non renseignement' pour un enregistrement donné. Une chaîne vide ou un 0 ne donnant pas forcément l'intention voulue, NULL spécifie que la valeur de ce champ est indéfinie indépendamment du type.

Contraintes d'intégrité Une contrainte d'intégrité permet d'exercer un contrôle sur la valeur d'une colonne et vaêtre définie lors de la création d'une colonne de la manière suivante

```
contrainte d'intégrité de colonne ::=
[CONSTRAINT nom_de_contrainte]
{ [NOT] NULL |
{ UNIQUE | PRIMARY KEY}
| REFERENCES [schema.]table[[colonne)]
[ON DELETE CASCADE]
{EXCEPTIONS INTO [schema.]table | DISABLE}
```

Il est permis de nommer les contraintes au travers du mot-clé CONSTRAINT. Cette contrainte sera alors plus facile de manipulation (notamment au travers de la table USER_CONSTRAINTS du dictionnaire des donnnées)

NULL spécifie que la colonne peut contenir des valeurs nulles. A contrario, NOT NULL interdit l'insertion de valeurs nulles pour la colonne considérée.

UNIQUE interdit deux lignes ayant la même valeur pour la colonne (ORACLE pose un index sur la colonne)

PRIMARY KEY indique une colonne porteuse de contrainte de clé primaire. Les valeurs de la colonne doivent alors être uniques et non nulles. (pose d'un index sur la colonne de manière automatique par ORACLE)

CHECK indique une contrainte de domaine, la condition indiquée doit être vérifiée pour chacune des valeurs insérées.

REFERENCES définit une contrainte d'intégrité référentielle par rapport à une clé unique ou primaire. ON DELETE CASCADE est optionnelle et permet de maintenir la contrainte d'intégrité référentielle lors de la suppression de tuples.

EXCEPTIONS INTO désigne une table d'exception existante dans laquelle ORACLE place des informations sur les lignes qui violent une contrainte d'intégrité activée

DISABLE est un mot-clé permettant de désactiver la contrainte. Par défaut, ORACLE active cette contrainte.

Définition d'index Les index ou accélérateurs d'accès aux données sont définis par :

```
CREATE [UNIQUE] INDEX <index>
ON [<propriétaire>.]<table> (liste <colonne> [ASC/DESC]);
```

Remarque : l'option UNIQUE spécifie que l'ensemble des attributs de l'index est un identifiant de la table.

Définition de vue

```
CREATE VIEW [<propriétaire>.]<vue> [liste (<colonne>) ] AS <commande SELECT> ;
```

Définition de synonymes

Nommer autrement une table ou une vue est possible par :

```
CREATE [PUBLIC] SYNONYM <synonyme> FOR [<propriétaire>.]<table>/<vue> ;
```

Modification de de la structure d'une table Modifier une table revient soit à ajouter un ou plusieurs attributs, soit à modifier le type d'un attribut, soit encore à supprimer un attribut.
La suppression d'un attribut n'est pas exprimable directement en SQL ; elle peut être réalisée d'une façon indirecte par la création d'une nouvelle table en omettant de recopier l'attribut.

Ajout de nouveaux attributs dans une table : pour ajouter un attribut, il suffit de le nommer et de donner son type et éventuellement sa taille et des contraintes (NOT NULL) ;

```
ALTER TABLE <table> ADD (liste <colonne> <type> [NULL/NOT NULL]...) ;
```

Par exemple, ajout de l'attribut Type_Licence dans la table pilote se fait par : ALTER TABLE pilote ADD (Type_Licence CHAR (40)) ;
Modification du type d'un attribut : cette modification peut concerner

- la réduction de la taille : seulement si l'attribut ne contient que des valeurs 'nulles'
- la modification du type : même condition que précédemment ;
- l'augmentation de la taille est toujours possible ;
- la suppression de l'interdiction de présence de valeurs nulles (passage de NOT NULL à NULL) : toujours possible.

La modification est réalisée par la commande :

```
ALTER TABLE <table> MODIFY (liste <colonne> [<type>] [NULL/NOT NULL] ) ;
```

Renommer une table On peut changer le nom d'une table en la renommant. Mais il faut prendre soin de répercuter cette modification au niveau des applications et des vues définies sur cette table.

```
RENAME <ancienne_table> TO <nouvelle_table> ;
```

La possibilité de définir des synomymes permet de pallier aux problèmes posés par le renommage d'une table. En effet, une table peut-être référencée soit par son nom initial soit par un synonyme.

Mise à jour des données d'une table

Ajout d'un ou plusieurs n-uplet (lignes) littéraux dans une table :

```
INSERT INTO <table> [liste <colonne>] VALUES (liste >valeur>) ;
```

Ajout d'une ou plusieurs lignes sélectionnées à partir d'autres tables :

```
INSERT INTO <table> <commande SELECT>;
```

Mise à jour de données à l'intérieur d'une table :

```
UPDATE <table>
        SET liste <colonne>=<expression>
        [WHERE condition ];
```

Remarque : si la clause WHERE n'existe pas, la mise à jour concerne toutes les lignes de la table.

Suppression d'une table C'est une opération rare et qui est à manipuler avec prudence surtout lorsqu'il existe des vues définies sur cette table. Cette commande est utile pour supprimer des tables de travail.

```
DROP TABLE <table> [CASCADE CONSTRAINTS];
```

Elle a pour effet de supprimer la table spécifiée, et tous les index, synonymes correspondants. Il est possible de supprimer le synonyme d'une table :

```
DROP [PUBLIC] SYNONYM <synonyme>  ;
```

Suppression d'un index :

```
DROP INDEX <index> [ON <table>] ;
```

Suppression d'une vue :

```
DROP VIEW <vue> ;
```

Suppression de données à l'intérieur d'une table :

```
DELETE FROM <table> [WHERE <condition>] ;
```

Remarque : si la clause WHERE n'existe pas, la suppression concerne toutes les lignes de la table.

Les droits L'accès d'un utilisateur est contrôlé au moment de la connexion. (login et password). Les droits d'accès à un objet (table, vue, ..) dépendent :

- de son autorité administrative ;
- du fait d'être ou non propriétaire des objets ;
- d'autorisations explicites accordées.

Donner des droits

```
GRANT liste <droit>/ALL ON <ressource> TO liste <usager>/PUBLIC [WITH GRANT OPTION] ;
```

GRANT donne à un ou plusieurs usagers (ou à tous, PUBLIC) les droits d'accès spécifiés sur les ordres SQL suivants : ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, ALL.

Supprimer des droits

```
REVOKE liste <droit >/ALL ON <ressource> FROM liste <usager>/PUBLIC ;
```

Créer des utilisateurs

Les utilisateurs possédant l'autorité d'un DBA (administrateur de Base) peuvent créer de nouveaux utilisateurs par la commande :

```
GRANT [CONNECT,][RESOURCE,][DBA] TO liste <user> [IDENTIFIED BY liste <passwd> ] ;
```