Département Informatique de l'Institut Universitaire de Technologie de l'Université Bordeaux 1

Bases de Données

Cours

Version 2 du 16 janvier 2018 à 14h12

Olivier Guibert

Généralités

Introduction (bibliographie, définitions préliminaires, historique, SGF *vs* SGBD, niveaux de description de l'ANSI)

Exemple « jouet »

Conception d'une BD

Le modèle entité-association

- Définitions et représentation graphique
- Passage d'un schéma entité-association à un schéma relationnel

Le MOD de MERISE/2

Le MLD (global et réparti) de MERISE/2

Le modèle navigationnel

Le modèle relationnel

- Définitions
- Contraintes d'intégrité
- Normalisation (dépendances fonctionnelles et formes normales)
- Passage d'un schéma relationnel à un schéma entité-association

Utilisation d'une BD

L'algèbre relationnelle

- Opérations ensemblistes (produit cartésien, union, intersection, différence, complément, division)
- Opérations relationnelles (projection, sélection, jointures)

Le calcul relationnel

SQL

- Présentation (introduction, généralités, éléments de base, session)
- SQL interactif : LMD (requêtes d'interrogation et de mise à jour), LDD, LCD
- SQL procédural et SQL programmé (intégré, dynamique, module)
- SQL relationnel-objet

Approfondissements

Fonctionnement d'un SGBDR

- Critères d'un SGBDR
- Le gestionnaire des transactions
- Indexation
- L'optimisation d'une requête d'interrogation

SGBD réparti et Client/Serveur

- SGBD⋆
- Architecture C/S

SGBD spécifiques

- SGBD déductif
- Entrepôt de données et fouille de données
- BD du *World Wide Web* et multimédia
- SGBDO

1FN : première FN

2FN : deuxième FN

3FN : troisième FN

4FN : quatrième FN

5FN : cinquième FN ou FN de projection-jointure *(project-join normal form (JD/NF))*

ACSE : *Association Control Service Element*

AGL *(CASE)* : Atelier de Génie Logiciel *(Computer Aided Software Environment)*

ANSI : *American National Standardisation Institute*

API : *Application Programming Interface* (interface de programmation d'application i. e. interface applicative)

BD *(DB)* : Bases de Données *(Database)*

BD$\star$: BD répartie

BDO : BD Objet

BI : *Business Intelligence* (informatique décisionnelle)

BLOB : *Binary Large Object* (champ binaire long)

C/S : Client/Serveur

CAE : *Common Application Environment*

CAO : Conception Assistée par Ordinateur

CLI : *Call-Level Interface* ou Client Interface

CLOB : *Character Large Object* (champ chaîne de caractères long)

CODASYL DBTG : *Conference On Data System Languages Database Task Group*

DCE : *Distributed Computing Environment* (architecture distribuée)

DFS : *Distributed File System* (système de fichiers distribués)

DTD : *Document Type Definition*

DTP : *Distributed Transaction Processing*

DSS : *Decision Support System* (management du système d'information)

FAO : Fabrication Assistée par Ordinateur

FN *(NF)* : Forme Normale *(Normal Form)*

FNBC *(BCNF)* : FN de Boyce-Codd *(Boyce-Codd NF)*

HTML : *HyperText Markup Language*

HTTP : *HyperText Transfer Protocol*

IA : Intelligence Artificielle

IDAPI : *Integrated Database Application Programming Interface*

IDL : *Interface Definition Language* (langage de définition d'interface)

IEC : *International Electronical Commission*

IEEE : *Institute of Electrical and Electronic Engineers*

IHM : Interface Homme-Machine

ISO : *International Standardization Organization* (organisation internationale de normalisation)

JDBC : *Java Database Connectivity*

JRT : *Routines and Types using the Java TM Programming Language*

L3G : Langage de 3^e Génération

L4G : Langage de 4^e Génération

LCD *(DCL)* : Langage de Contrôle des Données *(Data Control Language)*

LCT *(TCL)* : Langage de Contrôle des Transactions *(Transaction Control Language)*

LDD *(DDL)* : Langage de Définition des Données *(Data Definition Language)*

LMD *(DML)* : Langage de Manipulation des Données *(Data Manipulation Language)*

LOB : *Large Object* (champ long)

MCD : Modèle Conceptuel des Données

MED : *Management of External Data*

MLD : Modèle Logique des Données

MLD$\star$: MLD réparti

MM : *Multimedia*

MOD : Modèle Organisationnel des Données

MOLAP : *Multidimensional OLAP*

MROLAP : *Multidimensional Relational OLAP*

NDL : *Network Data Language*

NF^2 : *Non First Normal Form*

OBJ : *Object Language Binding*

OLB : *Object Language Bindings*

ODBC : *Open Database Connectivity*

ODL : *Object Definition Language*

ODMG : *Object Database Management Group*

OLAP : *On-Line Analysis Processing* (analyse [multidimensionnelle] interactive de données)

OLTP : *On-Line Transaction Processing*

OMG : *Object Management Group*

OML : *Object Manipulation Language*

OQL : *Object Query Language*

OSF : *Open Software Foundation*

OSI : *Open Systems Interconnection*

PSM : *Persistant Stored Modules*

RDA : *Remote Data Access*

ROLAP : *Relational OLAP*

RPC : *Remote Procedure Call* (appel de procédure à distance)

SAG : *SQL Access Group*

SGBC : Système de Gestion de Bases de Connaissances

SGBD *(DBMS)* : Système de Gestion de Bases de Données *(Database Management System)*

SGBD$\star$: SGBD réparti *(distributed DBMS)*

SGBDO : SGBD Objet

SGBDOO : SGBD Orienté-Objet

SGBDR : SGBD Relationnel

SGBDRO : SGBD Relationnel-Objet

SGF : Système de Gestion de Fichiers

SIG : Système d'Information Géographique

SPI : sans perte d'information

SQL : *Structured Query Language*

TP : *Transaction Processing* (traitement transactionnel)

UDT : *User Data Type* (type de données utilisateur)

UTC : *Coordinated Universal Time* (temps universel coordonné)

XA : *Interface Specialization*

XML : *eXtensible Markup Language*

XQL : *XML Query Language*

Bibliographie (1/3)

A. ABDELLATIF, M. LIMAME, A. ZEROUAL

Oracle 7 : Langages - Architecture - Administration
EYROLLES, 1/1/1994, 464 p., ISBN-13 : 978-2212088328
Note : administration d'une ancienne version d'Oracle

ACSIOME

Modélisation dans la conception des systèmes d'information
DUNOD, 1/12/1997, 318 p., ISBN-13 : 978-2225819704
Note : cours et exercices d'ACSI (et de BD)

M. BOUZEGHOUB, G. GARDARIN, P. VALDURIEZ

Les Objets : concepts, langages, bases de données, méthodes, interfaces
EYROLLES, 25/6/1998, 450 p., ISBN-13 : 978-2212089578
Note : cours sur les objets (BD notamment)

G. BRIARD

Oracle8 pour Windows NT
EYROLLES, 1998
Note : ancienne version d'Oracle pour un ancien système d'exploitation

F. BROUARD [SQLpro], R. BRUCHEZ, C. SOUTOU

SQL
PEARSON Education, 23/5/2008 (2^e édition), 246 p., ISBN-13 : 978-2744073182
Note : cours et exercices en SQL

C. CHRISMENT, K. PINEL-SAUVAGNAT, O. TESTE, M. TUFFERY

Bases de données relationnelles : concepts, mise en œuvre et exercices
HERMÈS Science Publications, 10/6/2008, 494 p., ISBN-13 : 978-2746220867
Note : cours et exercices (pour légèrement plus de la moitié de l'ouvrage) de BD

C. J. DATE

Introduction aux bases de données
VUIBERT, 6/12/2004 (8^e édition), 1045 p., ISBN-13 : 978-2711748389
Note : cours de BD, la référence internationale

P. DELMAL

SQL2 - SQL3 : Applications à Oracle
De Boeck Université, 15/03/2001 (3^e édition), 512 p., ISBN-13 : 978-2804135614
Note : cours et exercices en SQL

C. DELOBEL

Bases de données et systèmes relationnels
DUNOD, 14/1/2007, 449 p., ISBN-13 : 978-2040116286
Note : cours de BD, l'une des deux références historiques françaises

C. DELOBEL, C. LECLUSE, P. RICHARD

Bases de données : des systèmes relationnels aux systèmes objets

InterÉditions, 1991, 460 p., ISBN-13 : 978-2729603717

J. GABILLAUD

SQL Server 2005 : Administration d'une base de données avec SQL Server Management Studio

Éditions ENI, 1/6/2006, 455 p., ISBN-13 : 978-2746030565

J. GABILLAUD

SQL Server 2005 : Entraînez-vous à administrer une base de données

Éditions ENI, 6/2/2007, 272 p., ISBN-13 : 978-2746035263

G. GARDARIN

Bases de données : les systèmes et leurs langages

EYROLLES, 1983, 265 p., ASIN : B000XA8FFY

Note : cours de BD, l'une des deux références historiques françaises

G. GARDARIN

Bases de données : Objet et Relationnel

EYROLLES, 10/4/2003 (5^e édition), 787 p., ISBN-13 : 978-2212112818

Note : cours de BD (avec très peu d'exercices), la référence française

G. GARDARIN

Internet, Intranet et bases de données : Data Web, Data Media, Data Warehouse, Data Mining

EYROLLES, avril 2000, 246 p., ISBN-13 : 978-2212090697

Note : cours sur des SGBD spécifiques

G. GARDARIN, O. GARDARIN

Le Client-Serveur

EYROLLES, 1/3/1996, 487 p., ISBN-13 : 978-2212088762

Note : cours sur l'architecture C/S

G. GARDARIN, P. VALDURIEZ

Bases de données relationnelles : Analyse et comparaison des systèmes (Architecture des systèmes d'informations)

EYROLLES, 1987 (2^e édition), 325 p., ASIN : B0014MXJ5Q

G. GARDARIN, P. VALDURIEZ

Systèmes de gestion des bases de données avancés

EYROLLES, 1989

Bibliographie (3/3)

O. HEURTEL

Oracle 10g : Administration

Éditions ENI, 2/3/2005, 489 p., ISBN-13 : 978-2746027787

C. MARÉE, G. LEDANT

SQL 2 : Initiation, programmation

ARMAND COLIN, 14/1/2007 (2^e édition), 326 p., ISBN-13 : 978-2200214111

Note : cours et exercices en SQL

C. NOIRAULT

Oracle 10g : Entraînez-vous à administrer une base de données

Éditions ENI, 17/5/2005, 212 p., ISBN-13 : 978-2746028708

R. ORFALI, D. HARKEY, J. EDWARDS

Client/Serveur : Guide de survie

VUIBERT Informatique, 1/12/1999 (3^e édition), 782 p., ISBN-13 : 978-2711786497

J. PANTTAJA, M. PANTTAJA, B. PRENDERGAST

MicroSoft SQL Server : Guide de survie

John Wiley & Sons - International Thomson Publishing France, 20/11/1996, 449 p., ISBN-13 : 978-2841801596

Base de Données (BD)

Définition [informaticien]

Ensemble des données (de l'organisation) structurées et liées entre elles :

stocké sur support à accès direct (disque magnétique),

géré par un SGBD,

accessible par un ensemble d'applications

Autre définition [informaticien]

Organisation cohérente (i. e. satisfaisant l'ensemble des contraintes d'intégrité) de données permanentes (supposant la gestion de la mémoire auxiliaire et des caches) accessibles par des utilisateurs concurrents (gestion des transactions), fournissant une indépendance physique (entre les programmes d'application et la description des données) et optimisant automatiquement (rôle de l'optimiseur) des requêtes d'interrogation écrites dans un langage déclaratif de haut niveau (SQL)

Définition [utilisateur]

Il s'agit de toutes les informations de l'organisation, indépendamment du support (classeur, bande magnétique, etc.) et du mode de gestion (manuel ou automatisé)

Système de Gestion de Bases de Données (SGBD)

Ensemble de programmes permettant la gestion d'une BD, rendant transparent la structuration physique des données sur le support (disque magnétique)

Exemples

- AIM-P *(Advanced Information Management Prototype)* [IBM]
- DB2 [IBM]
- Firebird [logiciel libre]
- Gemstone [Gemstone System Inc.]
- IDMS [Computer Associates]
- IDS *(Integrated Data Storage)* [General Electric]
- IDS II [Bull]
- IMS/DB *(Information Management System)* = DL/1 [IBM]
- Informix [IBM]
- Ingres *(Intelligent Graphic Relational System)* [logiciel libre]
- Interbase [Borland]
- MySQL [logiciel libre ou propriétaire [Sun Microsystems] selon l'utilisation]
- O2 [O2 Technology ... Ardent Software ... Informix ...]
- Objectivity/DB [Objectivity Inc.]
- ObjectStore [Object Design Fr.]
- ONTOS [ONTOS]
- Oracle [Oracle Corporation]
- ORION [Itasca Systems]
- Poet = *FastObjects by Poet* [Poet Software Corporation]
- PostGreSQL *(post-Ingres)* [Univ. Berkeley, logiciel libre]
- SQL/DS [IBM]
- SQL Server [Microsoft]
- Sybase [Sybase]
- Versant [Versant Object Tech.]

Outil bureautique : Access [Microsoft]

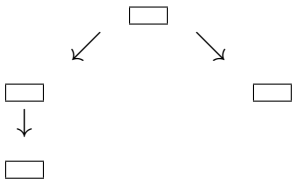
Historique (1/2)

(préhistoire) : Système de Gestion de Fichiers (SGF)

1^{re} génération : SGBD hiérarchique et réseau

Domaine de recherche dans les années [19]60, commercialisés à la fin des années [19]60

SGBD hiérarchique



$\rightsquigarrow$ arbre

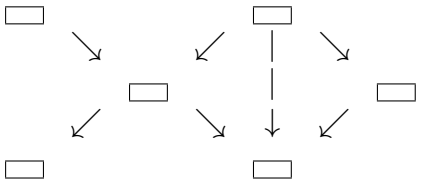
Remarque : temps de réponse optimal pour toute recherche qui suit le sens hiérarchique

Dépendance physique : contiguïté des données (et parfois pointeurs)

Exemple : IMS

SGBD réseau

1970 : *Conference On Data System Languages Database Task Group* (CODASYL DBTG)



$\rightsquigarrow$ graphe (orienté acyclique)

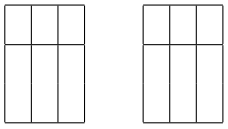
Dépendance logique : pointeurs

Exemples : IDS II, IDMS

2^{e} génération : SGBDR (SGBD Relationnel) et SGBD réparti (SGBD$\star$)

Domaine de recherche dès 1970 (E. F. CODD), commercialisés depuis 1980

SGBDR



Dépendance sémantique : valeurs

Exemples : DB2, Informix, Ingres, Oracle, SQL/DS, Sybase

SGBD$\star$

Exemples : Oracle$\star$, Ingres$\star$, etc.

Historique (2/2)

3^e (voire 4^e) génération SGBD Objet (SGBDO) i. e. SGBDRO et SGBDOO

Domaine de recherche dès le début des années [19]80, commercialisés depuis les années [19]80

SGBDRO i. e. extensions objet des SGBDR (exemples : DB2, MySQL, Oracle, SQL Server, Sybase)

SGBDOO i. e. approche objet d'un SGBD

Système à objets persistants (exemples : Gemstone, ObjectStore)

SGBDOO pur (exemples : AIM-P, O2, Objectivity/DB, ONTOS, ORION, Poet, PostGreSQL, Versant)

... meilleur support Internet, données semi-structurées, objets multimédias, aide à la prise de décisions, extraction de connaissances à partir des données

Remarque : les SGBDRO sont très majoritairement utilisés aujourd'hui et servent souvent de point de départ aux futurs SGBD actuellement imaginés

Principe

À chaque application correspond un (voire plusieurs) fichier(s) propre(s)

Inconvénient majeur

Redondance (i. e. duplication des données)

- Problème des mises à jour (insertion, modification, suppression)
- Perte d'espace disque

Conséquences

Inconsistance : les copies de la même information peuvent ne pas être toutes identiques

Incohérence : les contraintes d'intégrité peuvent ne pas être vérifiées

Éclatement (et donc difficulté pour regrouper les informations localisées dans plusieurs applications différentes)

Autres inconvénients

Programmation (en L3G) : l'information n'est donc accessible que par programmation (et donc inaccessible pour les utilisateurs en dehors de l'application)

Vision partielle (et non globale) de l'informatisation : liée à celle d'une application

Problèmes

Intégrité : difficulté à vérifier les contraintes d'intégrité sur des informations éclatées et dupliquées

Sécurité : les droits des utilisateurs sur une même information peuvent être différents d'une application à l'autre

Concurrence : les accès simultanés à une même information sont plus complexes à gérer à cause des informations dupliquées

Principe

Les programmes d'application font tous appels au SGBD qui lui seul a accès aux données (i. e. à la BD) : le SGBD sert donc d'interface entre le disque et les programmes

Fonctionnalités de base

Description de l'implantation physique (sur disque) des données

Description du schéma conceptuel des données

Manipulation des données : interrogation (i. e. recherche) et mises à jour de données (insertion, modification, suppression)

Fonctionnalités essentielles

Garantir l'intégrité (des données) : vérification globale (i. e. centralisation des contraintes entre les informations) et permanente des contraintes d'intégrité

Assurer la sécurité : vérification globale des droits (i. e. centralisation des contrôles d'accès) pour chaque information

Gestion de la concurrence : contrôler la cohérence des accès partagés des opérations de manipulation des données (mises à jour et même interrogation)

Avantages

Non redondance (et donc simplification des opérations de mises à jour)

Consistance (exemple : si un utilisateur manipule une donnée, celle-ci ne pourra pas être mise à jour par un autre utilisateur durant toute la durée de sa transaction)

Centralisation simplifiant l'administration (car 1! BD)

Compacité : gain en espace de stockage (car 1! BD)

Vision globale du système d'information (car 1! BD)

Fiabilité : le SGBD en est le garant

Indépendance données/traitements (cf. méthodes systémiques ou cartésiennes ... mais autorise approche objet)

Interrogation conviviale (langage autonome SQL interactif normalement accessible à un utilisateur et non aux seuls informaticiens)

Programmation (en L3G hôte de SQL, avec des générateurs d'applications, menus, écrans, rapports)

Extensibilité : modification du schéma des données dynamiquement

Politique globale d'informatisation (1! BD et donc applications non indépendantes)

Données partagées (entre plusieurs utilisateurs)

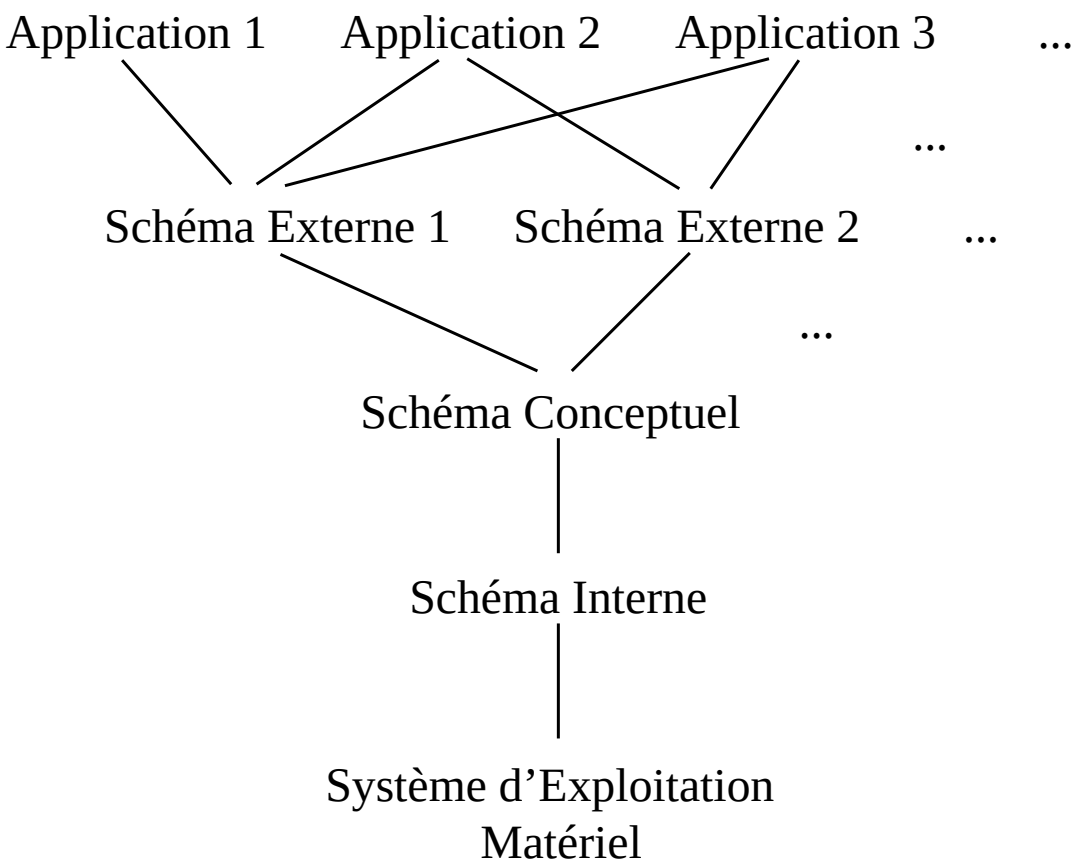
Inconvénients

Nécessité de gérer des ressources importantes : mémoire auxiliaire (pour le stockage des données), mémoire principale (code du SGBD, données partagées), processus (algorithmes du SGBD complexes, devant s'interconnecter avec les applications et le système d'exploitation)

L'information d'une organisation a tendance à être décentralisée

Niveaux de description (1/2)

(selon le rapport de l'ANSI/X3/SPARC Study Group on Database Management Systems)



3 niveaux de schéma

- Externe : vision simplifiée de l'information (à destination des utilisateurs notamment)
  - Vue : l'information est restructurée pour correspondre à la vision qu'en a l'utilisateur (cela revient à considérer un sous-schéma du schéma conceptuel, éventuellement avec des informations calculées)
  - Schéma
  - Manipulation des données : interrogation et mises à jour
- Conceptuel : structuration des données du monde réel de l'organisation i. e. intégralité de l'information (la BD !)
  - Relation (i. e. table)
  - Contrainte d'intégrité : de clé (primaire), d'intégrité référentielle, d'unicité, existentielle, etc.
  - Privilège (i. e. droit)
- Interne : implantation physique de l'information
  - Index (pour optimiser les interrogations)
  - Audit (pour l'étude des performances par exemple)

2 niveaux d'indépendance

- Indépendance logique (i. e. entre schémas externe et conceptuel)
  - Exemples : nouvelles vues, nouvelles informations
- Indépendance physique (i. e. entre schémas conceptuel et interne)
  - Exemple : réorganisation des données (index, *cluster*) pour accélérer les traitements

Niveaux de description (2/2)

Les intervenants

Utilisateur : utilise soit une application (*via* un programme), soit du SQL interactif

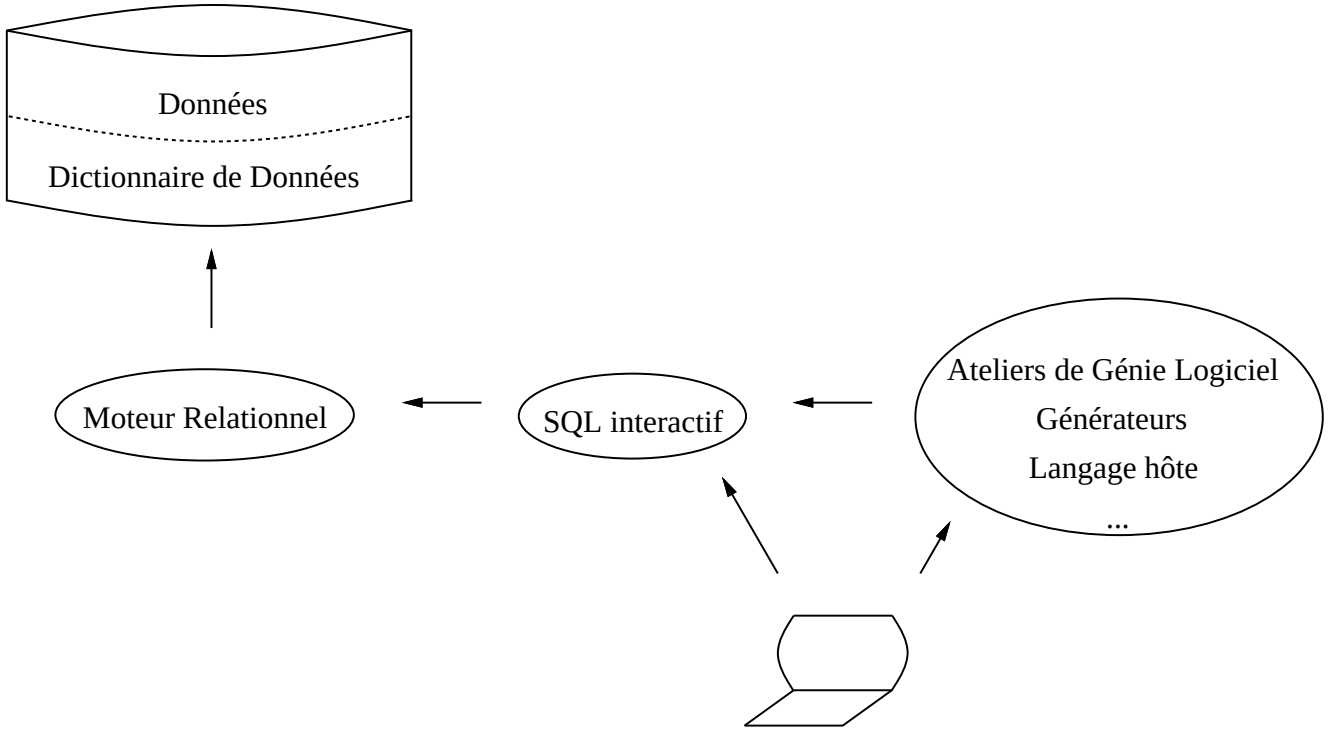
Développeur : réalise les programmes des applications (et les requêtes associées)

Administrateur des Données : responsable des données externes (exemple : typage)

Analyste-Concepteur : établit un schéma conceptuel satisfaisant le besoin des utilisateurs

Administrateur de la Base de Données : définit les données aux niveaux conceptuel (création et évolution du schéma conceptuel) et interne (emplacement et dimensionnement des fichiers physiques), gère les droits des utilisateurs, optimise les performances, gère les sauvegardes/restaurations, etc.

Ingénieur système : support technique pour la description physique des données et les performances du couplage système d'exploitation/SGBD



Commentaires

L'interface utilisateur permet d'accéder soit à une application, soit de saisir un ordre SQL interactivement

Dans les deux cas, la demande est formulée (*via* le programme ou directement) en SQL

L'ordre SQL est transmis au moteur relationnel qui sert donc d'interface entre les données et les utilisateurs

Le moteur relationnel accède aux données de l'organisation en les retrouvant grâce au dictionnaire des données

Rôles du Dictionnaire de Données

Description du schéma des données : schémas, relations, vues, contraintes d'intégrité, privilèges, déclencheurs, procédures stockées, etc.

Remarque : il s'agit donc de métadonnées i. e. des données (permettant la description de n'importe quel schéma de données) sur les données (du système d'information de l'organisation)

Stockage d'informations supplémentaires à des fins :

de surveillance (exemple : vérification des droits pour chacun des accès),

de statistique (temps de réponse d'une requête, requête la plus fréquemment exécutée, etc.),

d'optimisation (exploitation des statistiques pour indexer, créer des clusters, dénormaliser, etc.),

de restauration (en cas de panne),

etc.

Numéro INE de l'étudiant : *2*
Nom : *LEROI*
Département de naissance : *40*

Voitures possédées par l'étudiant :

Numéro d'immatriculation	Couleur

Diplômes obtenus par l'étudiant :

Intitulé abrégé	Intitulé complet	Année
BAC	*Baccalauréat*	*1980*
DEUG	*Diplôme d'Études Universitaires Générales*	*1982*

Numéro INE de l'étudiant : *3*
Nom : *DUPOND*
Département de naissance : *17*

Voitures possédées par l'étudiant :

Numéro d'immatriculation	Couleur

Diplômes obtenus par l'étudiant :

Intitulé abrégé	Intitulé complet	Année
BAC	*Baccalauréat*	*1981*
DUT	*Diplôme Universitaire de Technologie*	*1983*
MIAGe	*Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises*	*1985*

Numéro INE de l'étudiant : *4*
Nom : *MARTIN*
Département de naissance : *47*

Voitures possédées par l'étudiant :

Numéro d'immatriculation	Couleur
4747 LA 47	*rouge*

Diplômes obtenus par l'étudiant :

Intitulé abrégé	Intitulé complet	Année
BAC	*Baccalauréat*	*1977*
DEUG	*Diplôme d'Études Universitaires Générales*	*1980*
MIAGe	*Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises*	*1982*

Numéro INE de l'étudiant : *5*
Nom : *DURAND*
Département de naissance : *33*

Voitures possédées par l'étudiant :

Numéro d'immatriculation	Couleur
3333 BX 33	*rouge*
4040 NT 40	*jaune*

Diplômes obtenus par l'étudiant :

Intitulé abrégé	Intitulé complet	Année
BAC	*Baccalauréat*	*1981*
DUT	*Diplôme Universitaire de Technologie*	*1983*

Numéro INE de l'étudiant : *7*
Nom : *LEROI*
Département de naissance : *33*

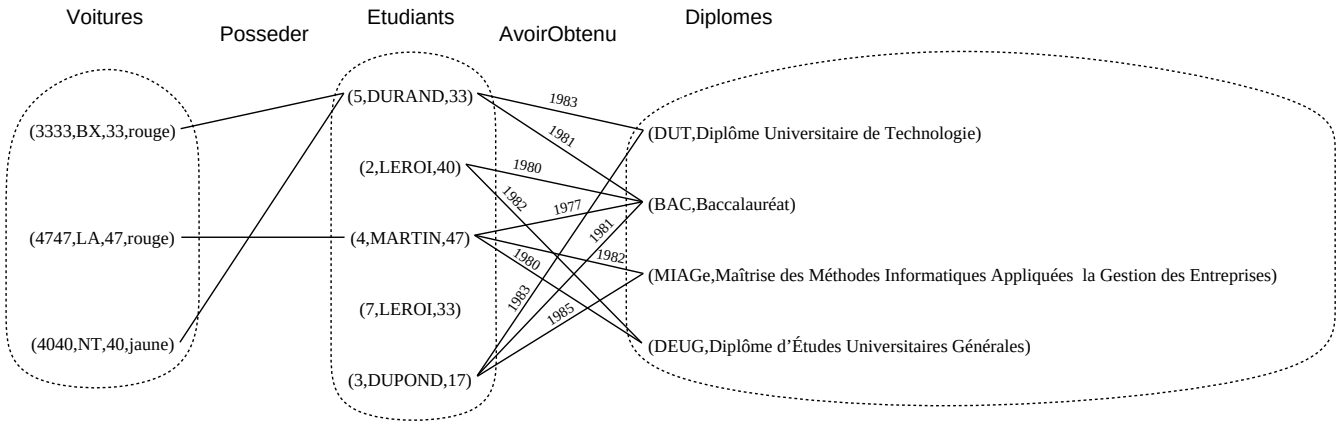
Voitures possédées par l'étudiant :

Numéro d'immatriculation	Couleur

Diplômes obtenus par l'étudiant :

Intitulé abrégé	Intitulé complet	Année

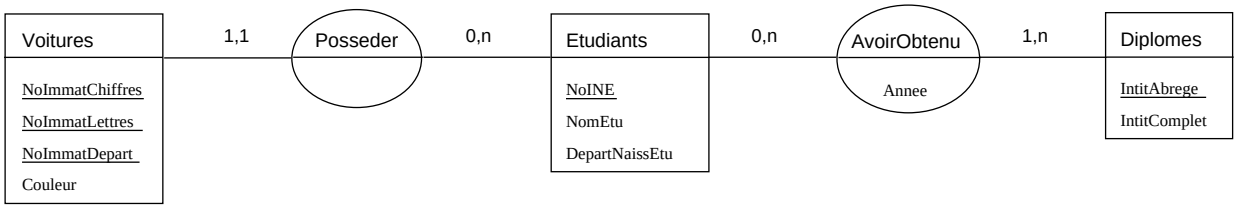
Le jour de la rentrée, le secrétariat de l'établissement avise les étudiants qu'ils ont jusqu'à la fin de la semaine pour apporter les originaux des diplômes qu'ils ont obtenus, ceci permettant de compléter les fiches des étudiants.


Voitures
Posseder
Etudiants
AvoirObtenu
Diplomes
(3333,BX,33,rouge)
(4747,LA,47,rouge)
(4040,NT,40,jaune)
(5,DURAND,33)
(2,LEROI,40)
(4,MARTIN,47)
(7,LEROI,33)
(3,DUPOND,17)
1983
1981
1980
1982
1977
1981
1982
1980
1983
1985
(DUT,Diplôme Universitaire de Technologie)
(BAC,Baccalauréat)
(MIAGe,Maîtrise des Méthodes Informatiques Appliquées la Gestion des Entreprises)
(DEUG,Diplôme d'Études Universitaires Générales)

Machine B

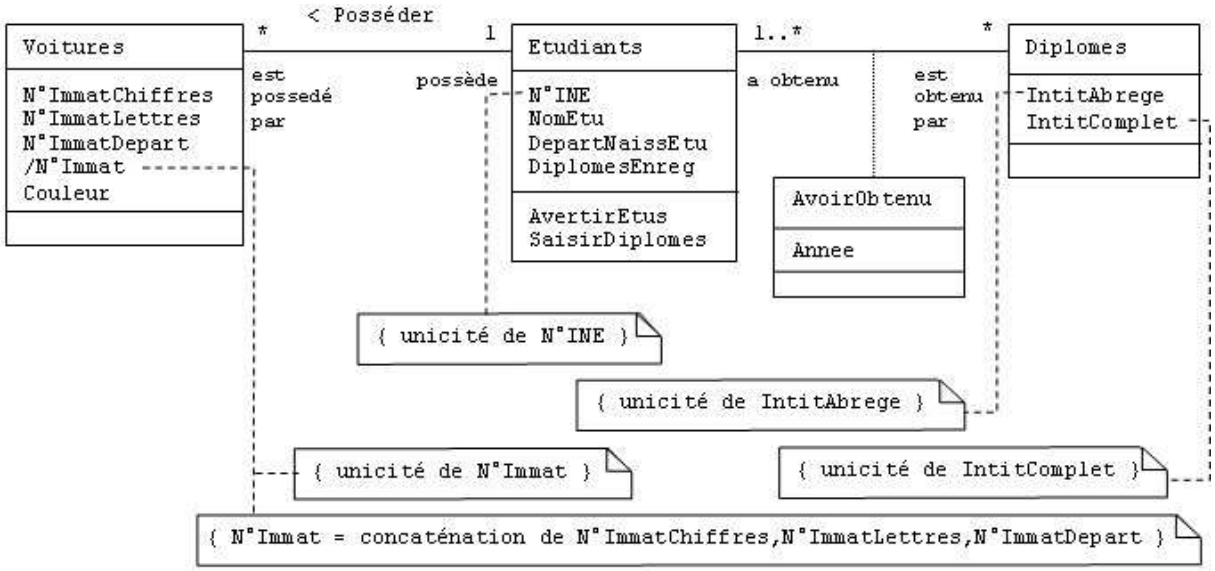
```
MACHINE Exemple_Jouet
SETS VOITURES ; ÉTUDIANTS ; DIPLÔMES
VARIABLES voitures, être_possédée_par, étudiants, avoir_obtenu, diplômes
INVARIANT voitures ⊆ VOITURES ∧
          étudiants ⊆ ÉTUDIANTS ∧
          diplômes ⊆ DIPLÔMES ∧
          être_possédée_par ∈ voitures ⟶ étudiants ∧
          avoir_obtenu ∈ étudiants ⟷ diplômes ∧
          ran(avoir_obtenu) = diplômes
```

Schéma entité-association



Contrainte d'intégrité supplémentaire : unicité de `IntitComplet`

Diagramme de classe



Exemple « jouet »

NoINE	NomEtu	DepartNaissEtu
5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17

...

NoImmatChiffres	NoImmatLettres	NoImmatDepart	Couleur
3333	'BX'	33	'rouge'
4040	'NT'	40	'jaune'
4747	'LA'	47	'rouge'

... ...

IntitAbrege	IntitComplet	Annee
'DUT'	'Diplôme Universitaire de Technologie'	1983
'BAC'	'Baccalauréat'	1981
'DEUG'	'Diplôme d'Études Universitaires Générales'	1982
'BAC'	'Baccalauréat'	1980
'DEUG'	'Diplôme d'Études Universitaires Générales'	1980
'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'	1982
'BAC'	'Baccalauréat'	1977
'DUT'	'Diplôme Universitaire de Technologie'	1983
'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'	1985
'BAC'	'Baccalauréat'	1981

...

La relation universelle permet la représentation de toutes les informations (i. e. sans perte) mais :

Zones variables (zéro, une ou plusieurs valeurs) engendrant des difficultés de stockage efficace

Exemple : les voitures d'un étudiant

Redondance

Exemple : les intitulés complets des diplômes

Schémas des relations

Etudiants (NoINE , NomEtu , DepartNaissEtu)

Voitures (NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur , NoINE)

Diplomes (IntitAbrege , IntitComplet)

AvoirObtenu (NoINE , IntitAbrege , Annee)

Contraintes d'intégrité

Contraintes de clé (primaire)

NoINE est la clé de la relation Etudiants

(NoImmatChiffres , NoImmatLettres , NoImmatDepart) est la clé de la relation Voitures

IntitAbrege est la clé de la relation Diplomes

(NoINE , IntitAbrege) est la clé de la relation AvoirObtenu

Contraintes d'intégrité référentielle

Voitures.NoINE référence Etudiants.NoINE

AvoirObtenu.NoINE référence Etudiants.NoINE

AvoirObtenu.IntitAbrege référence Diplomes.IntitAbrege

Contraintes d'unicité

Diplomes.IntitComplet est unique

Contraintes existentielles

Tous les attributs sont obligatoires

Autres contraintes

valeurs(Diplomes.IntitAbrege) = valeurs(AvoirObtenu.IntitAbrege)

Etudiants

NoINE	NomEtu	DepartNaissEtu
5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17

Voitures

NoImmatChiffres	NoImmatLettres	NoImmatDepart	Couleur	NoINE
3333	'BX'	33	'rouge'	5
4747	'LA'	47	'rouge'	4
4040	'NT'	40	'jaune'	5

Diplomes

IntitAbrege	IntitComplet
'DUT'	'Diplôme Universitaire de Technologie'
'BAC'	'Baccalauréat'
'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
'DEUG'	'Diplôme d'Études Universitaires Générales'

AvoirObtenu

NoINE	IntitAbrege	Annee
4	'DEUG'	1980
2	'DEUG'	1982
5	'DUT'	1983
4	'MIAGe'	1982
3	'DUT'	1983
2	'BAC'	1980
4	'BAC'	1977
3	'MIAGe'	1985
5	'BAC'	1981
3	'BAC'	1981

Ensemble des dépendances fonctionnelles

IntitAbrege $\rightarrow$ IntitAbrege$^{(r)}$,IntitComplet

IntitComplet$^{(s)}$ $\rightarrow$ IntitComplet$^{(r)}$,IntitAbrege

NoINE $\rightarrow$ NoINE$^{(r)}$,NomEtu,DepartNaissEtu

NoINE,IntitAbrege $\rightarrow$ NoINE$^{(r)}$,NomEtu$^{(t)}$,DepartNaissEtu$^{(t)}$,IntitAbrege$^{(r)}$,IntitComplet$^{(t)}$, Annee

NoINE,IntitComplet $\rightarrow$ NoINE$^{(r)}$,NomEtu$^{(t)}$,DepartNaissEtu$^{(t)}$,IntitComplet$^{(r)}$,IntitAbrege$^{(t)}$, Annee$^{(t)}$

NoImmatChiffres,NoImmatLettres,NoImmatDepart $\rightarrow$ NoImmatChiffres$^{(r)}$, NoImmatLettres$^{(r)}$,NoImmatDepart$^{(r)}$,Couleur,NoINE,NomEtu$^{(t)}$,DepartNaissEtu$^{(t)}$

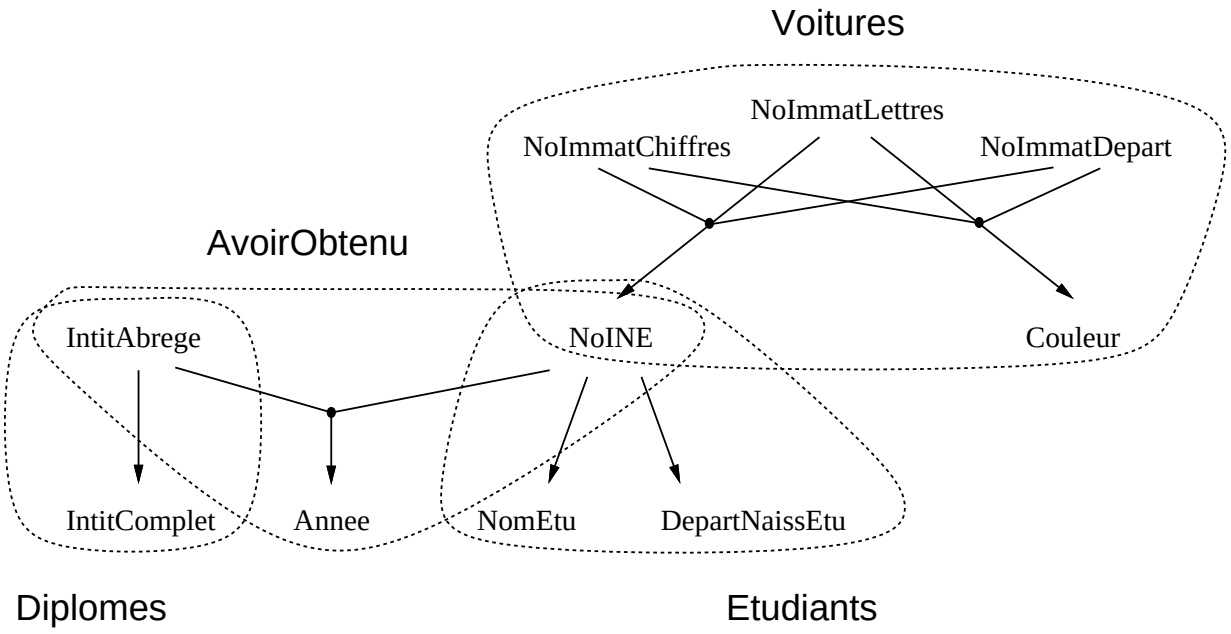
NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitAbrege $\rightarrow$ NoImmatChiffres$^{(r)}$, NoImmatLettres$^{(r)}$,NoImmatDepart$^{(r)}$,Couleur$^{(t)}$,NoINE$^{(t)}$,NomEtu$^{(t)}$,DepartNaissEtu$^{(t)}$, IntitAbrege$^{(r)}$,IntitComplet$^{(t)}$,Annee$^{(t)}$

NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitComplet $\rightarrow$ NoImmatChiffres$^{(r)}$, NoImmatLettres$^{(r)}$,NoImmatDepart$^{(r)}$,Couleur$^{(t)}$,NoINE$^{(t)}$,NomEtu$^{(t)}$,DepartNaissEtu$^{(t)}$, IntitComplet$^{(r)}$,IntitAbrege$^{(t)}$,Annee$^{(t)}$

Légende : $^{(r)}$ = par réflexivité, $^{(s)}$ = clé secondaire, $^{(t)}$ = par transitivité

Hypergraphe des dépendances fonctionnelles

(une fois enlevées les dépendances fonctionnelles de la clé secondaire et celles obtenues par réflexivité ou par transitivité)



Légende : sommet = attribut, hyper-arc = dépendance fonctionnelle

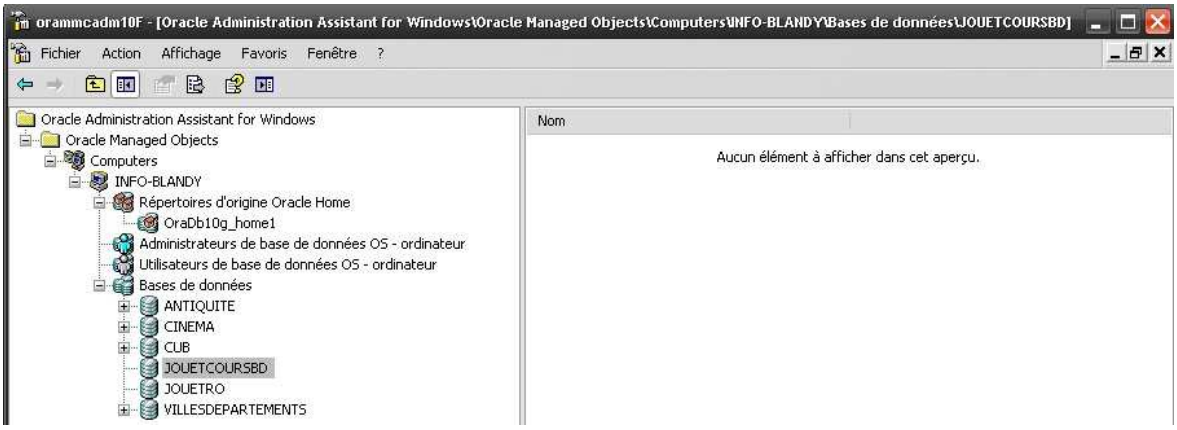
Oracle (1/4)

BD : création de `JouetCoursBD` avec l'assistant configuration de BD, en affectant le mot de passe de `SYSTEM`

Administration : création du compte d'administration `GUIBERT` avec le compte `SYSTEM`


```
ACCEPT pwdGuibert CHAR PROMPT 'Mot de passe de GUIBERT : ' HIDE
CREATE USER GUIBERT IDENTIFIED BY &pwdGuibert ;
GRANT DBA TO GUIBERT WITH ADMIN OPTION ;
```

Vérification à l'aide de l'assistant d'administration pour Windows



Vérification en consultant le fichier `tnsnames.ora` situé dans le répertoire `C:\oracle\product\10.2.0\db_1\NETWORK\ADMIN`

```
JOUETCOU =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
        (CONNECT_DATA = (SERVER = DEDICATED)
                        (SERVICE_NAME = JouetCoursBD) ) )
```

Oracle (2/4)

N. B. : les ordres SQL sont dorénavant exécutés à l'aide du compte `GUIBERT` (dans une session *SQL*Plus* par exemple)

LDD : création du schéma relationnel (1/2)

```
-- suppression des tables (et contraintes), index, vues
DROP VIEW Voitures_Etudiants ;
DROP INDEX Index_NomEtu ;
DROP INDEX Index_Voitures_Etudiants ;
DROP INDEX Index_AvoirObtenu_Etudiants ;
DROP INDEX Index_AvoirObtenu_Diplomes ;
DROP TABLE Voitures ;
DROP TABLE AvoirObtenu ;
DROP TABLE Etudiants ;
DROP TABLE Diplomes ;
-- création des tables et clés primaires, contraintes existentielles
CREATE TABLE Etudiants (
    NoINE          NUMBER(3)   NOT NULL ,
    NomEtu         VARCHAR(20) NOT NULL ,
    DepartNaissEtu NUMBER(2)            ,
        CONSTRAINT ClePrimaire_Etudiants PRIMARY KEY ( NoINE ) ) ;
CREATE TABLE Voitures (
    NoImmatChiffres NUMBER(4)   NOT NULL ,
    NoImmatLettres  CHAR(3)     NOT NULL ,
    NoImmatDepart   NUMBER(2)   NOT NULL ,
    Couleur         VARCHAR(10)          ,
    NoINE           NUMBER(3)   NOT NULL ,
        CONSTRAINT ClePrimaire_Voitures
            PRIMARY KEY ( NoImmatChiffres ,
                          NoImmatLettres ,
                          NoImmatDepart ) ) ;
CREATE TABLE Diplomes (
    IntitAbrege  CHAR(5)     NOT NULL ,
    IntitComplet VARCHAR(80) NOT NULL ,
        CONSTRAINT ClePrimaire_Diplomes PRIMARY KEY ( IntitAbrege ) ) ;
CREATE TABLE AvoirObtenu (
    NoINE       NUMBER(3) NOT NULL ,
    IntitAbrege CHAR(5)   NOT NULL ,
    Annee       NUMBER(4)          ,
        CONSTRAINT ClePrimaire_AvoirObtenu
            PRIMARY KEY ( NoINE , IntitAbrege ) ) ;
-- création des contraintes d'intégrité référentielles
ALTER TABLE Voitures ADD CONSTRAINT CIRef_Voitures_Etudiants
    FOREIGN KEY ( NoINE ) REFERENCES Etudiants ( NoINE ) ;
ALTER TABLE AvoirObtenu ADD CONSTRAINT CIRef_AvoirObtenu_Etudiants
    FOREIGN KEY ( NoINE ) REFERENCES Etudiants ( NoINE ) ;
ALTER TABLE AvoirObtenu ADD CONSTRAINT CIRef_AvoirObtenu_Diplomes
    FOREIGN KEY ( IntitAbrege ) REFERENCES Diplomes ( IntitAbrege ) ;
```

Oracle (3/4)

LDD : création du schéma relationnel (2/2)

```
-- création des index
CREATE INDEX Index_Voitures_Etudiants    ON Voitures    ( NoINE ASC ) ;
CREATE INDEX Index_AvoirObtenu_Etudiants ON AvoirObtenu ( NoINE ASC ) ;
CREATE INDEX Index_AvoirObtenu_Diplomes
    ON AvoirObtenu ( IntitAbrege ASC ) ;
CREATE INDEX Index_NomEtu ON Etudiants ( NomEtu ASC ) ;
-- création des vues
CREATE VIEW Voitures_Etudiants AS
    SELECT NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur ,
           NoINE , NomEtu , DepartNaissEtu
    FROM Voitures
    NATURAL JOIN Etudiants ; -- i. e. JOIN ON
                             -- Voitures.NoINE = Etudiants.NoINE
-- création des synonymes
CREATE PUBLIC SYNONYM Etudiants   FOR GUIBERT.Etudiants ;
CREATE PUBLIC SYNONYM Voitures    FOR GUIBERT.Voitures ;
CREATE PUBLIC SYNONYM Diplomes    FOR GUIBERT.Diplomes ;
CREATE PUBLIC SYNONYM AvoirObtenu FOR GUIBERT.AvoirObtenu ;
CREATE PUBLIC SYNONYM Voitures_Etudiants
    FOR GUIBERT.Voitures_Etudiants ;
```

Comptes : création des comptes utilisateurs

```
-- création du compte ETD
CREATE USER ETD IDENTIFIED BY ETD ;
GRANT CONNECT TO ETD ;
```

Droits : attribution des privilèges

```
-- attribution des privilèges au compte ETD
GRANT SELECT ON GUIBERT.Etudiants   TO ETD ;
GRANT SELECT ON GUIBERT.Voitures    TO ETD ;
GRANT SELECT ON GUIBERT.Diplomes    TO ETD ;
GRANT SELECT ON GUIBERT.AvoirObtenu TO ETD ;
GRANT SELECT ON GUIBERT.Voitures_Etudiants TO ETD ;
```

Oracle (4/4)

Données : insertion des tuples

```
-- suppression de toutes les données
DELETE FROM AvoirObtenu ;
DELETE FROM Diplomes ;
DELETE FROM Voitures ;
DELETE FROM Etudiants ;
COMMIT ;
-- insertion des données
INSERT INTO Etudiants VALUES ( 5 , 'DURAND' , 33 ) ;
INSERT INTO Etudiants VALUES ( 2 , 'LEROI'  , 40 ) ;
INSERT INTO Etudiants VALUES ( 4 , 'MARTIN' , 47 ) ;
INSERT INTO Etudiants VALUES ( 7 , 'LEROI'  , 33 ) ;
INSERT INTO Etudiants VALUES ( 3 , 'DUPOND' , 17 ) ;
COMMIT ;
INSERT INTO Voitures VALUES ( 3333 , 'BX' , 33 , 'rouge' , 5 ) ;
INSERT INTO Voitures VALUES ( 4747 , 'LA' , 47 , 'rouge' , 4 ) ;
INSERT INTO Voitures VALUES ( 4040 , 'NT' , 40 , 'jaune' , 5 ) ;
COMMIT ;
INSERT INTO Diplomes
    VALUES ( 'DUT'   , 'Diplome Universitaire de Technologie'     ) ;
INSERT INTO Diplomes
    VALUES ( 'BAC'   , 'Baccalaureat'                             ) ;
INSERT INTO Diplomes
    VALUES ( 'MIAGe' , 'Maitrise des Methodes Informatiques ' ||
                       'Appliquees a la Gestion des Entreprises'  ) ;
INSERT INTO Diplomes
    VALUES ( 'DEUG'  , 'Diplome d Etudes Universitaires Generales' ) ;
COMMIT ;
INSERT INTO AvoirObtenu VALUES ( 4 , 'DEUG'  , 1980 ) ;
INSERT INTO AvoirObtenu VALUES ( 2 , 'DEUG'  , 1982 ) ;
INSERT INTO AvoirObtenu VALUES ( 5 , 'DUT'   , 1983 ) ;
INSERT INTO AvoirObtenu VALUES ( 4 , 'MIAGe' , 1982 ) ;
INSERT INTO AvoirObtenu VALUES ( 3 , 'DUT'   , 1983 ) ;
INSERT INTO AvoirObtenu VALUES ( 2 , 'BAC'   , 1980 ) ;
INSERT INTO AvoirObtenu VALUES ( 4 , 'BAC'   , 1977 ) ;
INSERT INTO AvoirObtenu VALUES ( 3 , 'MIAGe' , 1985 ) ;
INSERT INTO AvoirObtenu VALUES ( 5 , 'BAC'   , 1981 ) ;
INSERT INTO AvoirObtenu VALUES ( 3 , 'BAC'   , 1981 ) ;
COMMIT ;
```

Conception d'une BD : schéma relationnel

On considère les informations suivantes destinées à compléter l'exemple « jouet »

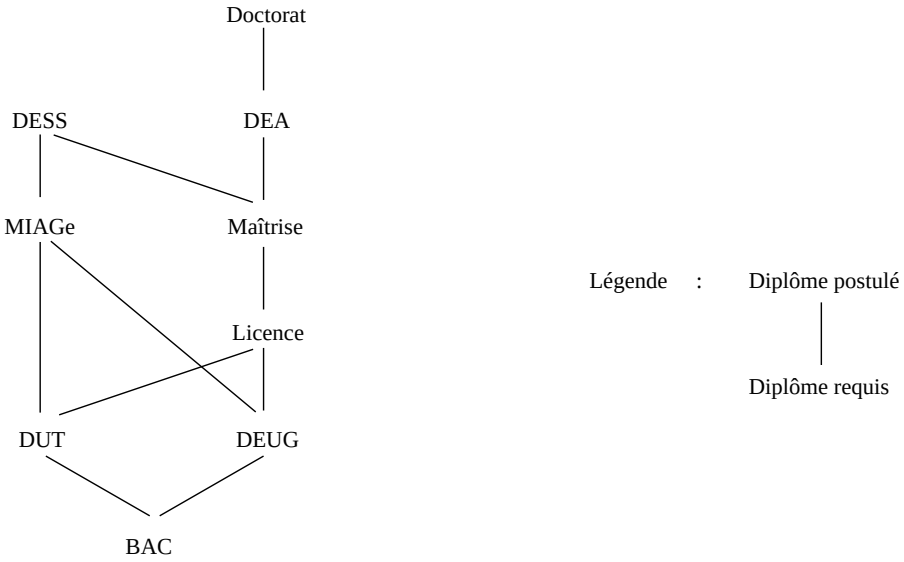
L'étudiant PICARD a acheté pour 28000 francs (soit 4268,57 €) une Twingo immatriculée 1717 LR 17 le 27 juin 1989

On désire savoir quels sont les étudiants qui peuvent conduire les voitures des étudiants (autres que leurs possesseurs)

L'information sur les diplômes est insuffisante : il faut également considérer les établissements (avec notamment leurs adresses) qui ont délivré ces diplômes aux étudiants ; de plus, tous les diplômes et tous les établissements concernent au moins un étudiant

On veut également pouvoir considérer les adresses des logements des étudiants. Toutefois, tous les étudiants n'ont pas forcément un tel logement et certains habitent chez leurs parents. En revanche, on désire connaître l'adresse des parents de tous les étudiants

On se propose de constituer une classification de poursuites d'études ; pour cela, nous disposons de l'esquisse suivante faisant apparaître les diplômes requis et postulés



Remarque : depuis la réforme du LMD (Licence/Master/Doctorat) c.-à-d. l'application au système français de la construction de l'espace européen de l'enseignement supérieur (dit processus de Bologne), la dénomination des diplômes a évolué : les DESS et DEA sont maintenant des M2 (2^e année de Master), la Maîtrise est maintenant un M1 (1^{re} année de Master), la Licence est maintenant une L3 (3^e année de Licence) et le DEUG est maintenant une L2 (2^e année de Licence)

Proposez un nouveau schéma relationnel (en compréhension)

Exercices (2/2)

Interrogation d'une BD : algèbre relationnelle, calcul relationnel et SQL

Écrivez en algèbre relationnelle, en calcul relationnel et en SQL les requêtes suivantes :

Les libellés complets des diplômes

Les voitures immatriculées en Gironde (33) et à Paris (75)

Les années d'obtention du DUT qui ne sont pas des années d'obtention du BAC

Les étudiants et leurs diplômes

La relation universelle

Les étudiants sans diplôme

Tous les diplômes auxquels peuvent prétendre tous les étudiants

Les étudiants ayant obtenu tous les diplômes

Pour chaque étudiant, les diplômes qu'il n'a pas obtenus

Schémas des relations

Logements (NoLogement , AdrLogement , NoINE)

Etudiants (NoINE , NomEtu , PrenomEtu , DepartNaissEtu , AdrParentsEtu)

Voitures (NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur , Marque , NoINE , DateAchat , MontantAchat)

Conduire (NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart)

Etablissements (NoEtablissement , NomEtablissement , AdrEtablissement)

EtreTitulaire (NoINE , IntitAbrege , NoEtablissement , Annee)

Diplomes (IntitAbrege , IntitComplet)

PoursuiteEtudes (IntitAbregeRequis , IntitAbregePostule)

Contraintes d'intégrité

Contraintes de clé (primaire)

NoLogement est la clé de la relation Logements

NoINE est la clé de la relation Etudiants

(NoImmatChiffres , NoImmatLettres , NoImmatDepart) est la clé de la relation Voitures

(NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart) est la clé de la relation Conduire

NoEtablissement est la clé de la relation Etablissements

(NoINE , IntitAbrege , NoEtablissement) est la clé de la relation EtreTitulaire

IntitAbrege est la clé de la relation Diplomes

(IntitAbregeRequis , IntitAbregePostule) est la clé de la relation PoursuiteEtudes

Contraintes d'intégrité référentielle

Logements.NoINE référence Etudiants.NoINE

Voitures.NoINE référence Etudiants.NoINE

Conduire.NoINE référence Etudiants.NoINE

(Conduire.NoImmatChiffres , Conduire.NoImmatLettres , Conduire.NoImmatDepart) référence (Voitures.NoImmatChiffres , Voitures.NoImmatLettres , Voitures.NoImmatDepart)

EtreTitulaire.NoINE référence Etudiants.NoINE

EtreTitulaire.IntitAbrege référence Diplomes.IntitAbrege

EtreTitulaire.NoEtablissement référence Etablissement.NoEtablissement

PoursuiteEtudes.IntitAbregeRequis référence Diplomes.IntitAbrege

PoursuiteEtudes.IntitAbregePostule référence Diplomes.IntitAbrege

Contraintes d'unicité

Logements.NoINE est unique

Diplomes.IntitComplet est unique

Contraintes existentielles

Tous les attributs sont obligatoires

Autres contraintes

valeurs(Voitures.NoINE,Voitures.NoImmatChiffres,Voitures.NoImmatLettres, Voitures.NoImmatDepart) $\cap$ valeurs(Conduire.NoINE,Conduire.NoImmatChiffres, Conduire.NoImmatLettres,Conduire.NoImmatDepart) = $\emptyset$

valeurs(Diplomes.IntitAbrege) = valeurs(EtreTitulaire.IntitAbrege)

valeurs(Etablissement.NoEtablissement) = valeurs(EtreTitulaire.NoEtablissement)

PoursuiteEtudes.IntitAbregeRequis et PoursuiteEtudes.IntitAbregePostule forment un graphe orienté acyclique (pas obligatoirement connexe)

Les libellés complets des diplômes

$\pi_{\text{IntitComplet}}(\text{Diplomes})$

$\{(\text{IntitComplet}) / \exists a : (a, \text{IntitComplet}) \in \text{Diplomes}\}$

```
SELECT IntitComplet -- DISTINCT inutile car unicité de IntitComplet
FROM Diplomes
```

Les voitures immatriculées en Gironde (33) et à Paris (75)

$\sigma_{\text{NoImmatDepart} \in \{33,75\}}(\text{Voitures})$

$\{(a, b, \text{NoImmatDepart}, c, d) : (a, b, \text{NoImmatDepart}, c, d) \in \text{Voitures} \wedge \text{NoImmatDepart} \in \{33, 75\}\}$

```
SELECT *
FROM Voitures
WHERE NoImmatDepart IN ( 33 , 75 )
```

Les années d'obtention du DUT qui ne sont pas des années d'obtention du BAC (avec une opération ensembliste)

$\pi_{\text{Annee}}(\sigma_{\text{IntitAbrege='DUT'}}(\text{AvoirObtenu})) \setminus$
$\pi_{\text{Annee}}(\sigma_{\text{IntitAbrege='BAC'}}(\text{AvoirObtenu}))$

$\{(\text{Annee}) / \exists \text{IntitAbrege}, c : (\text{IntitAbrege}, c, \text{Annee}) \in \text{AvoirObtenu} \wedge \text{IntitAbrege='DUT'}\} \setminus$
$\{(\text{Annee}) / \exists \text{IntitAbrege}, c : (\text{IntitAbrege}, c, \text{Annee}) \in \text{AvoirObtenu} \wedge \text{IntitAbrege='BAC'}\}$

```
SELECT Annee
FROM AvoirObtenu
WHERE IntitAbrege = 'DUT'
EXCEPT
SELECT Annee
FROM AvoirObtenu
WHERE IntitAbrege = 'BAC'
```

Les étudiants et leurs diplômes

$\text{Etudiants} \bowtie \text{AvoirObtenu} \bowtie \text{Diplomes}$

$\{(\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}, \text{IntitAbrege}, \text{IntitComplet}, \text{Annee}) :$
$(\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) \in \text{Etudiants} \wedge (\text{NoINE}, \text{IntitAbrege}, \text{Annee}) \in \text{AvoirObtenu} \wedge (\text{IntitAbrege}, \text{IntitComplet}) \in \text{Diplomes}\}$

```
SELECT *
FROM Etudiants
NATURAL JOIN AvoirObtenu
NATURAL JOIN Diplomes
```

La relation universelle (en algèbre relationnelle et en SQL uniquement)

$\text{Voitures} \overset{\leftrightarrow}{\bowtie} \text{Etudiants} \overset{\leftrightarrow}{\bowtie} \text{AvoirObtenu} \overset{\leftrightarrow}{\bowtie} \text{Diplomes}$

```
SELECT *
FROM Voitures
FULL OUTER NATURAL JOIN Etudiants
FULL OUTER NATURAL JOIN AvoirObtenu
FULL OUTER NATURAL JOIN Diplomes
```

Les étudiants sans diplôme (avec une jointure externe ou une sous-requête)

$\pi_{\text{NoINE,NomEtu,DepartNaissEtu}}(\sigma_{\text{IntitAbrege pas renseigné}}(\text{Etudiants} \overset{\leftarrow}{\bowtie} \text{AvoirObtenu}))$

$\{(\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) : (\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) \in \text{Etudiants} \wedge \text{NoINE} \notin \{(\text{NoINE})/\exists d, a : (\text{NoINE}, d, a) \in \text{AvoirObtenu}\}\}$

```
-- jointure externe
SELECT E.*
FROM Etudiants E
LEFT OUTER JOIN AvoirObtenu
WHERE IntitAbrege IS NULL
-- sous-requête
SELECT *
FROM Etudiants E
WHERE NoINE NOT IN ( SELECT NoINE
                     FROM AvoirObtenu )
```

Tous les diplômes auxquels peuvent prétendre tous les étudiants

$\text{Diplomes} \times \text{Etudiants}$

$\{(\text{IntitAbrege}, \text{IntitComplet}, \text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) : (\text{IntitAbrege}, \text{IntitComplet}) \in \text{Diplomes} \wedge (\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) \in \text{Etudiants}\}$

```
SELECT *
FROM Diplomes , Etudiants
```

Les étudiants ayant obtenu tous les diplômes

$\pi_{\text{NoINE,NomEtu,DepartNaissEtu}}((\text{Etudiants} \bowtie \text{AvoirObtenu}) \div \text{Diplomes})$

$\{(\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) : (\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) \in \text{Etudiants} \wedge \forall(\text{IntitAbrege}, \text{IntitComplet}) \in \text{Diplomes} \exists a : (\text{NoINE}, \text{IntitAbrege}, a) \in \text{AvoirObtenu}\}$

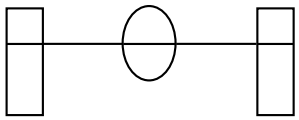
```
SELECT E.*
FROM Etudiants E
NATURAL JOIN AvoirObtenu
GROUP BY E.*
HAVING COUNT(*) = ( SELECT COUNT(*)
                    FROM Diplomes   )
```

Pour chaque étudiant, les diplômes qu'il n'a pas obtenus

$$(\text{Diplomes} \times \text{Etudiants}) \setminus \pi_{\text{NoINE,NomEtu,DepartNaissEtu,IntitAbrege,IntitComplet}}(\text{Diplomes} \bowtie \text{AvoirObtenu} \bowtie \text{Etudiants})$$

$$\{(\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}, \text{IntitAbrege}, \text{IntitComplet}) : (\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) \in \text{Etudiants} \wedge (\text{IntitAbrege}, \text{IntitComplet}) \in \text{Diplomes} \wedge \nexists a : (\text{NoINE}, \text{IntitAbrege}, a) \in \text{AvoirObtenu}\}$$

```
SELECT *
FROM Etudiants
CROSS JOIN Diplomes
EXCEPT
SELECT E.* , D.*
FROM Etudiants
NATURAL JOIN AvoirObtenu
NATURAL JOIN Diplomes
```



Autres dénominations

Modèle entité-liaison

Modèle objet-relation

Modèle individuel [TARDIEU]

Entity-Relationship model [CHEN]

Modèles voisins

Modèle organisationnel des données (MOD)

Modèle logique des données (MLD)

Modèle relationnel

Modèle navigationnel

Modèle objet

Diagramme de classes

Origine

1976 : CHEN

1976 : MOULIN, RANDON, SAVOYSKY, SPACCAPIETRA, TARDIEU, TEBOUL

Normalisation

Modèle retenu par l'*International Standardization Organization* (ISO) pour décrire les aspects conceptuels statiques des données dans la conception des systèmes d'information

Remarques

Modèle conceptuel des données (MCD) des méthodes MERISE et AXIAL notamment

Outil de dialogue

Représentation graphique simple et standard

Assure une communication efficace entre d'une part les participants à la conception (utilisateurs et concepteurs) et d'autre part les participants à la mise en œuvre de la solution informatisée (informaticiens)

Approche descendante

Définitions (1/3)

Entité : individu ou objet concret ou objet abstrait qui possède une existence intrinsèque et une certaine stabilité permettant de le repérer au cours du temps

Exemples : l'étudiant DURAND, la voiture immatriculée 4040 NT 40, le diplôme DEUG

Association : regroupement d'entités dans lequel chaque entité joue un rôle précis

Exemples : l'étudiant MARTIN possède la voiture immatriculée 4747 LA 47, les étudiants MARTIN et DUPOND ont tous deux obtenu une MIAGe (respectivement en 1982 et 1985)

Propriété : caractéristique d'une entité ou d'une association

Exemples : le département de naissance de l'étudiant DURAND est 33, la couleur de la voiture immatriculée 4747 LA 47 est rouge, l'intitulé complet du diplôme DUT est Diplôme Universitaire de Technologie

Remarque : une propriété est désignée par un nom et une valeur

Son type peut être simple, structuré ou répétitif

Type de propriétés (ou propriété-type) : nom d'une propriété

Exemples : le département de naissance des étudiants, la couleur des voitures, l'intitulé complet des diplômes, l'année d'obtention des diplômes par les étudiants

Remarque : contraintes d'unicité sur les valeurs des propriétés (toute propriété prend une valeur et une seule) et d'élémentarité sur les types de propriétés (entier, réel, booléen, caractère, texte, date, etc. mais pas tableau, structure, liste, pile, file, arbre, graphe, etc.)

Type d'entités (ou entité-type) : ensemble d'entités définies par un même ensemble de types de propriétés jouant un rôle identique et représentant une classe naturelle d'objets

Exemples : les types d'entités Etudiants, Voitures et Diplômes

Type d'associations (ou association-type) : ensemble d'associations ayant la même sémantique

Exemples : Posseder entre les types d'entités Etudiants et Voitures, AvoirObtenu entre les types d'entités Etudiants et Diplômes

Remarque : type d'associations binaire ou n-aire (i. e. mettant en jeu deux ou plusieurs types d'entités, de dimension 2 ou n), réflexive (i. e. mettant en jeu au moins deux fois le même type d'entités)

Lien : couple type d'entités et type d'associations, dont la signification est donnée par le rôle si nécessaire

Cardinalités : pour un lien donné, nombre minimum (cardinalité minimale) et maximum (cardinalité maximale) d'occurrences du type d'associations pouvant exister pour une seule occurrence du type d'entités
(i. e. les nombres minimum —au moins— et maximum —au plus— des cardinaux des images de l'application qui fait correspondre à un élément du type d'entités les éléments des autres types d'entités qui lui sont reliés par le type d'associations)

La cardinalité minimale prend pour valeur 0 ou 1 tandis que la cardinalité maximale prend pour valeur 1 ou n (n pour plusieurs)

Exemple : Les cardinalités minimale et maximale du lien entre le type d'entités Etudiants et le type d'associations Posseder sont respectivement 0 et n. En effet, il suffit de choisir les bonnes affirmations parmi les suivantes

Un étudiant peut ne pas posséder de voiture (cardinalité minimale = 0) [bonne réponse] ou tout étudiant possède au moins une voiture (cardinalité minimale = 1) [mauvais choix]

Tout étudiant possède au plus une voiture (cardinalité maximale = 1) [mauvais choix] ou un étudiant peut posséder plusieurs voitures (cardinalité maximale = n)) [bonne réponse]

Classification des liens d'un type d'associations binaire

Liens 1 : 1 (un à un) [injection partielle] : les deux cardinalités maximales ont pour valeur 1
(i. e. qu'à une occurrence de l'un des deux types d'entités peut correspondre par le type d'associations au plus une occurrence de l'autre type d'entités, et réciproquement)

Liens 1 : n (un à plusieurs) [fonction partielle] : l'une des deux cardinalités maximales a pour valeur 1 tandis que l'autre a pour valeur n
(i. e. qu'à une occurrence de l'un des deux types d'entités peut correspondre par le type d'associations au plus une occurrence de l'autre type d'entités, mais qu'à une occurrence de celui-ci correspond plusieurs occurrences du premier type d'entités cité)

Liens n : n (plusieurs à plusieurs) [relation quelconque] : les deux cardinalités maximales ont pour valeur n
(i. e. qu'à une occurrence de l'un des deux types d'entités peut correspondre par le type d'associations plusieurs occurrences de l'autre type d'entités, et réciproquement)

Identifiant : type(s) de propriété dont les valeurs permettent de distinguer toutes les occurrences du type d'entités correspondant

Exemples : numéro INE des étudiants, numéro d'immatriculation complet (chiffres, lettres et département) des voitures

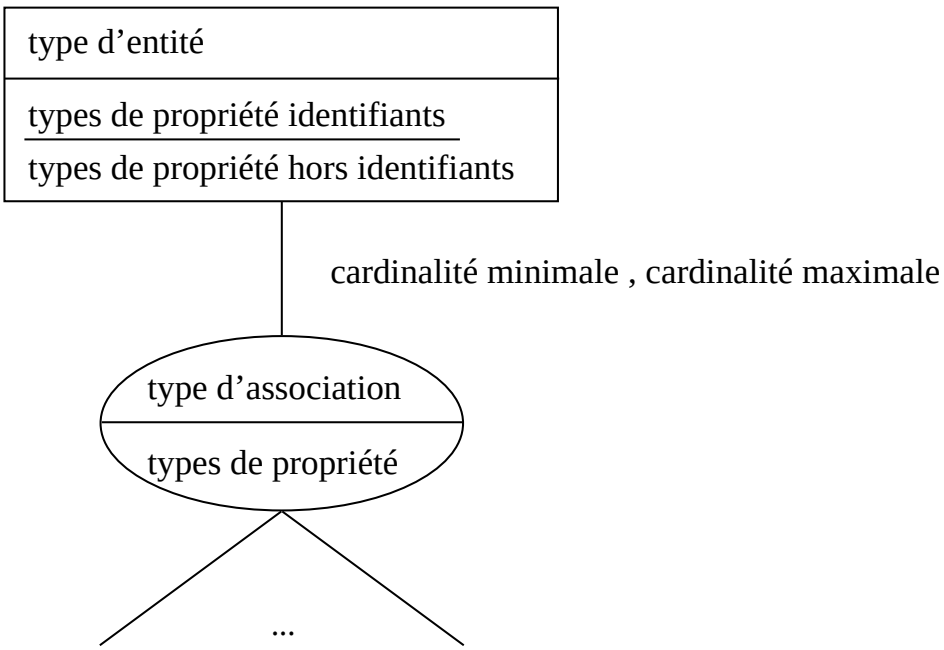
Contre-exemple : nom des étudiants (homonymies)

Remarque : l'identifiant d'un type d'associations s'obtient implicitement par composition de tous les identifiants des types d'entités reliés

Le dictionnaire des données est une représentation textuelle de toutes les informations d'un système d'information ; il se subdivise en trois sous-dictionnaires pour les types d'entités, les types d'associations et les types de propriétés

Définitions (3/3)

Le schéma entité-association est une représentation graphique du résultat de la modélisation du système d'information



Remarque : par commodité de langage, les termes entité, association, propriété désignent respectivement les termes type d'entités, type d'associations, type de propriétés

Partir de l'univers du discours

Reconnaître les types d'entités et les types d'associations

Un objet est reconnu soit en tant que type d'entités, soit en tant que type d'associations

Chaque type d'associations relie deux ou plusieurs (voire plusieurs fois le même) types d'entités

Plusieurs types d'associations différents peuvent relier les mêmes types d'entités, à condition d'avoir des significations différentes

Le schéma est normalement connexe

Les types d'entités et types d'associations sont des classes d'objets de sorte qu'ils possèdent tous plusieurs occurrences

Problème : perception de la réalité !

Les livres suivants sont disposés sur une étagère :

RACINE *Phèdre*	RACINE *Phèdre*	ZOLA *Germinal*

Le bibliothécaire voit trois volumes tandis que le libraire voit deux œuvres

Reconnaître les types de propriétés, les attribuer aux types d'entités ou types d'associations, choisir les identifiants

Ne relever que les types de propriétés pertinents pour l'organisation :

pas de synonyme, polysème, paramètre

pas d'information calculable (ou sinon impérativement munie d'une contrainte d'intégrité)

Un type de propriétés est attribué une fois et une seule, soit à un type d'entités, soit à un type d'associations

Pour chaque occurrence de type d'entités et de type d'associations, une et une seule valeur doit être affectée (à terme) à chaque type de propriétés

Chaque type de propriétés d'un type d'associations doit décrire exactement (en nombre et en nature) les types d'entités reliés

Chaque type d'entités doit posséder un identifiant (éventuellement composé de plusieurs types de propriétés)

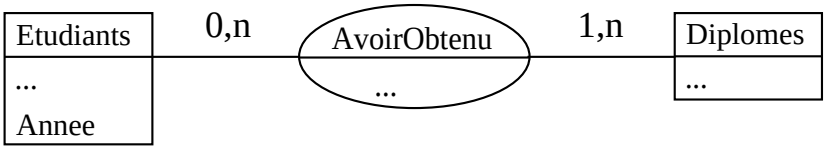
Aucun type d'associations ne doit posséder d'identifiant (il est implicitement donné par la composition des identifiants des types d'entités reliés) mais peut éventuellement posséder des types de propriétés

Pour chaque type d'entités, choix d'un identifiant (minimal) parmi les types de propriétés candidats de préférence celui couramment utilisé dans l'organisation ou à défaut création d'un type de propriétés identifiant

Tous les types de propriétés hors identifiant doivent dépendre pleinement de l'identifiant

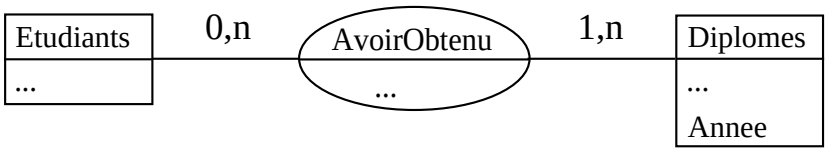
Chaque type de propriétés hors identifiant doit dépendre directement de l'identifiant

Problème : où attribuer un type de propriétés ?



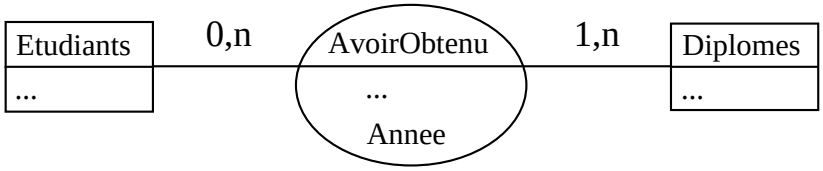
Un étudiant peut avoir obtenu plusieurs diplômes différents deux à deux, et de même un diplôme peut avoir été obtenu par plusieurs étudiants différents deux à deux (liens $n : n$)

Chaque étudiant n'a qu'une année possible pour l'obtention de tous ses diplômes (le type de propriétés Annee appartient au type d'entités Etudiants)



Un étudiant peut avoir obtenu plusieurs diplômes différents deux à deux, et de même un diplôme peut avoir été obtenu par plusieurs étudiants différents deux à deux (liens $n : n$)

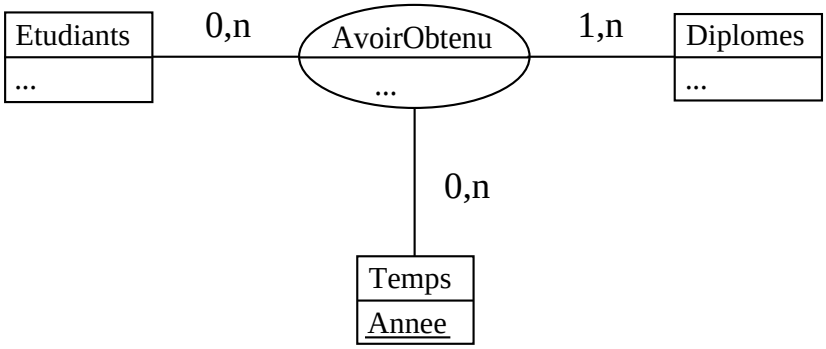
Chaque diplôme n'a qu'une année possible pour avoir été obtenu par tous les étudiants concernés (le type de propriétés Annee appartient au type d'entités Diplomes)



Un étudiant peut avoir obtenu plusieurs diplômes différents deux à deux, et de même un diplôme peut avoir été obtenu par plusieurs étudiants différents deux à deux (liens $n : n$)

L'année d'obtention peut varier pour un diplôme donné ou encore pour un étudiant donné (le type de propriétés Annee appartient au type d'associations AvoirObtenu)

Remarque : on parle de représentation synchronique (i. e. vision instantanée)



Un étudiant donné peut avoir obtenu un même diplôme à des années différentes

Remarque : on parle de représentation diachronique (i. e. vision historique)

Conclusion : ces quatre solutions sont toutes valables (même si dans ce contexte les deux premières semblent peu réalistes) mais ont des significations différentes ; le choix dépendra donc toujours de la réalité à modéliser

Démarche de conception (3/3)

Déterminer les cardinalités

Pour chaque lien, se demander si une occurrence du type d'entités est forcément liée par le type d'associations, et combien de liens au maximum sont associés à une occurrence du type d'entités

Valider le schéma (ou quelques règles supplémentaires à vérifier)

Supprimer un type d'associations redondant, c.-à-d. ayant la même sémantique que plusieurs autres types d'associations reliant transitivement par des liens $1,1$ les mêmes types d'entités

Compléter le schéma par des contraintes d'intégrité

Corriger quelques erreurs courantes

Ne pas confondre les types d'entités et les acteurs du système d'information

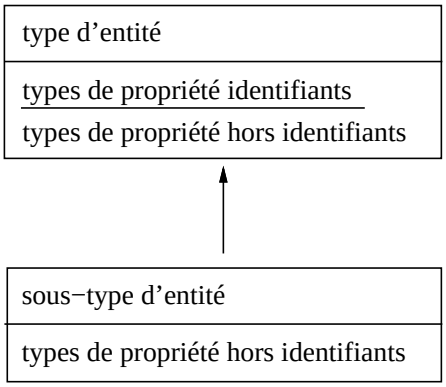
Ne pas confondre les types d'associations et les traitements

Remarque

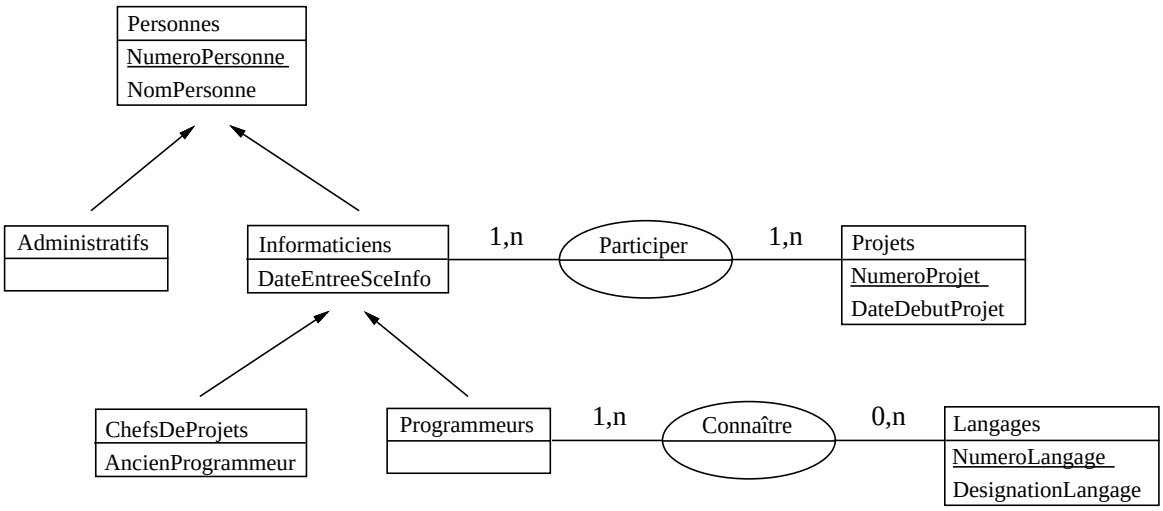
Conception facile (il est rapide de définir une solution possible) mais délicate (choix difficiles pour exprimer correctement la réalité)

Sous-type d'entités

Sous-ensemble d'occurrences d'un type d'entités qui sont dotées de types d'associations et/ou de types de propriétés spécifiques (en plus des types de propriétés héritées du type d'entités concerné)
On parle de type d'entités générique ou sur-type (d'entités) pour le type d'entités et de type d'entités spécialisé ou sous-type (d'entités) pour le sous-type d'entités

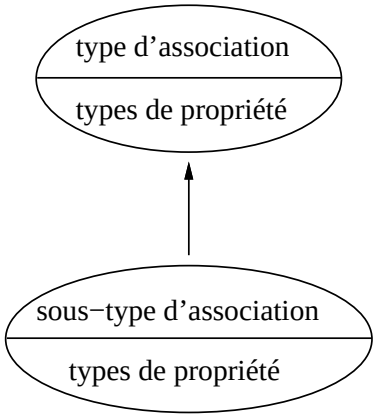


Exemple :

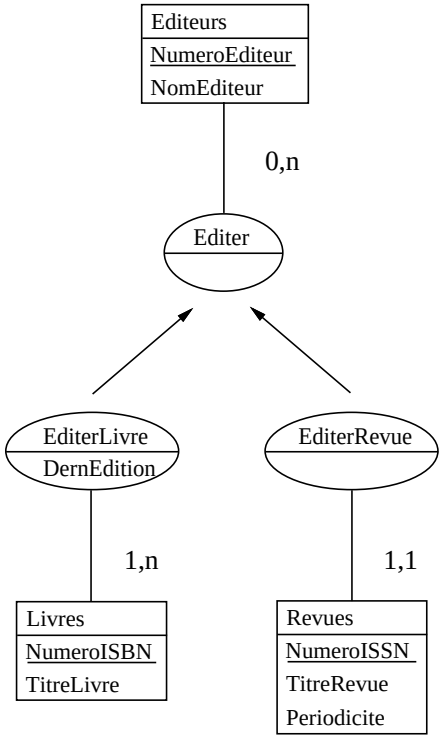


Sous-type d'associations

Sous-ensemble d'occurrences d'un type d'associations qui sont dotées de types de propriétés et/ou de cardinalités spécifiques (en plus des types de propriétés héritées du type d'associations concerné)
On parle de type d'associations générique pour le type d'associations et de type d'associations spécialisée pour le sous-type d'associations



Exemple :



Un livre peut être édité par plusieurs éditeurs tandis qu'une revue n'est éditée que par un et un seul éditeur

Contraintes s'appliquant aux sous-types d'entités

Partition 

Toute occurrence du type d'entités appartient à une et une seule occurrence du sous-type d'entités

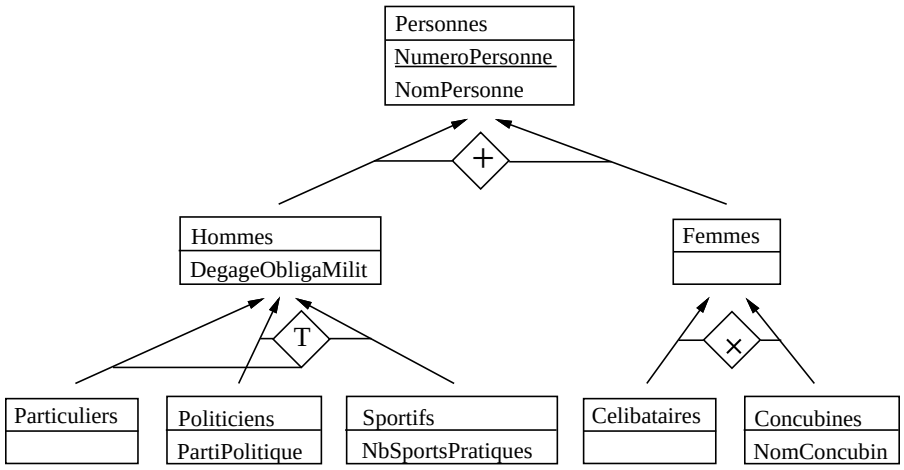
Totalité 

Toute occurrence du type d'entités appartient à au moins une occurrence du sous-type d'entités

Exclusion 

Toute occurrence du type d'entités appartient à au plus une occurrence du sous-type d'entités

Exemple :



 : une personne est soit un homme, soit une femme

 : un homme peut être à la fois un particulier, un politicien et un sportif, ou encore appartenir à deux des trois catégories ; de plus, une personne appartient à au moins l'une des trois catégories

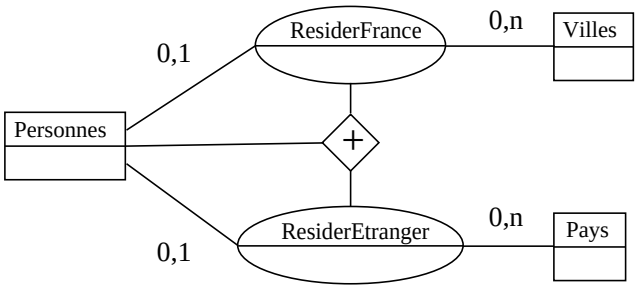
 : une femme ne peut pas être à la fois célibataire et concubine, mais peut n'être ni l'une ni l'autre (mariée par exemple)

Contraintes s'appliquant aux types d'associations (1/2)

Partition 

Toute occurrence du(des) type(s) d'entités correspond à une et une seule des occurrences des types d'associations

Exemple :

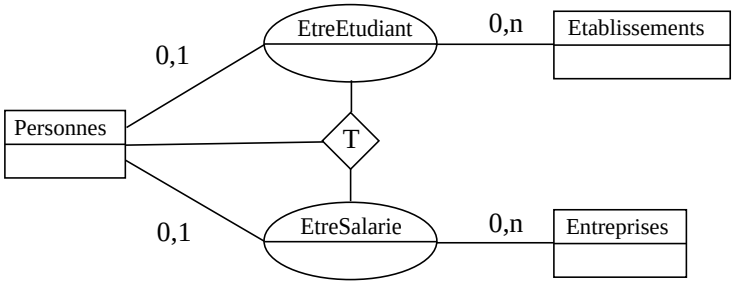


Chaque personne réside soit en France, soit à l'étranger

Totalité 

Toute occurrence du(des) type(s) d'entités correspond à au moins une des occurrences des types d'associations

Exemple :

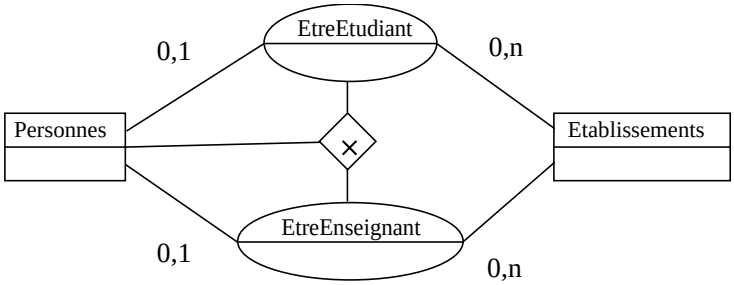


Chaque personne est soit étudiant dans un établissement, soit salarié dans une entreprise, soit les deux à la fois

Exclusion 

Toute occurrence du(des) type(s) d'entités correspond à au plus une des occurrences des types d'associations

Exemple :



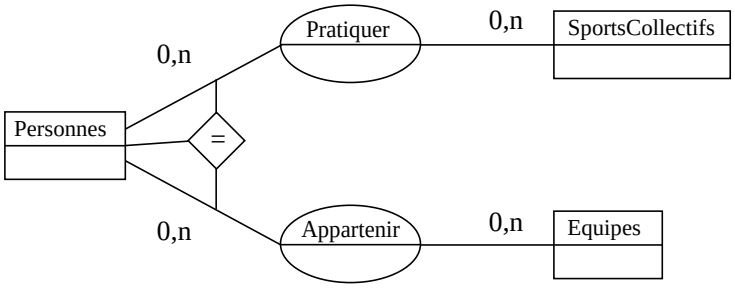
Une personne ne peut pas être à la fois étudiant et enseignant

Contraintes s'appliquant aux types d'associations (2/2)

Égalité 

Les occurrences des types d'associations sont exactement les mêmes relativement au(x) type(s) d'entité impliqué(s)

Exemple :

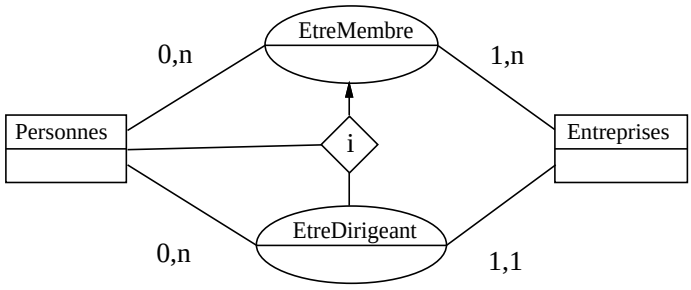


Une personne pratique un sport collectif si et seulement si elle appartient à une équipe

Inclusion 

Les occurrences du type d'associations pointé font toutes parties des occurrences du(des) autre(s) type(s) d'association relativement au(x) type(s) d'entité impliqué(s)

Exemple :

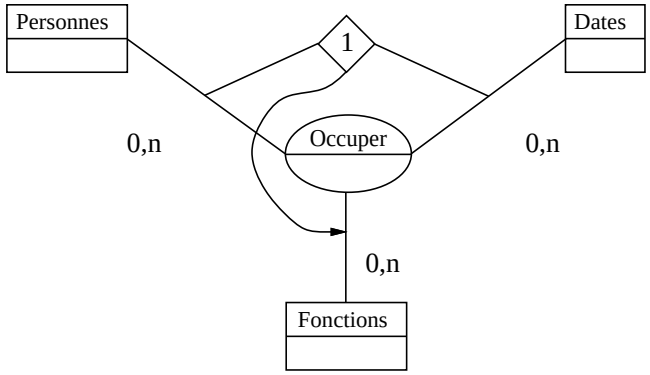


Le dirigeant d'une entreprise fait partie des membres de cette entreprise

Unicité 

Les occurrences du type d'associations pointé sont uniques relativement au(x) type(s) d'entité reliés par l(es) autre(s) lien(s)

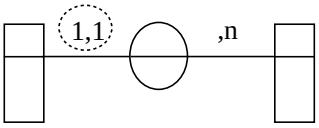
Exemple :



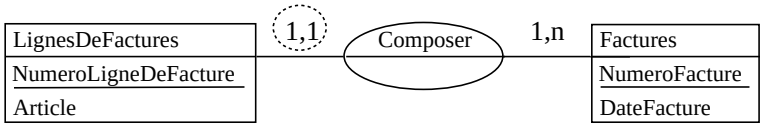
Une personne à une date donnée ne peut occuper qu'une et une seule fonction

Héritage d'identifiant

Un type d'entités a pour identifiant son propre identifiant composé avec ceux hérités des autres types d'entités reliés par un lien identifiant



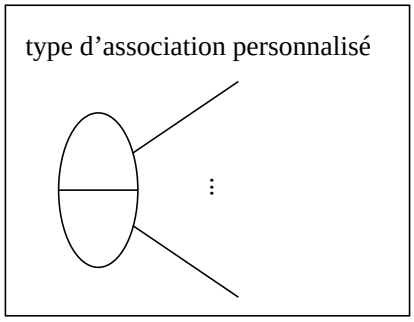
Exemple :



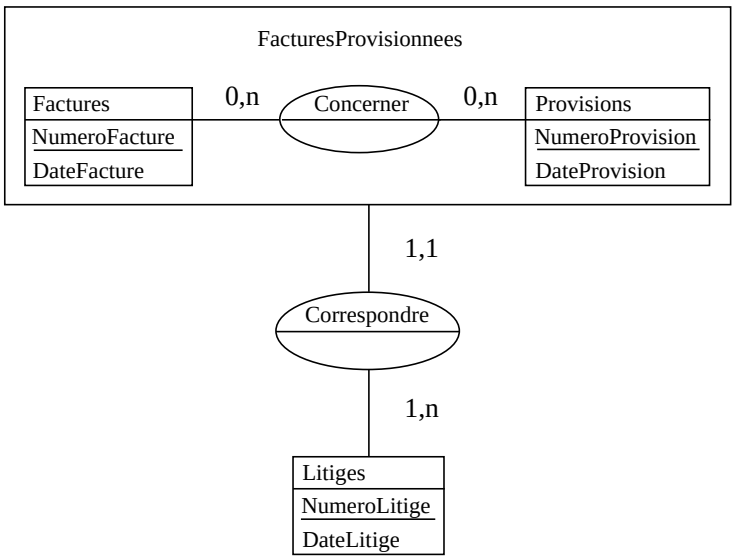
Ainsi, l'identifiant du type d'entités LignesDeFactures est le couple d'attributs (NumeroFacture , NumeroLigneDeFacture)

Type d'associations personnalisé

Regroupement d'un sous-schéma (de types d'entités et de types d'associations reliés) en l'équivalent d'un type d'entités



Exemple :



Un litige correspond à des factures provisionnées

Extensions (7/7)

Contraintes s'appliquant aux types de propriétés

Intervalle de valeurs d'un type de propriétés

Exemple : plus de 1901

Énumération des valeurs d'un type de propriétés

Exemple : roi, dame, tour, cavalier, fou, pion

Contraintes entre valeurs de plusieurs types de propriétés

Exemple : date de retour d'un questionnaire postérieure ou égale à sa date d'envoi

Format d'un type de propriétés

Exemple : le code produit est composé de quatre caractères, les trois premiers sont des lettres mnémoniques et le dernier est un chiffre séquentiel

...

Type de propriétés agrégé

Un type de propriétés agrégé est composé à partir d'autres types de propriétés

Exemple : le type de propriétés Adresse est en fait une agrégation des types de propriétés Numéro, Voie, CodePostal, Ville

Type de propriétés multi-valué

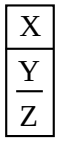
Pour une occurrence du type d'entités ou du type d'associations, la valeur du type de propriétés est une liste de valeurs appartenant chacune au même domaine

Exemple : les prénoms de chaque étudiant

Entité-association ⇝ relationnel (1/7)

(passage d'un schéma entité-association à un schéma relationnel)

Transformation de tous les types d'entités en relations

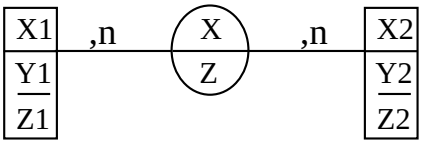


devient

X (Y , Z)
Y est la clé de la relation X

Le type d'entités X devient la relation X, les types de propriétés identifiants Y deviennent les attributs de la clé Y, les types de propriétés hors identifiants Z deviennent les attributs hors clé Z

Transformation de tous les types d'associations de liens $n : n$ en relations

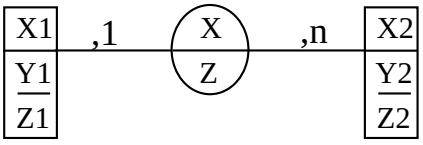


devient

`X1 ( Y1 , Z1 )`
`X2 ( Y2 , Z2 )`
X (Y1 , Y2 , Z)
`Y1 est la clé de la relation X1`
`Y2 est la clé de la relation X2`
(Y1 , Y2) est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2

Le type d'associations X devient la nouvelle relation X dont la clé (X.Y1,X.Y2) est composée des types de propriétés identifiants Y1 et Y2 des types d'entités associés X1 et X2 respectivement, et qui récupère comme attributs hors clé X.Z les types de propriétés Z du type d'associations X ; de plus, des contraintes d'intégrité référentielles sont à considérer pour chaque attribut X.Y1 et X.Y2 composant la clé de la nouvelle relation X relativement aux clés des relations X1.Y1 et X2.Y2 issues des types d'entités X1 et X2

Transformation de tous les types d'associations de liens $1 : n$ en attributs supplémentaires



devient

`X1 ( Y1 , Z1 , Y2 , Z )`
`X2 ( Y2 , Z2 )`
`Y1 est la clé de la relation X1`
`Y2 est la clé de la relation X2`
X1.Y2 référence X2.Y2

Les types de propriétés identifiants Y2 du type d'entités X2 impliqué dans le lien $,n$ constituent une clé étrangère X1.Y2 pour la relation X1 issue du type d'entités X1 impliqué dans le lien ,1, et engendre donc une contrainte d'intégrité référentielle puisque la clé étrangère X1.Y2 référence la clé primaire X2.Y2 ; les types de propriétés Z du type d'associations X suivent le mouvement de la clé étrangère et forment les attributs X1.Z

Légende : cet algorithme procède en deux passes, l'une pour les types d'entités et l'autre pour les types d'associations ; pour les transformations des types d'associations, le résultat obtenu par la première passe est ainsi écrit : `transformations des types d'entités`

Notation : A.B référence C.D signifie que l'ensemble des valeurs que peut prendre l'attribut B de la relation A est inclus dans ou égal à l'ensemble des valeurs que prend l'attribut D de la relation C, et ceci à tout moment : il s'agit d'une contrainte d'intégrité référentielle à respecter

Les contraintes d'intégrité suivantes s'appliquent :

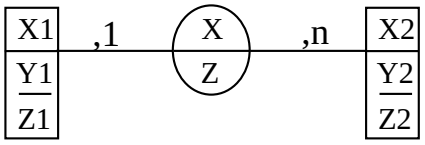
Chaque clé (primaire), éventuellement composée de plusieurs attributs, est unique

Chaque attribut composant la clé (primaire) est obligatoire

Chaque attribut composant la clé étrangère est obligatoire si le lien correspondant admet la valeur 1 pour sa cardinalité minimale

Exercice : on peut retrouver la règle de transformation des types d'associations de liens $1 : n$ à partir de celle des types d'associations de liens $n : n$ (sans oublier la contrainte d'unicité supplémentaire)

À partir de



On obtient tout d'abord

X1 (Y1 , Z1)
X2 (Y2 , Z2)
X (Y1 , Y2 , Z)
Y1 est la clé de la relation X1
Y2 est la clé de la relation X2
(Y1 , Y2) est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2
unicité de X.Y1

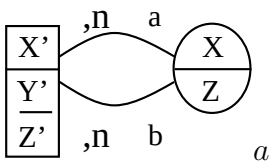
On simplifie les deux contraintes d'unicité car l'unicité de X.Y1 $\Longrightarrow$ l'unicité de (X.Y1 , X.Y2)

X1 (Y1 , Z1)
X2 (Y2 , Z2)
X (Y1 , Y2 , Z)
Y1 est la clé de la relation X1
Y2 est la clé de la relation X2
Y1 est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2

On fusionne X1 et X qui ont exactement même la clé Y1

X1 (Y1 , Z1 , Y2 , Z)
X2 (Y2 , Z2)
Y1 est la clé de la relation X1
Y2 est la clé de la relation X2
X1.Y2 référence X2.Y2

Type d'associations binaire réflexive de liens $n : n$



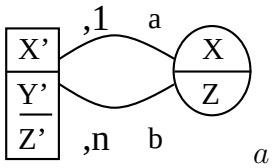
a et b sont des rôles

devient

```
X' ( Y' , Z' ) de clé Y'
```
X (Y'a , Y'b , Z)
(Y'a , Y'b) est la clé de la relation X
X.Y'a référence X'.Y'
X.Y'b référence X'.Y'

Remarque : une telle association décrit un graphe orienté (connexe ou non, mais ayant au plus une fois le même arc)

Type d'associations binaire réflexive de liens $1 : n$



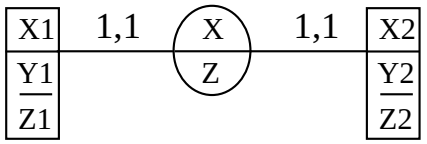
a et b sont des rôles

devient

```
X' ( Y' , Z' , Y'a , Z )
Y' est la clé de la relation X'
```
X'.Y'a référence X'.Y'

Remarque : une telle association décrit une endofonction i. e. un ensemble de composants, chaque composant étant soit un seul élément soit un cycle (d'un ou plusieurs) élément(s), chaque élément (seul ou du cycle) étant la racine d'un arbre (éventuellement vide)

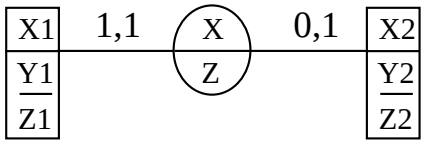
Type d'associations binaire de liens $1,1 : 1,1$



devient

X1 (Y1 , Z1 , Y2 , Z2 , Z)
Y1 est la clé de la relation X1
unicité de X1.Y2
pas d'indétermination pour X1.Y2

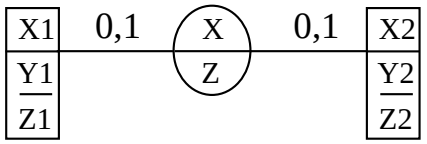
Type d'associations binaire de liens $1,1 : 0,1$



devient

```
X1 ( Y1 , Z1 , Y2 , Z )
X2 ( Y2 , Z2 )
Y1 est la clé de la relation X1
Y2 est la clé de la relation X2
```
X1.Y2 référence X2.Y2
unicité de X1.Y2
pas d'indétermination pour X1.Y2

Type d'associations binaire de liens $0,1 : 0,1$



devient

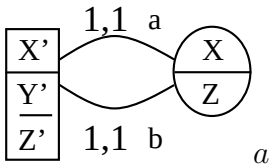
```
X1 ( Y1 , Z1 )
X2 ( Y2 , Z2 )
X ( Y1 , Y2 , Z )
Y1 est la clé de la relation X1
Y2 est la clé de la relation X2
```
Y1 est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2
unicité de X.Y2
pas d'indétermination pour X.Y2

ou

```
X1 ( Y1 , Z1 , Y2 , Z )
X2 ( Y2 , Z2 )
Y1 est la clé de la relation X1
Y2 est la clé de la relation X2
```
X1.Y2 référence X2.Y2
unicité de X1.Y2
indétermination pour X1.Y2 si et seulement si indétermination pour X1.Z

N. B. : la seconde solution n'est à retenir que si le nombre d'entités de X1 non liées à des entités de X2 par l'association X est très faible

Type d'associations binaire réflexive de liens $1,1 : 1,1$



a et b sont des rôles

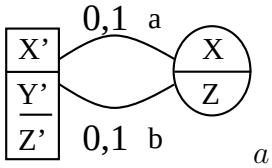
devient

```
X' ( Y' , Z' , Y'ab , Z )
Y' est la clé de la relation X'
```
X'.Y'ab référence X'.Y'
unicité de X'.Y'ab
pas d'indétermination pour X'.Y'ab

Remarque : une telle association décrit un ensemble de cycles (i. e. une bijection)

Remarque : Les types d'associations binaires réflexives de liens $1,1 : 0,1$ ne peuvent pas satisfaire simultanément les contraintes sur les cardinalités

Type d'associations binaire réflexive de liens $0,1 : 0,1$ (une seule solution est proposée)



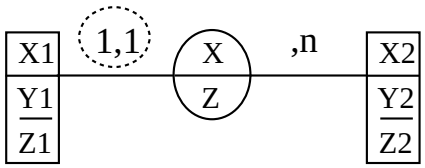
a et b sont des rôles

devient

```
X' ( Y' , Z' )
```
X (Y'a , Y'b , Z)
```
Y' est la clé de la relation X'
```
Y'a est la clé de la relation X
X.Y'a référence X'.Y'
X.Y'b référence X'.Y'
unicité de X.Y'b
pas d'indétermination pour X.Y'b

Remarque : une telle association décrit un ensemble de composants, chaque composant étant soit un sommet isolé, soit une chaîne (d'au moins un arc), soit un cycle (d'au moins un arc)

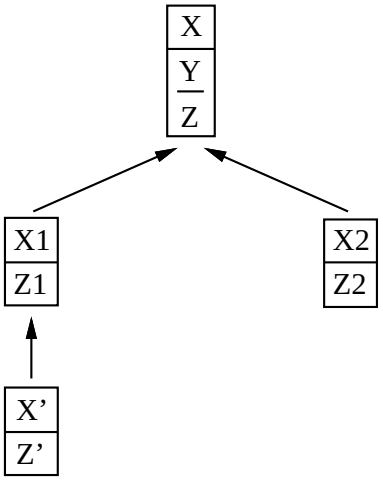
Type d'associations binaire avec un lien identifiant



devient

`X1 ( Y1 , Y2 , Z1 , Z )`
`X2 ( Y2 , Z2 )`
(Y1 , Y2) est la clé de la relation X1
`Y2 est la clé de la relation X2`
X1.Y2 référence X2.Y2

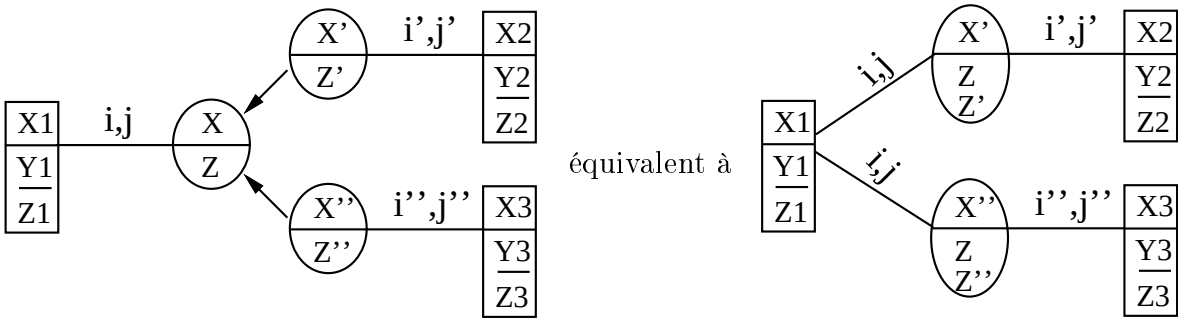
Sous-type d'entités



devient

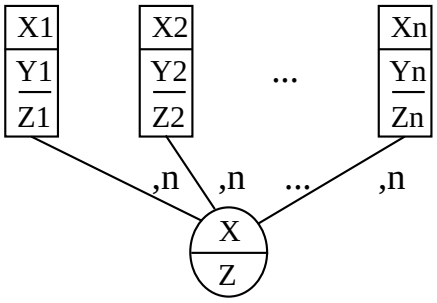
`X ( Y , Z )`
X1 (Y , Z1)
X2 (Y , Z2)
X' (Y , Z')
`Y est la clé de la relation X`
Y est la clé de la relation X1
Y est la clé de la relation X2
Y est la clé de la relation X'
X1.Y référence X.Y
X2.Y référence X.Y
X'.Y référence X1.Y

Sous-type d'associations



Discutable : à confronter avec Win'Design ou Power*AMC par exemple

Type d'associations n-aire (avec $n > 2$) de liens $n : n : \ldots : n$



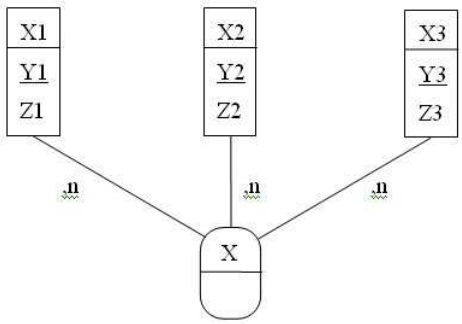
devient

```
X1 ( Y1 , Z1 ) de clé Y1
X2 ( Y2 , Z2 ) de clé Y2
⋮
Xn ( Yn , Zn ) de clé Yn
```

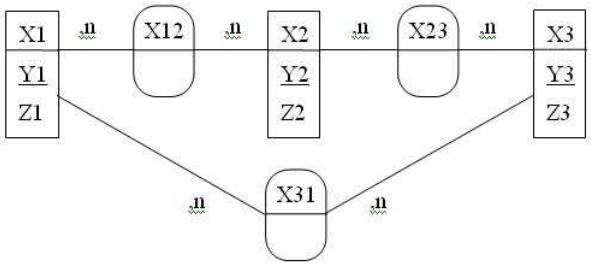
X (Y1 , Y2 , ..., Yn , Z)
(Y1 , Y2 , ..., Yn) est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2
⋮
X.Yn référence Xn.Yn

Une association ternaire *vs* trois associations binaires

Une association ternaire ne peut pas être remplacée par trois associations binaires où toute occurrence (a,b,c) de X serait substituée par trois occurrences (a,b), (b,c) et (c,a) dans X12, X23 et X31 respectivement

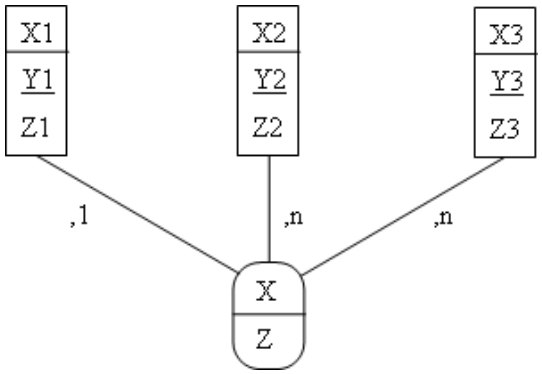


$\neq$



En effet, par exemple, pour X = $\{(1,\alpha,+),(1,\beta,-),(2,\alpha,-)\}$ comme pour X = $\{(1,\alpha,+),(1,\beta,-),(2,\alpha,-),(1,\alpha,-)\}$, on obtient X12 = $\{(1,\alpha),(1,\beta),(2,\alpha)\}$, X23 = $\{(\alpha,+),(\beta,-),(\alpha,-)\}$ et X31 = $\{(+,1),(-,1),(-,2)\}$

Type d'associations ternaire de liens $1 : n : n$



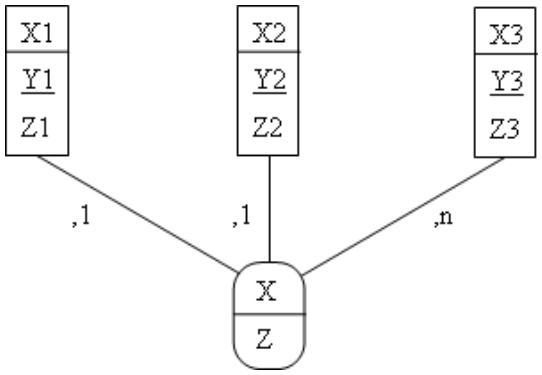
devient

```
X1 ( Y1 , Z1 ) de clé Y1
X2 ( Y2 , Z2 ) de clé Y2
X3 ( Y3 , Z3 ) de clé Y3
```

X (Y1 , Y2 , Y3 , Z)
Y1 est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2
X.Y3 référence X3.Y3

Remarque : on pourrait fusionner X1 et X dans une même relation

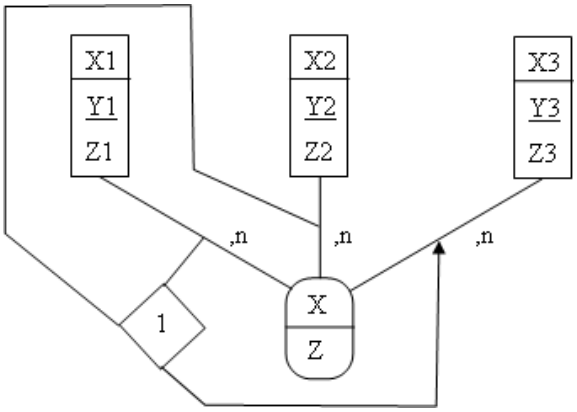
Type d'associations ternaire de liens $1:1:n$



devient

```
X1 ( Y1 , Z1 ) de clé Y1
X2 ( Y2 , Z2 ) de clé Y2
X3 ( Y3 , Z3 ) de clé Y3
```
X (Y1 , Y2 , Y3 , Z)
Y1 est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2
X.Y3 référence X3.Y3
unicité de X.Y2

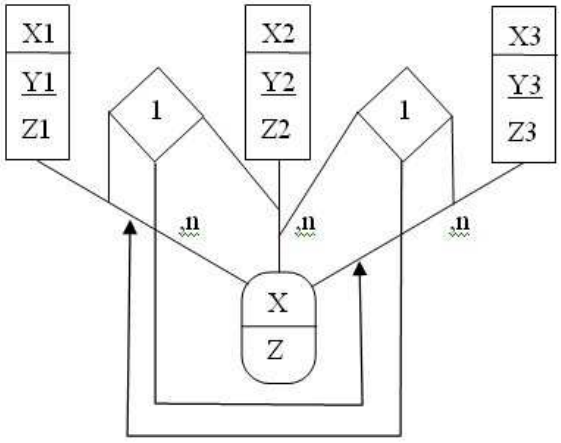
Type d'associations ternaire de liens $n:n:n$ avec une contrainte d'unicité



devient

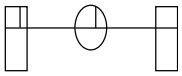
```
X1 ( Y1 , Z1 ) de clé Y1
X2 ( Y2 , Z2 ) de clé Y2
X3 ( Y3 , Z3 ) de clé Y3
```
X (Y1 , Y2 , Y3 , Z)
(Y1 , Y2) est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2
X.Y3 référence X3.Y3

Type d'associations ternaire de liens $n:n:n$ avec deux contraintes d'unicité



devient

```
X1 ( Y1 , Z1 ) de clé Y1
X2 ( Y2 , Z2 ) de clé Y2
X3 ( Y3 , Z3 ) de clé Y3
```
X (Y1 , Y2 , Y3 , Z)
(Y1 , Y2) est la clé de la relation X
X.Y1 référence X1.Y1
X.Y2 référence X2.Y2
X.Y3 référence X3.Y3
unicité de (X.Y2 , X.Y3)



Modèles voisins

Modèle entité-association

Modèle logique des données (MLD)

Modèle relationnel

Modèle navigationnel

Modèle objet

Diagramme de classes

Objectif

Représentation des données conceptuelles et organisationnelles

Remarque

Modèle organisationnel des données de la méthode MERISE/2

Composants

1 MOD global

Différentes vues du MOD pour un type de site ou un type de poste

Formalisme

Reprend celui du modèle entité-association étendu, complété de contraintes organisationnelles

Entité - Type d'entités - Association - Type d'associations - Lien - Cardinalités - Propriété - Type de propriétés - Identifiant

(cf. le modèle entité-association)

Type d'acteur

Regroupement d'acteurs exerçant des activités identiques

Type de site

Regroupement géographique et/ou fonctionnel de types d'acteurs

Type de poste

Rapprochement entre un type d'acteur et un type de site

Nature

Précisions sur la répartition homme/machine et sur le support d'historisation

Informatisé (), manuel (`M`), historisation sur support informatique (`H`), historisation sur support manuel (`HM`)

Type d'accès

Restriction sur un type d'entités, un type d'associations ou un type de propriétés

Création (`C`), interrogation (`I`), modification (`M`), suppression (`S`)

Restriction d'accès

Un type de site/poste n'a accès qu'à un sous-ensemble de types de propriétés (de type d'entités ou de type d'associations)

Un type de site/poste n'a accès qu'à un sous-ensemble d'occurrences (de type d'entités ou de type d'associations)

Un type de site/poste n'est autorisé à effectuer que certains types d'accès (sur un type d'entités, un type d'associations ou un type de propriétés)

Système d'autorisation d'utilisation des données

Représente la vue sur les données de chaque type de site et de chaque type de poste

Vue de type d'entités ou de type d'associations

Représentation de l'ensemble des données du type d'entités ou du type d'associations qu'un type de site ou qu'un type de poste peut manipuler, suivant un type d'accès

Il se restreint à un sous-ensemble de types de propriétés ou d'occurrences

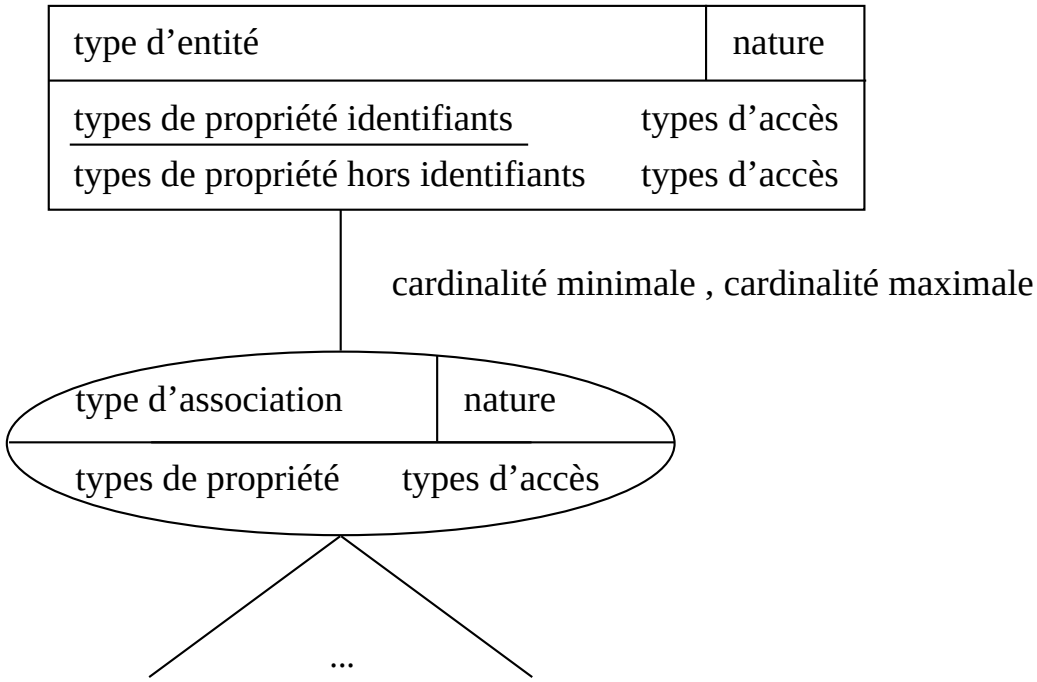
Groupe de données

Partition homogène de la visibilité d'un type de site ou d'un type de poste sur un ensemble de données

Données privées (mises à jour par le type de site et inaccessibles aux autres types de sites), protégées (mises à jour par le type de site et consultables par d'autres types de sites), partagées (mises à jour par plusieurs types de sites et consultables par plusieurs types de sites), consultables (pour le type de site)

Les données privées d'un type de site peuvent être privées ou protégées ou partagées pour ses types d'acteurs, les données protégées d'un type de site peuvent être protégées ou partagées pour ses types d'acteurs, et les données partagées d'un type de site sont partagées pour ses types d'acteurs

Représentation graphique du MOD (global)



Représentation tabulaire d'une vue de type d'entités ou de type d'associations par un type de site ou un type de poste

Vue du type de site/poste :				
type d'entités / type d'associations	Vues			types d'accès
	nom de la vue	prédicat de sélection	type de propriétés	

Modèles voisins

Modèle entité-association

Modèle organisationnel des données (MOD)

Modèle relationnel

Modèle navigationnel

Modèle objet

Diagramme de classes

Objectif

Représentation des données à mémoriser sur des supports informatiques permanents

Remarque

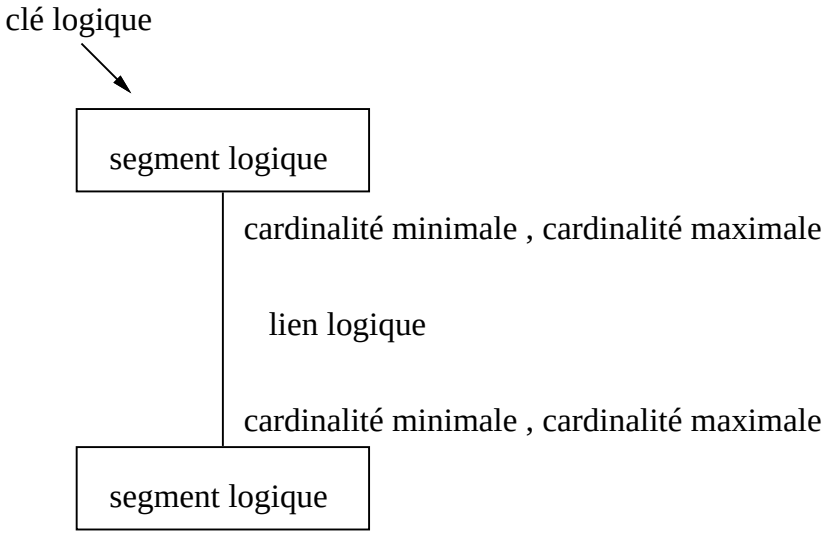
Modèle logique des données de la méthode MERISE/2

Formalismes

Formalisme générique dans un contexte hétérogène

Formalisme spécifique au SGBD cible (modèle relationnel pour un SGBDR, etc.)

Représentation graphique (formalisme générique) - Définitions



Segment logique

Ensemble de données élémentaires

Lien logique

Permet de retrouver les segments logiques qu'il relie

Data-item

Plus petit élément logique d'information

Data-type

Caractéristiques d'un *data-item* (format, nature, etc.)

Structure de données

Structure complexe de *data-items*

Clé logique

Data-item ou ensemble de *data-items* permettant de retrouver une occurrence de segment logique

MLD de la méthode MERISE/2 précisant l'implantation logique des données permanentes sur chacune des machines logiques types du système

Formalisme générique (du MLD global), puis dans ceux spécifiques aux SGBD cibles

Composé de plusieurs MLD locaux (i. e. MLD propres à chacune des machines logiques types)

Définitions

Machine logique type

Ensemble de ressources matérielles et logicielles non séparables permettant d'effectuer des traitements et de stocker des données persistantes

Segment logique réparti

Segment logique (du MLD global) utilisé par des machines logiques types et stocké en tant que segment de référence ou cliché

Segment logique de type référence : les mises à jour ne se font que sur la machine logique type de référence le gérant

Segment logique de type cliché : les mises à jour se font soit périodiquement sur les machines logiques types qui l'utilisent (sous contrôle de la machine logique type le gérant), soit à la demande des machines logiques types

Segment logique de type dossier : segment logique de type référence sur l'une des machines logiques types et de type cliché sur d'autres, à un instant donné

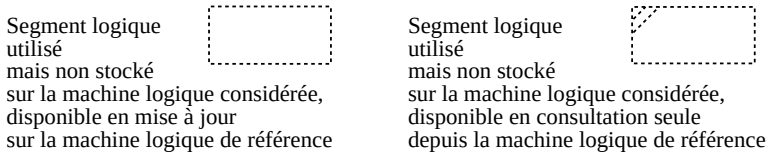
Lien logique réparti

Lien logique reliant deux segments logiques répartis

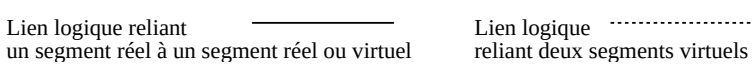
Représentation graphique (formalisme générique)

Segment logique stocké sur la machine logique type considérée (segment logique réel)

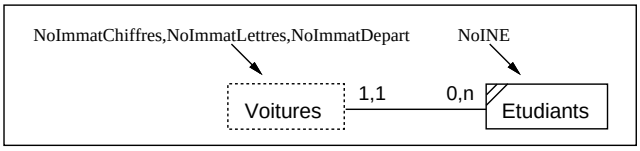
Segment logique utilisé mais non stocké sur la machine logique type considérée (segment logique virtuel)



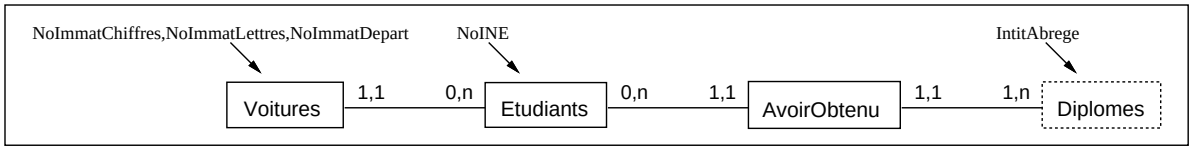
Lien logique



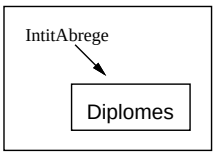
Machine logique type «Étudiants»



Machine logique type «Secrétariat»



Machine logique type «Rectorat»



Le segment logique `Diplomes` est tenu à jour (référence) par la machine logique type « Rectorat », lu par la machine logique type « Secrétariat » à chaque utilisation, ignoré de la machine logique type « Étudiants »

Le segment logique `Etudiants` est rafraîchi périodiquement par la machine logique type « Secrétariat » sur la machine logique type « Étudiants », ignoré de la machine logique type « Rectorat »

Autre dénomination

Modèle réseau

Modèles voisins

Modèle entité-association
Modèle organisationnel des données (MOD)
Modèle logique des données (MLD)
Modèle relationnel
Modèle objet
Diagramme de classes

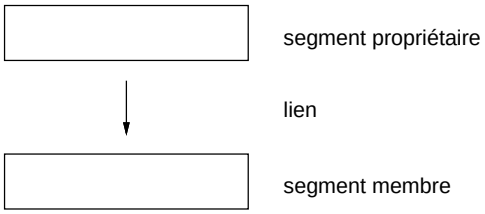
Origines

C. W. BACHMAN
CODASYL DBTG

Remarque

Un des modèles logiques des données de MERISE

Représentation graphique - Définitions



Segment ou article *(record)*

Collection nommée de plusieurs propriétés

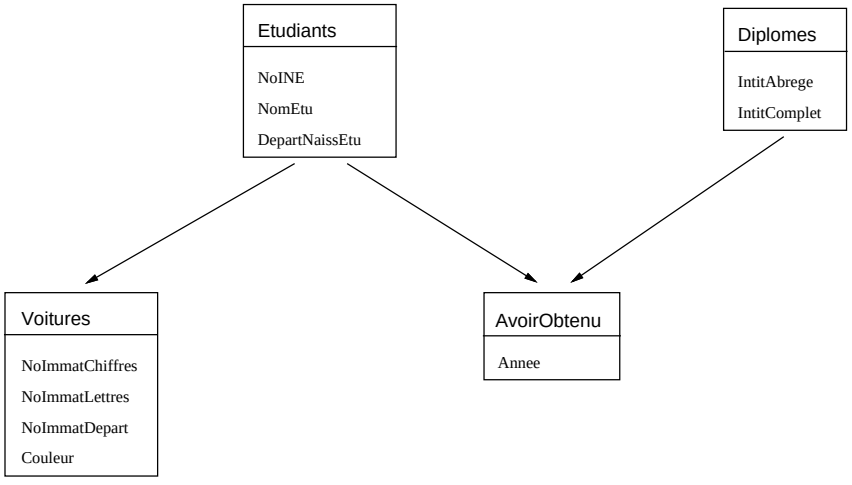
Lien *(set)*

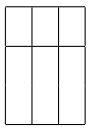
Relie un segment propriétaire *owner* et un segment membre *member*

Signification : une occurrence du segment propriétaire peut « posséder » plusieurs occurrences du segment membre

Remarque : tout segment peut être propriétaire de plusieurs segments et réciproquement tout segment peut être membre de plusieurs segments

Exemple « jouet »





Modèles voisins

Modèle entité-association

Modèle organisationnel des données (MOD)

Modèle logique des données (MLD)

Modèle navigationnel

Modèle objet

Diagramme de classes

Origine

1970 : E. F. CODD

Remarques

Un des modèles logiques des données de MERISE

Modèle assez formalisé

Ensemble : collection d'éléments (sans répétition ni ordre)

L'ensemble vide est noté $\{\}$ ou $\emptyset$

Exemple : les chiffres

Définition en extension

Exemple : $Chiffres = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Définition en compréhension

Exemple : $Chiffres = \{x \in \mathbb{N} : x < 10\}$

Produit cartésien de deux ensembles

Exemple :

Soit $A = \{a, b\}$

Soit $B = \{1, 2, 3\}$

Alors, $A \times B = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$ de cardinal $card(A).card(B)$

Partie d'un ensemble

Exemple :

Soit $A = \{a, b\}$

Soit $B = \{1, 2, 3\}$

Alors, $\{(a, 2), (b, 2), (b, 3)\}$ est l'une des $2^{card(A).card(B)}$ parties de $\mathcal{P}\ (A \times B)$

Union de deux ensembles

Exemple :

Soit $A = \{a, b\}$

Soit $B = \{b, c\}$

Alors, $A \cup B = \{a, b, c\}$

Intersection de deux ensembles

Exemple :

Soit $A = \{a, b\}$

Soit $B = \{b, c\}$

Alors, $A \cap B = \{b\}$

Différence de deux ensembles

Exemple :

Soit $A = \{a, b\}$

Soit $B = \{b, c\}$

Alors, $A \backslash B = \{a\}$

Différence symétrique de deux ensembles

Exemple :

Soit $A = \{a, b\}$

Soit $B = \{b, c\}$

Alors, $(A \backslash B) \cup (B \backslash A) = (A \cup B) \backslash (A \cap B) = \{a, c\}$

Définitions (1/2)

Domaine D : ensemble de valeurs

Exemples : chaînes de caractères de longueur au plus 25, entiers, $\{"Bonjour", e, i, \pi, 1, 0, \sqrt{2}, 'z'\}$

Attribut A (ou colonne) : nom de constituant (ou variable) prenant ses valeurs dans un domaine

Exemples : département de naissance d'un étudiant, NoINE d'un étudiant, couleur d'une voiture

Relation R (ou table) (définition en extension)

Partie (ou sous-ensemble) du produit cartésien $D_1 \times D_2 \times \cdots \times D_n$ sur les attributs $A_1, A_2, \ldots, A_n$ dont chaque élément $(d_1, d_2, \ldots, d_n)$ est un tuple i. e. n-uplet (ou ligne) avec $d_i \in D_i$ le domaine de l'attribut A_i pour tout $1 \leq i \leq n$

N. B. : une relation est donc simplement un ensemble de tuples

R

A_1	A_2	$\cdots$	A_n
d_1	d_2	$\cdots$	d_n
d'_1	d'_2	$\cdots$	d'_n
$\vdots$	$\vdots$	$\ddots$	$\vdots$
d''_1	d''_2	$\cdots$	d''_n

avec $\{d_i, d'_i, \ldots, d''_i\} \subseteq D_i, \forall\ 1 \leq i \leq n$

Chaque cellule, intersection d'une ligne (i. e. d'un tuple) et d'une colonne (i. e. d'un attribut), admet une et une seule valeur (en fait, cette condition est relâchée en au plus une valeur)

Degré d'une relation : nombre d'attributs

Cardinal d'une relation : nombre de tuples

Exemple :

Etudiants

NoINE	NomEtu	DepartNaissEtu
5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17

D_1 et D_3 : $\mathbb{N}^*$ (entiers positifs)
D_2 : chaînes de caractères

degré = 3
cardinal = 5

Schéma de Relation $R(A_1, A_2, \ldots, A_n)$ (définition en compréhension)

Liste des attributs d'une relation

Exemple : Etudiants (NoINE , NomEtu , DepartNaissEtu)

Remarque : la définition d'une relation (en terme ensembliste) implique que tous les tuples soient différents deux à deux ; les SGBDR vérifient cette obligation en choisissant systématiquement une clé (parmi les clés candidates) pour chaque relation

Définitions (2/2)

Clé (primaire ou principale)

Attribut ou groupe d'attributs minimal qui identifie de façon unique chaque tuple d'une relation

Exemples : numéro INE de la relation Etudiants, numéro d'immatriculation complet (chiffres, lettres et département) de la relation Voitures

Contre-exemple : nom des étudiants (homonymies)

Clé secondaire

Attribut ou groupe d'attributs minimal qui identifie de façon unique chaque tuple d'une relation, candidate non retenue pour être la clé primaire

Exemple : intitulé complet de la relation Diplomes

Clé étrangère

Attribut(s) d'une relation qui correspond à la clé (primaire) d'une autre (voire de la même) relation

Exemple : le NoINE de la relation Voitures (indiquant quel étudiant possède cette voiture)

Contrainte d'intégrité

Condition à vérifier par le SGBD pour toutes les relations (dépendance entre attributs, prédicat sur les valeurs d'un ou plusieurs attributs des tuples d'une relation, etc.)

Les contraintes d'intégrité peuvent être statiques (à vérifier tout le temps) ou dynamiques (à vérifier lors des changements d'état de la BD), et porter sur les tuples ou attributs ou relations ou inter-relations

Schéma [de la BD] relationnel[le]

Collection de schémas de relations et de contraintes d'intégrité, qui représentent l'univers réel du système d'information

Contraintes d'intégrité (1/2)

Contraintes d'intégrité statiques

Contrainte de clé (primaire) : une relation possède une clé primaire (éventuellement composée de plusieurs attributs) qui doit être unique (i. e. sans doublons)

> Exemples : l'attribut numéro INE pour la relation Etudiants, le triplet d'attributs du numéro d'immatriculation pour la relation Voitures

Contrainte de clé étrangère : l(es) attribut(s) d'une relation ont la même sémantique que la clé primaire correspondante d'une autre (voire de la même) relation

> Exemple : le numéro INE des relations Voitures et Etudiants

Contrainte d'intégrité référentielle : l'ensemble des valeurs d'une clé étrangère est inclus dans ou égal à l'ensemble des valeurs de la clé primaire correspondante

> Exemple : le numéro INE de la relation Voitures référence le numéro INE de la relation Etudiants ; en effet, valeurs(Voitures.NoINE) $= \{5, 4\} \subseteq$ valeurs(Etudiants.NoINE) $= \{5, 2, 4, 7, 3\}$

Contrainte d'unicité : les valeurs prises par un ou plusieurs attribut(s) sont uniques (i. e. sans doublons i. e. toutes différentes deux à deux)

> N. B. : cette contrainte s'applique notamment systématiquement aux clés primaires et aux clés secondaires
>
> Exemples : le numéro INE de la relation Etudiants, l'intitulé complet de la relation Diplomes

Contrainte existentielle : les valeurs d'un attribut sont définies pour tous les tuples de la relation (c.-à-d. que l'attribut est obligatoire i. e. non facultatif)

> Exemple : valeur d'indétermination (`NULL`) interdite pour le nom de l'étudiant (qui est donc un attribut obligatoire)

Contrainte de domaine : précision sur le domaine des valeurs d'un attribut (liste ou intervalle de valeurs, formats, etc.)

> Exemples : la couleur d'une voiture doit être jaune ou orange ou rouge, un département (de naissance d'un étudiant ou d'immatriculation de voiture) doit être compris entre 1 et 97

Contrainte de valeur par défaut : valeur à affecter à un attribut lors de l'insertion d'un tuple (si l'instruction correspondante n'affecte pas de valeur !)

> Exemples : année courante pour l'année d'obtention d'un diplôme, 33 pour le département

Contrainte sur un tuple : prédicat (ou formule) basé sur plusieurs attributs

> Exemple : le prix de vente d'un produit doit être supérieur à son prix d'achat

Contrainte de dépendance fonctionnelle : vérification des dépendances fonctionnelles entre groupes d'attributs

Contrainte de normalité : vérification du respect d'une forme normale

Contrainte de cardinalité : borne pour le nombre de tuples d'une relation

> Exemple : un étudiant ne peut pas avoir obtenu plus de cinq diplômes

. . . et toute autre contrainte statique spécifique

Contraintes d'intégrité (2/2)

Contraintes d'intégrité dynamiques

Contrainte de type pré-post condition : condition vérifiée par la BD lors de son évolution, sous forme d'une condition explicite avant, pendant ou après une opération de mise à jour

Exemples : ne supprimer un diplôme (les intitulés) que si aucun étudiant ne l'a obtenu, n'insérer une voiture que si l'étudiant qui la possède est déjà enregistré

Règle active : action(s) à exécuter suite à un changement d'état de la BD

Exemple : dès qu'un étudiant est supprimé, supprimer les voitures qu'il possède et les diplômes qu'il a obtenus

. . . et toute autre contrainte dynamique spécifique

Dépendance fonctionnelle

Définition

Soient X et Y deux ensembles d'attributs. On dit que X détermine Y (ou Y dépend fonctionnellement de X) noté X → Y lorsque deux tuples qui ont une même valeur sur les attributs de X implique qu'ils ont même valeur sur les attributs de Y (ou encore qu'à toute valeur de X n'est associée qu'une et une seule valeur de Y, dans toute extension de la relation) ; plus formellement, soient $x', x'' \in$ X et $y', y'' \in$ Y, alors x' → y' ∧ x" → y" ∧ $x' = x'' \Longrightarrow y' = y''$

Exemples (extraits de l'exemple « jouet »)

IntitAbrege → IntitComplet

IntitComplet → IntitAbrege

NoINE → NomEtu,DepartNaissEtu

NoINE,IntitAbrege → Annee

NoINE,IntitComplet → Annee

NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur,NoINE,NomEtu,DepartNaissEtu

NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitAbrege → Annee

NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitComplet → Annee

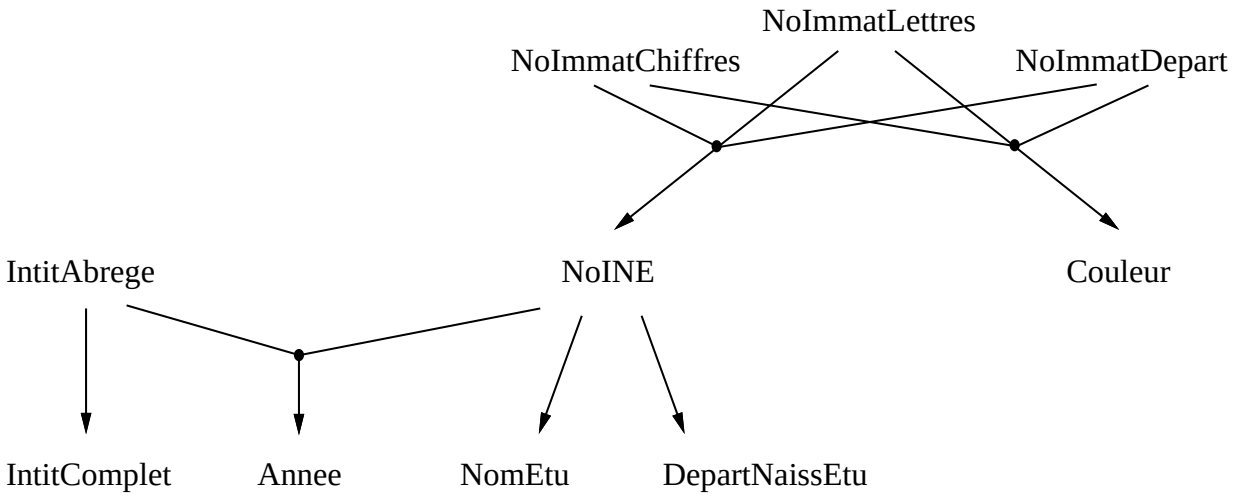
Contre-exemple

NomEtu ↛ NoINE à cause des homonymes (cf. LEROI)

Hypergraphe des dépendances fonctionnelles

Graphe orienté dans lequel chaque sommet correspond à un attribut et chaque hyper-arc (arc ayant plusieurs sources) correspond à une dépendance fonctionnelle canonique

Exemple :



pour les seules dépendances fonctionnelles canoniques suivantes :

IntitAbrege → IntitComplet

NoINE → NomEtu

NoINE → DepartNaissEtu

NoINE,IntitAbrege → Annee

NoImmatChiffres,NoImmatLettres,NoImmatDepart → NoINE

NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur

Propriétés des dépendances fonctionnelles (1/2)

Réflexivité

X → X' avec X' ⊆ X

Exemple : NoINE,IntitAbrege → NoINE

Augmentation

$X \to Y \Longrightarrow X,Z \to Y$

Exemple : NoINE → NomEtu ⟹ NoINE,IntitAbrege → NomEtu

Transitivité

$$\left\{ \begin{array}{l} X \to Y \\ Y \to Z \end{array} \right. \Longrightarrow X \to Z$$

Exemple : $\left\{ \begin{array}{l} \text{NoImmatChiffres,NoImmatLettres,NoImmatDepart} \to \text{NoINE} \\ \text{NoINE} \to \text{NomEtu} \end{array} \right. \Longrightarrow$ NoImmatChiffres,NoImmatLettres,NoImmatDepart → NomEtu

Décomposition

$X \to Y,Z \Longrightarrow X \to Y$

Exemple : NoINE → NomEtu,DepartNaissEtu ⟹ NoINE → NomEtu

Éclatement (ou projection)

$$X \to Y,Z \Longrightarrow \left\{ \begin{array}{l} X \to Y \\ X \to Z \end{array} \right.$$

Exemple : NoINE → NomEtu,DepartNaissEtu $\Longrightarrow \left\{ \begin{array}{l} \text{NoINE} \to \text{NomEtu} \\ \text{NoINE} \to \text{DepartNaissEtu} \end{array} \right.$

Union (ou fusion ou additivité)

$$\left\{ \begin{array}{l} X \to Y \\ X \to Z \end{array} \right. \Longrightarrow X \to Y,Z$$

Exemple : $\left\{ \begin{array}{l} \text{NoINE} \to \text{NomEtu} \\ \text{NoINE} \to \text{DepartNaissEtu} \end{array} \right. \Longrightarrow$ NoINE → NomEtu,DepartNaissEtu

Pseudo-transitivité

$$\left\{ \begin{array}{l} X \to Y \\ Y,W \to Z \end{array} \right. \Longrightarrow X,W \to Z$$

Exemple : $\left\{ \begin{array}{l} \text{NoImmatChiffres,NoImmatLettres,NoImmatDepart} \to \text{NoINE} \\ \text{NoINE,IntitAbrege} \to \text{Annee} \end{array} \right. \Longrightarrow$ NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitAbrege → Annee

Non-dépendance

$Y \not\to X'$ et $X' \subseteq X \Longrightarrow Y \not\to X$

Exemple : NomEtu $\not\to$ NoINE et NoINE ⊆ {NoINE,IntitAbrege} ⟹ NomEtu $\not\to$ NoINE,IntitAbrege

Théorème

Toutes les dépendances fonctionnelles peuvent être générées à partir d'un ensemble d'entre elles par les propriétés de réflexivité, augmentation et transitivité (axiomes d'AMSTRONG)

Propriétés des dépendances fonctionnelles (2/2)

Remarque

La relation sur les dépendances fonctionnelles est réflexive et transitive mais n'est ni antisymétrique (par exemple, IntitAbrege $\rightarrow$ IntitComplet et IntitComplet $\rightarrow$ IntitAbrege mais IntitAbrege $\neq$ IntitComplet) ni symétrique (par exemple, NoINE $\rightarrow$ NomEtu mais NomEtu $\not\rightarrow$ NoINE) ; ce n'est donc ni une relation d'ordre ni une relation d'équivalence

Typologie des dépendances fonctionnelles X $\rightarrow$ Y

Canonique

card(Y)=1 i. e. Y est réduit à un seul attribut

Exemple : NoINE,IntitAbrege $\rightarrow$ Annee

Contre-exemple : NoINE $\rightarrow$ NomEtu,DepartNaissEtu

Triviale

$Y \subseteq X$

Exemple : NoINE,IntitAbrege $\rightarrow$ NoINE

Contre-exemple : NoINE $\rightarrow$ NomEtu

Élémentaire

$\nexists$ X' $\subsetneq$ X / X' $\rightarrow$ Y

Exemple : NoINE,IntitAbrege $\rightarrow$ Annee

Contre-exemple : NoImmatChiffres,NoImmatLettres,NoImmatDepart,NoINE $\rightarrow$ NomEtu ; en effet, NoINE $\rightarrow$ NomEtu

Directe

$\nexists$ Z / X $\rightarrow$ Z $\wedge$ Z $\rightarrow$ Y $\wedge$ Z $\not\rightarrow$ X

Exemple : NoINE $\rightarrow$ NomEtu

Contre-exemple : NoImmatChiffres,NoImmatLettres,NoImmatDepart $\rightarrow$ NomEtu ; en effet, NoImmatChiffres,NoImmatLettres,NoImmatDepart $\rightarrow$ NoINE et NoINE $\rightarrow$ NomEtu et NoINE $\not\rightarrow$ NoImmatChiffres,NoImmatLettres,NoImmatDepart

Fermeture transitive et couverture minimale (1/7)

Fermeture transitive

La fermeture transitive $\mathcal{F}^+$ d'un ensemble de dépendances fonctionnelles élémentaires $\mathcal{F}$ s'obtient en appliquant systématiquement (tant que cela est possible) la propriété de transitivité

Remarque : deux ensembles de dépendances fonctionnelles élémentaires sont équivalents s'ils ont la même fermeture transitive

Couverture minimale

Une couverture minimale $\mathcal{C}$ d'un ensemble de dépendances fonctionnelles élémentaires $\mathcal{F}$ est telle qu'aucune dépendance fonctionnelle n'est redondante (pour toute dépendance fonctionnelle f, $\mathcal{F} \setminus \{f\} \neq \mathcal{F}$) et que toute dépendance fonctionnelle est dans la fermeture transitive de $\mathcal{F}$ ($\mathcal{F}^+$)

Remarque : il n'y a pas forcément unicité de la couverture minimale d'un ensemble de dépendances fonctionnelles élémentaires (cf. les deux premiers exemples de couvertures minimales différentes données par l'algorithme CouvMinDF() pour le même ensemble de dépendances fonctionnelles initial, l'ordre étant simplement modifié, ce qui change l'ordre d'exécution de l'algorithme)

Algorithme du calcul de la fermeture transitive d'un ensemble d'attributs

Fonction FermTransAttr($\mathcal{F}$,$\mathcal{A}$) : $\mathcal{A}^+$
// [E/] $\mathcal{F}$: un ensemble de dépendances fonctionnelles
// [E/] $\mathcal{A}$: un ensemble quelconque d'attributs
// [S/] $\mathcal{A}^+$: la fermeture transitive de $\mathcal{A}$ sous $\mathcal{F}$
$\mathcal{A}^+ \longleftarrow \mathcal{A}$
Répéter
    $\mathcal{A}^+$tmp $\longleftarrow \mathcal{A}^+$
    Pour chaque X $\rightarrow$ Y $\in \mathcal{F}$ faire
        Si X $\subseteq \mathcal{A}^+$ Alors
            $\mathcal{A}^+ \longleftarrow \mathcal{A}^+ \cup$ Y
Jusqu'à $\mathcal{A}^+ = \mathcal{A}^+$tmp
Retourner $\mathcal{A}^+$

Exemples de FermTransAttr()

Soit $\mathcal{F}$ = { c $\rightarrow$ a
c,e $\rightarrow$ b,d
a,c,d $\rightarrow$ b
d $\rightarrow$ e
b,c $\rightarrow$ d }

FermTransAttr($\mathcal{F}$,{}) = {}
FermTransAttr($\mathcal{F}$,{b,e}) = {b,e}
FermTransAttr($\mathcal{F}$,{c}) = {a,c}
FermTransAttr($\mathcal{F}$,{a,d}) = {a,d,e}
FermTransAttr($\mathcal{F}$,{c,d}) = {a,b,c,d,e}

Fermeture transitive et couverture minimale (2/7)

Algorithme du calcul d'une couverture minimale d'un ensemble de dépendances fonctionnelles

Fonction CouvMinDF($\mathcal{F}$) : $\mathcal{C}$
// [E/] $\mathcal{F}$: un ensemble de dépendances fonctionnelles
// [S/] $\mathcal{C}$: une couverture minimale de $\mathcal{F}$ i. e. vérifiant que :
// $\mathcal{C}^+ = \mathcal{F}^+$
// $\forall$ X $\rightarrow$ Y $\in \mathcal{C}$, card(Y)=1
// $\nexists$ f $\in \mathcal{C}$ / $\mathcal{C} \setminus$ {f} $\models$ f
// $\nexists$ X',X" $\rightarrow$ Y $\in \mathcal{C}$ / $\mathcal{C} \models$ X" $\rightarrow$ Y avec X'$\neq \emptyset \wedge$ X"$\neq \emptyset$
// Notation : $\models$ signifie
// "implique (en utilisant les propriétés des dépendances fonctionnelles)"
// N. B. : les étapes 2 et 3 peuvent être permutées
// 1. décomposition des membres droits des dépendances fonctionnelles
// en dépendances fonctionnelles canoniques
$\mathcal{C} \longleftarrow \emptyset$
Pour chaque X $\rightarrow$ Y_1 , ..., $Y_n \in \mathcal{F}$ tq card(Y_i)=1 $\forall 1 \leq i \leq n$ faire
  $\mathcal{C} \longleftarrow \mathcal{C} \cup$ { X $\rightarrow$ Y_1 , ..., X $\rightarrow$ Y_n }
// 2. élimination des dépendances fonctionnelles qui ne modifient pas sa fermeture
Pour chaque f $\in \mathcal{C}$ faire
  Si $\mathcal{C} \setminus$ {f} $\models$ f Alors // i. e. Y $\subseteq$? FermTransAttr($\mathcal{C}\setminus$ {f},X) où f = X $\rightarrow$ Y
    $\mathcal{C} \longleftarrow \mathcal{C} \setminus$ {f}
// 3. réduction des membres gauches des dépendances fonctionnelles
// qui ne modifient pas sa fermeture
Pour chaque X',X" $\rightarrow$ Y $\in \mathcal{C}$ avec card(X')=1 faire
// Remarque : souvent présenté pour chaque X $\rightarrow$ Y $\in \mathcal{C}$ tel que
// $\exists$ X' avec X'$\cup$X"=X $\wedge$ X'$\cap$X"=$\emptyset \wedge$ card(X')$\geq$1 $\wedge$ card(X")$\geq$1 / $\mathcal{C} \models$ X" $\rightarrow$ Y
  Si $\mathcal{C} \models$ X" $\rightarrow$ Y Alors
  // i. e. Y $\subseteq$? FermTransAttr($\mathcal{C}$,X") ou encore X' $\subseteq$? FermTransAttr($\mathcal{C}$,X")
    $\mathcal{C} \longleftarrow \mathcal{C} \setminus$ {X',X" $\rightarrow$ Y} $\cup$ {X" $\rightarrow$ Y}
Retourner $\mathcal{C}$

Remarque

On peut illustrer ces deux notions de fermeture transitive et de couverture minimale d'un ensemble de dépendances fonctionnelles élémentaires par un demi-treillis dont les éléments sont des ensembles de dépendances fonctionnelles élémentaires tels que deux éléments sont connectés lorsque l'on peut obtenir l'un à partir de l'autre en une seule simplification qui ne modifie pas sa fermeture (soit par élimination d'une dépendance fonctionnelle canonique, soit par réduction d'un attribut du membre gauche d'une dépendance fonctionnelle) ; dans ce demi-treillis, un ensemble de dépendances fonctionnelles élémentaires $\mathcal{F}$ est un élément quelconque, la fermeture transitive $\mathcal{F}^+$ est le seul élément maximal et les couvertures minimales $\mathcal{C}$ sont les éléments minimaux

Fermeture transitive et couverture minimale (3/7)

Premier exemple de CouvMinDF()

Soit $\mathcal{F}$ = { c → a
c,e → b,d
a,c,d → b
d → e
b,c → d }

À l'issue de l'étape 1, on a :

$\mathcal{C}$ = { c → a
c,e → b // issue de c,e → b,d
c,e → d // issue de c,e → b,d
a,c,d → b
d → e
b,c → d }

À l'issue de l'étape 2, on parvient à :

$\mathcal{C}$ = { c → a // conservée car a ∉ FermTransAttr($\mathcal{C}$\{c → a},{c}) = {c}
// c,e → b éliminée de $\mathcal{C}$ car
// b ∈ FermTransAttr($\mathcal{C}$\{c,e → b},{c,e}) = {a,b,c,d,e}
c,e → d // conservée car
// d ∉ FermTransAttr($\mathcal{C}$\{c,e → d},{c,e}) = {a,c,e}
a,c,d → b // conservée car
// b ∉ FermTransAttr($\mathcal{C}$\{a,c,d → b},{a,c,d}) =
// {a,c,d,e}
d → e // conservée car e ∉ FermTransAttr($\mathcal{C}$\{d → e},{d}) = {d}
b,c → d } // conservée car
// d ∉ FermTransAttr($\mathcal{C}$\{b,c → d},{b,c}) = {a,b,c}

À l'issue de l'étape 3, on obtient une couverture minimale :

$\mathcal{C}$ = { c → a
c,e → d // c gardé (X'={c}) car c ∉ FermTransAttr($\mathcal{C}$,{e}) = {e}
// e gardé (X'={e}) car e ∉ FermTransAttr($\mathcal{C}$,{c}) = {a,c}
c,d → b // a supprimé (X'={a})
// de a,c,d → b donc remplacée par c,d → b dans $\mathcal{C}$
// car a ∈ FermTransAttr($\mathcal{C}$,{c,d}) = {a,b,c,d,e}
// c gardé (X'={c}) car c ∉ FermTransAttr($\mathcal{C}$,{d}) = {d,e}
// d gardé (X'={d}) car d ∉ FermTransAttr($\mathcal{C}$,{c}) = {a,c}
d → e
b,c → d } // b gardé (X'={b}) car b ∉ FermTransAttr($\mathcal{C}$,{c}) = {a,c}
// c gardé (X'={c}) car c ∉ FermTransAttr($\mathcal{C}$,{b}) = {b}

Remarque : on peut obtenir $\mathcal{F}$ à partir de $\mathcal{C}$ en appliquant les propriétés des dépendances fonctionnelles d'augmentation c,d → b ⟹ a,c,d → b et de pseudo-transitivité
$\left\{ \begin{array}{l} \text{c,e} \rightarrow \text{d} \\ \text{c,d} \rightarrow \text{b} \end{array} \right. \Longrightarrow \text{c,e} \rightarrow \text{b}$

Fermeture transitive et couverture minimale (4/7)

Deuxième exemple de CouvMinDF()

Soit $\mathcal{F}$ = { c → a
a,c,d → b
c,e → b,d
d → e
b,c → d }

À l'issue de l'étape 1, la dépendance fonctionnelle c,e → b,d a été éclatée en deux dépendances fonctionnelles : c,e → b et c,e → d

À l'issue de l'étape 2, les dépendances fonctionnelles a,c,d → b et c,e → d ont été éliminées car b ∈ FermTransAttr($\mathcal{C}$\{a,c,d → b},{a,c,d}) = {a,b,c,d,e} et d ∈ FermTransAttr($\mathcal{C}$\{c,e → d},{c,e}) = {a,b,c,d,e} respectivement

L'étape 3 n'apporte aucune modification à

$\mathcal{C}$ = { c → a
c,e → b
d → e
b,c → d }

Remarque : on peut obtenir $\mathcal{F}$ à partir de $\mathcal{C}$ en appliquant les propriétés des dépendances fonctionnelles de pseudo-transitivité $\left\{ \begin{array}{l} \text{c,e} \rightarrow \text{b} \\ \text{b,c} \rightarrow \text{d} \end{array} \right. \Longrightarrow$ c,e → d et de pseudo-transitivité puis d'augmentation $\left\{ \begin{array}{l} \text{d} \rightarrow \text{e} \\ \text{c,e} \rightarrow \text{b} \end{array} \right. \Longrightarrow$ c,d → b ⟹ a,c,d → b

Fermeture transitive et couverture minimale (5/7)

Demi-treillis partiel que l'on obtient pour les deux premiers exemples de CouvMinDF() qui ont le même ensemble de dépendances fonctionnelles élémentaires $\mathcal{F}$, mais complet pour le sous-demi-treillis issu de $\mathcal{F}$

$\mathcal{F}^+ = \{$ c → a *ou* a,c
c,e → a,b,d *ou* a,b,c,d,e
a,c,d → b,e *ou* a,b,c,d,e
d → e *ou* d,e
b,c → a,d,e *ou* a,b,c,d,e $\}$

$\mathcal{F} = \{$ c → a
c,e → b,d
a,c,d → b
d → e
b,c → d $\}$

$\{$ c → a
c,e → b
a,c,d → b
d → e
b,c → d $\}$

$\{$ c → a
c,e → d
a,c,d → b
d → e
b,c → d $\}$

$\{$ c → a
c,e → b,d
c,d → b
d → e
b,c → d $\}$

$\{$ c → a
c,e → b
c,d → b
d → e
b,c → d $\}$

$\mathcal{C}1^{er} = \{$ c → a
c,e → d
c,d → b
d → e
b,c → d $\}$

$\{$ c → a
c,e → b,d
d → e
b,c → d $\}$

$\mathcal{C}2^{e} = \{$ c → a
c,e → b
d → e
b,c → d $\}$

Fermeture transitive et couverture minimale (6/7)

Troisième exemple de CouvMinDF() : l'exemple « jouet » (1/2)

Soit $\mathcal{F}$ = { IntitComplet → IntitAbrege
IntitAbrege → IntitComplet
NoINE → NomEtu,DepartNaissEtu
NoINE,IntitComplet → Annee
NoINE,IntitAbrege → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur,NoINE,NomEtu,DepartNaissEtu
NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitComplet → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitAbrege → Annee }

Fermeture transitive et couverture minimale (7/7)

Troisième exemple de CouvMinDF() : l'exemple « jouet » (2/2)

À l'issue de l'étape 1, NoINE → NomEtu,DepartNaissEtu $\Longrightarrow$
$\begin{cases} \text{NoINE} \rightarrow \text{NomEtu} \\ \text{NoINE} \rightarrow \text{DepartNaissEtu} \end{cases}$ et NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur,NoINE,NomEtu, DepartNaissEtu $\Longrightarrow$
$\begin{cases} \text{NoImmatChiffres,NoImmatLettres,NoImmatDepart} \rightarrow \text{Couleur} \\ \text{NoImmatChiffres,NoImmatLettres,NoImmatDepart} \rightarrow \text{NoINE} \\ \text{NoImmatChiffres,NoImmatLettres,NoImmatDepart} \rightarrow \text{NomEtu} \\ \text{NoImmatChiffres,NoImmatLettres,NoImmatDepart} \rightarrow \text{DepartNaissEtu} \end{cases}$

À l'issue de l'étape 2, les dépendances fonctionnelles

NoINE,IntitComplet → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart → NomEtu
NoImmatChiffres,NoImmatLettres,NoImmatDepart → DepartNaissEtu
NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitComplet → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitAbrege → Annee

ont été éliminées respectivement car

Annee ∈ FermTransAttr($\mathcal{C}$\{NoINE,IntitComplet → Annee}, {NoINE, IntitComplet}) = {IntitAbrege, IntitComplet, NoINE, NomEtu, DepartNaissEtu, Annee}

NomEtu ∈ FermTransAttr($\mathcal{C}$\{NoImmatChiffres,NoImmatLettres, NoImmatDepart → NomEtu}, {NoImmatChiffres,NoImmatLettres, NoImmatDepart}) = {NoINE, NomEtu, DepartNaissEtu, NoImmatChiffres, NoImmatLettres, NoImmatDepart, Couleur}

DepartNaissEtu ∈ FermTransAttr($\mathcal{C}$\{NoImmatChiffres,NoImmatLettres, NoImmatDepart → DepartNaissEtu}, {NoImmatChiffres,NoImmatLettres, NoImmatDepart}) = {NoINE, NomEtu, DepartNaissEtu, NoImmatChiffres, NoImmatLettres, NoImmatDepart, Couleur}

Annee ∈ FermTransAttr($\mathcal{C}$\{NoImmatChiffres,NoImmatLettres, NoImmatDepart,IntitComplet → Annee}, {NoImmatChiffres, NoImmatLettres,NoImmatDepart,IntitComplet}) = {IntitComplet, IntitAbrege, NoINE, NomEtu, DepartNaissEtu, Annee, NoImmatChiffres, NoImmatLettres, NoImmatDepart, Couleur}

Annee ∈ FermTransAttr($\mathcal{C}$\{NoImmatChiffres,NoImmatLettres, NoImmatDepart,IntitAbrege → Annee}, {NoImmatChiffres, NoImmatLettres,NoImmatDepart,IntitAbrege}) = {IntitComplet, IntitAbrege, NoINE, NomEtu, DepartNaissEtu, Annee, NoImmatChiffres, NoImmatLettres, NoImmatDepart, Couleur}

L'étape 3 n'apporte aucune modification à

$\mathcal{C}$ = { IntitComplet → IntitAbrege
IntitAbrege → IntitComplet
NoINE → NomEtu
NoINE → DepartNaissEtu
NoINE,IntitAbrege → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur
NoImmatChiffres,NoImmatLettres,NoImmatDepart → NoINE }

Remarque : les deux premières dépendances fonctionnelles créent un cycle et IntitComplet → IntitAbrege sera supprimée dans le schéma relationnel au profit d'une contrainte d'unicité car IntitComplet est en fait une clé secondaire de Diplomes (en choisissant IntitAbrege comme clé primaire)

Dépendance fonctionnelle multivaluée (1/3)

Définition

Soient R (X , Y , Z) une relation et X, Y, Z trois ensembles d'attributs. On dit que X multidétermine Y (ou qu'il y a une dépendance fonctionnelle multivaluée de Y sur X) noté X $\twoheadrightarrow$ Y si, étant donné des valeurs de X, il y a un ensemble de valeurs de Y associées qui est de plus indépendant des attributs de Z (i. e. $\forall(x,y,z) \in R \wedge (x,y',z') \in R \Longrightarrow (x,y',z) \in R \wedge (x,y,z') \in R$ i. e. $\forall x \in \pi_X(R), \{x\} \times \pi_Y(\sigma_{X=x}(R)) \times \pi_Z(\sigma_{X=x}(R)) = \sigma_{X=x}(R)$)

Dépendance fonctionnelle multivaluée (2/3)

Premier exemple : X ↠ Y

X	Y	Z
α	1	a
β	1	a
β	1	b
β	1	c
γ	1	a
γ	1	b
γ	2	a
γ	2	b

car

$\{\alpha\} \times \{1\} \times \{a\} =$

α	1	a

$\{\beta\} \times \{1\} \times \{a, b, c\} =$

β	1	a
β	1	b
β	1	c

$\{\gamma\} \times \{1, 2\} \times \{a, b\} =$

γ	1	a
γ	1	b
γ	2	a
γ	2	b

Second exemple

R

A	B	C	D	E
α	a	a'	a''	1
α	a	a'	b''	1
α	a	b'	a''	1
α	a	b'	b''	1
α	b	a'	a''	1
α	b	a'	b''	1

avec

A ↠ D	D ↠ A	B,D ↠ A	A,C,E ↠ B
A ↠ E	D ↠ E	B,D ↠ C	A,C,E ↠ D
A ↠ B,C	D ↠ A,E	B,D ↠ E	B,C,D ↠ A
A ↠ D,E	D ↠ B,C	B,D ↠ A,C	B,C,D ↠ E
A ↠ B,C,D	D ↠ A,B,C	B,D ↠ A,E	B,C,E ↠ A
A ↠ B,C,E	D ↠ B,C,E	B,D ↠ C,E	B,C,E ↠ D
B ↠ A	E ↠ A	B,E ↠ A	B,D,E ↠ A
B ↠ C	E ↠ D	B,E ↠ C	B,D,E ↠ C
B ↠ D	E ↠ A,D	B,E ↠ D	C,D,E ↠ A
B ↠ E	E ↠ B,C	B,E ↠ A,C	C,D,E ↠ B
B ↠ A,C	E ↠ A,B,C	B,E ↠ A,D	
B ↠ A,D	E ↠ B,C,D	B,E ↠ C,D	
B ↠ A,E	A,B ↠ C	C,D ↠ A	mais :
B ↠ C,D	A,B ↠ D	C,D ↠ B	A ↛ B
B ↠ C,E	A,B ↠ E	C,D ↠ E	A ↛ C
B ↠ D,E	A,B ↠ C,D	C,D ↠ A,B	A ↛ B,D
B ↠ A,C,D	A,B ↠ C,E	C,D ↠ A,E	A ↛ B,E
B ↠ A,C,E	A,B ↠ D,E	C,D ↠ B,E	A ↛ C,D
B ↠ A,D,E	A,C ↠ B	C,E ↠ A	A ↛ C,E
B ↠ C,D,E	A,C ↠ D	C,E ↠ B	⋮
C ↠ A	A,C ↠ E	C,E ↠ D	A,D ↛ B
C ↠ B	A,C ↠ B,D	C,E ↠ A,B	A,D ↛ C
C ↠ D	A,C ↠ B,E	C,E ↠ A,D	A,D ↛ B,E
C ↠ E	A,C ↠ D,E	C,E ↠ B,D	A,D ↛ C,E
C ↠ A,B	A,D ↠ E	D,E ↠ A	A,E ↛ B
C ↠ A,D	A,D ↠ B,C	D,E ↠ B,C	A,E ↛ C
C ↠ A,E	A,E ↠ D	A,B,C ↠ D	A,E ↛ B,D
C ↠ B,D	A,E ↠ B,C	A,B,C ↠ E	A,E ↛ C,D
C ↠ B,E	B,C ↠ A	A,B,D ↠ C	D,E ↛ B
C ↠ D,E	B,C ↠ D	A,B,D ↠ E	D,E ↛ C
C ↠ A,B,D	B,C ↠ E	A,B,E ↠ C	D,E ↛ A,B
C ↠ A,B,E	B,C ↠ A,D	A,B,E ↠ D	D,E ↛ A,C
C ↠ A,D,E	B,C ↠ A,E	A,C,D ↠ B	A,D,E ↛ B
C ↠ B,D,E	B,C ↠ D,E	A,C,D ↠ E	A,D,E ↛ C

Par exemple, on a A ↠ B,C car $\{\alpha\} \times \{(a, a'), (a, b'), (b, a')\} \times \{(a'', 1), (b'', 1)\} =$ R
De même, on a A,C,D ↠ B car $(\{(\alpha, a', a'')\} \times \{a, b\} \times \{1\}) \uplus (\{(\alpha, a', b'')\} \times \{a, b\} \times \{1\}) \uplus (\{(\alpha, b', a'')\} \times \{a\} \times \{1\}) \uplus (\{(\alpha, b', b'')\} \times \{a\} \times \{1\}) =$ R

Dépendance fonctionnelle multivaluée (3/3)

Contre-exemple :

R

X	Y	Z
ω	9	z
ω	9	y
ω	8	z

car $(\omega, 9, y) \in$ R et $(\omega, 8, z) \in$ R mais $(\omega, 8, y) \notin$ R

Axiomes d'inférence

Complémentation : R (X , Y , Z) $\wedge$ X $\twoheadrightarrow$ Y $\Longrightarrow$ X $\twoheadrightarrow$ Z

Exemple : dans le second exemple, on a A $\twoheadrightarrow$ D et A $\twoheadrightarrow$ B,C,E

Augmentation : X $\twoheadrightarrow$ Y $\wedge$ S' $\subseteq$ S $\Longrightarrow$ X,S $\twoheadrightarrow$ Y,S'

Exemple : dans le second exemple, pour A $\twoheadrightarrow$ B,C et S={C,D}, on obtient A,C,D $\twoheadrightarrow$ B (sans répéter les attributs C ou D dans le membre droit de la dépendance fonctionnelle multivaluée car ils y sont déjà dans le membre gauche)

Transitivité : X $\twoheadrightarrow$ Y $\wedge$ Y $\twoheadrightarrow$ Z $\Longrightarrow$ X $\twoheadrightarrow$ Z\Y

Exemple : dans le second exemple, on a A $\twoheadrightarrow$ B,C et B,C $\twoheadrightarrow$ D donc A $\twoheadrightarrow$ D

Règle additionnelle

Union : X $\twoheadrightarrow$ Y $\wedge$ X $\twoheadrightarrow$ Z $\Longrightarrow$ X $\twoheadrightarrow$ Y,Z

Exemple : dans le second exemple, on a A $\twoheadrightarrow$ D et A $\twoheadrightarrow$ B,C donc A $\twoheadrightarrow$ B,C,D

Dépendance fonctionnelle multivaluée élémentaire

Une dépendance fonctionnelle multivaluée X $\twoheadrightarrow$ Y est élémentaire si :

- Y $\neq \emptyset$
- X $\cap$ Y $= \emptyset$
- $\nexists$ X' $\twoheadrightarrow$ Y' $\neq$ X $\twoheadrightarrow$ Y / X' $\subseteq$ X $\wedge$ Y' $\subseteq$ Y

Théorème : X $\rightarrow$ Y $\Longrightarrow$ X $\twoheadrightarrow$ Y

Preuve : soit R (X , Y , Z) ; X $\rightarrow$ Y $\Longrightarrow \{(x,y,z) \in R \wedge (x,y',z') \in R \Longrightarrow y = y'\}$ $\Longrightarrow \{(x,y,z) \in R \wedge (x,y',z') \in R \Longrightarrow (x,y',z) \in R \wedge (x,y,z') \in R\} \Longleftrightarrow$ X $\twoheadrightarrow$ Y

Dépendance fonctionnelle de jointure (1/2)

Définition

Soient $S_1, \ldots, S_m$ des sous-ensembles des attributs d'une relation R. Il existe une dépendance fonctionnelle de jointure $*\{S_1, \ldots, S_m\}$ si $\pi_{S_1}(R) \bowtie \ldots \bowtie \pi_{S_m}(R) = \pi_{R.}*(R)$ i. e. la relation R une fois avoir réorganisé les attributs

Remarque : R peut alors être décomposée sans perte d'information (SPI) en S_1, $\ldots$, S_m

Théorème : les dépendances fonctionnelles multivaluées sont des cas particuliers des dépendances fonctionnelles de jointure

Preuve : R (X , Y , Z) avec $X \twoheadrightarrow Y$ (et donc $X \twoheadrightarrow Z$) vérifie $*\{\{X,Y\},\{X,Z\}\}$

Dépendance fonctionnelle de jointure triviale

Une dépendance fonctionnelle de jointure $*\{S_1, \ldots, S_m\}$ d'une relation R est triviale si $\exists i / S_i = R$

N. B. : R (A_1 , $\ldots$, A_n) admet toujours $*\{\{A_1,\ldots,A_n\},\ldots\}$ comme dépendance fonctionnelle de jointure

Remarque

Soit R (A_1 , $\ldots$, A_n) une relation de n attributs. Alors, considérer les dépendances fonctionnelles de jointure $*\{S_1, \ldots, S_m\}$ revient à choisir les $S_i \subseteq \{ A_1 , \ldots, A_n \}$ tels que :

$\forall\ 1 \leq i \leq m$, $S_i \neq \emptyset$ car R est probablement non vide

$\forall\ 1 \leq i \leq m$, $S_i \neq \{ A_1 , \ldots, A_n \}$
pour éviter les dépendances fonctionnelles de jointure triviales

$\forall\ 1 \leq i, j \leq m \wedge i \neq j$, $S_i \not\subseteq S_j$ car sinon $S_i \bowtie S_j = S_j$

$\forall\ 1 \leq i \leq m, \exists\ j \neq i \wedge 1 \leq j \leq m$ / $S_i \cap S_j \neq \emptyset$
pour que chaque S_i ait au moins une jointure

$\bigcup_{i=1}^{m} S_i = \{ A_1 , \ldots, A_n \}$ pour retrouver tous les attributs de R à partir des S_i

Corollaire : $\forall\ 1 \leq i \leq m$, $\text{card}(S_i) \geq 2$ d'après la 1^{re} ainsi que les 3^e et 4^e conditions

Par exemple, pour $n = 3$, toutes les dépendances fonctionnelles de jointure à considérer pour la relation R (A_1 , A_2 , A_3) sont $*\{\{A_1,A_2\},\{A_1,A_3\}\}$ (jointure sur A_1), $*\{\{A_1,A_2\},\{A_2,A_3\}\}$ (jointure sur A_2), $*\{\{A_1,A_3\},\{A_2,A_3\}\}$ (jointure sur A_3) et $*\{\{A_1,A_2\},\{A_2,A_3\},\{A_3,A_1\}\}$ (jointures sur A_2, sur A_3 et sur A_1)

Exemple (1/2)

R

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	b

a pour seule dépendance fonctionnelle de jointure $*\{\{X,Y\},\{Y,Z\},\{Z,X\}\}$

Dépendance fonctionnelle de jointure (2/2)

Exemple (2/2)

En effet, $\pi_{X,Y}(R) \bowtie \pi_{Y,Z}(R) \bowtie \pi_{Z,X}(R) =$

X	Y
α	1
α	2
β	1

$\bowtie$

Y	Z
1	a
1	b
2	b

$\bowtie$

Z	X
a	α
b	α
b	β

$=$

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	a
β	1	b

$\bowtie$

X	Z
α	a
α	b
β	b

ou

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	b
β	2	b

$\bowtie$

X	Y
α	1
α	2
β	1

ou

X	Y	Z
α	1	a
α	1	b
α	2	a
α	2	b
β	1	b

$\bowtie$

Y	Z
1	a
1	b
2	b

$=$

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	b

$=$ R mais ni *{{X,Y},{X,Z}} car $\{(\alpha,1),(\alpha,2),(\beta,1)\} \bowtie \{(\alpha,a),(\alpha,b),(\beta,b)\} = R \uplus \{(\alpha,2,a)\} \neq R$, ni *{{X,Y},{Y,Z}} car $\{(\alpha,1),(\alpha,2),(\beta,1)\} \bowtie \{(1,a),(1,b),(2,b)\} = R \uplus \{(\beta,1,a)\} \neq R$, ni *{{X,Z},{Y,Z}} car $\{(\alpha,a),(\alpha,b),(\beta,b)\} \bowtie \{(1,a),(1,b),(2,b)\} = R \uplus \{(\beta,2,b)\} \neq R$

Contre-exemple

R

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	b
γ	2	a

n'a aucune dépendance fonctionnelle de jointure non triviale : ni *{{X,Y},{X,Z}} notamment car $(\alpha,2,a) \in \left(\pi_{X,Y}(R) \bowtie \pi_{X,Z}(R)\right) \setminus R$, ni *{{X,Y},{Y,Z}} notamment car $(\beta,1,a) \in \left(\pi_{X,Y}(R) \bowtie \pi_{Y,Z}(R)\right) \setminus R$, ni *{{X,Z},{Y,Z}} notamment car $(\beta,2,b) \in \left(\pi_{X,Z}(R) \bowtie \pi_{Y,Z}(R)\right) \setminus R$, ni *{{X,Y},{Y,Z},{Z,X}} car $\pi_{X,Y}(R) \bowtie \pi_{Y,Z}(R) \bowtie \pi_{Z,X}(R) =$

α	1
α	2
β	1
γ	2

$\bowtie$

1	a
1	b
2	b
2	a

$\bowtie$

a	α
b	α
b	β
a	γ

$=$

α	1	a
α	1	b
α	2	b
β	1	b
α	2	a
γ	2	a

$= R \uplus \{(\alpha,2,a)\} \neq R$

Formes Normales (1/13)

Objectif : le respect des formes normales (FN) permet d'éliminer les redondances, sans perte d'information (SPI), et en essayant de préserver les dépendances fonctionnelles autant que possible

Notation : R (C_1 , ... , C_k , A_1 , ... , A_l) avec (C_1 , ... , C_k) clé primaire de R

Première forme normale (1FN)

Définition

Aucun attribut n'est à valeur multiple (i. e. tout attribut contient une valeur atomique)

Exemple et contre-exemple pour (NoINE , NomEtu) :

NoINE	NomEtu
5	'DURAND'
2	'LEROI'
4	'MARTIN'
7	'LEROI'
3	'DUPOND'

est en 1FN contrairement à

NoINE	NomEtu
5	'DURAND'
{ 2 , 7 }	'LEROI'
4	'MARTIN'
3	'DUPOND'

non en 1FN

Deuxième forme normale (2FN)

1FN et $C_1,\ldots,C_k \rightarrow A_1,\ldots,A_l$ élémentaires

Définition

La relation doit être en 1FN

Les attributs d'une relation ne faisant pas partie de sa clé primaire sont en dépendance fonctionnelle élémentaire avec la clé primaire de cette relation (i. e. tout attribut hors clé ne dépend pas fonctionnellement d'une partie stricte de la clé i. e. aucune partie stricte de la clé primaire ne détermine un attribut hors clé i. e. $\nexists$ C' $\subsetneqq \{C_1,\ldots,C_k\} \wedge A_j$ / C' $\rightarrow A_j$)

Exemple : (NoINE , NomEtu , DepartNaissEtu) de clé NoINE car NoINE $\rightarrow$ NomEtu,DepartNaissEtu

Contre-exemple : (NoINE , NomEtu , DepartNaissEtu) de clé (NoINE , NomEtu) car NoINE $\rightarrow$ DepartNaissEtu

Troisième forme normale (3FN)

2FN et $C_1,\ldots,C_k \rightarrow A_1,\ldots,A_l$ directes

Définition

La relation doit être en 2FN (et donc aussi en 1FN)

Les attributs d'une relation ne faisant pas partie de sa clé primaire sont en dépendance fonctionnelle directe avec la clé primaire de cette relation (i. e. $\nexists$ Z $\wedge$ A_j / $C_1,\ldots,C_k \rightarrow Z \wedge Z \rightarrow A_j \wedge Z \not\rightarrow C_1,\ldots,C_k$ i. e. il n'existe pas d'attributs hors clé qui dépendent fonctionnellement d'autres attributs hors clé car sinon on aurait A' $\rightarrow$ A" avec A' $\subsetneqq \{A_1,\ldots,A_l\}$ et A" $\subseteq \{A_1,\ldots,A_l\}$ et A" $\notin$ A' de sorte que $C_1,\ldots,C_k \rightarrow$ A" n'est pas directe où Z=A')

Exemple : (NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur , NoINE) de clé (NoImmatChiffres , NoImmatLettres , NoImmatDepart)

Contre-exemples : (NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur , NoINE , DepartNaissEtu) ou encore (NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur , DepartNaissEtu) de clé (NoImmatChiffres , NoImmatLettres , NoImmatDepart) car NoImmatChiffres,NoImmatLettres,NoImmatDepart $\rightarrow$ NoINE $\wedge$ NoINE $\rightarrow$ DepartNaissEtu $\wedge$ NoINE $\not\rightarrow$ NoImmatChiffres,NoImmatLettres,NoImmatDepart

Remarque : la phrase *The key, the whole key, nothing but the key* (la vérité, toute la vérité, rien que la vérité) [DATE citant SHAKESPEARE] fournit un moyen mnémonique pour retenir 1FN, 2FN, 3FN respectivement

Théorème : toute relation admet une décomposition (non unique) en 3FN, SPI, et qui préserve les dépendances fonctionnelles

Algorithme de décomposition d'un ensemble de dépendances fonctionnelles en 3FN

Fonction DecompoDFen3FN($\mathcal{F}$,$\mathcal{A}$) : $\mathcal{S}$
// [E/] $\mathcal{F}$: un ensemble de dépendances fonctionnelles
// [E/] $\mathcal{A}$: un ensemble d'attributs
// [S/] $\mathcal{S}$: un ensemble de relations, avec leurs contraintes d'intégrité de clé primaire,
// décomposées en 3FN (ou schéma relationnel partiel, sans les autres contraintes)
// $\mathcal{C}$: une couverture minimale de $\mathcal{F}$
// $\mathcal{B}$: ensemble temporaire d'attributs (de $\mathcal{F}$ et de $\mathcal{A}$ mais sans ceux de $\mathcal{C}$)
// 1. constitution d'une couverture minimale de $\mathcal{F}$, et utilisation de la propriété d'union
$\mathcal{C} \longleftarrow$ CouvMinDF($\mathcal{F}$)
Pour chaque $X \to Y \in \mathcal{C} \wedge X \to Z \in \mathcal{C}$ avec $Y \neq Z$ faire
  $\mathcal{C} \longleftarrow \mathcal{C} \setminus \{ X \to Y , X \to Z \} \cup \{ X \to Y \cup Z \}$ // i. e. $\mathcal{C} \longleftarrow \ldots \cup \{ X \to Y , Z \}$
// 2. traitement des attributs isolés
// (dans $\mathcal{A}$ mais pas dans $\mathcal{F}$, ou disparus en passant de $\mathcal{F}$ à $\mathcal{C}$)
$\mathcal{B} \longleftarrow \mathcal{A}$ // initialisation avec les attributs de $\mathcal{A}$
Pour chaque $X_1 , \ldots , X_k \to Y_1 , \ldots , Y_l \in \mathcal{F}$ faire // ajout des attributs de $\mathcal{F}$
  $\mathcal{B} \longleftarrow \mathcal{B} \cup \{ X_1 , \ldots , X_k , Y_1 , \ldots , Y_l \}$
Pour chaque $X_1 , \ldots , X_k \to Y_1 , \ldots , Y_l \in \mathcal{C}$ faire // suppression des attributs de $\mathcal{C}$
  $\mathcal{B} \longleftarrow \mathcal{B} \setminus \{ X_1 , \ldots , X_k , Y_1 , \ldots , Y_l \}$
$\mathcal{S} \longleftarrow \emptyset$
Pour chaque $Z \in \mathcal{B}$ faire // création d'autant de relations que d'attributs isolés
  $\mathcal{S} \longleftarrow \mathcal{S} \cup \{ Z (Z)$ de clé $Z \}$
// dans la littérature spécialisée, $\mathcal{S} \longleftarrow \{ \mathcal{B} (\mathcal{B})$ de clé $\mathcal{B} \}$ formant une seule relation
// 3. création des relations
Tant que $\mathcal{C} \neq \emptyset$ faire // ajout d'autant de relations que de dépendances fonctionnelles
  Choisir $X \to Y \in \mathcal{C}$ quelconque
    // dans la littérature, / $\forall$ $W \to Z \in \mathcal{C}$ card(X) $\geq$ card(W)
    // N. B. : $\mathcal{C}$ est couverture minimale $\Longrightarrow \nexists\ Y_i \to Y_j\ \forall i \neq j$ où $Y=\{Y_1,\ldots,Y_l\}$
  $\mathcal{S} \longleftarrow \mathcal{S} \cup \{ X (X , Y)$ de clé $X \}$
  $\mathcal{C} \longleftarrow \mathcal{C} \setminus \{X \to Y\}$
Retourner $\mathcal{S}$

Exemples de DecompoDFen3FN() (1/2)

Premier exemple de DecompoDFen3FN()

Soit $\mathcal{F}$ = { a → b,f et $\mathcal{A}$ = { a,c,e }
a,c,d → b
b,g → h }

À l'issue de l'étape 1, on a
$\mathcal{C}$ = { a → b,f
b,g → h }

À l'issue de l'étape 2, on a
$\mathcal{B}$ = { c,d,e } et
$\mathcal{S}$ = { c (c) de clé c
d (d) de clé d
e (e) de clé e }

À l'issue de l'étape 3, on a
$\mathcal{S}$ = { c (c) de clé c
d (d) de clé d
e (e) de clé e
a (a , b , f) de clé a
bg (b , g , h) de clé (b , g) }

Exemples de DecompoDFen3FN() (2/2)

Second exemple de DecompoDFen3FN() pour l'exemple « jouet »

Soit $\mathcal{F}$ = { IntitComplet → IntitAbrege
IntitAbrege → IntitComplet
NoINE → NomEtu,DepartNaissEtu
NoINE,IntitComplet → Annee
NoINE,IntitAbrege → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur,NoINE,NomEtu,DepartNaissEtu
NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitComplet → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart,IntitAbrege → Annee }

et $\mathcal{A} = \emptyset$ car il n'y a pas d'attributs isolés et dans ce cas, n'importe quel sous-ensemble de l'ensemble de tous les attributs figurant dans l'ensemble des dépendances fonctionnelles convient, le plus simple étant l'ensemble vide

À l'issue de l'étape 1, on a

$\mathcal{C}$ = { IntitComplet → IntitAbrege
IntitAbrege → IntitComplet
NoINE → NomEtu,DepartNaissEtu
NoINE,IntitAbrege → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur,NoINE }

À l'issue de l'étape 2, on a $\mathcal{B} = \emptyset$ et $\mathcal{S} = \emptyset$

À l'issue de l'étape 3, on a

$\mathcal{S}$ = { IntitComplet (IntitComplet , IntitAbrege) de clé IntitComplet
IntitAbrege (IntitAbrege , IntitComplet) de clé IntitAbrege
NoINE (NoINE , NomEtu , DepartNaissEtu) de clé NoINE
NoINE_IntitAbrege (NoINE , IntitAbrege , Annee) de clé (NoINE , IntitAbrege)
NoImmatChiffres_NoImmatLettres_NoImmatDepart (NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur , NoINE) de clé (NoImmatChiffres , NoImmatLettres , NoImmatDepart) }

Remarque : de façon générale, le schéma relationnel

R (A , B , C) de clé A
S (B , A , D) de clé B
// R.B référence S.B
// S.A référence R.A
(R.A , R.B) = (S.A , S.B)
// Contrainte d'unicité de R.B
// Contrainte d'unicité de S.A

peut être remplacé par

RS (A , B , C , D) de clé (primaire) A
Contrainte d'unicité de RS.B (clé secondaire)

; dans l'exemple, les deux relations IntitComplet (IntitComplet , IntitAbrege) de clé IntitComplet et IntitAbrege (IntitAbrege , IntitComplet) de clé IntitAbrege peuvent être remplacées, en supprimant la relation IntitComplet, par IntitAbrege (IntitAbrege , IntitComplet) de clé IntitAbrege et contrainte d'unicité de IntitComplet

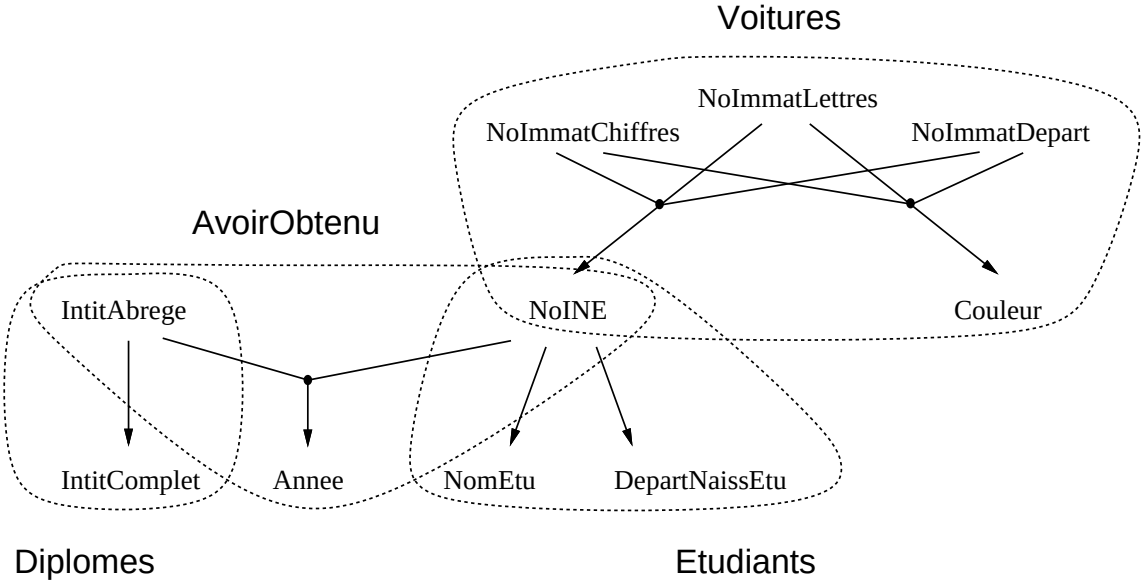
Remarque : passage de l'hypergraphe d'une couverture minimale des dépendances fonctionnelles à un schéma relationnel partiel dans les cas simples

Dans les cas simples, un schéma relationnel partiel (schémas des relations et contraintes d'intégrité restreintes aux contraintes de clés primaires et référentielles) se déduit de l'hypergraphe d'une couverture minimale des dépendances fonctionnelles. En effet, chaque nœud (i. e. sommet interne) de l'hypergraphe donne lieu à une relation dont la clé primaire correspond au nœud et dont les attributs hors clé correspondent aux sommets fils du nœud ; de plus, les attributs correspondant à des sommets fils qui sont eux-mêmes des nœuds sont des clés étrangères.

Pour l'exemple « jouet », à partir de l'ensemble des dépendances fonctionnelles (une fois enlevées les dépendances fonctionnelles de la clé secondaire et celles obtenues par réflexivité ou transitivité)

{ IntitAbrege → IntitComplet
NoINE → NomEtu,DepartNaissEtu
NoINE,IntitAbrege → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart → Couleur,NoINE }

de même couverture minimale (une fois la propriété d'union appliquée) dont l'hypergraphe (en nommant les regroupements source/cible) est



on obtient le schéma relationnel partiel

Diplomes (IntitAbrege , IntitComplet)
Etudiants (NoINE , NomEtu , DepartNaissEtu)
AvoirObtenu (NoINE , IntitAbrege , Annee)
Voitures (NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur , NoINE)
IntitAbrege est la clé de la relation Diplomes
NoINE est la clé de la relation Etudiants
(NoINE , IntitAbrege) est la clé de la relation AvoirObtenu
(NoImmatChiffres , NoImmatLettres , NoImmatDepart) est la clé de la relation Voitures
AvoirObtenu.NoINE référence Etudiants.NoINE
AvoirObtenu.IntitAbrege référence Diplomes.IntitAbrege
Voitures.NoINE référence Etudiants.NoINE

Forme normale de Boyce-Codd (FNBC) (1/3)

Définition

Les seules dépendances fonctionnelles élémentaires sont celles dont la clé entière détermine un attribut (hors clé)

Corollaires

Il n'existe pas d'attributs hors clé qui déterminent tout ou partie de la clé (i. e. $\nexists$ A' $\subseteq \{A_1,\ldots,A_l\} \wedge$ C' $\subseteq \{C_1,\ldots,C_k\}$ / A' $\rightarrow$ C')

Il n'existe pas une partie stricte (i. e. un sous-ensemble strict) de la clé qui détermine une autre partie de la clé (i. e. $\nexists$ C' $\subsetneq \{C_1,\ldots,C_k\} \wedge$ C" $\subseteq \{C_1,\ldots,C_k\} \wedge$ C' $\neq$ C" / C' $\rightarrow$ C")

Remarque : si la FNBC est respectée, la 3FN est également respectée

Preuve : on suppose la 1FN respectée ; la 2FN est vérifiée car la FNBC impose la clé entière pour les dépendances fonctionnelles élémentaires ; les dépendances fonctionnelles doivent être directes car sinon on aurait A' $\rightarrow$ A" (en plus de C $\rightarrow$ A') une dépendance fonctionnelle dont des attributs hors clé déterminent d'autres attributs hors clé

Exemple

(NoINE , IntitAbrege , Annee) de clé (NoINE , IntitAbrege) car Annee $\not\rightarrow$ NoINE,IntitAbrege et NoINE $\not\rightarrow$ IntitAbrege,Annee et IntitAbrege $\not\rightarrow$ NoINE,Annee et NoINE,Annee $\not\rightarrow$ IntitAbrege et IntitAbrege,Annee $\not\rightarrow$ NoINE

FNBC (2/3)

Contre-exemple

La relation Élèves_Matières (Élève , Matière , Professeur) de clé (Élève , Matière) d'extension

Élève	Matière	Professeur
x	'Maths'	1
x	'Physique'	2
y	'Maths'	1
y	'Physique'	3

avec Élève,Matière → Professeur et Professeur → Matière, est en 3FN mais pas en FNBC et, si on supprime le tuple (y,'Physique',3), on perd l'information que le professeur 3 enseigne la physique ; il suffit alors de créer deux relations Élèves_Professeurs (Élève , Professeur) de clé (Élève , Professeur) d'extension

Élève	Professeur
x	1
x	2
y	1
y	3

dont on peut supprimer le tuple (y,3) et Professeurs (Professeur , Matière) de clé Professeur d'extension

Professeur	Matière
1	'Maths'
2	'Physique'
3	'Physique'

toutes deux en FNBC, préservant la dépendance fonctionnelle Professeur → Matière tandis que la dépendance fonctionnelle Élève,Matière → Professeur est perdue, même si la décomposition est SPI car Élèves_Professeurs ⋈ Professeurs voire Élèves_Professeurs $\overrightarrow{\bowtie}$ Professeurs = $\pi_{\text{Élève,Professeur,Matière}}$ (Élèves_Matières) d'extension

Élève	Professeur	Matière
x	1	'Maths'
x	2	'Physique'
y	1	'Maths'
y	3	'Physique'

éventuellement sans le tuple (y,3,'Physique') voire

Élève	Professeur	Matière
x	1	'Maths'
x	2	'Physique'
y	1	'Maths'
	3	'Physique'

Théorème : toute relation admet une décomposition en FNBC, SPI, mais qui ne préserve généralement pas les dépendances fonctionnelles

Définitions

Une relation R (A) est en FNBC (relativement à un ensemble de dépendances fonctionnelles élémentaires $\mathcal{F}$) si chaque dépendance fonctionnelle (non triviale) X → Y de $\mathcal{F}^+$ avec X $\subsetneqq$ Y $\subseteq$ A vérifie également X → A (on dit que X est une superclé de R)

Un schéma relationnel est en FNBC (relativement à un ensemble de dépendances fonctionnelles élémentaires) si toutes ses relations sont en FNBC

FNBC (3/3)

Exemples :

(X , Y) sans dépendance fonctionnelle est en FNBC car rien ne l'interdit

(X , Y) avec X $\rightarrow$ Y est en FNBC car X $\rightarrow$ X,Y

(X , Y) avec X $\rightarrow$ Y et Y $\rightarrow$ X est en FNBC car X $\rightarrow$ X,Y et Y $\rightarrow$ X,Y

(X , Y , Z) avec X,Y $\rightarrow$ Z est en FNBC car X,Y $\rightarrow$ X,Y,Z

(X , Y , Z) avec X $\rightarrow$ Y,Z est en FNBC car X $\rightarrow$ X,Y,Z

Contre-exemples :

(X , Y , Z) avec X $\rightarrow$ Y n'est pas en FNBC car X $\not\rightarrow$ Z

(X , Y , Z) avec X $\rightarrow$ Y et Y $\rightarrow$ Z n'est pas en FNBC car Y $\not\rightarrow$ X (même si X $\rightarrow$ X,Y,Z)

(X , Y , Z , T) avec X $\rightarrow$ Y et Z $\rightarrow$ T n'est pas en FNBC car X $\not\rightarrow$ Z,T ou encore car Z $\not\rightarrow$ X,Y

Algorithme de décomposition d'un ensemble de dépendances fonctionnelles en un ensemble de relations en FNBC

Fonction DecompoDFenFNBC($\mathcal{F}$,$\mathcal{A}$) : $\mathcal{S}$
// [E/] $\mathcal{F}$: un ensemble de dépendances fonctionnelles (élémentaires ou non)
// [E/] $\mathcal{A}$: un ensemble d'attributs
// [S/] $\mathcal{S}$: un ensemble de relations décomposées en FNBC,
avec certaines des clés primaires
// $\mathcal{C}$: une couverture minimale de $\mathcal{F}$
// $\mathcal{B}$: ensemble des attributs de $\mathcal{F}$ et de $\mathcal{A}$
// $\mathcal{C}^+$: la fermeture transitive de $\mathcal{C}$ // non indispensable pour l'algorithme
// 1. initialisations :
couverture minimale / propriété d'union, attributs, fermeture transitive
$\mathcal{C} \longleftarrow$ CouvMinDF($\mathcal{F}$)
Pour chaque X $\rightarrow$ Y $\in \mathcal{C} \wedge$ X $\rightarrow$ Z $\in \mathcal{C}$ avec Y $\neq$ Z faire
$\mathcal{C} \longleftarrow \mathcal{C} \setminus \{$ X $\rightarrow$ Y , X $\rightarrow$ Z $\} \cup \{$ X $\rightarrow$ Y$\cup$Z $\}$ // i. e. $\mathcal{C} \longleftarrow \ldots \cup \{$ X $\rightarrow$ Y , Z $\}$
$\mathcal{B} \longleftarrow \mathcal{A}$ // initialisation avec les attributs de $\mathcal{A}$
Pour chaque X_1 , ..., $X_k \rightarrow Y_1$, ..., $Y_l \in \mathcal{F}$ faire // ajout des attributs de $\mathcal{F}$
$\mathcal{B} \longleftarrow \mathcal{B} \cup \{ X_1 , \ldots, X_k , Y_1 , \ldots, Y_l \}$
$\mathcal{C}^+ \longleftarrow \emptyset$
Pour chaque X $\rightarrow$ Y $\in \mathcal{C}$ faire
// ajout de la dépendance fonctionnelle ... en remplaçant Y
// par tous les attributs atteignables depuis X
// ce qui revient à appliquer la réflexivité et la transitivité
$\mathcal{C}^+ \longleftarrow \mathcal{C}^+ \cup \{$ X $\rightarrow$ FermTransAttr($\mathcal{C}$,X) $\}$
// donc X $\rightarrow$ Y devient X $\rightarrow$ Z avec (X$\cup$Y)$\subseteq$Z
// 2. création des relations
$\mathcal{S} \longleftarrow \{ \mathcal{B}$ ($\mathcal{B}$) $\}$ // on part de la relation universelle
Tant que $\exists$ XYZ (X , Y , Z) $\in \mathcal{S}$ non en FNBC
c.-à-d. $\exists$ X $\rightarrow$ X,Y $\in \mathcal{C}^+ \wedge$ X $\rightarrow$ Z $\notin \mathcal{C}^+$
i. e. (X$\cup$Y$\cup$Z)$\setminus$FermTransAttr($\mathcal{C}$,X) = Z $\neq \emptyset$ faire
// théorème de décomposition SPI car
// XYZ = XY $\bowtie$ XZ (i. e. = XY $\bowtie_{XY.X=XZ.X}$ XZ)
$\mathcal{S} \longleftarrow \mathcal{S} \setminus \{$ XYZ (X , Y , Z) $\} \cup \{$ XY (X , Y) de clé X , XZ (X , Z) $\}$
Retourner $\mathcal{S}$

Exemples de DecompoDFenFNBC() (1/2)

Premier exemple de DecompoDFenFNBC()

Soit $\mathcal{F}$ = { a → b, d → a } et $\mathcal{A}$ = { c }

À l'issue de l'étape 1, on a $\mathcal{C} = \mathcal{F}$, $\mathcal{B}$ = { a,b,c,d } et $\mathcal{C}^+$ = { a → a,b ; d → d,a,b }

L'étape 2 commence par constituer $\mathcal{S}$ = { abcd (a , b , c , d) } qui n'est pas en FNBC d'une part car a → a,b mais a ↛ c,d et d'autre part car d → d,a,b mais d ↛ c ; considérons ces deux alternatives

Première alternative : X={a}. Alors, $\mathcal{S}$ = { ab (a , b) de clé a ; acd (a , c , d) } n'est toujours pas en FNBC à cause de la (seule) relation acd puisque d → d,a mais d ↛ c. Ainsi, $\mathcal{S}$ = { ab (a , b) de clé a ; da (d , a) de clé d ; dc (d , c) } est en FNBC

Seconde alternative : X={d}. Alors, $\mathcal{S}$ = { dab (d , a , b) de clé d ; dc (d , c) } n'est toujours pas en FNBC à cause de la (seule) relation dab puisque a → a,b mais a ↛ d. Ainsi, $\mathcal{S}$ = { ab (a , b) de clé a ; ad (a , d) de clé d ; dc (d , c) } (car on peut conserver la clé d de dab pour ad) est en FNBC

Ainsi, pour cet exemple, les deux alternatives conduisent au même ensemble de relations

Pour illustrer cet exemple, on peut prendre a = Banque, b = BanqueAdresse, c = Client et d = Prêt

Deuxième exemple de DecompoDFenFNBC()

Soit $\mathcal{F}$ = { a → b, b → c, c → b } et $\mathcal{A} = \emptyset$

À l'issue de l'étape 1, on a $\mathcal{C} = \mathcal{F}$, $\mathcal{B}$ = { a,b,c } et $\mathcal{C}^+$ = { a → a,b,c ; b → b,c ; c → c,b }

L'étape 2 commence par constituer $\mathcal{S}$ = { abc (a , b , c) } qui n'est pas en FNBC d'une part car b → b,c mais b ↛ a et d'autre part car c → c,b mais c ↛ a ; étudions ces deux alternatives

Première alternative : X={b}. Alors, $\mathcal{S}$ = { bc (b , c) de clé b ; ba (b , a) } est en FNBC

Seconde alternative : X={c}. Alors, $\mathcal{S}$ = { cb (c , b) de clé c ; ca (c , a) } est en FNBC

Dans cet exemple, les deux alternatives conduisent à deux ensembles de relations différents, même s'ils modélisent les données du même système d'information

Exemples de DecompoDFenFNBC() (2/2)

Troisième exemple de DecompoDFenFNBC() pour l'exemple « jouet »

Soit $\mathcal{F}$ = { IntitAbrege → IntitComplet et $\mathcal{A} = \emptyset$
IntitComplet → IntitAbrege
NoINE → NomEtu,DepartNaissEtu
NoINE,IntitAbrege → Annee
NoImmatChiffres,NoImmatLettres,NoImmatDepart →
NoINE,Couleur }

À l'issue de l'étape 1, on a $\mathcal{C} = \mathcal{F}$, $\mathcal{B}$ = { IntitAbrege , IntitComplet , NoINE , NomEtu , DepartNaissEtu , Annee , NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur }

et $\mathcal{C}^+$ = { IntitAbrege → IntitAbrege,IntitComplet
IntitComplet → IntitComplet,IntitAbrege
NoINE → NoINE,NomEtu,DepartNaissEtu
NoINE,IntitAbrege →
NoINE,IntitAbrege,Annee,IntitComplet,NomEtu,DepartNaissEtu
NoImmatChiffres,NoImmatLettres,NoImmatDepart →
NoImmatChiffres,NoImmatLettres,NoImmatDepart,
NoINE,Couleur,NomEtu,DepartNaissEtu }

Voici le déroulement de l'étape 2 :

$\mathcal{S}$ = { ExempleJouet (IntitAbrege , IntitComplet , NoINE , NomEtu , DepartNaissEtu , Annee , NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur) } qui n'est pas en FNBC notamment car IntitAbrege → IntitAbrege,IntitComplet mais IntitAbrege $\not\rightarrow$ NoINE,NomEtu,DepartNaissEtu,Annee,NoImmatChiffres,NoImmatLettres, NoImmatDepart,Couleur

$\mathcal{S}$ = { Diplomes (IntitAbrege , IntitComplet) de clé IntitAbrege , R' (IntitAbrege , NoINE , NomEtu , DepartNaissEtu , Annee , NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur) } dont R' n'est pas en FNBC en particulier car NoINE → NoINE,NomEtu,DepartNaissEtu mais NoINE $\not\rightarrow$ IntitAbrege,Annee,NoImmatChiffres,NoImmatLettres,NoImmatDepart,Couleur

$\mathcal{S}$ = { Diplomes (IntitComplet , IntitAbrege) de clé IntitAbrege , Etudiants (NoINE , NomEtu , DepartNaissEtu) de clé NoINE , R" (NoINE , IntitAbrege , Annee , NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur) } dont R" n'est pas en FNBC puisque NoINE,IntitAbrege → NoINE,IntitAbrege,Annee mais NoINE,IntitAbrege $\not\rightarrow$ NoImmatChiffres,NoImmatLettres,NoImmatDepart,Couleur

$\mathcal{S}$ = { Diplomes (IntitComplet , IntitAbrege) de clé IntitAbrege , Etudiants (NoINE , NomEtu , DepartNaissEtu) de clé NoINE , AvoirObtenu (NoINE , IntitAbrege , Annee) de clé (NoINE,IntitAbrege) , R"' (IntitAbrege , NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur) } dont R"' n'est pas en FNBC avec NoImmatChiffres,NoImmatLettres,NoImmatDepart → NoImmatChiffres,NoImmatLettres,NoImmatDepart,NoINE,Couleur mais NoImmatChiffres,NoImmatLettres,NoImmatDepart $\not\rightarrow$ IntitAbrege

$\mathcal{S}$ = { Diplomes (IntitComplet , IntitAbrege) de clé IntitAbrege , Etudiants (NoINE , NomEtu , DepartNaissEtu) de clé NoINE , AvoirObtenu (NoINE , IntitAbrege , Annee) de clé (NoINE,IntitAbrege) , Voitures (NoImmatChiffres , NoImmatLettres , NoImmatDepart , NoINE , Couleur) de clé (NoImmatChiffres,NoImmatLettres,NoImmatDepart) , R"" (NoImmatChiffres , NoImmatLettres , NoImmatDepart , IntitAbrege) } en FNBC

Quatrième forme normale (4FN) (1/2)

Définition

Les seules dépendances fonctionnelles multivaluées élémentaires sont celles dont un sur-ensemble de la clé (i. e. une superclé) multidétermine un attribut

Corollaire : une relation R n'est pas en 4FN s'il existe $X \twoheadrightarrow Y$ où $X \not\supseteq$ la clé de R (par exemple, X contient un attribut ne faisant pas partie de la clé de R ou encore parce que X est une partie stricte de la clé de R)

Remarque : si la 4FN est respectée, la FNBC est également respectée

Remarque : R (X , Y , Z) $\wedge$ $X \twoheadrightarrow Y$ peut être remplacé par $\begin{cases} \text{R' (X , Y) de clé X} \\ \text{R" (X , Z) de clé X} \\ \text{R'.X = R".X} \end{cases}$

Exemple

La relation R (X , Y , Z) de clé (X , Y , Z) d'extension

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	b
γ	2	a

est en 4FN car il n'y a aucune dépendance fonctionnelle multivaluée (ni $X \not\twoheadrightarrow Y$ et $X \not\twoheadrightarrow Z$ car $(\alpha, 2, a) \notin$ R, ni $Y \not\twoheadrightarrow X$ et $Y \not\twoheadrightarrow Z$ car $(\beta, 1, a) \notin$ R, et ni $Z \not\twoheadrightarrow X$ et $Z \not\twoheadrightarrow Y$ car $(\beta, 2, b) \notin$ R)

Premier contre-exemple

La relation R (X , Y , Z) de clé (Y , Z) d'extension

X	Y	Z
α	1	a
α	1	b
α	2	a
α	2	b

n'est pas en 4FN car $X \twoheadrightarrow Y$ a pour membre gauche X qui ne fait pas partie de la clé (Y , Z) de R ou encore parce que $Y \twoheadrightarrow X$ a pour membre gauche Y une partie stricte de la clé (Y , Z) de R

4FN (2/2)

Second contre-exemple

La relation ÉTUDIANTS (N°Étd , Cours , Sport) de clé (N°Étd , Cours , Sport) d'extension

N°Étd	Cours	Sport
9	'Bases de données'	'Tennis'
9	'Bases de données'	'Handball'
4	'Bases de données'	'Handball'
4	'Mathématiques'	'Handball'
4	'Gestion'	'Handball'
4	'Bases de données'	'Natation'
4	'Mathématiques'	'Natation'
4	'Gestion'	'Natation'

n'est pas en 4FN car N°Étd $\twoheadrightarrow$ Cours (et donc N°Étd $\twoheadrightarrow$ Sport) a pour membre gauche N°Étd une partie stricte de la clé (N°Étd , Cours , Sport) de ÉTUDIANTS ; une solution consiste alors à créer les relations ÉTUDIANTS_COURS (N°Étd , Cours) de clé (N°Étd , Cours) d'extension

N°Étd	Cours
9	'Bases de données'
4	'Bases de données'
4	'Mathématiques'
4	'Gestion'

et ÉTUDIANTS_SPORTS (N°Étd , Sport) de clé (N°Étd , Sport) d'extension

N°Étd	Sport
9	'Tennis'
9	'Handball'
4	'Handball'
4	'Natation'

, les dépendances fonctionnelles multivaluées étant perdues, même si la décomposition est SPI car ÉTUDIANTS_COURS $\bowtie$ ÉTUDIANTS_SPORTS = ÉTUDIANTS

Théorème : toute relation admet une décomposition (pas forcément unique) en 4FN, SPI, mais qui ne préserve généralement pas les dépendances fonctionnelles (autres que celles multivaluées)

Cinquième forme normale (5FN) ou forme normale de projection-jointure *(project-join normal form (JD/NF))*

Définition

Toute dépendance fonctionnelle de jointure *{$S_1, \ldots, S_m$} d'une relation R est triviale ou tout S_i est un sur-ensemble de la clé (i. e. une superclé) de R

Remarque : 5FN généralise 4FN (i. e. si la 5FN est respectée, la 4FN est également respectée) car les dépendances fonctionnelles de jointure généralisent les dépendances fonctionnelles multivaluées

Exemple

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	b
γ	2	a

de clé (X , Y , Z) est en 5FN car cette relation n'a aucune dépendance fonctionnelle de jointure autre que *{{X,Y,Z}}

Premier contre-exemple

R

X	Y	Z
α	1	a
α	1	b
α	2	b
β	1	b

de clé (X , Y , Z) est en 4FN mais n'est pas en 5FN car elle a pour dépendance fonctionnelle de jointure *{{X,Y},{Y,Z},{Z,X}} qui d'une part n'est pas triviale car {X,Y} $\neq$ {X,Y,Z} $\wedge$ {Y,Z} $\neq$ {X,Y,Z} $\wedge$ {Z,X} $\neq$ {X,Y,Z} (où {X,Y,Z} est l'ensemble des attributs de R) et d'autre part car {X,Y} (ni même {Y,Z} ou encore {Z,X} d'ailleurs) $\not\supseteq$ {X,Y,Z} (où {X,Y,Z} est la clé de R) ; R peut donc être décomposée SPI en trois relations $\pi_{X,Y}(R)$, $\pi_{Y,Z}(R)$ et $\pi_{Z,X}(R)$

Pour illustrer cet exemple, on peut prendre R = BUVCRU, (X , Y , Z) = (Buveur , Cru , Producteur), $(\alpha, \beta) = (Ronald, Claude)$, $(1, 2) = (Chablis, Volnay)$ et $(a, b) = (Georges, Nicolas)$

Second contre-exemple

R

X	Y	Z
α	1	a
α	2	b
β	1	a
γ	2	b

de clé (X , Y) a pour dépendances fonctionnelles de jointure *{{X,Y},{Y,Z}}, *{{X,Z},{Y,Z}} et *{{X,Y},{Y,Z},{X,Z}} ; R n'est donc pas en 5FN car {Y,Z} $\not\subseteq$ {X,Y} (i. e. la clé de R)

Théorème : une relation qui n'est pas en 5FN peut être décomposée, SPI, et suit les dépendances fonctionnelles de jointure, mais ne préserve généralement pas les autres dépendances fonctionnelles

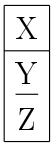
Théorème : toute relation en 5FN ne peut plus être décomposée SPI

Remarque : il existe d'autres formes normales (décompositions horizontales, dépendances algébriques, etc.) mais de moindre intérêt pour la conception d'une BD

Relationnel ⇝ entité-association (1/3)

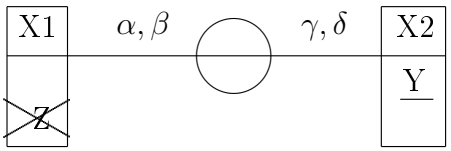
(passage d'un schéma relationnel à un schéma entité-association)

Transformation de toutes les relations en types d'entités

X (Y , Z) Y est la clé de la relation X	devient	

La relation X devient le type d'entités X, les attributs de la clé Y deviennent les types de propriétés identifiants Y, les attributs hors clé Z deviennent les types de propriétés hors identifiants Z

Transformation de toutes les contraintes d'intégrité référentielles en types d'associations

X1.Z référence X2.Y	devient	

avec

$$\alpha = \begin{cases} 1 & \text{si la valeur d'indétermination est interdite pour X1.Z} \\ & \text{(attribut obligatoire)} \\ 0 & \text{si la valeur d'indétermination est autorisée pour X1.Z} \\ & \text{(attribut facultatif)} \end{cases}$$

$\beta = 1$ car X1.Z admet au plus une valeur (comme tout attribut)

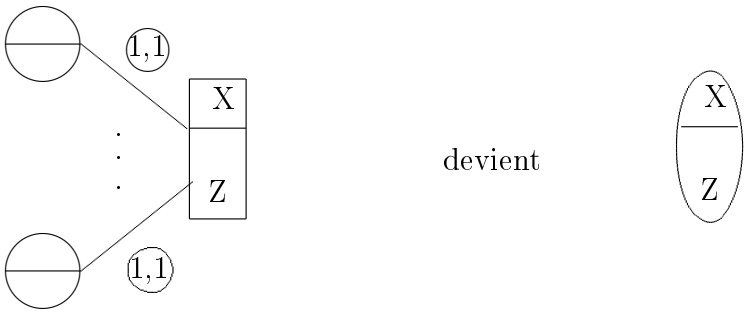
$$\gamma = \begin{cases} 1 & \text{si valeurs(X1.Z)} = \text{valeurs(X2.Y)} \\ 0 & \text{si valeurs(X1.Z)} \subsetneqq \text{valeurs(X2.Y)} \end{cases}$$

$$\delta = \begin{cases} 1 & \text{s'il existe une contrainte d'unicité sur X1.Z} \\ n & \text{s'il n'existe pas de contrainte d'unicité sur X1.Z} \end{cases}$$

De plus, si X1.Z fait partie de la clé de X1, alors α, β est un lien identifiant (on a $\alpha = 1$ car tout attribut faisant partie de la clé interdit la valeur d'indétermination i. e. est obligatoire

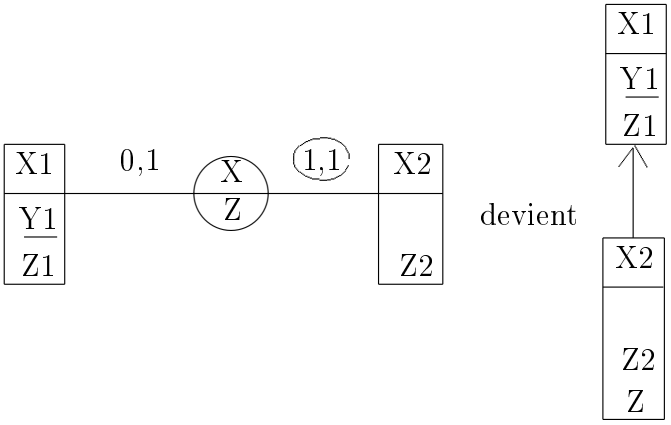
L'attribut clé étrangère X1.Z qui référence la clé primaire X2.Y est supprimé au profit d'un type d'associations de liens $1:1$ ou $1:n$ (selon la valeur de δ)

Amélioration du schéma entité-association (1/2)

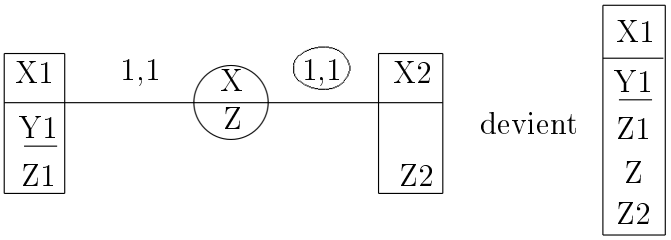


Si un type d'entités n'a pas de propriétés identifiantes propres et que tous (et au moins au nombre de deux) les liens sortant sont des liens identifiants, alors le type d'entités peut être transformé en type d'associations

Amélioration du schéma entité-association (2/2)



Si un type d'entités n'a pas de propriétés identifiantes propres et que le seul lien sortant est un lien identifiant et de cardinalité $0,1$ vers l'autre type d'entités, alors le type d'entités peut être remplacé par un sous-type d'entités



Si un type d'entités n'a pas de propriétés identifiantes propres et que le seul lien sortant est un lien identifiant et de cardinalité $1,1$ vers l'autre type d'entités, alors les deux types d'entités peuvent être fusionnés

Contraintes d'intégrité restantes

Les contraintes d'intégrité restantes sont ajoutées au schéma entité-association, sous forme graphique ou textuelle

Exercice : *Appliquez l'algorithme de rétro-conception des données au schéma relationnel suivant*

`W ( a , b , c )`
`X ( d , e )`
`Y ( f , g )`
`Z ( h , i )`
`a` est la clé de `W`
`d` est la clé de `X`
`(f,g)` est la clé de `Y`
`h` est la clé de `Z`
`c` référence `d`
`f` référence `d`
`g` référence `h`
`c` est le seul attribut autorisant la valeur d'indétermination
`valeurs(c) = valeurs(d)`
`valeurs(g) = valeurs(h)`
Unicité de `b`

Solution pas à pas

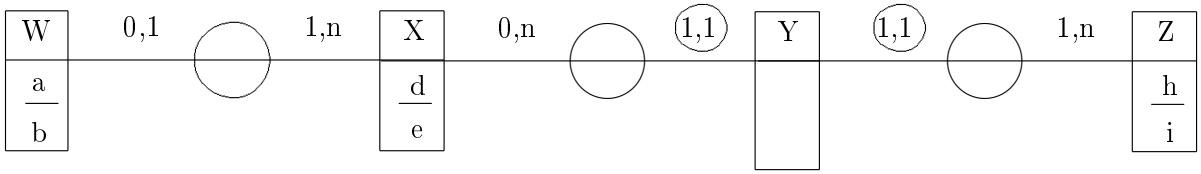
Les relations deviennent des types d'entités

Une première contrainte d'intégrité référentielle (`c` référence `d`) devient un type d'associations

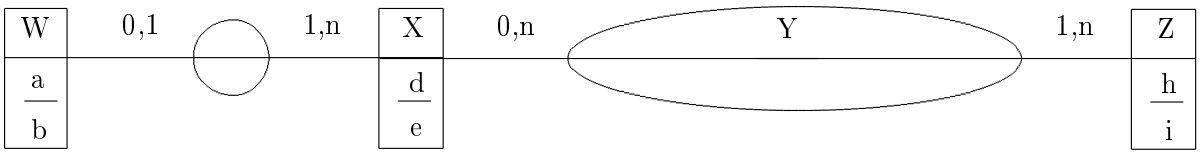
On a $\alpha = 0$ car `c` est facultatif, $\beta = 1$, $\gamma = 1$ car `valeurs(c) = valeurs(d)`, et $\delta = n$ (car il n'a pas unicité de `c`)

Les deux dernières contraintes d'intégrité référentielles (`f` référence `d` et `g` référence `h`) deviennent des types d'associations

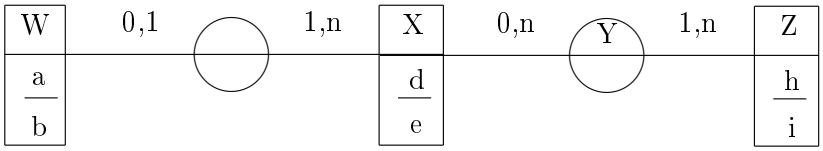
On a $\alpha = 1$ et $\beta = 1$ pour les deux liens sortant de `Y` car `f` et `g` sont tous deux obligatoires, les deux liens sortant de `Y` sont des liens identifiants car `f` et `g` font tous deux partie de la clé primaire de `Y`, $\gamma = 0$ du côté de `X` car `valeurs(f)` $\subsetneq$ `valeurs(d)` tandis que $\gamma = 1$ du côté de `Z` car `valeurs(g) = valeurs(h)`, et $\delta = n$ des deux côtés (car il n'a pas unicité ni de `f` ni de `g` puisque c'est le couple qui est unique et non chacune de ses composantes)



Amélioration du schéma entité-association en remplaçant un type d'entités sans propriétés identifiantes propres et dont les deux liens sortant sont des liens identifiants en type d'associations



Ajout des contraintes d'intégrité restantes



Contrainte d'intégrité supplémentaire : unicité de b

σ , π , $\bowtie$, $\times$, $\cup$, $\cap$, $\setminus$, $\div$, AGG

Origine

1970 : E. F. CODD, *A relational model for large shared data banks*

Objet

Modèle statique des traitements

Principe

S'appuie sur des données stockées sous forme de relations

À la base de nombreux langages de requêtes (L4G)

Remarques

Le résultat de toute opération est une nouvelle relation

Toutes les opérations s'appliquent à l'ensemble des tuples des relations

Notations

$R(X) = R(X_1, X_2, \cdots, X_n)$ est le schéma de la relation R

$S(Y) = S(Y_1, Y_2, \cdots, Y_m\ )$ est le schéma de la relation S

X' est un sous-ensemble de X

Y' est un sous-ensemble de Y

Z est l'ensemble des attributs communs à R et à S (i. e. $Z = X \cap Y$)

V est l'ensemble des attributs de R sauf ceux communs avec S (i. e. $V = X \setminus Z$)

W est l'ensemble des attributs de S sauf ceux communs avec R (i. e. $W = Y \setminus Z$)

θ est une condition

$domaine(A)$ est le domaine de l'attribut A

$card(R)$ est le cardinal de la relation R i. e. son nombre de tuples

Remarque

Les algorithmes proposés ci-après sont souvent qualifiés de naïfs (et parcourent séquentiellement les relations) ; en fait, on pourrait utiliser des algorithmes de tri/fusion, les index, ...ou encore laisser l'optimiseur effectuer son travail

Définition

L'arité est le nombre de relations qu'une opération manipule ; on parle d'opération unaire, binaire, etc.

Nouvelles relations (schéma en extension)

Enseignants

NoEns	NomEns	DepartNaissEns
5	'DURAND'	33
1	'LACOUR'	24
4	'ROUX'	40

// exactement même structure que Etudiants

Conduire

NoINE	NoImmatChiffres	NoImmatLettres	NoImmatDepart	
5	3333	'BX'	33	
8	3333	'BX'	33	// étudiant ∄
2	2424	'PX'	24	// voiture ∄
5	2424	'PX'	24	// voiture ∄

Formations

Ville	IntitAbrege
'BORDEAUX'	'BAC'
'BORDEAUX'	'DUT'
'BORDEAUX'	'DEUG'
'BORDEAUX'	'MIAGe'
'AGEN'	'BAC'
'AGEN'	'DUT'
'AGEN'	'DEUG'

Nouvelles contraintes d'intégrité

NoEns est la clé de la nouvelle relation Enseignants (NoEns , NomEns , DepartNaissEns)

(NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart) est la clé de la nouvelle relation Conduire (NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart)

Conduire.NoINE référence Etudiants.NoINE

> En fait, l'étudiant n^o 8 viole cette contrainte d'intégrité référentielle car il n'appartient pas à la relation Etudiants

Conduire.NoImmatChiffres,Conduire.NoImmatLettres,Conduire.NoImmatDepart référence Voitures.NoImmatChiffres,Voitures.NoImmatLettres,Voitures.NoImmatDepart

> En fait, la voiture immatriculée 2424 PX 24 viole cette contrainte d'intégrité référentielle car elle n'appartient pas à la relation Voitures

(Ville , IntitAbrege) est la clé de la nouvelle relation Formations (Ville , IntitAbrege)

Formations.IntitAbrege référence Diplomes.IntitAbrege

Remarque

La nouvelle BD de l'exemple « jouet » est donc incohérente car deux contraintes d'intégrité référentielles ne sont pas vérifiées ; nous avons cependant besoin de ces valeurs pour illustrer certaines opérations

Opérations ensemblistes (1/3)

Produit cartésien $\times$

Le résultat de l'opération (binaire) $R \times S$ est constitué de tous les $card(R) * card(S)$ tuples d'attributs $X1, X2, \ldots, Xn, Y1, Y2, \ldots, Ym$ obtenus par concaténation des tuples de R et de S

Exemple : *(tous) les étudiants avec pour chacun (toutes) les voitures*

Etudiants $\times$ Voitures

Etudiants.NoINE	NomEtu	DepartNaissEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart	Couleur	Voitures.NoINE
5	'DURAND'	33	3333	'BX'	33	'rouge'	5
5	'DURAND'	33	4747	'LA'	47	'rouge'	4
5	'DURAND'	33	4040	'NT'	40	'jaune'	5
2	'LEROI'	40	3333	'BX'	33	'rouge'	5
2	'LEROI'	40	4747	'LA'	47	'rouge'	4
2	'LEROI'	40	4040	'NT'	40	'jaune'	5
4	'MARTIN'	47	3333	'BX'	33	'rouge'	5
4	'MARTIN'	47	4747	'LA'	47	'rouge'	4
4	'MARTIN'	47	4040	'NT'	40	'jaune'	5
7	'LEROI'	33	3333	'BX'	33	'rouge'	5
7	'LEROI'	33	4747	'LA'	47	'rouge'	4
7	'LEROI'	33	4040	'NT'	40	'jaune'	5
3	'DUPOND'	17	3333	'BX'	33	'rouge'	5
3	'DUPOND'	17	4747	'LA'	47	'rouge'	4
3	'DUPOND'	17	4040	'NT'	40	'jaune'	5

Algorithme :

```
Pour chaque r ∈ R faire
    Pour chaque s ∈ S faire
        Écrire r,s
```

Union $\cup$

Le résultat de l'opération (binaire) $R \cup S$ est constitué des tuples appartenant à R ou à S, et en éliminant les doublons (sur les tuples entiers) éventuels

Remarque : $R \cup S = (R \cap S) \uplus (R \setminus S) \uplus (S \setminus R)$ où $\uplus$ désigne l'union disjointe

Exemple : *les enseignants et les étudiants*

Enseignants $\cup$ Etudiants

5	'DURAND'	33
1	'LACOUR'	24
4	'ROUX'	40
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17

Algorithme naïf :

```
// r ∈ R
Pour chaque r ∈ R faire
    Écrire r
// s ∈ S ∧ s ∉ R
Pour chaque s ∈ S faire
    PasTrouvé ⟵ VRAI
    Pour chaque r ∈ R et PasTrouvé faire
        Si r = s Alors
            PasTrouvé ⟵ FAUX
    Si PasTrouvé Alors
        Écrire s
```

Opérations ensemblistes (2/3)

Intersection $\cap$

Le résultat de l'opération (binaire) $R \cap S$ est constitué des tuples appartenant à la fois à R et à S

Exemple : *les enseignants et les étudiants qui ont exactement les mêmes caractéristiques*

Enseignants $\cap$ Etudiants

5	'DURAND'	33

Algorithme naïf :

// $r \in R \wedge r \in S$
Pour chaque $r \in R$ faire
    PasTrouvé $\longleftarrow$ VRAI
    Pour chaque $s \in S$ et PasTrouvé faire
        Si $s = r$ Alors
            Écrire r
            PasTrouvé $\longleftarrow$ FAUX

Différence $\setminus$

Le résultat de l'opération (binaire) $R \setminus S$ est constitué des tuples appartenant à R et n'appartenant pas à S

Exemple : *les enseignants ayant des caractéristiques différentes des étudiants*

Enseignants $\setminus$ Etudiants

1	'LACOUR'	24
4	'ROUX'	40

Algorithme naïf :

// $r \in R \wedge r \notin S$
Pour chaque $r \in R$ faire
    PasTrouvé $\longleftarrow$ VRAI
    Pour chaque $s \in S$ et PasTrouvé faire
        Si $s = r$ Alors
            PasTrouvé $\longleftarrow$ FAUX
    Si PasTrouvé Alors
        Écrire r

Remarque : cette opération n'est pas symétrique ; en revanche, on appelle différence symétrique le résultat de $(R \setminus S) \uplus (S \setminus R) = (R \cup S) \setminus (R \cap S)$

Remarque : pour les trois opérations union, intersection et différence, les domaines de chaque attribut des relations R et S en correspondance (deux à deux) doivent être identiques (ou compatibles) i. e. $domaine(X_i) = domaine(Y_i)$ pour tout $1 \leq i \leq n = m$

Complément $\neg$ (symbole non normalisé)

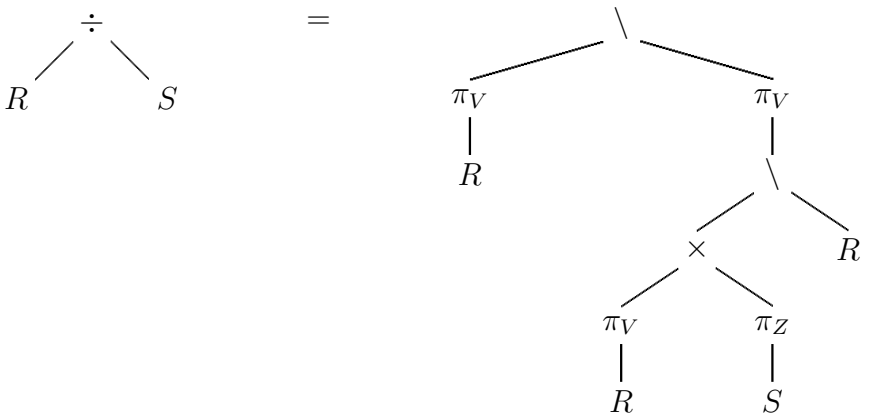
$\neg R = (domaine(X_1) \times \ldots \times domaine(X_n)) \setminus R$

Remarque : opération unaire

Division (ou quotient) ÷

Le résultat de l'opération (binaire) $R \div S$ est constitué des tuples r de R sur les attributs de V qui vérifient que pour tout tuple s de S sur les attributs de Z, le tuple rs (sur les attributs de X) appartient à R

Remarque : $R \div S = \pi_V(R) \setminus \pi_V((\pi_V(R) \times \pi_Z(S)) \setminus R)$



N. B. : on pourrait aussi considérer le graphe orienté acyclique (où chaque relation et chaque opération identique ne serait présente qu'une seule fois) plutôt que l'arbre

Exemple : *les villes formant à tous les diplômes*

Formations ÷ Diplomes

Ville
'BORDEAUX'

Algorithme naïf :

Pour chaque $r' \in R$ faire
// on cherche $r' \in R$ tel que $r'.Z \leq r.Z \ \forall r \in R / r.V = r'.V$
r'le+petit ⟵ VRAI
Pour chaque $r \in R$ et r'le+petit faire
Si $r.V = r'.V$ Alors
r'le+petit ⟵ $(r'.Z \leq r.Z)$ // vrai si et seulement si
// r' est le plus petit des Z ayant le même V
Si r'le+petit Alors
// on doit comparer tous les $s.Z$ de S avec tous les $r.Z$ de R tels que
// $r.V = r'.V$
sTousTrouvés ⟵ VRAI
Pour chaque $s \in S$ et sTousTrouvés faire
rPasTrouvé ⟵ VRAI
Pour chaque $r \in R$ et rPasTrouvé faire
Si $r.V = r'.V$ et $r.Z = s.Z$ Alors
rPasTrouvé ⟵ FAUX
Si rPasTrouvé Alors
sTousTrouvés ⟵ FAUX
Si sTousTrouvés Alors
Écrire $r.V$

Remarque : cette opération n'est pas symétrique

Projection π

Le résultat de l'opération (unaire) $\pi_{X'}(R)$ est constitué des tuples de R en ne conservant que les attributs de X', et en éliminant les doublons éventuels (restriction d'attributs i. e. de colonnes)

Exemple : *les couleurs des voitures*

$\pi_{\text{Couleur}}(\text{Voitures})$

Couleur
'rouge'
'jaune'

Algorithme naïf :

// $r \in R$ tels que $\nexists r' \in R / r.X' = r'.X' \wedge r'$ avant r
Rang1erDansR $\longleftarrow$ 0 // rang de r (premier) dans R
Pour chaque $r \in R$ faire
    Rang1erDansR $\longleftarrow$ Rang1erDansR + 1
    PasTrouvé $\longleftarrow$ VRAI
    Rang2ndDansR $\longleftarrow$ 1 // rang+1 de r' (second) dans R
    Pour chaque $r' \in R$ et Rang2ndDansR<Rang1erDansR et PasTrouvé faire
        Rang2ndDansR $\longleftarrow$ Rang2ndDansR + 1
        Si $r.X' = r'.X'$ Alors
            PasTrouvé $\longleftarrow$ FAUX
    Si PasTrouvé Alors
        Écrire $r.X'$

Sélection σ

Le résultat de l'opération (unaire) $\sigma_\theta(R)$ est constitué des tuples de R qui vérifient la condition θ (restriction de tuples i. e. de lignes)

Exemple : *les voitures rouges*

$\sigma_{\text{Couleur='rouge'}}(\text{Voitures})$

NoImmatChiffres	NoImmatLettres	NoImmatDepart	Couleur	NoINE
3333	'BX'	33	'rouge'	5
4747	'LA'	47	'rouge'	4

Exemple : *les couleurs et numéro d'étudiant des voitures rouge ou orange dont le numéro d'étudiant est 2, 4 ou 6*

$\pi_{\text{Couleur,NoINE}}(\sigma_{\text{Couleur}\in\{'rouge','orange'\}\wedge\text{NoINE}\in\{2,4,6\}}(\text{Voitures}))$

Couleur	NoINE
'rouge'	4

Algorithme :

Pour chaque $r \in R$ faire
    Si $r \models \theta$ Alors // où $\models$ est le symbole signifiant $satisfait$
        Écrire r

Remarque : la jointure est l'opération (binaire) fondamentale de l'algèbre relationnelle puisqu'elle permet en particulier d'associer deux relations, l'une sur sa clé primaire, l'autre sur sa clé étrangère ; cela revient à reconstituer l'information qui avait été éclatée lors de la normalisation (i. e. pour respecter les FN)

θ-jointure $\bowtie_\theta$

$R \bowtie_\theta S = \sigma_\theta(R \times S)$

Exemple : *les étudiants nés là où les voitures sont immatriculées*

Etudiants $\bowtie_{DepartNaissEtu=NoImmatDepart}$ Voitures

Etudiants.NoINE	NomEtu	DepartNaissEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart	Couleur	Voitures.NoINE
5	'DURAND'	33	3333	'BX'	33	'rouge'	5
2	'LEROI'	40	4040	'NT'	40	'jaune'	5
4	'MARTIN'	47	4747	'LA'	47	'rouge'	4
7	'LEROI'	33	3333	'BX'	33	'rouge'	5

Algorithme naïf :

Pour chaque $r \in R$ faire
  Pour chaque $s \in S$ faire
    Si $(r, s) \models \theta$ Alors
      Écrire r,s

Définitions

L'équijointure est un cas particulier de la θ-jointure, lorsque la condition θ est une égalité

L'auto-jointure est un cas particulier de la θ-jointure, lorsque $R = S$

Jointure $\bowtie$ (ou jointure naturelle)

Le résultat de l'opération $R \bowtie S$ est constitué des tuples

d'attributs de R et de S sans répétition (attributs de X et de Y où ceux de Z n'apparaissent qu'une fois)

obtenus en concaténant tout tuple de R avec tout tuple de S qui a même valeur que lui sur les attributs communs de R et de S (attributs de Z)

Remarque : $R \bowtie S = \pi_{X,W}(R \bowtie_{R.Z=S.Z} S)$

Exemple : *les voitures et pour chacune l'étudiant qui la possède*

Etudiants $\bowtie$ Voitures

NoINE	NomEtu	DepartNaissEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart	Couleur
5	'DURAND'	33	3333	'BX'	33	'rouge'
5	'DURAND'	33	4040	'NT'	40	'jaune'
4	'MARTIN'	47	4747	'LA'	47	'rouge'

Algorithme naïf :

Pour chaque $r \in R$ faire
  Pour chaque $s \in S$ faire
    Si $r.Z = s.Z$ Alors
      Écrire r,$s.W$

Semi-jointure gauche $\ltimes$

$R \ltimes S = \pi_X(R \bowtie S)$

Exemple : *les étudiants possédant (au moins) une voiture*
Etudiants $\ltimes$ Voitures

NoINE	NomEtu	DepartNaissEtu
5	'DURAND'	33
4	'MARTIN'	47

Algorithme naïf :

// $r \in R \wedge r.Z \in S.Z$
Pour chaque $r \in R$ faire
PasTrouvé $\longleftarrow$ VRAI
Pour chaque $s \in S$ et PasTrouvé faire
Si $s.Z = r.Z$ Alors
Écrire r
PasTrouvé $\longleftarrow$ FAUX

Remarque : cette opération n'est pas symétrique

Semi-jointure droite $\rtimes$

$R \rtimes S = \pi_Y(R \bowtie S)$

Exemple : *les étudiants diplômés*
AvoirObtenu $\rtimes$ Etudiants

NoINE	NomEtu	DepartNaissEtu
5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
3	'DUPOND'	17

Algorithme naïf :

// $s \in S \wedge s.Z \in R.Z$
Pour chaque $s \in S$ faire
PasTrouvé $\longleftarrow$ VRAI
Pour chaque $r \in R$ et PasTrouvé faire
Si $r.Z = s.Z$ Alors
Écrire s
PasTrouvé $\longleftarrow$ FAUX

Remarque : cette opération n'est pas symétrique

Jointure externe à gauche $\overset{\leftarrow}{\bowtie}$ (symbole non normalisé)

Le résultat de l'opération $R \overset{\leftarrow}{\bowtie} S$ est constitué des tuples d'attributs de R et de S sans répétition, obtenus en concaténant tout tuple de R avec tout tuple de S qui a même valeur que lui sur les attributs communs de R et de S mais avec des valeurs indéterminées s'il n'existe pas de tuple de S ayant même valeur que R sur les attributs communs

Remarque : $R \overset{\leftarrow}{\bowtie} S = (R \ltimes S) \uplus \sigma_{Z \notin \pi_Z(R \bowtie S)}(R)$

Exemple : *tous les étudiants, avec leurs voitures éventuelles*

Etudiants $\overset{\leftarrow}{\bowtie}$ Voitures

NoINE	NomEtu	DepartNaissEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart	Couleur
5	'DURAND'	33	3333	'BX'	33	'rouge'
5	'DURAND'	33	4040	'NT'	40	'jaune'
2	'LEROI'	40				
4	'MARTIN'	47	4747	'LA'	47	'rouge'
7	'LEROI'	33				
3	'DUPOND'	17				

Algorithme naïf :

Pour chaque $r \in R$ faire
  PasTrouvé $\longleftarrow$ VRAI
  Pour chaque $s \in S$ faire
    Si $r.Z = s.Z$ Alors
      Écrire r,$s.W$
      PasTrouvé $\longleftarrow$ FAUX
  Si PasTrouvé Alors
    Écrire r,$\underbrace{\texttt{NULL}, \ldots, \texttt{NULL}}_{card(W) fois}$

Remarque : cette opération n'est pas symétrique

Jointure externe à droite $\overset{\rightarrow}{\bowtie}$ (symbole non normalisé)

$R \overset{\rightarrow}{\bowtie} S = \pi_{V,Y}(S \overset{\leftarrow}{\bowtie} R)$

Remarque : $R \overset{\rightarrow}{\bowtie} S = (R \ltimes S) \uplus \sigma_{Z \notin \pi_Z(R \bowtie S)}(S)$

Exemple : *toutes les voitures pouvant être conduites, avec leurs conducteurs éventuels*

Etudiants $\overset{\rightarrow}{\bowtie}$ Conduire

NomEtu	DepartNaissEtu	NoINE	NoImmatChiffres	NoImmatLettres	NoImmatDepart
'DURAND'	33	5	3333	'BX'	33
		8	3333	'BX'	33
'LEROI'	40	2	2424	'PX'	24
'DURAND'	33	5	2424	'PX'	24

Algorithme naïf :

Pour chaque $s \in S$ faire
  PasTrouvé $\longleftarrow$ VRAI
  Pour chaque $r \in R$ faire
    Si $s.Z = r.Z$ Alors
      Écrire $r.V$,s
      PasTrouvé $\longleftarrow$ FAUX
  Si PasTrouvé Alors
    Écrire $\underbrace{\texttt{NULL}, \ldots, \texttt{NULL}}_{card(V) fois}$,$s$

Remarque : cette opération n'est pas symétrique

Jointure externe généralisée $\overset{\leftrightarrow}{\bowtie}$ (ou jointure externe complète)

(symbole non normalisé)

$R \overset{\leftrightarrow}{\bowtie} S = (R \overset{\leftarrow}{\bowtie} S) \cup (R \overset{\rightarrow}{\bowtie} S)$

Exemple : *les étudiants et les voitures pouvant être conduites, avec les étudiants qui ne conduisent aucune voiture et avec les voitures qui ne peuvent être conduites par aucun étudiant*

Etudiants $\overset{\leftrightarrow}{\bowtie}$ Conduire

NoINE	NomEtu	DepartNaissEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart
5	'DURAND'	33	3333	'BX'	33
5	'DURAND'	33	2424	'PX'	24
2	'LEROI'	40	2424	'PX'	24
4	'MARTIN'	47			
7	'LEROI'	33			
3	'DUPOND'	17			
8			3333	'BX'	33

Algorithme naïf :

// $R \overset{\leftarrow}{\bowtie} S$
Pour chaque $r \in R$ faire
  PasTrouvé $\longleftarrow$ VRAI
  Pour chaque $s \in S$ faire
    Si $r.Z = s.Z$ Alors
      Écrire r,$s.W$
      PasTrouvé $\longleftarrow$ FAUX
  Si PasTrouvé Alors
    Écrire r,$\underbrace{\texttt{NULL}, \ldots, \texttt{NULL}}_{card(W) fois}$

// $(R \overset{\rightarrow}{\bowtie} S) \setminus (R \bowtie S)$
Pour chaque $s \in S$ faire
  PasTrouvé $\longleftarrow$ VRAI
  Pour chaque $r \in R$ et PasTrouvé faire
    Si $s.Z = r.Z$ Alors
      PasTrouvé $\longleftarrow$ FAUX
  Si PasTrouvé Alors
    Écrire $\underbrace{\texttt{NULL}, \ldots, \texttt{NULL}}_{card(V) fois}$,$s$

Anti semi-jointure gauche *(left anti semi-join)* $\overline{\ltimes}$ (symbole non normalisé)

$R \overline{\ltimes} S = \pi_X((R \overleftarrow{\bowtie} S) \setminus (R \bowtie S))$

Exemple : *les étudiants sans voiture*
Etudiants $\overline{\ltimes}$ Voitures

NoINE	NomEtu	DepartNaissEtu
2	'LEROI'	40
7	'LEROI'	33
3	'DUPOND'	17

Algorithme naïf :

// $r \in R \wedge r.Z \notin S.Z$
Pour chaque $r \in R$ faire
  PasTrouvé $\longleftarrow$ VRAI
  Pour chaque $s \in S$ et PasTrouvé faire
    Si $s.Z = r.Z$ Alors
      PasTrouvé $\longleftarrow$ FAUX
  Si PasTrouvé Alors
    Écrire r

Remarque : cette opération n'est pas symétrique

Anti semi-jointure droite *(right anti semi-join)* $\overline{\rtimes}$ (symbole non normalisé)

$R \overline{\rtimes} S = \pi_Y((R \overrightarrow{\bowtie} S) \setminus (R \bowtie S))$

Exemple : *les étudiants sans diplôme*
AvoirObtenu $\overline{\rtimes}$ Etudiants

NoINE	NomEtu	DepartNaissEtu
7	'LEROI'	33

Algorithme naïf :

// $s \in S \wedge s.Z \notin R.Z$
Pour chaque $s \in S$ faire
  PasTrouvé $\longleftarrow$ VRAI
  Pour chaque $r \in R$ et PasTrouvé faire
    Si $r.Z = s.Z$ Alors
      PasTrouvé $\longleftarrow$ FAUX
  Si PasTrouvé Alors
    Écrire s

Remarque : cette opération n'est pas symétrique

Agrégation AGG (symbole non normalisé)

AGG (R ; [G] ; A [; W] [; H])

AGG : fonction d'agrégation (`COUNT`, `SUM`, `AVG`, `MIN` ou `MAX`)

R : relation

G : attributs de l'agrégation (i. e. du regroupement)

A : attribut pour lequel est calculée la fonction d'agrégation

W : condition de la sélection avant l'agrégation

H : condition de la sélection après l'agrégation

Exemple : *le nombre d'étudiants diplômés, pour chaque diplôme*
COUNT(AvoirObtenu;IntitAbrege;NoINE)

IntitAbrege	COUNT(NoINE)
'DUT'	2
'BAC'	4
'MIAGe'	2
'DEUG'	2

Exemple : *le plus petit des numéros INE des étudiants girondins*
MIN(Etudiants;;NoINE;DepartNaissEtu=33)

MIN(NoINE)
5

Exemple : *pour chaque étudiant ayant obtenu au moins deux diplômes autres que le baccalauréat, l'année du dernier diplôme*
MAX(AvoirObtenu;NoINE;Annee;IntitAbrege≠'BAC';COUNT(IntitAbrege)≥2)

NoINE	MAX(Annee)
4	1982
3	1985

Remarque : opération unaire

Éclatement *(split)* RESTRICT

$$\begin{cases} \text{R1} = \text{RESTRICT (R ,}\theta\text{)} \\ \text{R2} = \text{RESTRICT (R ,}\neg\theta\text{)} \end{cases}$$

Remarque : cette opération retourne deux relations comme résultat

Remarque : opération unaire

Fermeture transitive $^+$

R^+ est calculé par l'algorithme suivant :

// calcul de R^+ à partir de R composée de deux attributs ($n = 2$)
$\text{R}^+ \longleftarrow \text{R}$
Tant que $\exists a, b, c / (a, b) \in R \wedge (b, c) \in R \wedge (a, c) \notin R$ faire
$\quad \text{R}^+ \longleftarrow \text{R}^+ \cup \{(a, c)\}$

Exemple : R = (Parent , Enfant) et R^+ = ANCÊTRES (Ascendant , Descendant)

Remarque : cette opération n'a pas de définition en termes ensemblistes

Remarque : opération unaire

$\{(\ldots)/\exists \ldots : (\ldots) \in \ldots \wedge \ldots\}$

Objet

Modèle statique des traitements

Principes

Calcul des prédicats du premier ordre

S'appuie sur des données stockées sous forme de relations

Définitions

Atome

Constante ou variable

Exemples : 'LEROI', 33 (constantes) ou x, NomEtu (variables)

Prédicat (à plusieurs variables)

Égalité ($=$), différence ($\neq$), infériorité stricte ($<$) ou large ($\leq$), supériorité stricte ($>$) ou large ($\geq$), appartenance ($\in$) ou non appartenance ($\notin$), inclusion stricte ($\subsetneq$) ou large ($\subseteq$), etc.

Formule atomique

$p(a_1, \ldots, a_n)$
où p est un prédicat et les a_i sont des atomes

Exemple : NomEtu = 'LEROI'

Formule du calcul des prédicats du premier ordre (ou calcul relationnel)

f (formule), $\neg f$ (négation), $f \wedge g$ (et logique), $f \vee g$ (ou logique), $\forall x f(x)$ (quantificateur universel), $\exists x f(x)$ (quantificateur existentiel)
où x est une variable et f et g sont des formules atomiques ou du calcul des prédicats du premier ordre

Exemples (1/2)

Données

Etudiants = { (5 , 'DURAND' , 33) ,
(2 , 'LEROI' , 40) ,
(4 , 'MARTIN' , 47) ,
(7 , 'LEROI' , 33) ,
(3 , 'DUPOND' , 17) }

Voitures = { (3333 , 'BX' , 33 , 'rouge' , 5) ,
(4747 , 'LA' , 47 , 'rouge' , 4) ,
(4040 , 'NT' , 40 , 'jaune' , 5) }

Contraintes d'intégrité

Contrainte d'unicité

Le numéro d'étudiant est la clé primaire de Etudiants
$\forall(n, nom, dpt) \in \text{Etudiants} , \nexists(n, nom', dpt') \in \text{Etudiants} : (nom, dpt) \neq (nom', dpt')$

Contrainte d'intégrité référentielle

Toute voiture appartient à un (seul) étudiant
$\forall(i, j, k, c, \text{NoINE}) \in \text{Voitures} , \exists!(\text{NoINE}, n, d) \in \text{Etudiants}$

Requêtes d'interrogation (1/2)

Projection

Les couleurs des voitures
$\{(\text{Couleur})/\exists i, j, k, n : (i, j, k, \text{Couleur}, n) \in \text{Voitures}\}$
= { 'rouge' ,
'jaune' }

Sélection

Les voitures rouges
$\{(\text{NoImmatChiffres}, \text{NoImmatLettres}, \text{NoImmatDepart}, \text{Couleur}, \text{NoINE})/$
$(\text{NoImmatChiffres}, \text{NoImmatLettres}, \text{NoImmatDepart}, \text{Couleur}, \text{NoINE}) \in$
$\text{Voitures} \wedge \text{Couleur} = \text{'rouge'}\}$
$= \{(\text{NoImmatChiffres}, \text{NoImmatLettres}, \text{NoImmatDepart}, \text{'rouge'}, \text{NoINE})/$
$(\text{NoImmatChiffres}, \text{NoImmatLettres}, \text{NoImmatDepart}, \text{'rouge'}, \text{NoINE}) \in$
$\text{Voitures}\}$
= { (3333 , 'BX' , 33 , 'rouge' , 5) ,
(4747 , 'LA' , 47 , 'rouge' , 4) }

Jointure

Les étudiants et leurs voitures
$\{(\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}, \text{NoImmatChiffres}, \text{NoImmatLettres},$
$\text{NoImmatDepart}, \text{Couleur})/(\text{NoINE}, \text{NomEtu}, \text{DepartNaissEtu}) \in \text{Etudiants}$
$\wedge(\text{NoImmatChiffres}, \text{NoImmatLettres}, \text{NoImmatDepart}, \text{Couleur}, \text{NoINE}) \in$
$\text{Voitures}\}$
= { (5 , 'DURAND' , 33 , 3333 , 'BX' , 33 , 'rouge') ,
(5 , 'DURAND' , 33 , 4040 , 'NT' , 40 , 'jaune') ,
(4 , 'MARTIN' , 47 , 4747 , 'LA' , 47 , 'rouge') }

Exemples (2/2)

Requêtes d'interrogation (2/2)

Produit cartésien

Tous les étudiants et toutes les voitures

$\{(\text{NoINE\_Etudiants}, \text{NomEtu}, \text{DepartNaissEtu}, \text{NoImmatChiffres}, \text{NoImmatLettres}, \text{NoImmatDepart}, \text{Couleur}, \text{NoINE\_Voitures})/$
$(\text{NoINE\_Etudiants}, \text{NomEtu}, \text{DepartNaissEtu}) \in \text{Etudiants} \wedge$
$(\text{NoImmatChiffres}, \text{NoImmatLettres}, \text{NoImmatDepart}, \text{Couleur}, \text{NoINE\_Voitures}) \in$
$\text{Voitures}\}$

$=$ {
(5 , 'DURAND' , 33 , 3333 , 'BX' , 33 , 'rouge' , 5) ,
(5 , 'DURAND' , 33 , 4747 , 'LA' , 47 , 'rouge' , 4) ,
(5 , 'DURAND' , 33 , 4040 , 'NT' , 40 , 'jaune' , 5) ,
(2 , 'LEROI' , 40 , 3333 , 'BX' , 33 , 'rouge' , 5) ,
(2 , 'LEROI' , 40 , 4747 , 'LA' , 47 , 'rouge' , 4) ,
(2 , 'LEROI' , 40 , 4040 , 'NT' , 40 , 'jaune' , 5) ,
(4 , 'MARTIN' , 47 , 3333 , 'BX' , 33 , 'rouge' , 5) ,
(4 , 'MARTIN' , 47 , 4747 , 'LA' , 47 , 'rouge' , 4) ,
(4 , 'MARTIN' , 47 , 4040 , 'NT' , 40 , 'jaune' , 5) ,
(7 , 'LEROI' , 33 , 3333 , 'BX' , 33 , 'rouge' , 5) ,
(7 , 'LEROI' , 33 , 4747 , 'LA' , 47 , 'rouge' , 4) ,
(7 , 'LEROI' , 33 , 4040 , 'NT' , 40 , 'jaune' , 5) ,
(3 , 'DUPOND' , 17 , 3333 , 'BX' , 33 , 'rouge' , 5) ,
(3 , 'DUPOND' , 17 , 4747 , 'LA' , 47 , 'rouge' , 4) ,
(3 , 'DUPOND' , 17 , 4040 , 'NT' , 40 , 'jaune' , 5) }

Opération ensembliste

Les couleurs et numéro d'étudiant des voitures rouge ou orange dont le numéro d'étudiant est 2, 4 ou 6

$\{(\text{Couleur}, \text{NoINE})/\exists i, j, k : (i, j, k, \text{Couleur}, \text{NoINE}) \in \text{Voitures}\} \cap (\{'rouge', 'orange'\} \times \{2, 4, 6\})$

$=$ { ('rouge' , 4) }

Requêtes de mises à jour

Insertion

Insertion de la voiture 2424 PX 24

$\text{Voitures} = \text{Voitures} \cup \{(2424, 'PX', 24, 'rouge', 5)\}$

$=$ {
(3333 , 'BX' , 33 , 'rouge' , 5) ,
(4747 , 'LA' , 47 , 'rouge' , 4) ,
(4040 , 'NT' , 40 , 'jaune' , 5) ,
(2424 , 'PX' , 24 , 'rouge' , 5) }

Suppression

Suppression de la voiture 2424 PX 24 (après son insertion)

$\text{Voitures} = \text{Voitures} \setminus \{(2424, 'PX', 24, 'rouge', 5)\}$

$=$ {
(3333 , 'BX' , 33 , 'rouge' , 5) ,
(4747 , 'LA' , 47 , 'rouge' , 4) ,
(4040 , 'NT' , 40 , 'jaune' , 5) }

Historique

Naissance de SQL

Création du langage SQUARE entre 1972 et 1975, devenu SQL en 1976, pour le prototype relationnel SYSTEM-R d'IBM

1979 : première version de SQL commercialisée par Relational Software Inc. (devenue Oracle Corp.)

SQL-86 (ou SQL 1)

Première norme de l'*American National Standardisation Institute* (ANSI) ANSI X3.135 en 1986 ratifiée en 1987 par l'*International Electronical Commission* (IEC) de l'*International Standardization Organization* (ISO) ISO/IEC 9075

SQL-89

Norme ANSI et ISO

SQL-92 (ou SQL 2) : ISO/IEC 9075:1992

Norme ANSI et ISO pour un véritable langage de gestion des données

Nouveaux types de données

Définition déclarative complète des contraintes d'intégrité

Gestion professionnelle des transactions

Normalisation de SQL dynamique

Langage universel des SGBD, notamment pour les applications Client/Serveur (C/S)

Trois niveaux d'imbrication du respect de la norme

Entry : omissions de SQL 1

Intermediate : schéma, SQL dynamique, domaines, jointure externe, mises à jour en cascade

Full : véritable langage de BD

SQL-99 (ou SQL 3) : ISO/IEC 9075:1999 (début de l'approche relationnelle-objet)

Ajout de nouvelles fonctionnalités : expression rationnelle, requêtes récursives, déclencheurs, types non scalaires, quelques fonctionnalités orientées objet

SQL 2003 : ISO/IEC 9075:2003

Nouveautés : interface client *(client interface)*, procédure stockée *(stored procedure)*, objets SQL *(SQL object)*, requête récursive *(recursive query)*, déclencheur *(trigger)*

Intégration de concepts objet, de fonctionnalités multimédias, de fonctionnalités pour optimiser les performances, de la gestion de la répartition, de la gestion de grands volumes de données ; exemples : ajout de fonctions pour XML, ordres standardisés et attributs avec valeurs auto-produites

SQL 2008 : ISO/IEC 9075:2008

Ajout de fonctions de fenêtrage, limitation du nombre de tuples (i. e. de lignes)

Améliorations sur les types discrets, curseurs, auto-incréments

SQL Access Group (SAG) et normes ISO/IEC (1/2)

Le SAG

Organisme réunissant une quarantaine de constructeurs détenant 70 % du marché, qui a pour objectif de développer un langage SQL commun et un programme de conversion pour chacun de leurs dialectes de SQL

Exemples : *Open Database Connectivity* (ODBC) de MicroSoft et al. (1993), *Integrated Database Application Programming Interface* (IDAPI) de Borland (1994)

ISO/IEC 9075:2008 *(Information technology – Database languages – SQL – Part n)* (au 17/7/2008)

SQL/Framework (Part 1: Framework)

Description (non technique) de la structuration du document et définition des concepts communs

SQL/Foundation (Part 2: Foundation)

Fondements de la norme et l'essentiel du langage SQL : fonctionnalités de base pour définir les autres composants *Call-Level Interface* (CLI) et *Persistant Stored Modules* (PSM), déclencheurs, relations imbriquées, types abstraits de données, possibilités orientées objet

SQL/CLI (Part 3: Call-Level Interface)

Interface orientée objet pour accéder aux BD, outils C/S (bibliothèques dynamiques)

SQL/PSM (Part 4: Persistant Stored Modules)

Langage procédural étendant SQL : structures de contrôle, gestion des exceptions, déclaration et affectation de variable, gestion des zones de diagnostic, création de procédures stockées, support des possibilités orientées objet

SQL/MED (Part 9: Management of External Data)

Caractérisation de l'utilisation de SQL pour des données non SQL

SQL/OLB (Part 10: Object Language Bindings)

SQL/Schemata (Part 11: Information and Definition Schemas)

SQL/JRT (Part 13: SQL Routines and Types Using the Java TM Programming Language)

SQL/XML (Part 14: XML-Related Specifications)

ISO/IEC 9075:1999

SQL/Bindings (Part 5: Host Language Bindings) et *SQL/OLAP (Part 5/Amd 1: On-Line Analytical Processing)*

SQL intégré *(embedded)* et dynamique

SQL/XA Interface Specialization

Standardisation d'une API entre un gestionnaire de transactions *(transaction manager)* et un gestionnaire de ressources SQL *(SQL ressource manager)*, dont la gestion des transactions distribuées *via* le protocole d'achèvement en deux phases

SQL/Temporal

Spécifications relatives au temps

SQL/OBJ Object Language Binding

Spécifications des possibilités de manipulation des données à partir d'un langage orienté objet

SQL/MM Multimedia : multimédia

SQL/RDA Remote Data Access : protocole de transfert de données C/S

ISO/IEC 9079:2000 *Remote databases access for SQL with security enhancement*

ISO/IEC 13249 (depuis 1999) *(SQL Multimedia and application packages – Part n)*

Extension de SQL par une bibliothèque de types abstraits de données spécifiques au multimédia (exemples : `Full text`, `Graphic still`, `Animation`, `Image still`, `Full motion video`, `Audio`, `Spatial 2D`, `Spatial 3D`, `Music`, `Data mining`)

ISO/IEC 19125:2004 *Geographic information – Simple feature access – Part 2: SQL option*

SQL interactif : résultat obtenu immédiatement après l'exécution d'une requête

Langage de Définition des Données (LDD) : description de la structure de la BD

Langage de Manipulation des Données (LMD) : manipulations (i. e. interrogation et mises à jour (insertion, suppression, modification)) des données des relations

Langage de Contrôle des Données (LCD) : gestion des droits d'accès

Langage de Contrôle des Transactions (LCT) : gestion des transactions

... : connexion à un serveur, paramétrage d'une session

SQL procédural : PSM, CLI, etc.

Modules pour les routines (fonctions et procédures stockées), en plus des déclencheurs et méthodes, et interaction avec les langages hôtes

SQL programmé : requêtes SQL associées à un langage (hôte) de troisième génération

SQL intégré

Les requêtes SQL sont plongées *(embedded)* dans le programme (une pré-compilation transforme les ordres SQL en instructions du langage hôte)

SQL dynamique

Constitution et exécution de requêtes connues uniquement à l'exécution (et non à la compilation) du programme

Module SQL

Deux modules sont développés, l'un pour les requêtes SQL, l'autre pour les instructions du langage hôte

Quelques ordres SQL classés

SQL interactif					SQL programmé	
LDD	LMD	LCD	LCT	...	SQL intégré	SQL dynamique
`CREATE`	`SELECT`	`GRANT`	`COMMIT`	`CONNECT`	`DECLARE`	`PREPARE`
`DROP`	`INSERT`	`REVOKE`	`ROLLBACK`	`SET`	`CURSOR`	`DESCRIBE`
`ALTER`	`DELETE`				`FETCH`	`EXECUTE`
	`UPDATE`					

Remarques

Langage de 4^e génération (L4G) : déclaratif et non procédural ; ni structure de contrôle, ni variable

Langage complet de gestion (incluant l'interrogation) d'une BD

Critiques

Interface (interactive) *in fine* peu conviviale

Processus de normalisation/jointure « ésotérique »

Il faut être un développeur (et non un utilisateur) ne serait-ce que pour interroger la BD en utilisant correctement et efficacement les jointures, les sous-requêtes, les opérations ensemblistes, etc.

Remarques générales

Commentaires : `--` ◁ commentaire ▷ ◁ retour chariot ▷, `/*` ◁ commentaire ▷ `*/`

Séparateurs : espace, tabulation, retour chariot

Séparateur de requêtes : ;

Mise en facteur : `(` ... `)`

Constantes

Prédéfinie : `NULL` (indétermination i. e. ni `0` (zéro), ni '' (chaîne de caractères vide), ni `' '` (chaîne de caractères composée d'espaces ou tabulations ou retours chariot), etc. mais l'absence de valeur) ; `TRUE` (vrai), `FALSE` (faux) et `UNKNOWN` (inconnu i. e. prédicat non évaluable ; ni vrai ni faux mais ce troisième état)

Utilisateur : `98` ; `7E6` pour $7 * 10^6 = 7000000$; `-543.21` ; `'14:36:52.7'` ; `'2009-09-16'` ; `'2010-06-08 09:47:53'` ; `'Mercredi'` ; `N'Le SQL, c''est facile !'` (de type `NATIONAL CHARACTER VARYING`) ; `X'A0FA3'` ou `B'10100000111110100011'` (de type `BINARY VARYING`) pour $10 * 16^4 + 0 * 16^3 + 15 * 16^2 + 10 * 16^1 + 3 * 16^0 = 659363$

Types de données (1/3)

Numériques

`INTEGER` : entier ($\mathbb{Z}$), souvent sur 32 bits (soit alors un entier compris entre $-2^{31} = -2147483648$ et $2^{31} - 1 = 2147483647$)

`SMALLINT` : petit entier ($\mathbb{Z}$), souvent sur 16 bits

`BIGINT` : grand entier ($\mathbb{Z}$), souvent sur 64 bits

`NUMERIC`[`(`◁ entier ▷[`,`◁ entier ▷]`)`] : nombre décimal, le premier entier (1 par défaut) donnant le nombre de chiffres significatifs i. e. la précision, le second entier (0 par défaut) donnant le nombre de chiffres décimaux i. e. l'échelle

`DECIMAL`[`(`◁ entier ▷[`,`◁ entier ▷]`)`] : nombre décimal comme `NUMERIC` mais dont le codage interne par le SGBDR peut manipuler des nombres inférieurs ou supérieurs à ceux spécifiés

`REAL` : réel ($\mathbb{R}$)

`DOUBLE PRECISION` : réel double précision ($\mathbb{R}$)

`FLOAT`[`(`◁ entier ▷`)`] : flottant, l'entier donnant le nombre de chiffres binaires significatifs

Exemples :

`NUMERIC(5,2)` pour un nombre décimal allant de -999.99 à 999.99

`DECIMAL` pour un nombre allant de -9 à 9

`FLOAT(8)` pour un réel allant de -128.0 à 127.0

Types de données (2/3)

Temps

`TIME`[`STAMP`] [`WITH TIME ZONE`] [(◁ entier ▷)] : heure (heure, minute et seconde) ou (`TIMESTAMP`) « horodate » (date et heure i. e. année, mois, jour, heure, minute et seconde), et avec éventuellement le fuseau horaire précisant l'écart au *Coordinated Universal Time* (UTC) i. e. au temps universel coordonné

`INTERVAL` ◁ intervalle ▷ [(◁ entier ▷)] [`TO` ◁ intervalle ▷] : intervalle de temps où ◁ intervalle ▷ peut être `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE` ou `SECOND`, en n'autorisant que les combinaisons *Y2M*, *M*, *D2h*, *D2m*, *D2s*, *h2m*, *h2s*, *m2s* et *s*

Exemples :

`TIME(1)` pour un temps allant de 00:00:00.0 à 23:59:59.9

`TIMESTAMP WITH TIME ZONE + 01:00` pour une horodate ayant 1 heure d'écart avec le méridien de Greenwich

`INTERVAL YEAR (3) TO MONTH` pour un intervalle allant de 0 an et 0 mois à 99 ans et 11 mois

Chaînes (de caractères et de chiffres binaires)

[`NATIONAL`] `CHARACTER` [`VARYING`] (◁ entier ▷) : chaîne de caractères d'une longueur fixe égale à l'entier ou [option `VARYING`] variable au plus égale à l'entier, chaque caractère étant codé sur 1 octet (ASCII ou EBCDIC) ou [option `NATIONAL`] 2 octets (UNICODE)

[`NATIONAL`] `CHARACTER LARGE OBJECT` (◁ entier ▷) : grande chaîne de caractères d'une longueur au plus égale à l'entier exprimé généralement en `K`, `M` ou `G` pour kilo, Méga ou Giga respectivement

`BIT` [`VARYING`] [(◁ entier ▷)] : chaîne de chiffres binaires d'une longueur fixe égale à l'entier ou [option `VARYING`] variable au plus égale à l'entier)

`BINARY LARGE OBJECT` [(◁ entier ▷)] : grande chaîne de chiffres binaires d'une longueur (en octets) au plus égale à l'entier

Exemples :

`CHARACTER(100)` pour une chaîne d'exactement 100 caractères (chacun codé sur 1 octet)

`NVARCHAR(30)` pour une chaîne d'au plus (`NATIONAL CHARACTER VARYING`) 30 caractères (chacun codé sur 2 octets)

`CLOB(20M)` pour une grande chaîne (`CHARACTER LARGE OBJECT`) d'au plus 20 millions (méga) de caractères

`BLOB(4G)` pour une grande chaîne de chiffres binaires (`BINARY LARGE OBJECT`) d'au plus 4 milliards (giga) de caractères

Booléen

`BOOLEAN` : booléen

Types de données (3/3)

Types de données abstraits

◁ attribut ▷ ◁ type ▷ `ARRAY[`◁ entier ▷`]` : tableau à une dimension ayant l'entier comme nombre d'éléments

◁ attribut ▷ `ROW(`◁ attributs et leurs types ▷`)` : structure composée de plusieurs attributs

type collection et patron (i. e. objet complexe) `MULTISET` : collection non ordonnée sans doublon (`SET`), collection non ordonnée avec doublons (`BAG`), collection ordonnée avec doublons (`LIST`), collection ordonnée et indicée (`ARRAY`), pile *(stack)*, file *(queue)*, tableaux dynamiques *(varray)*, tableaux insérables *(iarray)*

`CREATE TABLE` ◁ relation ▷ `UNDER` ◁ relation ▷ : relation héritant d'une relation

types utilisateurs *User Data Type* (UDT) : types distincts i. e. typage fort, types hérités et types abstraits

type relation (i. e. type table) ou table objet : relation définie à partir d'un type utilisateur

type référence `REF` : attribut pointant directement vers un objet ligne d'une relation

type pointeur `LOCATOR` : caractéristique que tout attribut peut posséder, permettant notamment de stocker des données du côté du client pour réduire le trafic réseau avec le serveur

type référençant un fichier externe `DATALINK` : attribut désignant un fichier (image, son, vidéo, binaire, etc.) externe au SGBD

type `XML` : stockage et interrogation de grappes XML

Exemples :

`CARDINALITY(CONCATENATE('éléments.',ARRAY['Tableau','de','quatre']))` vaut 4 à savoir le cardinal du tableau constitué de la concaténation (i. e. l'ajout) d'un élément (une chaîne de caractères) et d'un tableau composé de trois éléments (trois chaînes de caractères)

`ROW('UN',ARRAY['Deux','Trois','Quatre'],5)` est un triplet composé d'une chaîne de caractères, d'un tableau composé de trois chaînes de caractères et d'un entier

`MULTISET[`$e^{i*\pi}$`,+,1,=,0]` est une collection de cinq expressions

`MULTISET(SELECT DISTINCT ROW(Nom,Prenoms[1]) FROM Personnes)` est une collection de couples distincts composés du nom et du premier prénom des personnes

Opérateurs arithmétiques

Opérateurs arithmétiques (par ordre croissant de priorité)

`*` (multiplication) et `/` (division)

`+` (addition) et `-` (soustraction)

Opérateurs logiques (1/2)

Combinaison de prédicats (par ordre décroissant de priorité)

`NOT` (négation)

`AND` (et logique)

`OR` (ou logique)

Comparaisons

`IS NULL` : indétermination

`IS UNKNOWN` : prédicat non évaluable

`=` (égal) ; `<>` (différent)

`<` (inférieur) ; `<=` (inférieur ou égal) ; `>=` (supérieur ou égal) ; `>` (supérieur)

`BETWEEN` [`SYMETRIC`] ◁ valeur ▷ `AND` ◁ valeur ▷ : appartenance à un intervalle allant de la première valeur à la seconde ou [option `SYMETRIC`] entre la plus petite et la plus grande des deux valeurs, bornes comprises

`LIKE` : respect d'un certain format avec `%` (pour n'importe quelle chaîne de caractères, y compris la chaîne vide) et `_` (pour n'importe quel caractère)

`SIMILAR TO` : respect d'une expression régulière avec `-` (intervalle), `[]` (appartenance à la liste), `^[]` (non appartenance à la liste), `:ALPHA` (caractère alphabétique indépendamment de la casse), `:UPPER` ou `:LOWER` (caractère alphabétique vérifiant la casse, respectivement en majuscule ou en minuscule), `:DIGIT` (chiffre), `:ALNUM` (caractère alphanumérique quelconque i. e. imprimable), `:SPACE` (espace), `:WHITESPACE` (caractère non imprimable tel l'espace, la tabulation, le retour chariot, etc.), `|` (ou), `||` (concaténation)

`COLLATE` ◁ collation ▷ : permet de comparer des expressions de collations différentes ou d'imposer une collation (langue ou codes de caractères, sensibilité à la casse (majuscule/minuscule), sensibilité aux diacritiques (À,à,Â,â,Ç,ç,É,é,È,è,Ê,ê,Ë,ë,i,Î,î,Ï,ï,j,Ô,ô,Ù,ù,Û,û,Ü,ü) et aux ligatures (Æ,æ,Œ,œ), etc.)

Exemples :

`a IS NOT NULL` indique si a est renseigné

`NULL = 0` vaut `UNKNOWN`

`( b <> 2 OR c BETWEEN SYMETRIC 4 AND d ) AND e IN (8,16,32)` i. e. $(b \neq 2 \vee \min\{4,d\} \leq c \leq \max\{4,d\}) \wedge e \in \{8,16,32\}$

`f LIKE 'SOIXANT_-%_'` où f est une chaîne de caractères commençant par $SOIXANT$ suivie d'un caractère quelconque puis d'un trait d'union puis d'une chaîne de longueur quelconque et terminée par un caractère quelconque

`g SIMILAR TO '[DVTQCS]||^[AEIOUY]%[AEIOUY]|[U-Z]'` où g est une chaîne de caractères commençant par l'une des lettres de $\{D,V,T,Q,C,S\}$ suivie d'un caractère autre qu'une voyelle puis d'une chaîne de longueur quelconque et terminée par une voyelle ou une lettre comprise entre U et Z

Utilisez les collations pour éviter `Prénom SIMILAR TO '[Ll][EeÉéÈèÊêËë][AaÀàÂâ]'` : `Prénom = 'Lea' COLLATE FRENCH_CI_AI` en SQL Server où `CI` et `AI` signifient respectivement *Case Insensitive* et *Accent Insensitive*, `UPPER(Prénom,'NLS_SORT=FRENCH') = 'LEA'` en Oracle, etc.

Opérateurs logiques (2/2)

Tests

`IN` : appartenance à un ensemble de valeurs (vaut `UNKNOWN` si l'ensemble contient au moins un élément à `NULL`)

`MATCH` : appartenance d'un n-uplet à un ensemble de n-uplets

`EXISTS` : existence i. e. vrai si la sous-requête retourne au moins un tuple

`UNIQUE` : unicité sans considérer les indéterminations i. e. vrai si toutes les valeurs renseignées retournées par la sous-requête sont distinctes

`DISTINCT` : unicité y compris pour l'indétermination i. e. vrai si tous les tuples retournés par la sous-requête sont distincts)

`ALL` : tous i. e. vrai si la comparaison (généralement une inégalité) est vraie pour toutes les valeurs retournées par la sous-requête (vaut `UNKNOWN` si l'ensemble contient au moins un élément à `NULL`)

`ANY` (ou `SOME`) : au moins un i. e. vrai si la comparaison (généralement une inégalité) est vraie avec au moins l'une des valeurs retournées par la sous-requête (vaut `UNKNOWN` si l'ensemble contient au moins un élément à `NULL`)

Remarques :

`<> ALL` est équivalent à `NOT IN`

`= ANY` est équivalent à `IN`

Les tests d'inégalités (strictes ou larges) avec `ALL` et `ANY` (i. e. `> ALL`, `> ANY`, `>= ALL`, etc.) peuvent s'exprimer avec `EXISTS` s'il n'y a pas de `NULL` voire comme une comparaison avec le minimum ou le maximum de l'ensemble

Exemples :

`'LEROI' IN (SELECT Nom FROM Personnes)` indique s'il existe au moins une personne de nom *LEROI*

`('LEROI','Arthur') MATCH (SELECT Nom,Prénom FROM Personnes)` indique s'il existe au moins une personne de nom *LEROI* et de prénom *Arthur*
Remarque : ce prédicat est différent de `'LEROI' IN (SELECT Nom FROM Personnes) AND 'Arthur' IN (SELECT Prénom FROM Personnes)` et de `'LEROI' IN (SELECT Nom FROM Personnes) OR 'Arthur' IN (SELECT Prénom FROM Personnes)` ; par exemple, pour l'ensemble des couples des noms et prénoms des personnes $\{('LEROI','Bob'),('MARTIN','Arthur')\}$, le `MATCH` retourne faux tandis que les deux autres prédicats retournent vrai
N. B. : si votre SGBDR ne connaît pas `MATCH`, on peut écrire `'LEROI'||'&'||'Arthur' IN (SELECT TRIM(Nom)||'&'||TRIM(Prénom) FROM Personnes)` où `&` est un caractère qui ne doit figurer dans aucun nom ou prénom

`EXISTS (SELECT * FROM Personnes WHERE Numéro < 9)` indique s'il y a au moins une personne de numéro inférieur à 9

`NOT UNIQUE (SELECT Nom FROM Personnes)` indique s'il y a des homonymes (au moins deux personnes ont le même nom i. e. que les noms des personnes ne sont pas tous différents deux à deux)

`8 > ALL (SELECT Numéro FROM Personnes)` indique si 8 est supérieur à tous les numéros des personnes (i. e. $8 > \max\{$`Personnes.Numéro`$\}$)

`7 <= ANY (SELECT Numéro FROM Personnes)` indique si 7 est inférieur ou égal à au moins l'un des numéros des personnes (i. e. $7 \leq \max\{$`Personnes.Numéro`$\}$)

`6 >= ANY ( 5 , NULL , 4 , NULL , 3 )` vaut `UNKNOWN`

Fonctions (1/5)

Diverses

`CURRENT_PATH` : chemin d'accès courant au schéma

`[CURRENT_]USER` : utilisateur courant

`SESSION_USER` : utilisateur autorisé

`SYSTEM_USER` : utilisateur système

`CAST(`◁ expression ▷ `AS` ◁ type ▷`)` : transtype une expression (dans un type)

`COALESCE(`◁ expressions ▷`)` : retourne la première valeur de la liste d'expressions dont la valeur n'est pas absente

`NULLIF(`◁ expression ▷`,`◁ valeur ▷`)` : retourne `NULL` si l'expression vaut la valeur, ou l'expression sinon

Exemples :

`CAST('98' AS INTEGER)` pour 98

`COALESCE(Pays,Département,'inconnu')` retourne le pays s'il est renseigné, le département s'il est renseigné lorsque pays n'est pas renseigné, *inconnu* si ni le pays ni le département n'est renseigné

`NULLIF(Département,'inconnu')` retourne `NULL` si le département vaut *inconnu*, ou le département sinon

Numériques

`ABS(`◁ réel ▷`)` : valeur absolue du nombre réel

`MOD(`◁ entier ▷`,`◁ entier ▷`)` : reste (modulo) de la division entière du premier entier (dividende) par le second entier (diviseur)

`POWER(`◁ réel ▷`,`◁ réel ▷`)` : premier nombre réel élevé à la puissance du second nombre réel

`EXP(`◁ réel ▷`)` : exponentielle du nombre réel

`LN(`◁ réel ▷`)` : logarithme népérien du nombre réel

`FLOOR(`◁ réel ▷`)` : plus grand entier inférieur ou égal au nombre réel

`CEIL(`◁ réel ▷`)` : plus petit entier supérieur ou égal au nombre réel

`WIDTH_BUCKET(`◁ expression ▷`,`◁ entier ▷`,`◁ entier ▷`,`◁ entier ▷`)` : distribution des valeurs de l'expression du premier entier au deuxième entier en autant de tranches que le troisième entier ; ainsi, `WIDTH_BUCKET(e,d,f,t)` (où e est l'expression, d la valeur de début, f la valeur de fin, t le nombre de tranches et soit $p = \lfloor \frac{f-d}{t} \rfloor$ le pas) vaut 0 si $e < d$ ou $1 + \lfloor \frac{e-d}{p} \rfloor$ sinon

Exemples :

`MOD(97,8)` vaut 1 car $97 = 12 * 8 + 1$

`POWER(4,5)` vaut $4^5 = 1024$

`FLOOR(2.6)` vaut $\lfloor 2.6 \rfloor = 2$

`ABS(CEIL(-31.0/4.0))` vaut $|\lceil -\frac{31}{4} \rceil| = |\lceil -7.75 \rceil| = |-7| = 7$

`WIDTH_BUCKET(g,21,51,3)` retourne 0, 1, 2, 3, 4, 5, etc. pour g allant respectivement jusqu'à 20, de 21 à 30, de 31 à 40, de 41 à 50, de 51 à 60, de 61 à 70, etc.

Fonctions (2/5)

Temps

`CURRENT_TIME` : heure système

`CURRENT_DATE` : date système

`CURRENT_TIMESTAMP` : horodate système

`EXTRACT(`◁ type ▷ `FROM` ◁ variable ▷`)` : extraction, à partir d'une variable (de type `DATE`, `TIME` ou `INTERVAL`), d'une information de type donné (`DATE`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `TIMEZONE_HOUR` ou `TIMEZONE_MINUTE`)

`(`◁ variable ▷`,`◁ variable ▷`) OVERLAPS (`◁ variable ▷`,`◁ variable ▷`)` : recouvrement (i. e. chevauchement) de périodes (même partiellement), les première et troisième variables étant de type `TIMESTAMP`, les deuxième et quatrième étant de type `TIMESTAMP` ou `INTERVAL`) ; en fait, on a `(d1,f1) OVERLAPS (d2,f2)` équivalent à `(d1<d2 AND d2<f1)) OR (d2<d1 AND d1<f2) OR (d1=d2 AND f1 IS NOT NULL AND f2 IS NOT NULL)` (le premier test `d1<d2<f1` considère les cas où la seconde période débute (`d2`) durant la première période (`]d1;f1[`) tandis que le deuxième test `d2<d1<f2` considère les cas où la première période (`d1`) débute durant la seconde période (`]d2;f2[`))

Exemples :

`EXTRACT(MINUTE FROM CURRENT_TIME)` vaut le nombre de minutes de l'heure actuelle

`INTERVAL(CAST('12:34:56' AS TIME) - INTERVAL '78' MINUTE)` vaut 11 h 16 min 56 s

`('14:00:00','15:50:00') OVERLAPS ('16:10:00','18:00:00')` est faux

`( CURRENT_DATE , CURRENT_DATE ) OVERLAPS ( CURRENT_DATE - INTERVAL '1' DAY , CURRENT_DATE + INTERVAL '1' DAY )` est vrai

`( CURRENT_TIMESTAMP - INTERVAL '48' HOUR - INTERVAL '120' MINUTE , CURRENT_TIMESTAMP + INTERVAL '1' DAY ) OVERLAPS ( CURRENT_TIMESTAMP , CURRENT_TIMESTAMP + INTERVAL '1' MONTH )` est vrai

Fonctions (3/5)

Chaînes

`CHAR[ACTER]_LENGTH(`◁ chaîne ▷`)` : nombre de caractères de la chaîne

`POSITION (`◁ chaîne ▷ `IN` ◁ chaîne ▷`)` : indice de la première occurrence de la première chaîne de caractères dans la seconde chaîne de caractères

`SUBSTRING(`◁ chaîne ▷ `FROM` ◁ entier ▷ [`FOR` ◁ entier ▷]`)` : extraction de la chaîne de caractères à partir de l'indice donné par le premier entier d'une longueur donnée par le second entier

`OVERLAY(`◁ chaîne ▷ `PLACING` ◁ chaîne ▷ `FROM` ◁ entier ▷ [`FOR` ◁ entier ▷]`)` : insère la seconde chaîne de caractères dans la première chaîne de caractères à partir de l'indice donné par le premier entier d'une longueur donnée par le second entier

`TRANSLATE(`◁ chaîne ▷ `USING` ◁ translation ▷`)` : remplace les caractères dans la chaîne de caractères tel qu'indiqué dans la translation

`CONVERT(`◁ chaîne ▷ `USING` ◁ fonction de conversion ▷`)` : remplace les caractères dans la chaîne de caractères en utilisant une fonction de conversion (valable uniquement dans la clause `SELECT`)

`TRIM(`[◁ restriction ▷] ◁ chaîne ▷`)` : supprime les espaces à gauche et à droite de la chaîne de caractères ou [option ◁ restriction ▷ vaut `LEADING`] que ceux en tête ou [option ◁ restriction ▷ vaut `TAILING`] que ceux en queue

`LOWER(`◁ chaîne ▷`)` : transformation de la chaîne de caractères en minuscules

`UPPER(`◁ chaîne ▷`)` : transformation de la chaîne de caractères en majuscules

◁ chaîne ▷`||`◁ chaîne ▷ : concaténation des deux chaînes de caractères

`OCTET_LENGTH(`◁ chaîne ▷`)` : nombre d'octets de la chaîne

`BIT_LENGTH(`◁ chaîne ▷`)` : nombre de chiffres binaires de la chaîne

Exemples :

`CHARACTER_LENGTH('BONJOUR')` vaut 7

`POSITION('O' IN 'BONJOUR')` vaut 2

`SUBSTRING('BONJOUR' FROM 5 FOR 2)` vaut OU

`OVERLAY('BONJR' PLACING 'OU' FROM 5 FOR 2)` vaut $BONJOUR$

`TRANSLATE('BÔNJÔÜR','ÀÂÉÈÊËÎÏÔÙÛÜ','AAEEEEIIOUUU')` vaut $BONJOUR$

`TRIM(UPPER('  Bon')||'JOUR ')` vaut $BONJOUR$ (chaîne composée de sept et non dix caractères)

Fonctions (4/5)

Fonctions d'agrégation (ou de regroupement)

Définition : un agrégat est un sous-ensemble des tuples d'une relation pour lesquels les valeurs d'un groupe d'attributs sont identiques (ligne à ligne)

`COUNT(*)` : nombre de tuples (i. e. de lignes)

`COUNT(`[`DISTINCT`] ◁ attribut ▷`)` : nombre d'occurrences des valeurs renseignées et [option `DISTINCT`] différentes deux à deux de l'attribut

`SUM(`[`DISTINCT`] ◁ attribut ▷`)` : somme des valeurs renseignées et [option `DISTINCT`] différentes deux à deux de l'attribut

`AVG(`[`DISTINCT`] ◁ attribut ▷`)` : moyenne des valeurs renseignées et [option `DISTINCT`] différentes deux à deux de l'attribut

`MAX(`◁ attribut ▷`)` : plus grande des valeurs renseignées de l'attribut

`MIN(`◁ attribut ▷`)` : plus petite des valeurs renseignées de l'attribut

`STDDEV_POP(`[`DISTINCT`] ◁ attribut ▷`)` : écart-type de la population des valeurs renseignées de l'attribut

`STDDEV_SAMP(`[`DISTINCT`] ◁ attribut ▷`)` : écart-type d'un échantillon des valeurs renseignées de l'attribut

`VAR_POP(`[`DISTINCT`] ◁ attribut ▷`)` : variance de la population des valeurs renseignées de l'attribut

`VAR_SAMP(`[`DISTINCT`] ◁ attribut ▷`)` : variance d'un échantillon des valeurs renseignées de l'attribut

`COVAR_POP(`[`DISTINCT`] ◁ attribut ▷`,`◁ attribut ▷`)` : covariance de la population des valeurs renseignées de deux attributs

`COVAR_SAMP(`[`DISTINCT`] ◁ attribut ▷`,`◁ attribut ▷`)` : covariance d'un échantillon des valeurs renseignées de deux attributs

`CORR(`[`DISTINCT`] ◁ attribut ▷`,`◁ attribut ▷`)` : corrélation des valeurs renseignées de deux attributs

`EVERY(`◁ condition ▷`)` : indique si toutes les valeurs sont vraies

`ANY(`◁ condition ▷`)` : indique si au moins une des valeurs est vraie

Remarques :

- Excepté pour `COUNT(*)`, tous les `NULL` sont éliminés du calcul de la fonction d'agrégation
- L'option `DISTINCT` élimine les répétitions du calcul de la fonction d'agrégation ; par défaut [option `ALL`], les répétitions sont conservées

Exemples : soit $R = (NULL, 2, 5, 9, NULL, 2, 5, NULL, 2, NULL)$

- `COUNT(*)` vaut 10 i. e. le nombre d'éléments de R
- `COUNT(R)` vaut 6 i. e. le nombre d'éléments renseignés de R c.-à-d. de $(2, 5, 9, 2, 5, 2)$
- `COUNT(DISTINCT R)` vaut 3 i. e. le nombre d'éléments renseignés et sans répétition de R i. e. $card(\{2, 5, 9\})$
- `SUM(R)` vaut $25 = 2 + 5 + 9 + 2 + 5 + 2$
- `SUM(DISTINCT R)` vaut $16 = 2 + 5 + 9$
- `MIN(R)` vaut 2
- `EVERY(COALESCE(R,0)<4)` est faux car au moins une valeur renseignée est supérieure ou égale à 4
- `ANY(COALESCE(R,9)<4)` est vrai car au moins une valeur renseignée est inférieure à 4

Fonctions (5/5)

Fonctions pour l'analyse multidimensionnelle OLAP

`CUBE(`◁ attributs ▷`)` : effectue des sous-totaux des agrégats de toutes les combinaisons de tous les attributs

`ROLLUP(`◁ attributs ▷`)` : effectue des sous-totaux des agrégats de toutes les combinaisons de certains des attributs

`GROUPING SETS({CUBE,ROLLUP}` ◁ attributs ▷`)` : effectue des sous-totaux des agrégats de toutes les combinaisons des attributs spécifiés

Exemples :

`CUBE(Nom,Prénom)` fait un regroupement sur les couples nom et prénom, puis pour chaque nom, ensuite pour chaque prénom, et enfin pour toute la population

`GROUPING SETS(CUBE(A,ROLLUP(B,C))` effectue un regroupement de type `CUBE` sur les couples composés de A et de (B, C), ce dernier bénéficiant d'un regroupement de type `ROLLUP` pour chacune de ses composantes B et C

Fonctions de fenêtrage

Objectif

Définir une fenêtre de données exploitable ensuite par des fonctions d'agrégation ou d'ordonnancement, soit implicitement, soit explicitement par la directive `WINDOW` dans le `SELECT` immédiatement avant la directive `ORDER BY`

◁ fenêtre de données ▷ := `OVER (`[`PARTITION BY (`◁ attributs ▷`)`] `ORDER BY` ◁ attributs ▷`)` : définition d'une fenêtre de données, éventuellement [option `PARTITION BY`] en la partionnant

`ROW_NUMBER()` ◁ fenêtre de données ▷ : numérotation des lignes (i. e. tuples) basée sur la liste d'attributs

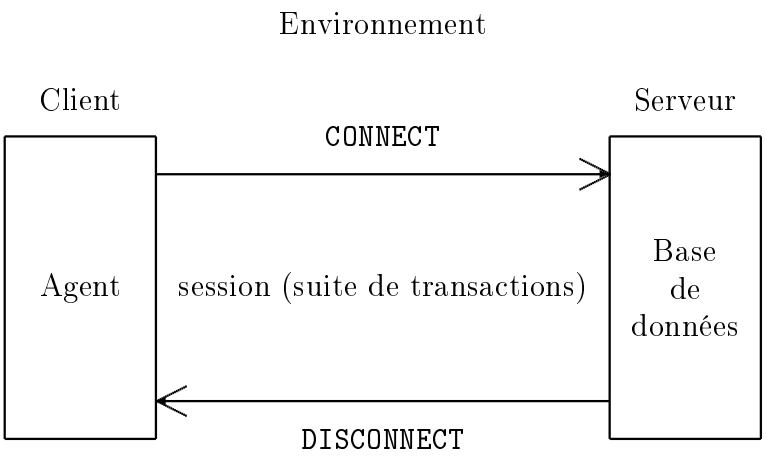
[`{DENSE_,PERCENT_}`]`RANK()` ◁ fenêtre de données ▷ : classement basé sur la liste d'attributs avec ou [fonction `DENSE_RANK`] sans les *ex æquo*, ou [fonction `PERCENT_RANK`] en pourcentage

`CUME_DIST()` ◁ fenêtre de données ▷ : distribution cumulative basée sur la liste d'attributs

Exemples :

`ROW_NUMBER() OVER (ORDER BY Nom DESC NULLS LAST)` : numérotation dans l'ordre alphabétique décroissant des noms en finissant par les noms non renseignés

`RANK() OVER (PARTITION BY Prénom ORDER BY Nom)` : numérotation (séquentielle en recommençant à 1 pour chaque prénom différent) des noms selon l'ordre alphabétique, pour chaque paquet de prénoms identiques ; on suppose donc le résultat trié d'abord sur le prénom et ensuite sur le nom




```
---------------------------------
-- initialisation d'une connexion
---------------------------------
-- connexion au serveur UnServeur
CONNECT TO UnServeur ; -- CONNECT TO DEFAULT

-------------------------------------------------------------------------
-- modification des paramètres par défaut de la session
-- (i. e. une connexion activée) qui précise l'identifiant d'autorisation,
-- le catalogue, le schéma, le fuseau horaire et le jeu de caractères
-------------------------------------------------------------------------
-- choix du schéma EtudiantsVoituresDiplomes
SET SCHEMA EtudiantsVoituresDiplomes ;

-- toutes les contraintes (déférables ou non) sont à appliquer à la fin de
-- chaque requête (et non en fin de transaction)
SET CONSTRAINTS ALL IMMEDIATE ; -- plutôt que DEFERRED

-- toute manipulation (interrogation ou mise à jour) est permise avec le plus
-- haut niveau d'isolation (résout tous les problèmes excepté l'interblocage)
SET SESSION CHARACTERISTICS AS -- ou SET TRANSACTION
    READ WRITE -- plutôt que READ ONLY
    ISOLATION LEVEL SERIALIZABLE ;  -- ni READ UNCOMMITED
                                    -- ni READ COMMITED
                                    -- ni REPEATED READ

-------------------------------------------------------------------------
-- requêtes (LDD, LMD dont requêtes sont regroupées en transactions, LCD, LCT)
-------------------------------------------------------------------------

...

---------------
-- déconnexion
---------------
-- termine toutes les connexions
DISCONNECT ALL ; -- DISCONNECT CURRENT ne fermerait que la connexion actuelle
```

Remarque : l'analogie avec l'algèbre relationnelle constitue un abus de langage

Opérations relationnelles

[WITH [RECURSIVE] ◁ nom requête ▷ (◁ attributs ▷) AS (◁ requête d'interrogation ▷)] -- expression de table
SELECT ◁ attributs et/ou fonctions d'agrégation, voire expression ▷ -- projection
FROM ◁ relations ▷ -- relations (produit cartésien)
[JOIN ◁ relations et conditions de jointure ▷]* -- jointures
[WHERE ◁ condition ▷] -- sélection n'utilisant pas de fonction d'agrégation
[GROUP BY ◁ attributs ▷] -- agrégation ; au moins tous attributs du SELECT
[HAVING ◁ condition ▷] -- sélection utilisant des fonctions d'agrégation
[WINDOW ◁ fenêtre de données ▷] -- fenêtrage
[ORDER BY ◁ attributs ▷] -- tri

Projection (1/2)

```
-- les noms des étudiants (sans répétition)
SELECT DISTINCT NomEtu
FROM Etudiants
```

'DURAND'
'LEROI'
'MARTIN'
'DUPOND'

```
-- les régions de naissance des étudiants
SELECT * , CASE
    WHEN DepartNaissEtu IN ( 33 , 24 , 47 , 40 , 64 ) THEN 'Aquitaine'
    WHEN DepartNaissEtu IN ( 79 , 86 , 17 , 16 ) THEN 'Poitou-Charentes'
    ELSE NULL
FROM Etudiants
```

5	'DURAND'	33	'Aquitaine'
2	'LEROI'	40	'Aquitaine'
4	'MARTIN'	47	'Aquitaine'
7	'LEROI'	33	'Aquitaine'
3	'DUPOND'	17	'Poitou-Charentes'

```
-- les numéros d'immatriculation des voitures et une répartition des
-- numéros d'immatriculation en chiffre par milliers (les tranches
-- [0;1000[, [1000;2000[, [2000;3000[, ... affichant respectivement
-- 0, 1, 2, ...)
-- N. B. : WIDTH_BUCKET() aurait ici pu être remplacé, par exemple, par
--         FLOOR(NoImmatChiffres/1000) ou encore par
--         (NoImmatChiffres-MOD(NoImmatChiffres,1000))/1000
SELECT CAST(NoImmatChiffres AS CHAR) || ' ' || NoImmatLettres || ' ' ||
    CAST(NoImmatDepart AS CHAR) ,
    WIDTH_BUCKET(NoImmatChiffres,1000,10000,10)
FROM Voitures
```

'3333 BX 33'	3
'4747 LA 47'	4
'4040 NT 40'	4

Projection (2/2)

```
-- les intitulés abrégés des diplômes ainsi que les quatre premiers et
-- cinq derniers caractères de l'intitulé complet du diplôme
-- séparés par des points de suspension
SELECT IntitAbrege , SUBSTRING(IntitComplet FROM 1 FOR 4) || '...' ||
    SUBSTRING(IntitComplet FROM CHARACTER_LENGTH(IntitComplet)-4 FOR 5)
FROM Diplomes
```

'DUT'	'Dipl...logie'
'BAC'	'Bacc...uréat'
'MIAGe'	'Maît...rises'
'DEUG'	'Dipl...rales'

```
-- les numéros, noms (en minuscule) et départements de naissance
-- (en remplaçant par NULL les girondins) des étudiants
SELECT NoINE , LOWER(NomEtu) , NULLIF(DepartNaissEtu,33)
FROM Etudiants
```

5	'durand'	
2	'leroi'	40
4	'martin'	47
7	'leroi'	
3	'dupond'	17

Tri

```
-- les étudiants triés sur leur département de naissance (i. e. le 3e
-- champ) en ordre décroissant et, en cas d'égalité, sur leur nom en
-- ordre croissant
SELECT NomEtu , NoINE , DepartNaissEtu
FROM Etudiants
ORDER BY 3 DESC , NomEtu ASC
```

'MARTIN'	4	47
'LEROI'	2	40
'DURAND'	5	33
'LEROI'	7	33
'DUPOND'	3	17

Sélection (1/2)

```
-- les numéros des étudiants, les intitulés des diplômes obtenus et
-- leurs années d'obtention pour ceux obtenus il y a au moins trente ans
-- en 2009 ou dont l'intitulé abrégé du diplôme est un mot compris entre
-- 'C' et 'E' (avec 'BAC' < 'C' < 'DEUG' < 'DUT' < 'E' < 'MIAGe')
SELECT NoINE , IntitAbrege , Annee
FROM AvoirObtenu
WHERE EXTRACT(YEAR FROM CURRENT_DATE) - Annee >= 30
   OR IntitAbrege BETWEEN 'C' AND 'E'
```

4	'DEUG'	1980
2	'DEUG'	1982
5	'DUT'	1983
3	'DUT'	1983
4	'BAC'	1977

Sélection (2/2)

```
-- les voitures girondines rouges ou dont le numéro d'étudiant est
-- compris entre le chiffre des unités et des dizaines du numéro
-- d'immatriculation du département (entre le minimum et le maximum des
-- deux), et en affichant également la valeur absolue de la différence
-- entre d'une part la partie entière supérieure de la division du
-- numéro d'immatriculation en chiffre par le numéro d'immatriculation
-- du département et d'autre part le cube du numéro d'étudiant
SELECT * , ABS(FLOOR(NoImmatChiffres/NoImmatDepart)-POWER(NoINE,3))
FROM Voitures
WHERE ( NoImmatDepart , Couleur ) = ( 33 , 'rouge' ) OR NoINE
   BETWEEN SYMETRIC MOD(NoImmatDepart,10) AND CEIL(NoImmatDepart/10)
```

3333	'BX'	33	'rouge'	5	24
4747	'LA'	47	'rouge'	4	37

```
-- les étudiants nés dans la région Aquitaine dont le nom contient la
-- lettre N
SELECT *
FROM Etudiants
WHERE DepartNaissEtu IN (24,33,40,47,64) AND NomEtu LIKE '%N%'
```

5	'DURAND'	33
4	'MARTIN'	47

```
-- les diplômes dont l'intitulé abrégé commence par une majuscule suivie
-- d'un caractère autre que la première lettre de l'alphabet (minuscule
-- ou majuscule) et dont l'intitulé complet contient le mot
-- "universitaire" (en ignorant la casse) jusqu'en douzième position
SELECT IntitAbrege , IntitComplet
FROM Diplomes
WHERE IntitAbrege SIMILAR TO '[A-Z]^[aA]%'
  AND POSITION('universitaire' IN LOWER(IntitComplet)) BETWEEN 1 AND 12
```

'DUT'	'Diplôme Universitaire de Technologie'

Agrégation (1/4)

```
-- le nombre d'étudiants
SELECT COUNT(*)
FROM Etudiants
```

5

```
-- le nombre d'étudiants dont le numéro est supérieur à 9
SELECT COUNT(*)
FROM Etudiants
WHERE NoINE > 9
```

0

// l'un des rares cas où `COUNT(*)` vaut zéro (car généralement strictement positif)

```
-- le numéro d'étudiant le plus petit et le plus grand
SELECT MIN(NoINE) , MAX(NoINE)
FROM Etudiants
```

2	7

Agrégation (2/4)

```
-- pour chaque étudiant, le nombre de ses diplômes ainsi que
-- les première et dernière années d'obtention
SELECT NoINE , COUNT(*) , MIN(Annee) , MAX(Annee)
FROM AvoirObtenu
GROUP BY NoINE
```

4	3	1977	1982
2	2	1980	1982
5	2	1981	1983
3	3	1981	1985

```
-- compte, pour chaque diplôme obtenu :
-- le nombre de tuples pour *, NoINE (le nombre de valeurs de NoINE est
--   le nombre de tuples car NoINE est obligatoire), DISTINCT NoINE (le
--   nombre de valeurs distinctes de NoINE qui est obligatoire et unique
--   pour chacune des valeurs de IntitAbrege issues du regroupement),
--   IntitAbrege (attribut obligatoire), Annee (attribut obligatoire)
-- 1 pour DISTINCT IntitAbrege (toutes les valeurs sont identiques)
-- le nombre d'années différentes deux à deux pour DISTINCT Annee
-- N. B. : pour éviter certaines d'erreurs, privilégiez COUNT(*) qui
--         compte le nombre de tuples ou encore COUNT(NoINE) qui fait de
--         même car les tuples sont identifiés par (IntitAbrege,NoINE)
--         le premier élément étant dans le GROUP BY le second étant
--         dans le COUNT()
SELECT IntitAbrege , COUNT(*) , COUNT(NoINE) , COUNT(DISTINCT NoINE) ,
       COUNT(IntitAbrege) , COUNT(DISTINCT IntitAbrege) ,
       COUNT(Annee) , COUNT(DISTINCT Annee)
FROM AvoirObtenu
GROUP BY IntitAbrege
```

'DEUG'	2	2	2	2	1	2	2
'DUT'	2	2	2	2	1	2	1
'MIAGe'	2	2	2	2	1	2	2
'BAC'	4	4	4	4	1	4	3

```
-- les noms des étudiants (sans répétition)
-- N. B. : à bannir ; utilisez DISTINCT dans le SELECT et sans GROUP BY
SELECT NomEtu
FROM Etudiants
GROUP BY NomEtu
```

'DURAND'
'LEROI'
'MARTIN'
'DUPOND'

```
-- indique si le nom de tous les étudiants est en majuscule
SELECT EVERY( NomEtu = UPPER(NomEtu) )
FROM Etudiants
```

TRUE

Agrégation (3/4)

```
-- les numéros des étudiants, intitulés abrégés des diplômes,
-- nombre de tuples et années minimale, moyenne et maximale
-- en effectuant tous les cumuls possibles :
-- pour chaque étudiant auquel cas le 2e champ est NULL,
-- pour chaque diplôme auquel cas le 1er champ est NULL, et
-- pour tous auquel cas les 1er et 2e champs sont NULL
SELECT NoINE , IntitAbrege , COUNT(*) , MIN(Annee) , AVG(Annee) ,
    MAX(Annee)
FROM AvoirObtenu
GROUP BY CUBE( NoINE , IntitAbrege )
ORDER BY NoINE ASC NULLS LAST , IntitAbrege ASC NULLS LAST
```

2	'BAC'	1	1980	1980	1980
2	'DEUG'	1	1982	1982	1982
2		2	1980	1981	1982
3	'BAC'	1	1981	1981	1981
3	'DUT'	1	1983	1983	1983
3	'MIAGe'	1	1985	1985	1985
3		3	1981	1983	1985
4	'BAC'	1	1977	1977	1977
4	'DEUG'	1	1980	1980	1980
4	'MIAGe'	1	1982	1982	1982
4		3	1977	1979.66...	1982
5	'BAC'	1	1981	1981	1981
5	'DUT'	1	1983	1983	1983
5		2	1981	1982	1983
	'BAC'	4	1977	1979.75	1981
	'DEUG'	2	1980	1981	1982
	'DUT'	2	1983	1983	1983
	'MIAGe'	2	1982	1983.5	1985
		10	1977	1981.4	1985

```
-- les seuls cumuls pour chaque étudiant, pour chaque diplôme et pour
-- tous les numéros des étudiants, intitulés abrégés des diplômes et
-- nombre de tuples
SELECT NoINE , IntitAbrege , COUNT(*)
FROM AvoirObtenu
GROUP BY GROUPING SETS ( NoINE , IntitAbrege , () )
```

2		2
3		3
4		3
5		2
	'BAC'	4
	'DEUG'	2
	'DUT'	2
	'MIAGe'	2
		10

Agrégation (4/4)

```
-- les numéros des étudiants, intitulés abrégés des diplômes,
-- et nombre de tuples en effectuant les cumuls
-- pour chaque étudiant et pour tous (mais pas pour chaque diplôme)
SELECT NoINE , IntitAbrege , COUNT(*)
FROM AvoirObtenu
GROUP BY ROLLUP( NoINE , IntitAbrege )
```

2	'BAC'	1
2	'DEUG'	1
2		2
3	'BAC'	1
3	'DUT'	1
3	'MIAGe'	1
3		3
4	'BAC'	1
4	'DEUG'	1
4	'MIAGe'	1
4		3
5	'BAC'	1
5	'DUT'	1
5		2
		10

Sélection après agrégation

```
-- les noms des étudiants homonymes
SELECT NomEtu
FROM Etudiants
GROUP BY NomEtu
HAVING COUNT(*) >= 2
```

'LEROI'

```
-- les étudiants ayant au moins deux voitures rouges
SELECT NoINE
FROM Voitures
WHERE Couleur = 'rouge'
GROUP BY NoINE -- ou NoINE , Couleur car 1! couleur, sans changer SELECT
HAVING COUNT(*) >= 2
```


```
-- pour tous les éléments de AvoirObtenu, leur nombre et les années
-- d'obtention minimale, moyenne et maximale si le nombre est supérieur
-- à 8 et si l'année moyenne d'obtention est comprise entre 1980 et 1982
SELECT COUNT(*) , MIN(Annee) , AVG(Annee) , MAX(Annee)
FROM AvoirObtenu
HAVING COUNT(*) > 8 AND AVG(Annee) BETWEEN 1980 AND 1982
```

10	1977	1981.4	1985

```
-- le nombre d'étudiants s'il y a au moins un étudiant non girondin
SELECT COUNT(*)
FROM Etudiants
HAVING ANY( DepartNaissEtu <> 33 )
```

5

Produit cartésien

```
-- (toutes) les voitures avec pour chacune (tous) les étudiants
SELECT *
FROM Voitures
CROSS JOIN Etudiants
-- équivalent en SQL-89 : SELECT * FROM Voitures , Etudiants
```

3333	'BX'	33	'rouge'	5	5	'DURAND'	33
3333	'BX'	33	'rouge'	5	2	'LEROI'	40
3333	'BX'	33	'rouge'	5	4	'MARTIN'	47
3333	'BX'	33	'rouge'	5	7	'LEROI'	33
3333	'BX'	33	'rouge'	5	3	'DUPOND'	17
4747	'LA'	47	'rouge'	4	5	'DURAND'	33
4747	'LA'	47	'rouge'	4	2	'LEROI'	40
4747	'LA'	47	'rouge'	4	4	'MARTIN'	47
4747	'LA'	47	'rouge'	4	7	'LEROI'	33
4747	'LA'	47	'rouge'	4	3	'DUPOND'	17
4040	'NT'	40	'jaune'	5	5	'DURAND'	33
4040	'NT'	40	'jaune'	5	2	'LEROI'	40
4040	'NT'	40	'jaune'	5	4	'MARTIN'	47
4040	'NT'	40	'jaune'	5	7	'LEROI'	33
4040	'NT'	40	'jaune'	5	3	'DUPOND'	17

Jointure naturelle (i. e. sur des attributs de même nom, souvent les clés primaire et étrangère)

```
-- les voitures et pour chacune l'étudiant qui la possède
SELECT *
FROM Voitures
NATURAL JOIN Etudiants
-- équivalent en SQL-89 : SELECT *
--                        FROM Voitures , Etudiants
--                        WHERE Voitures.NoINE = Etudiants.NoINE
```

3333	'BX'	33	'rouge'	5	'DURAND'	33
4040	'NT'	40	'jaune'	5	'DURAND'	33
4747	'LA'	47	'rouge'	4	'MARTIN'	47

Jointure conditionnelle (1/2)

```
-- les couples d'étudiants dont le département de naissance du premier
-- est plus petit que les départements de naissance du second qui doit
-- avoir le numéro 2 ou 7
SELECT E.* , E2.*
FROM Etudiants E -- on renomme Etudiants par E
JOIN Etudiants E2 ON E.DepartNaissEtu < E2.DepartNaissEtu
WHERE E2.NoINE IN ( 2 , 7 )
```

5	'DURAND'	33	2	'LEROI'	40
7	'LEROI'	33	2	'LEROI'	40
3	'DUPOND'	17	2	'LEROI'	40
3	'DUPOND'	17	7	'LEROI'	33

Jointure conditionnelle (2/2)

```
-- les étudiants nés là où les voitures sont immatriculées
SELECT *
FROM Etudiants
JOIN Voitures ON DepartNaissEtu = NoImmatDepart
```

3333	'BX'	33	'rouge'	5	5	'DURAND'	33
3333	'BX'	33	'rouge'	5	7	'LEROI'	33
4747	'LA'	47	'rouge'	4	4	'MARTIN'	47
4040	'NT'	40	'jaune'	5	2	'LEROI'	40

Jointures externes :

Jointure [naturelle] externe à gauche : `LEFT OUTER [NATURAL] JOIN`
Jointure [naturelle] externe à droite : `RIGHT OUTER [NATURAL] JOIN`
Jointure [naturelle] externe généralisée : `FULL OUTER [NATURAL] JOIN`

```
-- les étudiants et leurs voitures,
-- en faisant apparaître les étudiants qui n'ont pas de voitures
SELECT *
FROM Etudiants
LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
-- i. e. LEFT OUTER NATURAL JOIN Voitures
```

5	'DURAND'	33	3333	'BX'	33	'rouge'
5	'DURAND'	33	4040	'NT'	40	'jaune'
2	'LEROI'	40				
4	'MARTIN'	47	4747	'LA'	47	'rouge'
7	'LEROI'	33				
3	'DUPOND'	17				

```
-- les étudiants qui n'ont pas de diplôme
SELECT Etudiants.*
FROM Etudiants
LEFT OUTER NATURAL JOIN AvoirObtenu
WHERE IntitAbrege IS NULL
```

7	'LEROI'	33

```
-- les étudiants et l'intitulé de leurs diplômes,
-- en précisant "<aucun>" si un étudiant n'a pas obtenu de diplôme
SELECT E.* , COALESCE(CAST(IntitAbrege AS CHAR),'<aucun>')
FROM Etudiants E
LEFT OUTER NATURAL JOIN AvoirObtenu
ORDER BY E.NoINE , IntitAbrege
```

2	'LEROI'	40	'BAC'
2	'LEROI'	40	'DEUG'
3	'DUPOND'	17	'BAC'
3	'DUPOND'	17	'DUT'
3	'DUPOND'	17	'MIAGe'
4	'MARTIN'	47	'BAC'
4	'MARTIN'	47	'DEUG'
4	'MARTIN'	47	'MIAGe'
5	'DURAND'	33	'BAC'
5	'DURAND'	33	'DUT'
7	'LEROI'	33	'<aucun>'

Requête imbriquée ou sous-requête (1/4)

Remarques

Sous-requête au niveau des ordres `SELECT`, `FROM`, `WHERE` ou `HAVING`

Une sous-requête est dite corrélative lorsque la sous-requête utilise un (ou plusieurs) attribut(s) de la requête principale

```
-- les étudiants dont le département de naissance est plus petit que
-- le département de naissance de l'étudiant numéro 2
-- N. B. : préférer une proposition utilisant une auto-théta-jointure
SELECT *
FROM Etudiants
WHERE DepartNaissEtu < ( -- infériorité ==> 1! tuple 1! attribut
    SELECT DepartNaissEtu
    FROM Etudiants
    WHERE NoINE = 2 )
```

5	'DURAND'	33
7	'LEROI'	33
3	'DUPOND'	17

```
-- les étudiants qui possèdent au moins une voiture
-- N. B. : aussi mauvaise qu'une des requêtes suivantes
--         car une jointure suffit
SELECT *
FROM Etudiants
WHERE NoINE IN (
    SELECT DISTINCT NoINE -- peut retourner 0, 1 ou plusieurs tuples
    FROM Voitures )
```

5	'DURAND'	33
4	'MARTIN'	47

```
-- le nombre le plus élevé de diplômes obtenus
-- N. B. : SELECT MAX(COUNT(...)) est interdit
SELECT MAX(Nombre)
FROM ( SELECT COUNT(*) Nombre
       FROM AvoirObtenu
       GROUP BY NoINE )
```

3

```
-- les étudiants et leur nombre de voitures, pour ceux ayant moins de
-- voitures que le nombre maximum de voitures possédées par les
-- étudiants
-- N. B. : privilégiez l'une des requêtes suivantes utilisant une
--         expression de table
SELECT E.* , COUNT(*)
FROM Etudiants E
NATURAL JOIN Voitures
GROUP BY E.*
HAVING COUNT(*) <  ( SELECT MAX(NbVoitParEtu)
                     FROM ( SELECT COUNT(*) NbVoitParEtu
                            FROM Voitures
                            GROUP BY NoINE ) )
```

4	'MARTIN'	47	1

Requête imbriquée ou sous-requête (2/4)

```
-- les étudiants qui ont obtenu plus de diplômes (et au moins un)
-- que la moyenne du nombre de diplômes par étudiant
SELECT E.* , COUNT(*) NombreDeDiplomes
FROM Etudiants E
NATURAL JOIN AvoirObtenu
GROUP BY E.*
HAVING COUNT(*) > ( ( SELECT COUNT(*) FROM AvoirObtenu ) /
                    ( SELECT COUNT(*) FROM Etudiants ) )
```

4	'MARTIN'	47	3
3	'DUPOND'	17	3

```
-- les étudiants ayant obtenu au moins un diplôme en 1983
-- N. B. : une jointure suffit
SELECT *
FROM Etudiants
WHERE ( NoINE , 1983 ) MATCH ( SELECT NoINE , Annee
                               FROM AvoirObtenu )
```

5	'DURAND'	33
3	'DUPOND'	17

```
-- les voitures dont le numéro d'immatriculation du département
-- correspond au département de naissance de l'étudiant qui possède
-- la voiture, et s'il a obtenu un et un seul diplôme après 1980
SELECT *
FROM Voitures
WHERE ( NoImmatDepart , NoINE ) MATCH UNIQUE (
    SELECT DepartNaissEtu , NoINE
    FROM Etudiants
    NATURAL AvoirObtenu
    WHERE Annee > 1980 )
```

4747	'LA'	47	'rouge'	4

```
-- les étudiants qui possèdent au moins une voiture
-- N. B. : aussi mauvaise qu'une des requêtes précédentes
--         car 1 jointure suffit
SELECT *
FROM Etudiants
WHERE EXISTS (
    SELECT NULL -- ou 'z' ou 9 ou * ou
                -- NoImmatChiffres,NoImmatLettres,NoImmatDepart
    FROM Voitures
    WHERE Voitures.NoINE = Etudiants.NoINE )
```

5	'DURAND'	33
4	'MARTIN'	47

Requête imbriquée ou sous-requête (3/4)

```
-- les étudiants qui possèdent au plus une voiture
SELECT *
FROM Etudiants E
WHERE UNIQUE ( SELECT ALL V.NoINE -- ou 'z' ou 9 ; ALL est facultatif
               FROM Voitures V
               WHERE V.NoINE = E.NoINE )
```

2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17

```
-- les étudiants
-- N. B. : très mauvais car une seule relation suffit
SELECT *
FROM Etudiants
WHERE UNIQUE (
    SELECT * -- ou NoImmatChiffres,NoImmatLettres,NoImmatDepart
    FROM Voitures V
    WHERE Voitures.NoINE = Etudiants.NoINE )
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17

```
-- les diplômes obtenus plusieurs fois la même année,
-- sans considérer l'année 1983 qui est sortie des valeurs
-- N. B. : Annee est un attribut obligatoire
SELECT *
FROM Diplomes
WHERE NOT UNIQUE (
    SELECT NULLIF(Annee,1983)
    FROM AvoirObtenu
    WHERE AvoirObtenu.IntitAbrege = Diplomes.IntitAbrege )
```

'DUT'	'Diplôme Universitaire de Technologie'
'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
'DEUG'	'Diplôme d'Études Universitaires Générales'

```
-- les diplômes obtenus plusieurs fois la même année
SELECT *
FROM Diplomes
WHERE NOT DISTINCT (
    SELECT Annee
    FROM AvoirObtenu
    WHERE AvoirObtenu.IntitAbrege = Diplomes.IntitAbrege )
```

'DUT'	'Diplôme Universitaire de Technologie'
'BAC'	'Baccalauréat'

Requête imbriquée ou sous-requête (4/4)

```
-- les étudiants non girondins ayant obtenu plus de diplômes
-- que tous les étudiants girondins
SELECT E.*
FROM Etudiants E
NATURAL JOIN AvoirObtenu
WHERE DepartNaiss <> 33
GROUP BY E.*
HAVING COUNT(*) > ALL (
    SELECT COUNT(*)
    FROM Etudiants E
    NATURAL JOIN AvoirObtenu
    WHERE DepartNaiss = 33
    GROUP BY E.* )
```

4	'MARTIN'	47
3	'DUPOND'	17

```
-- les étudiants ayant obtenu un diplôme
-- avant l'une quelconque des années d'obtention du baccalauréat
SELECT DISTINCT E.*
FROM Etudiants E
NATURAL JOIN AvoirObtenu
WHERE Annee < ANY ( SELECT Annee
                    FROM AvoirObtenu
                    WHERE IntitAbrege = 'BAC' )
```

2	'LEROI'	40
4	'MARTIN'	47

Collection

```
-- somme des premiers nombres impairs, à l'aide d'un MULTISET transformé
-- en relation grâce à la fonction UNNEST
SELECT 'Somme des' , COUNT(Impairs.Impair) ,
       'premiers nombres impairs =' , SUM(Impairs.Impair)
FROM UNNEST ( MULTISET ( 1 , 3 , 5 , 7 , 9 , 11 , 13 ) )
     AS Impairs ( Impair )
```

'Somme des'	7	'premiers nombres impairs ='	49

```
-- pour chaque étudiant, toutes ses informations suivies de la liste de
-- toutes les couleurs (différentes deux à deux) des voitures
SELECT * ,
       MULTISET ( SELECT DISTINCT Couleur FROM Voitures ) AS Couleurs
FROM Etudiants
```

5	'DURAND'	33	['rouge' , 'jaune']
2	'LEROI'	40	['rouge' , 'jaune']
4	'MARTIN'	47	['rouge' , 'jaune']
7	'LEROI'	33	['rouge' , 'jaune']
3	'DUPOND'	17	['rouge' , 'jaune']

Fenêtrage et ordonnancement

```
-- les diplômes et les étudiants qui les ont obtenus, avec un numéro
-- d'ordre pour chaque diplôme selon l'année, le numéro et le nom de
-- l'étudiant, l'année, le nombre de diplômés ainsi que l'année minimale
-- et maximale pour le diplôme, et le nombre total de tuples
SELECT IntitAbrege ,
       RANK() OVER ( PARTITION BY IntitAbrege ORDER BY Annee ) ,
       NoINE , NomEtu , Annee ,
       COUNT(*) OVER ( PARTITION BY IntitAbrege ) ,
       MIN(Annee) OVER ( PARTITION BY IntitAbrege ) ,
       MAX(Annee) OVER ( PARTITION BY IntitAbrege ) ,
       COUNT(*) OVER ()
FROM AvoirObtenu
NATURAL JOIN Etudiants
ORDER BY IntitAbrege , 2
```

'BAC'	1	4	'MARTIN'	1977	4	1977	1981	10
'BAC'	2	2	'LEROI'	1980	4	1977	1981	10
'BAC'	3	3	'DUPOND'	1981	4	1977	1981	10
'BAC'	3	5	'DURAND'	1981	4	1977	1981	10
'DEUG'	1	4	'MARTIN'	1980	2	1980	1982	10
'DEUG'	2	2	'LEROI'	1982	2	1980	1982	10
'DUT'	1	3	'DUPOND'	1983	2	1983	1983	10
'DUT'	1	5	'DURAND'	1983	2	1983	1983	10
'MIAGe'	1	4	'MARTIN'	1982	2	1982	1985	10
'MIAGe'	2	3	'DUPOND'	1985	2	1982	1985	10

```
-- les étudiants avec un numéro de ligne dans l'ordre du numéro
-- d'étudiant, le numéro, le nom, et deux numéros d'ordre sur le nom
-- (avec des ex æquo pour les homonymes) le premier ne récupérant pas
-- les numéros laissés libres par les ex æquo contrairement au second
SELECT ROW_NUMBER() OVER ( ORDER BY NoINE ) , NoINE , NomEtu ,
       RANK() OVER ( ORDER BY NomEtu ) ,
       DENSE_RANK() OVER ( ORDER BY NomEtu )
FROM Etudiants
ORDER BY NomEtu
```

2	3	'DUPOND'	1	1
4	5	'DURAND'	2	2
1	2	'LEROI'	3	3
5	7	'LEROI'	3	3
3	4	'MARTIN'	5	4

Expression de table

```
-- les étudiants ayant moins de voitures que le nombre maximum de
-- voitures possédées par les étudiants
WITH VoituresParEtudiant ( NoINE , NbVoitParEtu ) AS (
    SELECT NoINE , COUNT(*)
    FROM Voitures
    GROUP BY NoINE )
SELECT E.*
FROM Etudiants E
NATURAL JOIN VoituresParEtudiant
WHERE NbVoitParEtu <
    ( SELECT MAX(NbVoitParEtu)
      FROM VoituresParEtudiant )
```

4	'MARTIN'	47	1

```
-- les personnes avec leur niveau dans l'arborescence des parrainages
WITH RECURSIVE Parrainages ( NoPers , Niveau ) AS (
    -- racine de niveau 1
    SELECT NoPers , 1
    FROM Personnes
    WHERE NoPersParrain IS NULL -- la racine n'a pas de parrain
    UNION
    -- enfants d'un niveau donné
    SELECT Personnes.NoPers , Niveau+1
    FROM Personnes
    JOIN Parrainages ON Personnes.NoPersParrain = Parrainages.NoPers )
SELECT *
FROM Parrainages
```

Pour

Personnes

NoPers	NoPersParrain
5	2
2	4
4	7
7	
3	2

, on obtient

Parrainages

NoPers	Niveau
7	1
4	2
2	3
5	4
3	4

Opérations ensemblistes (1/2)

◁ requête d'interrogation ▷ { `UNION` [`ALL`] , `INTERSECT` , `EXCEPT` } ◁ requête d'interrogation ▷ ...

Remarques

Toutes les clauses `SELECT` doivent avoir le même nombre d'attributs

Le premier attribut de toutes les clauses `SELECT` doivent être de types compatibles ; il doit en être de même pour les deuxième, troisième, ..., dernier attributs

Il y a au plus une clause `ORDER BY` située à la fin de la requête ensembliste

Union (1/2)

```
-- les numéros des étudiants, intitulés abrégés des diplômes, le nombre
-- de tuples regroupés et un codage des différents SELECT de la requête,
-- en effectuant tous les cumuls possibles (pour chaque étudiant, pour
-- chaque diplôme et pour tous)
-- N. B. : privilégiez la version utilisant CUBE()
SELECT NoINE , IntitAbrege , 1 , 'a' -- ( NoINE , IntitAbrege ) est la
                                     -- clé d'AvoirObtenu
FROM AvoirObtenu
UNION
SELECT NoINE , NULL , COUNT(*) , 'b'
FROM AvoirObtenu
GROUP BY NoINE
UNION
SELECT NULL , IntitAbrege , COUNT(*) , 'c'
FROM AvoirObtenu
GROUP BY IntitAbrege
UNION
SELECT NULL , NULL , COUNT(*) , 'd'
FROM AvoirObtenu
ORDER BY 1 ASC NULLS LAST , 2 ASC NULLS LAST
```

2	'BAC'	1	'a'
2	'DEUG'	1	'a'
2		2	'b'
3	'BAC'	1	'a'
3	'DUT'	1	'a'
3	'MIAGe'	1	'a'
3		3	'b'
4	'BAC'	1	'a'
4	'DEUG'	1	'a'
4	'MIAGe'	1	'a'
4		3	'b'
5	'BAC'	1	'a'
5	'DUT'	1	'a'
5		2	'b'
	'BAC'	4	'c'
	'DEUG'	2	'c'
	'DUT'	2	'c'
	'MIAGe'	2	'c'
		10	'd'

Opérations ensemblistes (2/2)

Union (2/2)

```
-- tous les numéros de départements
SELECT DepartNaissEtu -- de naissance des étudiants
FROM Etudiants
UNION -- sans répétition par défaut, UNION ALL sinon
SELECT NoImmatDepart -- d'immatriculation des voitures
FROM Voitures
ORDER BY 1
```

17
33
40
47

Intersection

```
-- les numéros de départements communs
SELECT DepartNaissEtu -- de naissance des étudiants
FROM Etudiants
INTERSECT
SELECT NoImmatDepart -- d'immatriculation des voitures
FROM Voitures
ORDER BY 1
```

33
40
47

Différence

```
-- les numéros de départements de naissance des étudiants qui ne sont
-- pas des numéros de départements d'immatriculation de voitures
SELECT DepartNaissEtu
FROM Etudiants
EXCEPT
SELECT NoImmatDepart
FROM Voitures
```

17

```
-- les numéros de départements d'immatriculation de voitures qui ne sont
-- pas des numéros de départements de naissance des étudiants
SELECT NoImmatDepart
FROM Voitures
EXCEPT
SELECT DepartNaissEtu
FROM Etudiants
```


Exécution pas à pas requêtes interroga° (1/9)

Remarque : l'optimiseur

Le moteur SQL dispose en réalité d'un optimiseur de requête d'interrogation qui effectue un traitement plus efficace que les exécutions pas à pas présentées ci-après ; cependant, cette démarche est fondamentale pour la bonne compréhension d'une requête d'interrogation et plus particulièrement d'une requête d'interrogation complexe

Plusieurs requêtes d'interrogation différentes peuvent effectuer exactement le même traitement (et donc permettre d'obtenir le même résultat d'exécution) mais leurs temps de réponse peuvent varier considérablement (les stratégies de l'optimiseur ne seront pas les mêmes suivant l'écriture de la requête d'interrogation)

Principe

Pour une requête d'interrogation simple (ni opération ensembliste, ni sous-requête, etc.), le résultat se constitue en exécutant successivement les clauses `FROM` (produit cartésien entre toutes les relations), `JOIN` (jointures), `WHERE` (sélections), `GROUP BY` (agrégation), `HAVING` (sélections après l'agrégation), `SELECT` (projection et dénomination), `ORDER BY` (tri)

Pour une requête d'interrogation plus complexe, il faut évaluer indépendamment les requêtes reliées par une opération ensembliste et, pour les sous-requêtes non corrélées, la sous-requête la plus imbriquée pour reconstituer la requête principale au fur et à mesure des résultats obtenus pour les différentes sous-requêtes

Premier exemple (1/4)

```
-- pour chaque diplôme, excepté le DEUG et ceux dont la première année
-- d'obtention est strictement inférieure à 1983, le nombre d'étudiants
-- (sauf celui de numéro 2) l'ayant obtenu suivi des intitulés du diplôme
-- et de la dernière année d'obtention, le résultat étant trié sur ce
-- dernier critère en ordre croissant
SELECT COUNT(*) NbDiplomes , AO.IntitAbrege , IntitComplet ,
       MAX(Annee) DernAnnee
FROM AvoirObtenu AO
NATURAL JOIN Diplomes
WHERE IntitAbrege <> 'DEUG' AND NoINE <> 2
GROUP BY AO.IntitAbrege , IntitComplet
HAVING MIN(Annee) < 1983
ORDER BY 4 ASC
```

Exécution du produit cartésien

NoINE	AO.IntitAbrege	Annee	Diplomes.IntitAbrege	IntitComplet
4	'DEUG'	1980	'DUT'	'Diplôme Universitaire de Technologie'
4	'DEUG'	1980	'BAC'	'Baccalauréat'
4	'DEUG'	1980	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
4	'DEUG'	1980	'DEUG'	'Diplôme d'Études Universitaires Générales'
2	'DEUG'	1982	'DUT'	'Diplôme Universitaire de Technologie'
2	'DEUG'	1982	'BAC'	'Baccalauréat'
2	'DEUG'	1982	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
2	'DEUG'	1982	'DEUG'	'Diplôme d'Études Universitaires Générales'
5	'DUT'	1983	'DUT'	'Diplôme Universitaire de Technologie'
5	'DUT'	1983	'BAC'	'Baccalauréat'
5	'DUT'	1983	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
5	'DUT'	1983	'DEUG'	'Diplôme d'Études Universitaires Générales'
4	'MIAGe'	1982	'DUT'	'Diplôme Universitaire de Technologie'
4	'MIAGe'	1982	'BAC'	'Baccalauréat'
4	'MIAGe'	1982	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
4	'MIAGe'	1982	'DEUG'	'Diplôme d'Études Universitaires Générales'
3	'DUT'	1983	'DUT'	'Diplôme Universitaire de Technologie'
3	'DUT'	1983	'BAC'	'Baccalauréat'
3	'DUT'	1983	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
3	'DUT'	1983	'DEUG'	'Diplôme d'Études Universitaires Générales'
2	'BAC'	1980	'DUT'	'Diplôme Universitaire de Technologie'
2	'BAC'	1980	'BAC'	'Baccalauréat'
2	'BAC'	1980	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
2	'BAC'	1980	'DEUG'	'Diplôme d'Études Universitaires Générales'
4	'BAC'	1977	'DUT'	'Diplôme Universitaire de Technologie'
4	'BAC'	1977	'BAC'	'Baccalauréat'
4	'BAC'	1977	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
4	'BAC'	1977	'DEUG'	'Diplôme d'Études Universitaires Générales'
3	'MIAGe'	1985	'DUT'	'Diplôme Universitaire de Technologie'
3	'MIAGe'	1985	'BAC'	'Baccalauréat'
3	'MIAGe'	1985	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
3	'MIAGe'	1985	'DEUG'	'Diplôme d'Études Universitaires Générales'
5	'BAC'	1981	'DUT'	'Diplôme Universitaire de Technologie'
5	'BAC'	1981	'BAC'	'Baccalauréat'
5	'BAC'	1981	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
5	'BAC'	1981	'DEUG'	'Diplôme d'Études Universitaires Générales'
3	'BAC'	1981	'DUT'	'Diplôme Universitaire de Technologie'
3	'BAC'	1981	'BAC'	'Baccalauréat'
3	'BAC'	1981	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
3	'BAC'	1981	'DEUG'	'Diplôme d'Études Universitaires Générales'

Exécution de la jointure

NoINE	IntitAbrege	Annee	IntitComplet
4	'DEUG'	1980	'Diplôme d'Études Universitaires Générales'
2	'DEUG'	1982	'Diplôme d'Études Universitaires Générales'
5	'DUT'	1983	'Diplôme Universitaire de Technologie'
4	'MIAGe'	1982	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
3	'DUT'	1983	'Diplôme Universitaire de Technologie'
2	'BAC'	1980	'Baccalauréat'
4	'BAC'	1977	'Baccalauréat'
3	'MIAGe'	1985	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
5	'BAC'	1981	'Baccalauréat'
3	'BAC'	1981	'Baccalauréat'

Premier exemple (2/4)

Exécution de la sélection

NoINE	IntitAbrege	Annee	IntitComplet
5	'DUT'	1983	'Diplôme Universitaire de Technologie'
4	'MIAGe'	1982	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
3	'DUT'	1983	'Diplôme Universitaire de Technologie'
4	'BAC'	1977	'Baccalauréat'
3	'MIAGe'	1985	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'
5	'BAC'	1981	'Baccalauréat'
3	'BAC'	1981	'Baccalauréat'

Exécution de l'agrégation

IntitAbrege	IntitComplet	`COUNT(*)`	`MAX`(Annee)	`MIN`(Annee)
'DUT'	'Diplôme Universitaire de Technologie'	2	1983	1983
'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'	2	1985	1982
'BAC'	'Baccalauréat'	3	1981	1977

Exécution de la sélection après agrégation

IntitAbrege	IntitComplet	`COUNT(*)`	`MAX`(Annee)
'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'	2	1985
'BAC'	'Baccalauréat'	3	1981

Exécution de la projection

NbDiplomes	IntitAbrege	IntitComplet	DernAnnee
2	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'	1985
3	'BAC'	'Baccalauréat'	1981

Exécution du tri : résultat de la première requête

NbDiplomes	IntitAbrege	IntitComplet	DernAnnee
3	'BAC'	'Baccalauréat'	1981
2	'MIAGe'	'Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises'	1985

Corollaire : tous les attributs (i. e. les fonctions d'agrégation ne sont donc pas concernées) qui figurent dans la clause `SELECT` doivent être dans la clause `GROUP BY`

Premier exemple (3/4)

Oracle (SQL*Plus)

```
SQL> SELECT COUNT(*) NbDiplomes , IntitAbrege , IntitComplet , MAX(Annee) DernAnnee
  2  FROM AvoirObtenu
  3  NATURAL JOIN Diplomes
  4  WHERE IntitAbrege <> 'DEUG' AND NoINE <> 2
  5  GROUP BY IntitAbrege , IntitComplet
  6  HAVING MIN(Annee) < 1983
  7  ORDER BY 4 ASC
  8  ;

NBDIPLOMES INTIT INTITCOMPLET                                                    DERNANNEE
---------- ----- ------------------------------------------------------------- ----------
         3 BAC   Baccalaureat                                                       1981
         2 MIAGe Maitrise des Methodes Informatiques Appliquees a la Gestion des Entreprises      1985
```

Oracle (SQL Developer)

```
SELECT COUNT(*) NbDiplomes , IntitAbrege , IntitComplet , MAX(Annee) DernAnnee
FROM AvoirObtenu
NATURAL JOIN Diplomes
WHERE IntitAbrege <> 'DEUG' AND NoINE <> 2
GROUP BY IntitAbrege , IntitComplet
HAVING MIN(Annee) < 1983
ORDER BY 4 ASC
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	NBDIPLOMES	INTITABREGE	INTITCOMPLET	DERNANNEE
1	3	BAC	Baccalaureat	1981
2	2	MIAGe	Maitrise des Methodes Informatiques Appliquees a la Gestion des Entreprises	1985

SQL Server

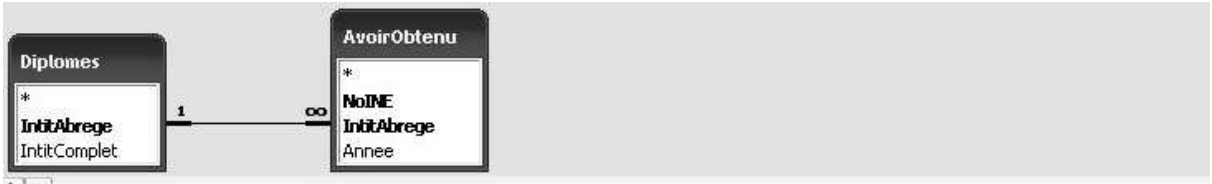
```
SELECT COUNT(*) NbDiplomes , AO.IntitAbrege , IntitComplet , MAX(Annee) DernAnnee
FROM AvoirObtenu AO
JOIN Diplomes ON AO.IntitAbrege = Diplomes.IntitAbrege
WHERE AO.IntitAbrege <> 'DEUG' AND NoINE <> 2
GROUP BY AO.IntitAbrege , IntitComplet
HAVING MIN(Annee) < 1983
ORDER BY 4 ASC
```

Résultats | Messages

	NbDiplomes	IntitAbrege	IntitComplet	DernAnnee
1	3	BAC	Baccalaureat	1981
2	2	MIAGe	Maitrise des Methodes Informatiques Appliquees a la Gestion des Entreprises	1985

Premier exemple (4/4)

Access



Champ :	NbDiplomes: Compte(*)	IntitAbrege	IntitComplet	DernAnnee: Annee	IntitAbrege	NoINE	Annee
Table :		Diplomes	Diplomes	AvoirObtenu	Diplomes	AvoirObtenu	AvoirObtenu
Opération :	Expression	Regroupement	Regroupement	Max	Où	Où	Min
Tri :				Croissant			
Afficher :	☑	☑	☑	☑	☐	☐	☐
Critères :					<>'DEUG'	<>2	<1983
Ou :							

```
SELECT Count(*) AS NbDiplomes, Diplomes.IntitAbrege, Diplomes.IntitComplet, Max(AvoirObtenu.Annee) AS DernAnnee
FROM Diplomes INNER JOIN AvoirObtenu ON Diplomes.IntitAbrege=AvoirObtenu.IntitAbrege
WHERE (((Diplomes.IntitAbrege)<>'DEUG') AND ((AvoirObtenu.NoINE)<>2))
GROUP BY Diplomes.IntitAbrege, Diplomes.IntitComplet
HAVING (((Min(AvoirObtenu.Annee))<1983))
ORDER BY Max(AvoirObtenu.Annee);
```

NbDiplomes	IntitAbrege	IntitComplet	DernAnnee
3	BAC	Baccalauréat	1981
2	MIAGe	Maîtrise des Méthodes Informatiques Appliquées à la Gestion des Entreprises	1985

Second exemple (1/4)

```
-- tous les étudiants avec leurs voitures éventuelles sauf ceux qui ont au
-- moins une voiture et au moins un diplôme en prenant parmi toutes les
-- combinaisons possibles de leurs voitures par leurs diplômes celles dont
-- le diplôme autre que le baccalauréat a été obtenu par au moins un
-- étudiant de numéro autre que 2 et 4
SELECT Etudiants.NoINE , NomEtu ,
       NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
EXCEPT
SELECT Etudiants.NoINE , NomEtu ,
       NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
NATURAL JOIN Voitures
NATURAL JOIN AvoirObtenu
WHERE IntitAbrege IN (
    SELECT IntitAbrege
    FROM AvoirObtenu
    WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4 ) )
ORDER BY Etudiants.NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
```

Exécution de la première partie de l'opération ensembliste (avant `EXCEPT`)

Etudiants.NoINE	NomEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart
5	'DURAND'	3333	'BX'	33
5	'DURAND'	4040	'NT'	40
2	'LEROI'			
4	'MARTIN'	4747	'LA'	47
7	'LEROI'			
3	'DUPOND'			

Exécution de la partie imbriquée de la sous-requête (entre `IN (` et `) ORDER BY`)

IntitAbrege
'DUT'
'DUT'
'MIAGe'

Exécution de la seconde partie de l'opération ensembliste (après `EXCEPT`)

Etudiants.NoINE	NomEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart
5	'DURAND'	3333	'BX'	33
5	'DURAND'	4040	'NT'	40
4	'MARTIN'	4747	'LA'	47

où l'étudiant de n° 5 apparaît deux fois car il a deux voitures (3333 BX 33 et 4040 NT 40) et un seul des diplômes retournés par la sous-requête (DUT) tandis que l'étudiant de n° 4 apparaît une seule fois car il a une voiture (4747 LA 47) et un seul des diplômes retournés par la sous-requête (MIAGe)

Exécution de l'opération ensembliste

Etudiants.NoINE	NomEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart
2	'LEROI'			
7	'LEROI'			
3	'DUPOND'			

Exécution du tri : résultat de la seconde requête

Etudiants.NoINE	NomEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart
2	'LEROI'			
3	'DUPOND'			
7	'LEROI'			

Second exemple (2/4)

Oracle (SQL*Plus)

```
SQL> SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
  2  FROM Etudiants
  3  LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
  4  MINUS
  5  SELECT NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
  6  FROM Etudiants
  7  NATURAL JOIN Voitures
  8  NATURAL JOIN AvoirObtenu
  9  WHERE IntitAbrege IN ( SELECT IntitAbrege
 10                         FROM AvoirObtenu
 11                         WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4 ) )
 12  ORDER BY NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
 13  ;
```

```
NOINE NOMETU NOIMMATCHIFFRES NOI NOIMMATDEPART
----- ------ ---------------- --- --------------
    2 LEROI
    3 DUPOND
    7 LEROI
```

Oracle (SQL Developer)

```
SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
MINUS
SELECT NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
NATURAL JOIN Voitures
NATURAL JOIN AvoirObtenu
WHERE IntitAbrege IN ( SELECT IntitAbrege
                       FROM AvoirObtenu
                       WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4 ) )
ORDER BY NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	NOINE	NOMETU	NOIMMATCHIFFRES	NOIMMATLETTRES	NOIMMATDEPART
1	2	LEROI	(null)	(null)	(null)
2	3	DUPOND	(null)	(null)	(null)
3	7	LEROI	(null)	(null)	(null)

Second exemple (3/4)

SQL Server

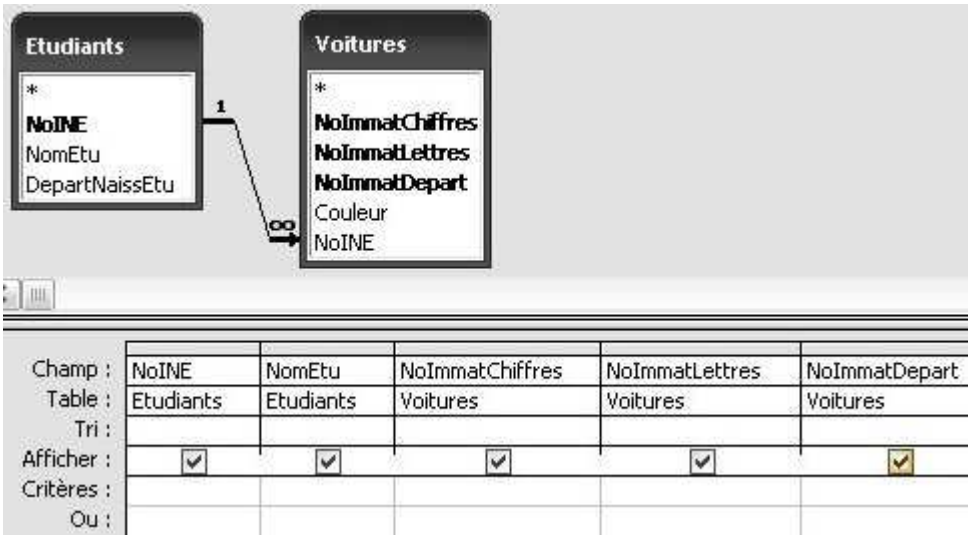
```
SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
EXCEPT
SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
JOIN AvoirObtenu ON Etudiants.NoINE = AvoirObtenu.NoINE
WHERE IntitAbrege IN ( SELECT IntitAbrege
                       FROM AvoirObtenu
                       WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4) )
ORDER BY Etudiants.NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
```

Résultats | Messages

	NoINE	NomEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart
1	2	LEROI	NULL	NULL	NULL
2	3	DUPOND	NULL	NULL	NULL
3	7	LEROI	NULL	NULL	NULL

Second exemple (4/4)

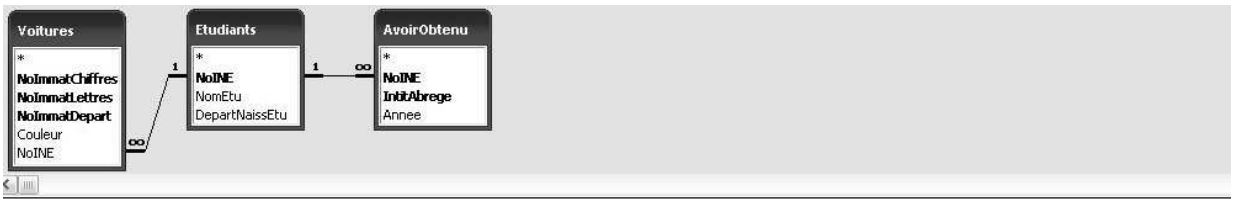
Access : en deux parties car l'opération de différence ensembliste n'existe pas



Champ :	NoINE	NomEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart
Table :	Etudiants	Etudiants	Voitures	Voitures	Voitures
Tri :					
Afficher :	☑	☑	☑	☑	☑
Critères :					
Ou :					

```
SELECT Etudiants.NoINE, Etudiants.NomEtu, Voitures.NoImmatChiffres, Voitures.NoImmatLettres, Voitures.NoImmatDepart
FROM Etudiants LEFT JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE;
```

NoINE	NomEtu	NoImmatChiffre	NoImmatLettres	NoImmatDepart
2	LEROI			
3	DUPOND			
4	MARTIN	4747	LA	47
5	DURAND	3333	BX	33
5	DURAND	4040	NT	40
7	LEROI			



Champ :	NoINE	NomEtu	NoImmatChiffres	NoImmatLettres	NoImmatDepart	IntitAbrege
Table :	Etudiants	Etudiants	Voitures	Voitures	Voitures	AvoirObtenu
Tri :						
Afficher :	☑	☑	☑	☑	☑	☐
Critères :						In (SELECT IntitAbrege FROM AvoirObtenu WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN (2 , 4))
Ou :						

```
SELECT Etudiants.NoINE, Etudiants.NomEtu, Voitures.NoImmatChiffres, Voitures.NoImmatLettres, Voitures.NoImmatDepart
FROM (Etudiants INNER JOIN AvoirObtenu ON Etudiants.NoINE = AvoirObtenu.NoINE) INNER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
WHERE (((AvoirObtenu.IntitAbrege) In (SELECT IntitAbrege FROM AvoirObtenu WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4 ) )));
```

NoINE	NomEtu	NoImmatChiffre	NoImmatLettres	NoImmatDepart
4	MARTIN	4747	LA	47
5	DURAND	3333	BX	33
5	DURAND	4040	NT	40

Insertion de tuples dans une relation (1/3)

`INSERT INTO` ◁ relation ▷
{ `VALUES (` ◁ expressions ▷ `)` , ◁ requête d'interrogation ▷ }

N. B. : les insertions sont effectuées successivement à partir des données initiales de l'exemple « jouet »

```
-- insérer l'étudiant numéro 1 JOURLY né en Dordogne
INSERT INTO Etudiants ( NoINE , NomEtu , DepartNaissEtu )
VALUES ( 1 , 'JOURLY' , 24 )
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17
1	'JOURLY'	24

```
-- insérer l'étudiant numéro 9 BILMET sans préciser son département de
-- naissance
INSERT INTO Etudiants ( NoINE , NomEtu )
VALUES ( 9 , 'BILMET' ) -- valeur par défaut pour DepartNaissEtu
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17
1	'JOURLY'	24
9	'BILMET'	33

```
-- insérer simultanément les étudiants numéros
-- 8 DUPENA né dans le Lot-et-Garonne et 6 RUNEN né en Charente-Maritime
INSERT INTO Etudiants
VALUES ( ( 8 , 'DUPENA' , 47 ) , ( 6 , 'RUNEN' , 17 ) )
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17
1	'JOURLY'	24
9	'BILMET'	33
8	'DUPENA'	47
6	'RUNEN'	17

Insertion de tuples dans une relation (2/3)

```
-- insérer l'étudiant LETOUR né dans les Pyrénées-Atlantiques en utilisant
-- l'auto-incrémentation du NoINE via le générateur de la séquence
-- SequenceNoINE
INSERT INTO Etudiants -- ( NoINE , NomEtu , DepartNaissEtu )
VALUES ( NEXT VALUE FOR SequenceNoINE , 'LETOUR' , 64 )
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17
1	'JOURLY'	24
9	'BILMET'	33
8	'DUPENA'	47
6	'RUNEN'	17
10	'LETOUR'	64

```
-- insérer l'étudiant MONLIX né dans les Landes en utilisant
-- l'auto-incrémentation du NoINE via le générateur par défaut de
-- l'attribut Etudiants.NoINE
INSERT INTO Etudiants ( NomEtu , DepartNaissEtu )
VALUES ( 'MONLIX' , 40 )
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17
1	'JOURLY'	24
9	'BILMET'	33
8	'DUPENA'	47
6	'RUNEN'	17
10	'LETOUR'	64
11	'MONLIX'	40

Insertion de tuples dans une relation (3/3)

```
-- insérer les voitures parmi les étudiants
INSERT INTO Etudiants
SELECT NoImmatDepart , TRIM(CAST(NoImmatChiffres AS VARCHAR)) ||
       NoImmatLettres || TRIM(CAST(NoImmatDepart AS VARCHAR)) ,
       NoImmatDepart
FROM Voitures
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'DUPOND'	17
1	'JOURLY'	24
9	'BILMET'	33
8	'DUPENA'	47
6	'RUNEN'	17
10	'LETOUR'	64
11	'MONLIX'	40
33	'3333BX33'	33
47	'4747LA47'	47
40	'4040NT40'	40

Modification de tuples d'une relation (1/2)

`UPDATE` ◁ relation ▷
`SET` ◁ attributs ▷ = { ◁ expressions ▷ , ◁ résultat d'une requête ▷ }
[`WHERE` ◁ condition ▷]

N. B. : les modifications sont effectuées successivement à partir des données initiales de l'exemple « jouet »

```
-- modifier tous les diplômes en mettant leurs intitulés
-- (abrégé et complet) en majuscule
UPDATE Diplomes
SET IntitAbrege = UPPER(IntitAbrege) , IntitComplet = UPPER(IntitComplet)
```

'DUT'	'DIPLÔME UNIVERSITAIRE DE TECHNOLOGIE'
'BAC'	'BACCALAURÉAT'
'MIAGE'	'MAÎTRISE DES MÉTHODES INFORMATIQUES APPLIQUÉES À LA GESTION DES ENTREPRISES'
'DEUG'	'DIPLÔME D'ÉTUDES UNIVERSITAIRES GÉNÉRALES'

```
-- modifier le nom des étudiants numéros 3 et 5 en ROUGE
UPDATE Etudiants
SET NomEtu = 'ROUGE'
WHERE NoINE IN ( 3 , 5 )
```

5	'ROUGE'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33
3	'ROUGE'	17

Modification de tuples d'une relation (2/2)

```
-- modifier le département de naissance des étudiants n'ayant pas un nom de
-- couleur de voiture (indépendamment de la casse) en affectant comme
-- valeur le plus petit des départements d'immatriculation des voitures
UPDATE Etudiants
SET DepartNaissEtu = (
    SELECT MIN(NoImmatDepart)
    FROM Voitures )
WHERE UPPER(NomEtu) NOT IN (
    SELECT DISTINCT UPPER(Couleur)
    FROM Voitures )
```

5	'ROUGE'	33
2	'LEROI'	33
4	'MARTIN'	33
7	'LEROI'	33
3	'ROUGE'	17

Insertion et modification de tuples dans une relation (1/2)

```
MERGE INTO ◁ relation ▷
USING ◁ requête d'interrogation ▷
ON ◁ condition ▷
WHEN NOT MATCHED THEN -- ◁ condition ▷ vaut TRUE
    INSERT VALUES ( ◁ expressions ▷ )
WHEN MATCHED THEN -- ◁ condition ▷ vaut FALSE ou UNKNOWN
    UPDATE SET ◁ attributs ▷ = ◁ expressions ▷
```

N. B. : les insertions/modifications sont effectuées successivement à partir des données initiales de l'exemple « jouet »

```
-- insérer ou modifier les diplômes BAC, L1, L2, L3 et MIAGe
-- selon que cet intitulé est absent ou non parmi les données
-- en rendant pour chacun d'eux, et dans tous les cas (insertion ou
-- modification), l'intitulé complet identique à l'intitulé abrégé
MERGE INTO Diplomes
USING
    SELECT Intit
    FROM UNNEST ( MULTISET ( 'BAC' , 'L1' , 'L2' , 'L3' , 'MIAGe' ) )
        AS DiplomesAFusionner ( Intit )
ON Intit = IntitAbrege
WHEN NOT MATCHED THEN
    INSERT ( IntitAbrege , IntitComplet ) VALUES ( Intit , Intit )
WHEN MATCHED THEN
    UPDATE SET IntitComplet = IntitAbrege
```

'DUT'	'Diplôme Universitaire de Technologie'
'BAC'	'BAC'
'MIAGe'	'MIAGe'
'DEUG'	'Diplôme d'Études Universitaires Générales'
'L1'	'L1'
'L2'	'L2'
'L3'	'L3'

Insertion et modification de tuples dans une relation (2/2)

```
-- insérer ou modifier les étudiants de numéro au plus 3 parmi les
-- voitures, selon que le département de naissance est différent ou égal au
-- numéro d'immatriculation du département,
-- en affectant dans tous les cas (insertion ou modification) la couleur
-- orange, en insérant comme voiture le département de naissance de
-- l'étudiant pour le numéro d'immatriculation en chiffre (en le
-- multipliant par 101) et pour le numéro d'immatriculation du département,
-- les deux premières lettres du nom pour le numéro d'immatriculation en
-- lettre, et le numéro d'étudiant pour lui-même
-- N. B. : la contrainte d'unicité de la clé de la relation Voitures est
--         violée lorsque l'on veut insérer plusieurs étudiants nés dans le
--         même département et dont les deux premières lettres du nom sont
--         identiques
MERGE INTO Voitures
USING
    SELECT NoINE , UPPER(SUBSTRING(NomEtu FROM 1 FOR 2)) DebutNomEtu ,
           DepartNaissEtu
    FROM Etudiants
    WHERE NoINE <= 3
ON DepartNaissEtu = NoImmatDepart
WHEN NOT MATCHED THEN
    INSERT ( NoImmatChiffres , NoImmatLettres , NoImmatDepart , Couleur ,
             NoINE )
    VALUES ( 101*DepartNaissEtu , DebutNomEtu , DepartNaissEtu , 'orange' ,
             NoINE )
WHEN MATCHED THEN
    UPDATE SET Couleur = 'orange'
```

3333	'BX'	33	'rouge'	5
4747	'LA'	47	'rouge'	4
4040	'NT'	40	'orange'	5
1717	'DU'	17	'orange'	3

Suppression de tuples d'une relation

`DELETE FROM` ◁ relation ▷
[`WHERE` ◁ condition ▷]

N. B. : les suppressions sont effectuées successivement à partir des données initiales de l'exemple « jouet »

```
-- supprimer tous les diplômes des étudiants de nom DUPOND ou LEROI
DELETE FROM AvoirObtenu
WHERE NoINE IN (
    SELECT NoINE
    FROM Etudiants
    WHERE NomEtu IN ( 'DUPOND' , 'LEROI' ) )
```

4	'DEUG'	1980
5	'DUT'	1983
4	'MIAGe'	1982
4	'BAC'	1977
5	'BAC'	1981

```
-- supprimer tous les étudiants nés en dehors de l'Aquitaine
DELETE FROM Etudiants
WHERE DepartNaissEtu NOT IN ( 24 , 33 , 40 , 47 , 64 )
```

5	'DURAND'	33
2	'LEROI'	40
4	'MARTIN'	47
7	'LEROI'	33

```
-- supprimer toutes les voitures
DELETE FROM Voitures
```


Catalogue

Description d'une BD

Ensemble de schémas

Description des fichiers physiques (disque, nom, taille, etc.) pour les données et le journal notamment

La norme ne décrit pas les ordres de création et suppression d'un catalogue laissant libre les éditeurs de SGBD mais la plupart proposent `CREATE DATABASE` et `DROP DATABASE`

Schéma

Ensemble cohérent de relations, vues, contraintes, domaines, jeux de caractères et collations (définissant l'ordre entre les caractères, l'égalité étant possible) et translations (permettant de remplacer des caractères par d'autres), privilèges, déclencheurs, routines (fonctions et procédures stockées), etc.

`CREATE SCHEMA` ◁ schéma ▷

`DROP SCHEMA` ◁ schéma ▷ { `CASCADE` , `RESTRICT` }

```
-- créer le schéma de l'exemple "jouet"
-- en utilisant par défaut le jeu de caractères ISO Latin 1
CREATE SCHEMA EtudiantsVoituresDiplomes DEFAULT CHARACTER SET 'Latin-1'
-- supprimer le schéma de l'exemple "jouet",
-- mais uniquement s'il n'y a pas de données
DROP SCHEMA EtudiantsVoituresDiplomes RESTRICT
```

Séquence : `CREATE SEQUENCE` ◁ séquence ▷ `AS` ◁ type ▷ `(` ◁ générateur ▷ `)`

```
-- séquence de 1 à 99 par pas de 1 en bouclant une fois le dernier numéro
-- attribué
CREATE SEQUENCE SequenceNoINE AS INT
    ( START WITH 1 INCREMENT BY 1 MAXVALUE 99 MINVALUE 1 CYCLE )
```

Contraintes d'intégrité

Remarque : les contraintes d'intégrité peuvent être déclaratives (définies en même temps que le schéma) ou algorithmiques (déclencheur, procédure stockée, fonction, méthode)

Contrainte de clé (primaire) : `PRIMARY KEY`

Chaque attribut composant la clé primaire est obligatoire (`NOT NULL`)

Tous les k-uplets formés des k attributs composant la clé primaire sont uniques (`UNIQUE`) ; en revanche, chaque attribut composant la clé primaire n'est pas unique (pour $k > 1$) sauf s'il compose la clé à lui seul ($k = 1$)

Contrainte d'intégrité référentielle

`FOREIGN KEY (` ◁ attributs ▷ `) REFERENCES` ◁ relation ▷ `(` ◁ attributs ▷ `)`
[`MATCH` ◁ option validation référence ▷] [`ON UPDATE` ◁ action màj ▷] [`ON DELETE` ◁ action màj ▷]
[[`NOT`] `DEFERRABLE`] [`INITIALLY {` `IMMEDIATE` , `DEFERRED` `}`]

◁ option validation référence ▷ := { `SIMPLE` , `PARTIAL` , `FULL` } précise, lorsque plusieurs attributs sont concernés (i. e. que la clé est composée de plusieurs attributs), que la contrainte s'applique : à tous les attributs si et seulement si ils sont tous renseignés (`SIMPLE`), à tous les attributs renseignés (`PARTIAL`), toujours sauf si aucun attribut n'est renseigné (`FULL`)

◁ action màj ▷ := { `NO ACTION` , `RESTRICT` , `CASCADE` , `SET NULL` , `SET DEFAULT` } pour interdire la modification ou la suppression (dans les deux relations, qui référence et qui est référencée) en fin de transaction (`NO ACTION`) ou dès la violation de contrainte (`RESTRICT`), ou pour répercuter la modification ou la suppression dans la relation qui est référencée dans celle qui référence en cascade (`CASCADE`) ou en enlevant la valeur (`SET NULL`) ou en affectant à la valeur par défaut (`SET DEFAULT`)

La contrainte peut ne pas être déférée (`NOT DEFERRABLE`) et doit alors s'appliquer immédiatement, ou peut être déférée (`DEFERRABLE`) et s'appliquer alors soit dès l'ordre de mise à jour (`INITIALLY IMMEDIATE`) soit en fin de transaction (`INITIALLY DEFERRED`)

Contrainte d'unicité : `UNIQUE`

Contrainte existentielle : `NOT NULL`

Contrainte de valeur par défaut : `DEFAULT`

Contrainte de « vérification » : `CHECK (` ◁ contrainte ▷ `)`

Contrainte de domaine : `CREATE DOMAIN` ◁ domaine ▷ ◁ type et contraintes ▷

```
CREATE DOMAIN DomaineNumero INT NOT NULL
CREATE DOMAIN DomaineIntitAbrege CHAR(8) NOT NULL
CREATE DOMAIN DomaineDepart INT DEFAULT 33
    CONSTRAINT ContrainteDomaineDepart
        CHECK ( VALUE BETWEEN 1 AND 95 OR VALUE BETWEEN 971 AND 974 OR VALUE = 9
    -- il faudrait en fait choisir CHAR(3) DEFAULT '33' pour avoir les
    -- 101 départements français depuis le 31 mars 2011 : '01' ... '09'
    -- '10' ... '19' '2A' '2B' '21' ... '95' '971' ... '974' '976'
```

Assertion : `CREATE ASSERTION` ◁ assertion ▷ `CHECK (` ◁ contrainte multi-relations ▷ `)`

Contrainte concernant des attributs de différentes relations

```
CREATE ASSERTION ContrainteDiplomeImpliqPasVoiture
    CHECK ( NOT EXISTS ( SELECT * FROM Voitures WHERE NoINE NOT IN
                            ( SELECT NoINE FROM AvoirObtenu ) ) )
```

Création d'une relation, avec ses attributs et des contraintes d'intégrité (1/2)

`CREATE TABLE` ◁ relation ▷ (◁ attributs avec leurs types et contraintes ▷ , ◁ contraintes ▷)

```
-- créer la relation des étudiants
CREATE TABLE Etudiants (
    NoINE DomaineNumero CONSTRAINT ContrainteIdEtudiants PRIMARY KEY
        GENERATED BY DEFAULT AS IDENTITY
            ( START WITH 1 INCREMENT BY 1 MAXVALUE 99 MINVALUE 1 CYCLE ) ,
    NomEtu VARCHAR(25) NOT NULL
        ContrainteNomEtuCollation COLLATE 'Latin-1' ,
    DepartNaissEtu DomaineDepart NOT NULL )
```

```
-- créer la relation des voitures
CREATE TABLE Voitures (
    NoImmatChiffres DomaineNumero
        CONSTRAINT ContrainteNoImmatChiffresBornes
            CHECK ( VALUE BETWEEN 1 AND 9999 ) ,
    NoImmatLettres CHAR(3) NOT NULL
        CONSTRAINT ContrainteNoImmatLettresMAJ
            CHECK ( VALUE = UPPER(VALUE) ) ,
    NoImmatDepart DomaineDepart NOT NULL ,
    Couleur VARCHAR(10) NOT NULL
        CONSTRAINT ContrainteListeCouleurs
            CHECK ( VALUE IN ( 'rouge' , 'jaune' , 'orange' ) )
        CONSTRAINT ContrainteDefautCouleur DEFAULT 'rouge' ,
    NoINE DomaineNumero
        CONSTRAINT ContrainteRefVoitures2Etudiants
            REFERENCES Etudiants ( NoINE )
                ON DELETE CASCADE ON UPDATE CASCADE ,
    CONSTRAINT ContrainteIdVoitures
        PRIMARY KEY ( NoImmatChiffres , NoImmatLettres , NoImmatDepart ) )
```

```
-- créer la relation des diplômes
CREATE TABLE Diplomes (
    IntitAbrege DomaineIntitAbrege
        CONSTRAINT ContrainteIdDiplomes PRIMARY KEY ,
    IntitComplet VARCHAR(100) NOT NULL
        CONSTRAINT ContrainteUniqIntitComplet UNIQUE ,
    CONSTRAINT ContrainteIntitMemesInitiales
        CHECK ( SUBSTR(IntitAbrege FROM 1 FOR 1) =
                SUBSTRING(IntitComplet FROM 1 FOR 1) ) )
```

Création d'une relation, avec ses attributs et des contraintes d'intégrité (2/2)

```
-- créer la relation des diplômes obtenus par des étudiants
CREATE TABLE AvoirObtenu (
    NoINE DomaineNumero ,
    IntitAbrege DomaineIntitAbrege ,
    Annee SMALLINT NOT NULL
        CONSTRAINT ContrainteAnneeAuMoins1950 CHECK ( VALUE >= 1950 )
        CONSTRAINT ContrainteDefautAnnee EXTRACT(YEAR FROM CURRENT_DATE) ,
    CONSTRAINT ContrainteIdAvoirObtenu
        PRIMARY KEY ( NoINE , IntitAbrege ) ,
    CONSTRAINT ContrainteRefAvoirObtenu2Etudiants
        FOREIGN KEY ( NoINE ) REFERENCES Etudiants ( NoINE )
            ON UPDATE CASCADE ON DELETE CASCADE ,
    CONSTRAINT ContrainteRefAvoirObtenu2Diplomes
        FOREIGN KEY ( IntitAbrege ) REFERENCES Diplomes ( IntitAbrege )
            ON UPDATE CASCADE ON DELETE CASCADE ,
    CONSTRAINT ContrainteAuPlus5DiplomesParEtudiant
        CHECK ( MAX ( SELECT COUNT(*) FROM AvoirObtenu GROUP BY NoINE )
                <= 5 ) )
```

Modification (de la structure) d'une relation :

`ALTER TABLE` ◁ relation ▷ { `ADD` , `DROP` , `MODIFY` } ◁ attribut ▷ ⋯

```
-- ajouter le prénom des étudiants
ALTER TABLE Etudiants ADD PrenomEtu VARCHAR(10)
-- changer la taille du prénom des étudiants
ALTER TABLE Etudiants MODIFY PrenomEtu VARCHAR(25)
-- supprimer le prénom des étudiants
ALTER TABLE Etudiants DROP PrenomEtu
```

Suppression d'une relation : `DROP TABLE` ◁ relation ▷

```
-- supprimer la relation des étudiants
DROP TABLE Etudiants
```

Photographie ou relation temporaire *(temporary base table)*

Relation contenant des données déduites de « relations de base » ou d'autres photographies

Syntaxe : `CREATE TEMPORARY TABLE` ◁ relation ▷ ...

Problèmes

- Perte d'espace de stockage
- Risque d'incohérence entre la photographie et les données initiales mises à jour ensuite

Cadre d'utilisation

- Accès privé à l'application qui l'utilise (et donc pas de concurrence)
- Vide en début de session, et effacée automatiquement à la fin de chaque session
- Peut être vidée à chaque `COMMIT`
- Modifiable même au sein d'une transaction en lecture seule

Vue

Permet de représenter la BD sous une forme plus proche de la vision qu'en a l'utilisateur (par exemple, des jointures sont réalisées pour rendre une information plus globale ou des informations calculables sont retournées ; de même, les données superflues ne sont pas affichées pour rendre une information plus simple)

Permet d'assurer une plus grande sécurité des données (par exemple, seules les informations présentées peuvent être mises à jour)

Remarque : une vue est dynamique (contrairement à une relation temporaire par exemple) i. e. seul le script de la vue est stocké, le résultat étant recherché à chaque utilisation de la vue

Il est rarement possible d'exécuter une mise à jour sur une vue, car cela est impossible (limitation théorique) ou parce que l'éditeur du SGBD n'a pas programmé l'algorithme correspondant ; la norme précise les règles à respecter pour qu'une vue puisse être mise à jour :

- `SELECT` sans `DISTINCT` ni expression ni fonction
- `FROM` et `JOIN` sur des relations ou vues qui puissent elles-mêmes être mises à jour
- Ni opération ensembliste (excepté `UNION ALL`), ni `GROUP BY` ou `HAVING`
- Une requête imbriquée ne peut l'être que dans le `WHERE`

`CREATE VIEW` ◁ vue ▷ [`(` ◁ attributs ▷ `)`] `AS` ◁ requête d'interrogation ▷ [`WITH CHECK OPTION`]

- Si `WITH CHECK OPTION` est précisé, les valeurs mises à jour par la vue doivent satisfaire la condition du `WHERE` comme s'il s'agissait d'une contrainte

`DROP VIEW` ◁ vue ▷

```
-- créer une vue pour les voitures des étudiants
CREATE VIEW VueVoituresDesEtudiants AS
SELECT * FROM Voitures NATURAL JOIN Etudiants
-- utiliser la vue pour les voitures des étudiants nés en Gironde
SELECT NoImmatChiffres , NoImmatLettres , NoImmatDepart , NomEtu
FROM VueVoituresDesEtudiants
WHERE DepartNaissEtu = 33
-- créer une vue pour les couleurs des voitures des étudiants nés en
-- Aquitaine
-- N. B. : vue basée sur une vue
CREATE VIEW VueCouleursVoituresDesEtudiantsAquitains AS
SELECT DISTINCT Couleur
FROM VueVoituresDesEtudiants
WHERE DepartNaissEtu IN ( 24 , 33 , 40 , 47 , 64 )
-- supprimer la vue des voitures des étudiants
DROP VIEW VueVoituresDesEtudiants
```

```
-- créer une vue pour les diplômes dont intitulé abrégé contient un chiffre
CREATE VIEW VueDiplomesContenant0a9 AS
SELECT *
FROM Diplomes
WHERE IntitAbrege SIMILAR TO '%[0-9]%' WITH CHECK OPTION
-- utilisation sans erreur de cette vue car "L3" contient le chiffre 3
INSERT INTO VueDiplomesContenant0a9 VALUES ( 'L3' , 'Licence 3e année' )
-- utilisation avec erreur de cette vue car pas de chiffre dans "Doc"
INSERT INTO VueDiplomesContenant0a9 VALUES ( 'Doc' , 'Doctorat' )
```

Dictionnaire des données (1/2)

Schéma composé des métadonnées décrivant complètement tous les schémas d'un catalogue

Schéma (dénommé `INFORMATION_SCHEMA`) stocké lui-même sous la forme d'un schéma relationnel (et donc consultable par des ordres SQL `SELECT`)

Il doit notamment posséder les relations ou vues suivantes :

`SCHEMATA ( CATALOG_NAME , SCHEMA_NAME , SCHEMA_OWNER , ... )` : schémas,

`TABLES ( TABLE_CATALOG , TABLE_SCHEMA , TABLE_NAME , TABLE_TYPE , ... )` : relations (i. e. tables) et vues,

`VIEWS` : vues,

`VIEW_TABLE_USAGE` : relations utilisées par des vues,

`VIEW_COLUMN_USAGE` : attributs utilisés par des vues,

`COLUMNS (TABLE_CATALOG , TABLE_SCHEMA , TABLE_NAME , COLUMN_NAME , ORDINAL_POSITION , COLUMN_DEFAULT , IS_NULLABLE , DATA_TYPE , CHARACTER_MAXIMUM_LENGTH , CHARACTER_OCTET_LENGTH , NUMERIC_PRECISION , NUMERIC_PRECISION_RADIX , NUMERIC_SCALE , DATETIME_PRECISION , CHARACTER_SET_CATALOG , CHARACTER_SET_SCHEMA , CHARACTER_SET_NAME , COLLATION_CATALOG , COLLATION_SCHEMA , COLLATION_NAME , DOMAIN_CATALOG , DOMAIN_SCHEMA , DOMAIN_NAME)` : attributs (i. e. colonnes),

`TABLE_CONSTRAINTS ( TABLE_CATALOG , TABLE_SCHEMA , TABLE_NAME , CONSTRAINT_CATALOG , CONSTRAINT_SCHEMA , CONSTRAINT_NAME , CONSTRAINT_TYPE , ... )` : contraintes de relation,

`KEY_COLUMN_USAGE ( TABLE_CATALOG , TABLE_SCHEMA , TABLE_NAME , CONSTRAINT_CATALOG , CONSTRAINT_SCHEMA , CONSTRAINT_NAME , COLUMN_NAME , ... )` : attributs utilisés pour les contraintes de clé primaire, de clé étrangère et d'unicité,

`REFERENTIAL_CONSTRAINTS` : contraintes d'intégrité référentielle,

`ASSERTIONS ( CONSTRAINT_CATALOG , CONSTRAINT_SCHEMA , CONSTRAINT_NAME , IS_DEFERRABLE , INITIALLY_DEFERRED , ... )` : assertions,

`CONSTRAINT_TABLE_USAGE` : relations utilisées par une contrainte d'intégrité référentielle, d'unicité ou assertion,

`CONSTRAINT_COLUMN_USAGE` : attributs utilisés par une contrainte d'intégrité référentielle, d'unicité ou assertion,

`CHECK_CONSTRAINTS` : contraintes et assertions,

`CHECK_TABLE_USAGE` : relations utilisées par une contrainte de validation,

`CHECK_COLUMN_USAGE` : attributs utilisés par une contrainte de validation,

`DOMAINS` : domaines,

`DOMAINS_CONSTRAINTS` : contraintes de domaine,

`CHARACTER_SETS` : jeux de caractères,

`COLLATIONS` : collations,

`TRANSLATIONS` : translations,

`TRIGGERS` : déclencheurs,

`TRIGGER_TABLE_USAGE` : relations utilisées par des déclencheurs,

`TRIGGER_COLUMN_USAGE` : attributs utilisés par des déclencheurs,

Dictionnaire des données (2/2)

`ROUTINES ( TABLE_CATALOG , TABLE_SCHEMA , ROUTINE_NAME , ROUTINE_TYPE , SPECIFIC_CATALOG , SPECIFIC_SCHEMA , SPECIFIC_NAME , ... )` : fonctions et procédures stockées
`ROUTINE_TABLE_USAGE` : relations utilisées par des procédures stockées,
`ROUTINE_COLUMN_USAGE` : attributs utilisés par des procédures stockées,

`USERS` : utilisateurs et rôles,
`ROLES` : rôles,

`TABLE_PRIVILEGES ( TABLE_CATALOG , TABLE_SCHEMA , TABLE_NAME , COLUMN_NAME , PRIVILEGE_TYPE , GRANTOR , GRANTEE , IS_GRANTABLE , ... )` : privilèges sur les relations,
`COLUMNS_PRIVILEGES` : privilèges sur les attributs,
`USAGE_PRIVILEGES` : privilèges des autres objets de la base

Exemples

```
-- les relations, sans les vues
SELECT *
FROM TABLES
WHERE TABLE_TYPE = 'BASE TABLE' -- <> 'VIEW'
-- les relations et leurs attributs
SELECT *
FROM TABLES
JOIN COLUMNS ON
    ( TABLES.TABLE_CATALOG , TABLES.TABLE_SCHEMA , TABLES.TABLE_NAME ) =
    ( COLUMNS.TABLE_CATALOG , COLUMNS.TABLE_SCHEMA , COLUMNS.TABLE_NAME )
-- les contraintes de clé étrangère
SELECT TABLE_NAME , COLUMN_NAME , CONSTRAINT_NAME
FROM TABLE_CONSTRAINTS
WHERE CONSTRAINT_TYPE = 'FOREIGN KEY' -- <> 'PRIMARY KEY'
                                      -- AND <> 'UNIQUE'
-- les privilèges avec pour chacun
-- l'utilisateur qui l'a attribué et celui qui l'a reçu
SELECT GRANTOR , GRANTEE , TABLE_NAME , PRIVILEGE_TYPE
FROM TABLE_PRIVILEGES
-- les procédures stockées, sans les fonctions
SELECT *
FROM ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE' -- <> 'FUNCTION'
```

Principes

Un utilisateur (ou un rôle) ne peut effectuer une opération que s'il a le privilège approprié pour cette opération

Le créateur d'un schéma a tous les droits sur ce schéma et distribue les privilèges (i. e. les attribue `GRANT` ou les retire `REVOKE`) à des utilisateurs (ou des rôles)

On distingue les privilèges objet (pour manipuler les données) et les privilèges système (pour définir les données)

Utilisateurs et rôles

Un rôle est un ensemble de privilèges

Un utilisateur peut remplir plusieurs rôles et inversement un rôle peut être réalisé par plusieurs utilisateurs

La norme prévoit la création et la suppression de rôles . . . mais pas des utilisateurs !

Privilèges objet (1/2)

Syntaxe

`GRANT` ◁ privilèges objet ▷ `ON` ◁ objet ▷ `TO` ◁ utilisateurs ou rôles ▷ [`WITH GRANT OPTION`]

`REVOKE` ◁ privilèges objet ▷ `ON` ◁ objet ▷ `FROM` ◁ utilisateurs ou rôles ▷ { `RESTRICT` , `CASCADE` }

`GRANT OPTION` permet à un utilisateur de céder ou de retirer ses droits à ou d'un autre utilisateur

`CASCADE` permet d'éviter d'avoir des privilèges abandonnés. Supposons que $u1$ attribue un privilège à $u2$ puis que $u2$ attribue un privilège à $u3$ et qu'ensuite $u1$ retire le privilège à $u2$; $u3$ dispose alors d'un privilège abandonné car il découle d'un privilège qui n'existe plus

◁ privilèges objet ▷

`SELECT` : interrogation d'une relation ou d'une vue

`INSERT` : insertion dans une relation ou dans une vue

`UPDATE` : modification dans une relation ou dans une vue

`DELETE` : suppression dans une relation ou dans une vue

`REFERENCES` : respect d'une contrainte d'intégrité (référentielle ou non) d'une relation vers une autre relation

`USAGE` : utilisation d'un domaine, d'un jeu de caractères, d'une collation, d'une translation

`EXECUTE` : exécution d'une routine SQL (i. e. d'une fonction ou procédure stockée, d'une méthode voire d'un paquetage)

`UNDER` : utilisation de types utilisateurs (UDT)

Quelques autres privilèges non normalisés mais courants :

`DBA` : administrateur

`CONNECT` : nouvel utilisateur

`ALL` [`PRIVILEGES`] : tous les privilèges qui peuvent être donnés et être reçus

Privilèges objet (2/2)

◁ objet ▷

[TABLE] ◁ relation ou vue ▷

DOMAIN ◁ domaine ▷

CHARACTER_SET ◁ jeu de caractères ▷

COLLATION ◁ collation ▷

TRANSLATION ◁ translation ▷

TYPE ◁ type utilisateur ▷

◁ routine SQL ▷

Privilèges système

Syntaxe

GRANT ◁ privilèges système ▷ TO ◁ utilisateurs ou rôles ▷ [WITH GRANT OPTION]

REVOKE ◁ privilèges système ▷ FROM ◁ utilisateurs ou rôles ▷

GRANT OPTION permet à un utilisateur de céder ou de retirer ses droits à ou d'un autre utilisateur

◁ privilèges système ▷ := { CREATE , ALTER , DROP } { SESSION , ROLE , TABLE , VIEW , PROCEDURE , ... }

Exemples

```
-- créer un rôle UnRole en l'attribuant à l'utilisateur courant
CREATE ROLE UnRole WITH ADMIN CURRENT_USER
-- attribuer les privilèges objet d'interrogation et de mise à jour
-- sur les voitures à l'utilisateur gestionnaire des voitures
GRANT SELECT , INSERT , DELETE , UPDATE , REFERENCES
ON Voitures
TO GestionnaireVoitures
-- attribuer le privilège objet d'interrogation sur les voitures
-- à tous les utilisateurs
GRANT SELECT
ON Voitures
TO PUBLIC
-- retirer tous privilèges objet sur les voitures à l'utilisateur Maladroit
REVOKE ALL
ON Voitures
FROM Maladroit
-- supprimer le rôle UnRole
DROP ROLE UnRole
-- attribuer le privilège système de gérer une relation à l'utilisateur
-- UnPseudoDBA qui pourra redistribuer ces privilèges
GRANT CREATE SESSION , CREATE TABLE , DROP TABLE , ALTER TABLE
TO UnPseudoDBA
WITH ADMIN OPTION
-- attribuer le privilège d'interroger le dictionnaire des données
-- à tous les utilisateurs
GRANT SELECT ANY TABLE TO PUBLIC
-- retirer tous les privilèges à l'utilisateur Maladroit
REVOKE ALL PRIVILEGES FROM Maladroit
```

Constat

Le SQL interactif est extrêmement puissant du point de vue d'un programmeur mais un utilisateur a quant à lui du mal à exprimer une requête d'interrogation (il bute souvent sur les jointures et les requêtes imbriquées)

Le formatage de sortie du résultat d'une requête est insuffisant

La norme s'est considérablement enrichie de sorte qu'il est maintenant probablement possible d'exprimer tous les traitements uniquement avec des ordres SQL

Solution

Le SQL procédural ou programmé va prendre en charge les points suivants :

- L'algorithmique de l'application (structure de contrôle et structure de données ainsi que la gestion des erreurs et des exceptions)
- La gestion de l'interface utilisateur
- La gestion de la mémoire
- La gestion des communications réseau

Pour cela, les ordres SQL interactifs sont associés soit à d'autres instructions SQL, langage de 4^e génération (L4G), soit à des instructions écrites dans un langage de 3^e génération (L3G) dont la liste est fixée (ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, PL/1, Java)

N. B. : les exemples qui suivent en SQL programmé utilisent les langages SQL et C (et même SQLJ)

Norme

La norme traite le SQL procédural et programmé au niveau du LDD (déclencheur, méthode associée à un type utilisateur), des PSM (fonctions et procédures stockées), du SQL intégré *(embedded SQL)* et de CLI (exemple : les API pour se connecter au serveur ont donné lieu aux interfaces ODBC, OLEDB, DBE, dbExpress, JDBC, etc.)

Quelques mots-clés de *Persistant Stored Modules* (PSM)

Déclaration de variable : `DECLARE`

Affectation : `SET`

Bloc : `BEGIN` ... `END`

Remarque : un bloc peut contenir plusieurs transactions (`BEGIN NOT ATOMIC`) ou n'en faire qu'une seule auquel cas tout le code est exécuté ou rien ne l'est (`BEGIN ATOMIC`)

Structures de contrôle :

`IF`, `THEN`, `ELSIF`, `ELSE`, `END IF`

`LOOP`, `END LOOP`

`WHILE`, `DO`, `END WHILE`

`REPEAT`, `UNTIL`, `END REPEAT`

`ITERATIVE`, `LEAVE`

Appel d'une procédure stockée : `CALL`

Curseur

Un curseur permet de parcourir successivement les tuples du résultat d'une requête d'interrogation, les tuples pouvant être mis à jour

L'algorithme de base consiste à déclarer le curseur (`DECLARE`), ouvrir le curseur `OPEN`, lire le premier tuple `FETCH`, boucler en lisant un à un les tuples du curseur jusqu'à atteindre le dernier (en testant `SQLSTATE`), fermer le curseur `CLOSE`

`DECLARE` ◁ curseur ▷ [{ `SENSITIVE` , `INSENSITIVE` , `ASENSITIVE` }] [[`NOT`] `SCROLL`] `CURSOR` [`WITH`[`OUT`] `HOLD`] [`WITH`[`OUT`] `RETURN`] `FOR` ◁ requête d'interrogation ▷ `FOR` { `READ ONLY` , `UPDATE OF` ◁ attributs ▷ }

Les données peuvent être mises à jour par d'autres transactions (`SENSITIVE`) ou rester stables (`INSENSITIVE` ce qui impose `FOR READ ONLY`), ou laisser le SGBD choisir (`ASENSITIVE`)

Il est possible de naviguer dans les données (`SCROLL`) ou n'autoriser que la possibilité d'aller systématiquement au suivant (`NOT SCROLL`)

Le curseur peut ne pas être fermé une fois la transaction terminée (`WITH HOLD`) ou être fermé (`WITHOUT HOLD`)

Un jeu de résultats peut être retourné à l'appelant (`WITH RETURN`) ou ne rien retourner (`WITHOUT RETURN`)

Il est possible de mettre à jour les données (`FOR UPDATE` ce qui impose `SENSITIVE`) ou n'en autoriser que la lecture (`FOR READ ONLY`)

`OPEN` ◁ curseur ▷

`FETCH` ◁ curseur ▷ `INTO` ◁ variables ▷

`FETCH` { `NEXT` , `PRIOR` , `FIRST` , `LAST` , `RELATIVE` ◁ déplacement ▷ , `ABSOLUTE` ◁ déplacement ▷ } ◁ curseur « scrollable » ▷ `INTO` ◁ variables ▷ pour se déplacer dans le résultat de la requête (d'un tuple à un autre)

Modification ou suppression du tuple courant

`UPDATE SET` ◁ attributs ▷ = ◁ expressions ▷ `WHERE CURRENT OF` ◁ curseur ▷

`DELETE WHERE CURRENT OF` ◁ curseur ▷

`CLOSE` ◁ curseur ▷

Gestion des erreurs

Après l'exécution de chaque ordre SQL, le SGBD retourne dans la zone de communication *(SQL Communication Area)* `SQLCA` une information complète sur son déroulement (pas d'erreur, condition particulière, etc.)

Il est impératif d'en gérer les exceptions dans une application professionnelle

`SQLSTATE` : chaîne de 5 caractères contenant le code d'erreur

`'00000'` : ordre exécuté avec succès (i. e. aucune erreur détectée)

`'00`xxx`'` = `SUCCESS`

`'01`xxx`'` = `SUCCESS WITH INFO` : condition particulière obtenue (avertissement) ; cf. exception `SQLWARNING`

`'02`xxx`'` = `NO DATA` : pas de tuple traité ; cf. exception `NOT FOUND`
(`sqlca.sqlcode` vaut `1403` en Oracle et `100` pour beaucoup d'autres SGBD)

$\geq$ `'03000'` : hors norme ; cf. exception `SQLEXCEPTION`

N. B. : x est un caractère quelconque

Une zone de diagnostic est également accessible par l'ordre `GET DIAGNOSTICS`

L'entête qui donne des informations générales :

Nombre de lignes (`ROW_COUNT`)

Commande SQL (`COMMAND_FUNCTION`, `COMMAND_FUNCTION_CODE`)

Transaction (`TRANSACTION_ACTIVE`)

Exemple :

```
GET DIAGNOSTICS nblig = ROW\_COUNT , transac = TRANSACTION\_ACTIVE
```

Les détails :

Code SQL (`RETURNED_SQLSTATE`)

Message d'erreur (`MESSAGE_TEXT`)

Informations sur l'objet : ◁ o1 ▷`_NAME` où ◁ o1 ▷ := { `CONNECTION` , `SERVER` , `CATALOG` , `SCHEMA` , `TABLE` , `COLUMN` , `CURSOR` , `PARAMETER` } ; ◁ o2 ▷`_`◁ o3 ▷ où ◁ o2 ▷ := { `CONSTRAINST` , `TRIGGER` , `ROUTINE` } et ◁ o3 ▷ := { `CATALOG` , `SCHEMA` , `NAME` } ; etc.

Exemple :

```
GET DIAGNOSTICS EXCEPTION 1 etat = RETURNED\_SQLSTATE ,
                            svr = SERVER\_NAME ,
                            ci =  CONSTRAINST\_NAME
```

Gestion des exceptions

`WHENEVER` { `SQLWARNING` , `NOT FOUND` , `SQLEXCEPTION` , `SQLSTATE` (◁ classe et erreur ▷) } { `CONTINUE` , `GOTO` ◁ étiquette dans code ▷ }

Lorsque survient une exception, il est possible de la piéger, de la traiter et soit de continuer le programme soit de le faire poursuivre à un autre emplacement du code

Il est possible : de définir une constante pour une exception (`DECLARE` ◁ condition ▷ `CONDITION` [`FOR` ◁ valeur SQLSTATE ▷]), de définir une forme d'exception et son comportement (`DECLARE` ◁ *handler* ▷ `HANDLER` ...), de vider la zone de diagnostic (`SIGNAL`), de modifier la zone de diagnostic et de la faire suivre à un gestionnaire d'exceptions (`RESIGNAL`)

Déclencheur *(trigger)*

Procédure « compilée » et cataloguée (i. e. stockée) dans le dictionnaire, exécutée automatiquement à chaque fois que l'évènement déclenchant associé (à la relation) se produit

`CREATE TRIGGER` ◁ déclencheur ▷ { `BEFORE` , `AFTER` } { `INSERT` , `UPDATE` [`OF (` ◁ attributs ▷ `)`] , `DELETE` } `ON` ◁ relation ▷ [`REFERENCING` [`OLD` [`ROW`] ◁ anciennes valeurs ligne ▷] [`NEW` [`ROW`] ◁ nouvelles valeurs ligne ▷] [`OLD TABLE` ◁ anciennes valeurs relation ▷] [`NEW TABLE` ◁ nouvelles valeurs relation ▷]] `FOR EACH` { `ROW` , `STATEMENT` } [`WHEN` ◁ condition ▷] `BEGIN` [[`NOT`] `ATOMIC`] ◁ code ▷ `END`

Le déclenchement peut s'effectuer avant (`BEFORE`) ou après (`AFTER`) n'importe quelle opération de mise à jour (`INSERT`, `UPDATE` ou `DELETE`) de la relation

Il est possible (`REFERENCING`) d'utiliser les anciennes (`OLD` pour `UPDATE` et `DELETE`) et nouvelles (`NEW` pour `INSERT` et `UPDATE`) valeurs de la ligne ([`ROW`] ce qui impose `FOR EACH ROW`) ou de la relation (`TABLE` ce qui impose `FOR EACH STATEMENT`)

Le déclencheur peut s'appliquer à tous les tuples concernés i. e. globalement en une seule fois (`FOR EACH STATEMENT`) ou à chacun des tuples individuellement (`FOR EACH ROW`)

Une garde peut être spécifiée (`WHEN`) afin de ne pas déclencher le traitement si la condition n'est pas vérifiée

```
-- transforme, si nécessaire, le nom de l'étudiant en majuscule
-- après la modification du (seul) nom de l'étudiant
CREATE TRIGGER NomEtuMAJ_ApresModif
AFTER UPDATE OF ( NomEtu ) -- se déclenche après la modification
                           -- du nom de l'étudiant
ON Etudiants
REFERENCING NEW ROW nvlEtudiant -- tuple obtenu après la modification
FOR EACH STATEMENT -- pour tous les tuples concernés
                   -- i. e. globalement en une fois
WHEN nvlEtudiant.NomEtu <> UPPER(nvlEtudiant.NomEtu) -- si pas déjà en
                                                     -- majuscule
BEGIN ATOMIC -- en une seule transaction
    SET nvlEtudiant.NomEtu = UPPER(nvlEtudiant.NomEtu) ;
END
```

```
-- les étudiants ne doivent pas avoir plus de 3 voitures
CREATE TRIGGER AuPlus3VoitParEtd_ApresInser
AFTER INSERT ON Voitures -- se déclenche après l'insertion d'une voiture
REFERENCING NEW ROW nvllVoiture -- tuple obtenu après l'insertion
FOR EACH ROW -- pour chacune des insertions prise individuellement
BEGIN NOT ATOMIC -- pour permettre de traiter plusieurs transactions
    IF ( SELECT COUNT(*) FROM Voitures GROUP BY nvllVoiture.NoINE ) > 3
    THEN
        RESIGNAL SQLSTATE '90123' SET MESSAGE_TEXT =
            'Les étudiants ne doivent pas avoir plus de trois voitures' ;
    END IF
END
```

Procédure stockée *stored procedure* (1/3)

Procédure « compilée » et cataloguée (i. e. stockée) dans le dictionnaire, exécutée à la demande

`CREATE PROCEDURE` ◁ procédure ▷ `(` [[`IN`][`OUT`] ◁ paramètres ▷ ◁ types ▷] `)` `LANGUAGE` { `SQL` , `ADA` , `C` , `COBOL` , `FORTRAN` , `MUMPS` , `PASCAL` , `PL1` , `JAVA` } `PARAMETER STYLE` { `SQL` , `GENERAL` , `JAVA` } `SPECIFIC` ◁ procédure ▷ [`NOT`] `DETERMINISTIC` { `NO SQL` , `CONTAINS SQL` , `READS SQL DATA` , `MODIFIES SQL DATA` } `DYNAMIC RESULT SETS` ◁ entier naturel ▷ `BEGIN` [[`NOT`] `ATOMIC`] { ◁ code ▷ , ◁ référence code externe ▷ } `END`

- Chaque paramètre peut être en entrée (`IN`), en sortie (`OUT`) ou en entrée/sortie (`INOUT`)
- La procédure peut être écrite (`LANGUAGE`) en SQL (`SQL`) ou dans un autre langage (ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, PL1 ou JAVA) auquel cas le code sera une référence externe à l'objet qui contient le programme
- Il faut préciser de quelle façon passer les paramètres au code (`PARAMETER STYLE`) si la procédure est externe (`GENERAL` ou `JAVA`) ou en SQL (`SQL`)
- Les ordres `ALTER` et `DROP` reprennent le nom donné après `SPECIFIC`
- Il faut indiquer si la procédure retourne systématiquement les mêmes valeurs en [entrée/]sortie pour les mêmes valeurs des paramètres en entrée[/sortie] (`DETERMINISTIC`) ou non (`NOT DETERMINISTIC`) telle une instruction SQL du LMD ou l'appel à un séquenceur ou l'utilisation de la date ou heure courante
- Il faut mentionner si le code : ne contient pas de SQL (`NO SQL` telle une procédure externe), contient du SQL sans lecture ni écriture de données dans la BD (`CONTAINS SQL`), lit des données de la BD (`READS SQL DATA`), ou modifie des données de la BD (`MODIFIES SQL DATA`)
- La procédure peut retourner zéro, un ou plusieurs jeux de résultats (`DYNAMIC RESULT SETS`) pour un curseur resté ouvert à savoir les tuples non encore lus (curseur non navigable) ou tous (curseur navigable)

```
-- concaténation de tous les numéros d'immatriculation en lettres des
-- voitures
CREATE PROCEDURE ProcedureCursConcatNoImmatLettres ( OUT c VARCHAR )
LANGUAGE SQL
PARAMETER STYLE SQL
SPECIFIC ProcedureCursConcatNoImmatLettres
NOT DETERMINISTIC
READS SQL DATA
DYNAMIC RESULT SETS 0
BEGIN ATOMIC
    DECLARE i INTEGER DEFAULT NULL ;
    SET c = '' ;
    FOR i AS CursConcatTousNoImmatLettres
        INSENSITIVE -- données du curseur stables
        CURSOR FOR SELECT NoImmatLettres
                   FROM Voitures
                   ORDER BY NoImmatLettres
        FOR READ ONLY -- en lecture seule
    DO
        SET c = c || Voitures.NoImmatLettres ;
    END FOR
END
```

Procédure stockée (2/3)

```
-- utilisation d'un curseur navigant avec des modifications et des
-- suppressions
CREATE PROCEDURE ProcedureCursNavigModifSuppr ( )
LANGUAGE SQL -- et non ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, PL1, JAVA
PARAMETER STYLE SQL -- et non GENERAL ou JAVA
SPECIFIC ProcedureCursNavigModifSuppr -- pour ALTER et DROP
NOT DETERMINISTIC -- non déterministe car il y a des mises à jour notamment
MODIFIES SQL DATA -- car il y a des modifications et des suppressions
DYNAMIC RESULT SETS 0 -- ne retourne aucun jeu de résultat
BEGIN NOT ATOMIC -- tuples sont gérés individuellement (et non globalement)
    DECLARE SQLSTATE CHAR(5) DEFAULT NULL ; -- retour d'ordre SQL
    DECLARE SQLSTATE_SUCCESS CHAR(2) DEFAULT '00' ; -- retour sans problème
    DECLARE CursNoINE INT DEFAULT NULL ; -- NoINE récupéré du curseur
    DECLARE CursNomEtu VARCHAR(25) DEFAULT NULL ; -- NomEtu du curseur
    DECLARE NoINEMoy INT DEFAULT NULL ; -- moyenne n° étudiant min. et max.
    DECLARE CursEtudiants
        SENSITIVE -- les données du curseur peuvent être mises à jour par
                  -- d'autres transactions
        SCROLL -- pour naviguer dans le curseur, pas que vers le suivant
        CURSOR
        WITHOUT HOLD -- fermeture du curseur une fois transaction terminée
        WITHOUT RETURN -- sans retourner de résultats à l'appelant
        FOR SELECT NoINE , NomEtu FROM Etudiants
            FOR UPDATE OF NomEtu ; -- données pouvant être mises à jour
    OPEN CursEtudiants ; -- ouverture du curseur
    FETCH FIRST FROM CursEtudiants
        INTO CursNoINE , CursNomEtu ; -- lecture du premier tuple
    WHILE SUBSTRING(SQLSTATE FROM 1 FOR 2) = SQLSTATE_SUCCESS
    DO -- TantQue pas de problème Faire
        SET NoINEMoy = ( SELECT FLOOR((MIN(NoINE)+MAX(NoINE))/2)
                         FROM Etudiants ) ;
        IF CursNoINE <= NoINEMoy THEN
            BEGIN
            UPDATE SET NomEtu = UPPER(NomEtu)    -- modification, en
                WHERE CURRENT OF CursEtudiants ; -- restant sur même tuple
            FETCH RELATIVE +2 FROM CursEtudiants -- on avance de 2 tuples
                INTO CursNoINE , CursNomEtu ;    -- i. e. on en saute 1
            END
        ELSE
            BEGIN
            DELETE WHERE CURRENT OF CursEtudiants ;
                -- suppression et passage au tuple précédent
            FETCH LAST FROM CursEtudiants INTO CursNoINE , CursNomEtu ;
                -- on passe directement au dernier tuple
            END
        END IF
    END WHILE
    CLOSE CursEtudiants ; -- fermeture du curseur
END
```

Procédure stockée (3/3)

```
-- insertion des licences L1, L2 et L3 parmi les diplômes
CREATE PROCEDURE ProcedureSQLDynamiqInserLicences ( )
LANGUAGE SQL
PARAMETER STYLE SQL
SPECIFIC ProcedureSQLDynamiqInserLicences
NOT DETERMINISTIC
MODIFIES SQL DATA
DYNAMIC RESULT SETS 0
BEGIN ATOMIC
    DECLARE InsereDiplomeLicence VARCHAR DEFAULT NULL ; -- SQL dynamique
    DECLARE i SMALLINT DEFAULT NULL ;
    SET InsereDiplomeLicence = 'INSERT INTO Diplomes VALUES ( ? , ? )' ;
    PREPARE InsereDiplomeLicence ;
    SET i = 1 ;
    WHILE i <= 3
    DO
        EXECUTE InsereDiplomeLicence
            USING 'L' || TRIM(CAST(i AS VARCHAR)) ,
                  'Licence année ' || CAST(i AS VARCHAR) ;
        SET i = i + 1 ;
    END WHILE
END
```

Fonction

Fonction (SQL ou dans un langage externe) retournant une valeur à partir d'éventuels paramètres en entrée, exécutée à la demande ou dans une requête

`CREATE FUNCTION` ◁ fonction ▷ `(` [[`IN`] ◁ paramètres ▷ ◁ types ▷] `) RETURNS` ◁ type ▷ `LANGUAGE` { `SQL` , `ADA` , `C` , `COBOL` , `FORTRAN` , `MUMPS` , `PASCAL` , `PL1` , `JAVA` } `PARAMETER STYLE` { `SQL` , `GENERAL` , `JAVA` } `SPECIFIC` ◁ fonction ▷ [`NOT`] `DETERMINISTIC` { `NO SQL` , `CONTAINS SQL` , `READS SQL DATA` , `MODIFIES SQL DATA` } { `RETURN NULL` , `CALLED` } `ON NULL INPUT BEGIN` [[`NOT`] `ATOMIC`] { ◁ code ▷ , ◁ référence code externe ▷ } `END`

Les paramètres sont en entrée ([`IN`])

Une seule valeur est retournée (`RETURNS`)

La fonction peut être écrite (`LANGUAGE`) en SQL (`SQL`) ou dans un autre langage (ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, PL1 ou JAVA) auquel cas le code sera une référence externe à l'objet qui contient le programme

Il faut préciser de quelle façon passer les paramètres au code (`PARAMETER STYLE`) si la fonction est externe (`GENERAL` ou `JAVA`) ou en SQL (`SQL`)

Les ordres `ALTER` et `DROP` reprennent le nom donné après `SPECIFIC`

Il faut indiquer si la fonction retourne systématiquement la même valeur pour les mêmes valeurs des paramètres (`DETERMINISTIC`) ou non (`NOT DETERMINISTIC`) telle une instruction SQL du LMD ou l'appel à un séquenceur ou l'utilisation de la date ou heure courante

Il faut mentionner si le code : ne contient pas de SQL (`NO SQL` telle une fonction externe), contient du SQL sans lecture ni écriture de données dans la BD (`CONTAINS SQL`), lit des données de la BD (`READS SQL DATA`), ou modifie des données de la BD (`MODIFIES SQL DATA`)

Lorsque des paramètres n'ont pas de valeur (`ON NULL INPUT`), la fonction peut directement retourner `NULL` (`RETURN NULL`) ou traiter ce cas particulier dans le code (`CALLED`)

```
-- concaténation des composants du numéro d'immatriculation
CREATE FUNCTION FonctionNoImmat (
    NoImmatChiffres INT ,
    NoImmatLettres CHAR(3) ,
    NoImmatDepart INT )
RETURNS VARCHAR
LANGUAGE SQL
PARAMETER STYLE SQL
SPECIFIC FonctionNoImmat
DETERMINISTIC
CONTAINS SQL
RETURN NULL ON NULL INPUT
BEGIN ATOMIC
    RETURN CAST(NoImmatChiffres AS VARCHAR) || ' ' || UPPER(NoImmatLettres)
           || ' ' || CAST(NoImmatDepart AS VARCHAR) ;
END ;
```

Méthode

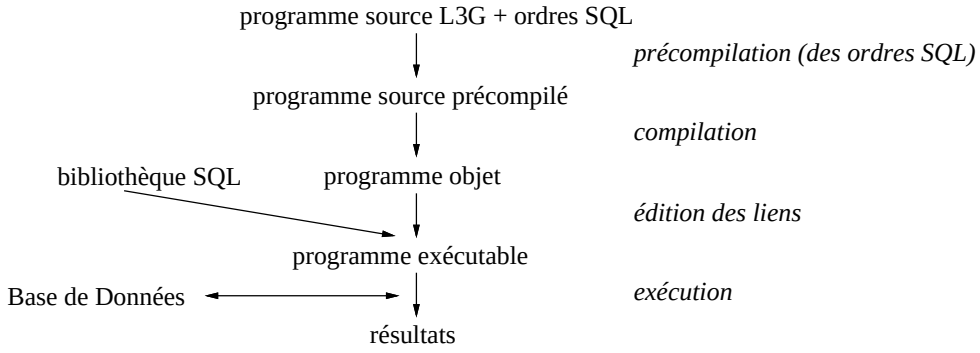
Programme codé en SQL ou dans un langage externe retournant une valeur à partir d'éventuels paramètres en entrée, attachée à un type utilisateur (UDT), utilisable uniquement comme un attribut d'une relation

`CREATE` { `INSTANCE` , `STATIC` , `CONSTRUCTOR` } `METHOD` ◁ méthode ▷ `(` [[`IN`] ◁ paramètres ▷ ◁ types ▷] `)` `RETURNS` ◁ type ▷ `FOR` ◁ type utilisateur ▷ `LANGUAGE` { `SQL` , `ADA` , `C` , `COBOL` , `FORTRAN` , `MUMPS` , `PASCAL` , `PL1` , `JAVA` } `PARAMETER STYLE` { `SQL` , `GENERAL` , `JAVA` } `SPECIFIC` ◁ méthode ▷ [`NOT`] `DETERMINISTIC` { `NO SQL` , `CONTAINS SQL` , `READS SQL DATA` , `MODIFIES SQL DATA` } { `RETURN NULL` , `CALLED` } `ON NULL INPUT` `BEGIN` [[`NOT`] `ATOMIC`] { ◁ code ▷ , ◁ référence code externe ▷ } `END`

- La méthode peut s'appliquer à un objet particulier (`INSTANCE`) ou au niveau du type de l'objet (`STATIC`) ou être destinée à instancier des objets (`CONSTRUCTOR`)
- Les paramètres sont en entrée ([`IN`])
- Une seule valeur est retournée (`RETURNS`)
- Il faut spécifier le type utilisateur (`FOR` ◁ type utilisateur ▷)
- La méthode peut être écrite (`LANGUAGE`) en SQL (`SQL`) ou dans un autre langage (ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, PL1 ou JAVA) auquel cas le code sera une référence externe à l'objet qui contient le programme
- Il faut préciser de quelle façon passer les paramètres au code (`PARAMETER STYLE`) si la méthode est externe (`GENERAL` ou `JAVA`) ou en SQL (`SQL`)
- Les ordres `ALTER` et `DROP` reprennent le nom donné après `SPECIFIC`
- Il faut indiquer si la méthode retourne systématiquement la même valeur pour les mêmes valeurs des paramètres (`DETERMINISTIC`) ou non (`NOT DETERMINISTIC`) telle une instruction SQL du LMD ou l'appel à un séquenceur ou l'utilisation de la date ou heure courante
- Il faut mentionner si le code : ne contient pas de SQL (`NO SQL` telle une méthode externe), contient du SQL sans lecture ni écriture de données dans la BD (`CONTAINS SQL`), lit des données de la BD (`READS SQL DATA`), ou modifie des données de la BD (`MODIFIES SQL DATA`)
- Lorsque des paramètres n'ont pas de valeur (`ON NULL INPUT`), la méthode peut directement retourner `NULL` (`RETURN NULL`) ou traiter ce cas particulier dans le code (`CALLED`)

```
-- numéro d'immatriculation composé des chiffres, lettres et département
CREATE TYPE TypeNoImmat AS (
    NoImmatChiffres INT ,
    NoImmatLettres CHAR(3) ,
    NoImmatDepart INT )
NOT INSTANTIABLE
NOT FINAL
METHOD MethodeNoImmat () RETURNS VARCHAR ;
-- .../...
-- concaténation des composants du numéro d'immatriculation
CREATE INSTANCE METHOD MethodeNoImmat RETURNS VARCHAR FOR TypeNoImmat
RETURN CAST(SELF.NoImmatChiffres AS VARCHAR) || ' ' ||
       UPPER(SELF.NoImmatLettres) || ' ' ||
       CAST(SELF.NoImmatDepart AS VARCHAR) ;
```

Processus de développement



Intégration des ordres SQL dans le programme source

Les ordres SQL déclaratifs et exécutables `EXEC SQL` ◁ ordre SQL ▷ ; sont insérés dans le programme source partout où cela est autorisé par le L3G hôte

Communication entre programme source et ordres SQL

Les ordres SQL et les instructions du programme source communiquent par l'intermédiaire de variables partagées situées dans la `DECLARE SECTION`

Remarque : un paramètre indicateur permet de fixer ou de savoir si un attribut n'a pas de valeur (`-1` pour `NULL`) ou a une valeur déterminée (`0`)

Appel d'une procédure stockée : `EXECUTE`

```
/* bibliothèques */
#include <stdio.h>
#include <string.h>
/* constantes */
#define EXECUTION_OK "00000" /* SQLSTATE pour OK */
#define FIN_DE_CURSEUR "02000" /* SQLSTATE pour NO DATA FOUND */
#define VALEUR_DETERMINEE 0 /* NOT NULL i. e. renseignée ; -1 sinon */
/* zone des déclarations communes au C et à SQL */
EXEC SQL BEGIN DECLARE SECTION;
    char compte[31];
    int numero_etu;
    VARCHAR nom_etu[26]; /* 25 + 1 pour '\0' */
    int depart_naiss_etu;
    int depart_naiss_etu_indic; /* pour tester l'indétermination */
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
/* variable incluse pour la gestion des erreurs */
EXEC SQL INCLUDE SQLCA;
/* programme principal */
void main () {
/* connexion */
EXEC SQL CONNECT TO LeServeur AS :compte;
/* affichage du nom de l'étudiant de numéro 3 */
numero_etu = 3;
EXEC SQL SELECT NomEtu INTO :nom_etu FROM Etudiants WHERE NoINE = :numero_etu;
printf("%s",nom_etu);
/* liste des étudiants */
EXEC SQL DECLARE CursEtu CURSOR FOR
    SELECT NoINE , NomEtu , DepartNaissEtu FROM Etudiants;
EXEC SQL OPEN CursEtu;
EXEC SQL FETCH CursEtu /* on se place sur le premier étudiant */
    INTO :numero_etu , :nom_etu , :depart_naiss_etu:depart_naiss_etu_indic;
while ( strcmp(SQLSTATE,FIN_DE_CURSEUR) ) {
    printf("%d\t%s",numero_etu,nom_etu); /* on traite l'étudiant */
    if ( depart_naiss_etu_indic == VALEUR_DETERMINEE )
        printf("\t%d",depart_naiss_etu);
    printf("\n");
    EXEC SQL FETCH CursEtu /* on passe à l'étudiant suivant */
        INTO :numero_etu , :nom_etu , :depart_naiss_etu:depart_naiss_etu_indic;
    }
EXEC SQL CLOSE CursEtu;
/* modification du nom de l'étudiant numéro 5 en "DURAND" */
EXEC SQL UPDATE Etudiants SET NomEtu = 'DURAND' WHERE NoINE = 5;
if ( strcmp(SQLSTATE,EXECUTION_OK) ) {
    EXEC SQL ROLLBACK; } /* invalidation définitive de la modification */
else {
    EXEC SQL COMMIT; } /* validation définitive de la modification */
/* déconnexion */
EXEC SQL DISCONNECT CURRENT;
}
```

Remarque : SQLJ (ISO/IEC 9075-10) permet d'intégrer des ordres SQL dans du code Java

```
// déclarations de variables de connexion (ici au DSN par défaut d'Oracle)
String url = "jdbc:oracle:thin:@localhost:1521:orcl";
String compte;
String mot_passe;
// connexion
connex.connect(url,compte,mot_passe);
// déclarations
int numero_etu; // numéro de l'étudiant
String nom_etu; // nom de l'étudiant
int depart_naiss_etu; // département de naissance de l'étudiant
// affichage du nom de l'étudiant de numéro 3
numero_etu = 3;
#sql { SELECT NomEtu INTO :nom_etu FROM Etudiants WHERE NoINE = :numero_etu };
System.out.println("N° :" + numero_etu);
// liste des étudiants : 1re version
#sql public iterator IteratCursEtuV1 ( int , String , int );
IteratCursEtuV1 CursEtuV1;
#sql CursEtuV1 = { SELECT NoINE , NomEtu , DepartNaissEtu FROM Etudiants };
while ( not CursEtuV1.endFetch() ) {
    // récupère les informations d'un étudiant
    #sql { FETCH :CursEtuV1 INTO :numero_etu , :nom_etu , :depart_naiss_etu };
    // affiche les informations sur l'étudiant
    System.out.println("N° :" + numero_etu +
                       ", Nom : " + nom_etu +
                       ", Département :" + depart_naiss_etu);
    }
CursEtuV1.close();
// liste des étudiants : 2de version
#sql public iterator IteratCursEtuV2
    (int num_etd, String nom_etd, int dpt_naiss_etd);
IteratCursEtuV2 CursEtuV2;
#sql CursEtuV2 = { SELECT NoINE , NomEtu , DepartNaissEtu FROM Etudiants };
while ( CursEtuV2.next() ) {
    // affiche les informations sur l'étudiant
    System.out.println("N° :" + CursEtuV2.num_etd() +
                       ", Nom : " + CursEtuV2.nom_etd() +
                       ", Département :" + CursEtuV2.dpt_naiss_etd());
    }
CursEtuV2.close();
// modifier le nom de l'étudiant numéro 5 en DURAND
#sql { UPDATE Etudiants SET NomEtu = 'DURAND' WHERE NoINE = 5 };
#sql { COMMIT };
// déconnexion
connex.close();
```

Objectif

Combiner un ordre SQL interactif à un programme écrit dans un L3G

La requête est inconnue lors de la compilation et est créée au cours de l'exécution du programme

La norme masque au programmeur toute l'allocation de mémoire (gérée par le SGBD)

La zone de description des états *(item descriptor area)*

Structure contenant toutes les informations requises pour gérer un paramètre (en entrée) ou un attribut (en sortie) : type, longueur, valeur d'indétermination autorisée ou non, nom, nom réel ou généré par le SGBD, catalogue, schéma, jeu de caractères, valeur, indicateur, etc.

Un descripteur est alloué pour l'ensemble des paramètres (en entrée) et un autre pour la liste des attributs du résultat (en sortie)

Remarques

`EXECUTE IMMEDIATE` ◁ ordre SQL autre qu'une requête d'interrogation ▷ permet d'éviter les deux étapes de préparation et d'exécution

Il existe des curseurs dynamiques (qu'il faut allouer puis libérer)

```
/* déclarations */
EXEC SQL BEGIN DECLARE SECTION;
    char desc_entr[128] , desc_sort[128];
    VARCHAR commande[512];
    int nb_params,nb_attrib , i , typ,lg,...;
EXEC SQL END DECLARE SECTION;

...

/* allocation des deux descripteurs */
EXEC SQL ALLOCATE DESCRIPTOR :desc_entr; /* entrées = paramètres */
EXEC SQL ALLOCATE DESCRIPTOR :desc_sort; /* sorties = attributs */

/* saisie de n'importe quelle commande de l'utilisateur */
gets(commande);

/* préparation */
EXEC SQL PREPARE ordre FROM :commande;

/* description des paramètres (en entrée) */
EXEC SQL DESCRIBE INPUT ordre USING SQL DESCRIPTOR :desc_entr;
EXEC SQL GET DESCRIPTOR :desc_entr :nb_params = COUNT;
for ( i = 0 ; i < nb_params ; i++ ) {
    EXEC SQL GET DESCRIPTOR :desc_entr VALUE :i :typ = TYPE , :lg = LENGTH ,
        ...; }

/* description du résultat (de sortie) */
EXEC SQL DESCRIBE OUTPUT ordre USING SQL DESCRIPTOR :desc_sort;
EXEC SQL GET DESCRIPTOR :desc_sort :nb_attrib = COUNT;

/* exécution de la commande */
if ( nb_attrib > 0 ) { /* requête d'interrogation SELECT */
    /* déclaration du curseur */
    EXEC SQL DECLARE curs INSENSITIVE SCROLL CURSOR FOR ordre;
    /* extraction successive des informations */
    EXEC SQL OPEN curs USING SQL DESCRIPTOR :desc_entr;
    EXEC SQL FETCH NEXT curs USING SQL DESCRIPTOR :desc_sort;
    while ( strcmp(SQLSTATE,"02000") ) {
        ... /* affichage des valeurs */
        EXEC SQL FETCH NEXT curs USING SQL DESCRIPTOR :desc_sort; }
    EXEC SQL CLOSE curs; }
else { /* ordre différent d'une requête d'interrogation */
    EXEC SQL EXECUTE ordre USING SQL DESCRIPTOR :desc_entr; }

/* libération de la préparation */
EXEC SQL DEALLOCATE PREPARE ordre;

/* libération des deux descripteurs */
EXEC SQL DEALLOCATE DESCRIPTOR :desc_entr;
EXEC SQL DEALLOCATE DESCRIPTOR :desc_sort;
```

Le module SQL

Syntaxe

```
-- entête
MODULE ◁ module ▷
NAMES ARE ◁ jeu de caractères ▷
LANGUAGE ◁ langage ▷
SCHEMA ◁ schéma ▷
AUTHORIZATION ◁ créateur ▷
-- déclarations de relations temporaires utilisées dans les procédures
...
-- déclarations des curseurs utilisés dans les procédures
...
-- procédures
PROCEDURE ◁ procédure ▷ ( ◁ paramètres ▷ ) ◁ ordre SQL unique ▷ ;
...
END MODULE
```

Exemple de procédure

```
-- suppression d'un étudiant
PROCEDURE supprime_un_etudiant ( SQLSTATE , :numero_etu INT )
    DELETE FROM Etudiants WHERE NoINE = :numero_etu ;
```

Le module C

Exemple d'appel à une procédure

```
/* appel de la procédure pour supprimer l'étudiant de numéro 3 */
void suppr_etu () {
    char code_retour[6];
    int num_etu = 3;
    supprime_un_etudiant(code_retour,num_etu);
    ...
    }
...
```

Types abstraits : type utilisateur, type relation, type référence (1/3)

```
-- définition du type des immatriculations des voitures
CREATE TYPE T_ImmatVoiture WITHOUT OID VISIBLE AS (
    -- attributs (tous PUBLIC par défaut ; ni PRIVATE, ni PROTECTED)
    Chiffres DomaineNumero ,
    Lettres CHAR(3) NOT NULL
        CONSTRAINT ContrainteNoImmatLettresMAJ
            CHECK ( VALUE = UPPER(VALUE) ) ,
    Depart DomaineDepart NOT NULL
    -- méthodes : constructeur et destructeur d'instance,
    -- acteur (observateur ici et non mutateur)
    CONSTRUCTOR FUNCTION T_ImmatVoiture_Constructeur
        RETURNS T_ImmatVoiture
        BEGIN
            DECLARE i T_ImmatVoiture
            SET i.Chiffres = NULL
            SET i.Lettres = NULL
            SET i.Depart = NULL
            RETURN i
        END FUNCTION
    DESTRUCTOR FUNCTION T_ImmatVoiture_Destructeur (i T_ImmatVoiture)
        RETURNS T_ImmatVoiture
        BEGIN
            DESTROY i
            RETURN i
        END FUNCTION
        -- concatène les composants du numéro d'immatriculation
    ACTOR FUNCTION FonctionNoImmat RETURNS VARCHAR )
CREATE FUNCTION FonctionNoImmat() FOR T_ImmatVoiture
    RETURNS VARCHAR
    SELF AS RESULT
    LANGUAGE SQL
    PARAMETER STYLE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURN NULL ON NULL INPUT
    RETURN CAST(T_ImmatVoiture.Chiffres AS VARCHAR) || ' ' ||
           UPPER(T_ImmatVoiture.Lettres) || ' ' ||
           CAST(T_ImmatVoiture.Depart AS VARCHAR)
-- définition du type des voitures
CREATE TYPE T_Voiture WITH OID VISIBLE AS (
    -- N. B. : REF() pour obtenir l'oid d'un objet
    NoImmat REF(T_ImmatVoiture) REFERENCES ImmatVoitures UNIQUE ,
    Couleur VARCHAR(10) NOT NULL
        CONSTRAINT ContrainteListeCouleurs
            CHECK ( VALUE IN ( 'rouge' , 'jaune' , 'orange' ) )
        CONSTRAINT ContrainteDefautCouleur DEFAULT 'rouge' )
```

Types abstraits : type utilisateur, type relation, type référence (2/3)

```
-- définition du type des étudiants
CREATE TYPE T_Etudiant WITH OID VISIBLE AS (
    NoINE DomaineNumero
        GENERATED BY DEFAULT AS IDENTITY
            ( START WITH 1 INCREMENT BY 1 MAXVALUE 99 MINVALUE 1 CYCLE ) ,
    NomEtu VARCHAR(25) NOT NULL
        ContrainteNomEtuCollation COLLATE 'Latin-1' ,
    DepartNaissEtu DomaineDepart NOT NULL ,
    VoituresPossedees SET ( T_Voiture ) ,
    ACTOR FUNCTION EstGirondin RETURNS BOOLEAN ) -- est-il girondin ?
CREATE FUNCTION EstGirondin() FOR T_Etudiant
    RETURNS BOOLEAN
    SELF AS RESULT
    LANGUAGE SQL
    PARAMETER STYLE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURN NULL ON NULL INPUT
    RETURN T_Etudiant.DepartNaissEtu = 33
-- définition des relations des immatriculations des voitures et des
-- étudiants (contenant leurs voitures)
CREATE TABLE ImmatVoitures ( NoImmat T_ImmatVoiture )
CREATE TABLE Etudiants OF T_Etudiant
    ( CONSTRAINT ContrainteIdEtudiants PRIMARY KEY ( NoINE ) )
-- insertion d'une immatriculation d'une voiture et d'un étudiant
    -- N. B. : notation .. pour accès à un champ composé ou à une fonction
INSERT INTO ImmatVoitures
    ( NoImmat..Chiffres , NoImmat..Lettres , NoImmat..Depart )
    VALUES ( 3333 , 'BX' , 33 )
INSERT
    INTO Etudiants ( NoINE , NomEtu , DepartNaissEtu , VoituresPossedees )
    VALUES ( 5 , 'DURAND' , 33 , T_Voiture (
        SELECT REF(NoImmat) , 'rouge'
        FROM ImmatVoitures
        WHERE NoImmat..Chiffres = 3333 AND NoImmat..Lettres = 'BX'
          AND NoImmat..Depart = 33
        UNION
        SELECT REF(NoImmat) , 'jaune'
        FROM ImmatVoitures
        WHERE NoImmat..Chiffres = 4040 AND NoImmat..Lettres = 'NT'
          AND NoImmat..Depart = 40 ) )
-- interrogation des noms des étudiants girondins
SELECT E.NomEtu FROM Etudiants E WHERE EstGirondin(E)
```

Types abstraits : type utilisateur, type relation, type référence (3/3)

```
-- interrogation des étudiants possédant une voiture rouge immatriculée en
-- Gironde
    -- N. B. : FROM TABLE <collection>
    --         pour utiliser une collection comme une relation
    -- N. B. : DEREF() ou -> pour obtenir l'objet pointé par son oid
SELECT E.*
FROM Etudiants E
WHERE 'rouge' IN (
    SELECT V..Couleur
    FROM TABLE ( E.VoituresPossedees ) V
    WHERE DEREF(V..NoImmat).Depart = 33  ) -- ou V->NoImmat.Depart
```

Types abstraits : héritage de types (sur-types et sous-types)

```
CREATE TYPE T_Personne AS (
    Id DomaineNumero ,
    Nom VARCHAR(25) NOT NULL )
    NOT INSTANTIABLE -- on ne peut pas déclarer un attribut ou une variable
                     -- de ce type
    NOT FINAL -- on peut définir un sous-type de ce type
              -- (i. e. noeud de l'arbre d'héritage)
CREATE TYPE T_Etudiant UNDER T_Personne AS (
    NoINE DomaineNumero ,
    DepartNaissEtu DomaineDepart NOT NULL )
    INSTANTIABLE -- on peut déclarer un attribut ou une variable de ce type
    NOT FINAL
CREATE TYPE T_Etudiant_InfoIUTBx1 UNDER T_Etudiant
    INSTANTIABLE
    FINAL -- on ne peut pas définir un sous-type de ce type
          -- (i. e. feuille de l'arbre d'héritage)
CREATE TYPE T_Enseignant UNDER T_Personne AS (
    Grade INT )
    INSTANTIABLE
    FINAL
```

Types abstraits : héritage de relations

```
CREATE TABLE Personnes (
    Id DomaineNumero NOT NULL
        CONSTRAINT ContrainteIdPersonnes PRIMARY KEY ,
    Nom VARCHAR(25) NOT NULL ContrainteNomEtuCollation COLLATE 'Latin-1' )
CREATE TABLE Etudiants UNDER Personnes (
    NoINE DomaineNumero CONSTRAINT ContrainteCle2ndaireEtudiants UNIQUE
        GENERATED BY DEFAULT AS IDENTITY
            ( START WITH 1 INCREMENT BY 1 MAXVALUE 99 MINVALUE 1 CYCLE ) ,
    DepartNaissEtu DomaineDepart NOT NULL )
```

Types abstraits : type ligne *(row)*

```
CREATE TABLE Voitures (
    NoImmat ROW (
        Chiffres DomaineNumero ,
        Lettres CHAR(3) NOT NULL
            CONSTRAINT ContrainteNoImmatLettresMAJ
                CHECK ( VALUE = UPPER(VALUE) ) ,
        Depart DomaineDepart NOT NULL )
        CONSTRAINT ContrainteIdVoitures PRIMARY KEY ,
    Couleur VARCHAR(10) NOT NULL
        CONSTRAINT ContrainteListeCouleurs
            CHECK ( VALUE IN ( 'rouge' , 'jaune' , 'orange' ) )
        CONSTRAINT ContrainteDefautCouleur DEFAULT 'rouge' ,
    NoINE DomaineNumero
        CONSTRAINT ContrainteRefVoitures2Etudiants
            REFERENCES Etudiants ( NoINE )
                ON DELETE CASCADE ON UPDATE CASCADE )
-- les chiffres et lettres des immatriculations et couleurs des voitures
-- landaises ou immatriculées 3333 BX 33
SELECT NoImmat.Chiffres , NoImmat.Lettres , Couleur
FROM Voitures
WHERE NoImmat.Depart = 40 OR NoImmat = ROW ( 3333 , 'BX' , 33 )
```

Critères d'un « vrai » SGBD Relationnel (1/2)

Fondements des SGBDR : 1+12 règles d'E. F. CODD, *Is your DBMS really relational?* et *Does your DBMS run by the rules? in Computer-world*, oct. 1985, vulgarisant ses travaux commencés en 1970

- Introduction (règle 0) : tout SGBDR doit gérer complètement les BD à l'aide de ses caractéristiques relationnelles
- Règle de l'information (règle 1) : toutes les informations sont représentées explicitement par des valeurs dans des relations
- Garantie d'accès (règle 2) : toute donnée atomique est accessible par : une ou plusieurs relations ou vues, un attribut, et une valeur de clé primaire
- L'indicateur `NULL` (règle 3) : `NULL` permet de représenter une information manquante ou inapplicable, indépendamment du type, du domaine de valeur ou d'une valeur par défaut
- Catalogue relationnel, dynamique et accessible directement (règle 4) : la description de la BD et de son contenu est représentée de la même manière que les données (relations et valeurs)
- Langage relationnel complet (règle 5) : au moins un langage du SGBDR doit pouvoir définir les données, vues et contraintes d'intégrité (LDD), manipuler les données (LMD), gérer les droits (LCD) et gérer les frontières des transactions (LCT)
- Règle de mise à jour des vues (règle 6) : toute vue qui peut théoriquement mettre à jour des données doit l'autoriser
- LMD ensembliste (règle 7) : un SGBDR retourne un ensemble d'éléments en réponse à une requête d'interrogation qui lui est soumise et doit pouvoir mettre à jour un ensemble d'éléments en exécutant une seule requête
- Indépendance physique des données (règle 8) : les applications restent opérationnelles lorsque les méthodes d'accès physique ou les structures de stockage sont modifiées
- Indépendance logique des données (règle 9) : les applications restent opérationnelles suite à n'importe quelle restructuration (apportées aux relations) qui préserve les informations et qui ne leur portent théoriquement aucune atteinte
- Indépendance vis-à-vis des contraintes d'intégrité (règle 10) : la définition des contraintes d'intégrité spécifiques à une BD doit être stockée dans le catalogue (et non dans les applications)
- Indépendance de distribution (règle 11) : le langage relationnel doit permettre aux applications et aux requêtes d'être identiques, que les données soient centralisées ou réparties
- Non-subversion (règle 12) : il doit être impossible de rendre la BD incohérente en utilisant un langage de plus bas niveau que le langage (relationnel) du SGBDR

Extensions des SGBDR : règles supplémentaires d'E. F. CODD

- 6 règles sur les catalogues, domaines, privilèges, etc.
- 12 règles sur les BD multidimensionnelles (reposant sur le modèle OLAP)

Remarque : les SGBDR respectent plus ou moins ces règles !

Critères d'un « vrai » SGBD Relationnel (2/2)

Noyau minimal d'un SGBDR

Relation : toutes les données (y compris le dictionnaire de données) doivent uniquement être organisées sous forme de relations

Algèbre relationnelle : toutes ses opérations (relationnelles et ensemblistes) doivent être disponibles

Langage SQL : l'interface doit reprendre (au minimum) ses fonctionnalités

Optimiseur de requêtes d'interrogation : indispensable pour améliorer les performances

Vues : possibilité de création de « relations virtuelles »

Fonctionnalités attendues de tout SGBD

Intégrité *(integrity)* : prise en charge continuelle par le SGBDR des contraintes d'intégrité

Sécurité *(security)* : attribution de privilèges à des groupes d'utilisateurs pour des parties de la BD (exemple : utilisation conjointe des vues et des privilèges)

Concurrence : le gestionnaire de transactions

- Récupération *(recovery)* suite à une panne
- Accès simultanés *(concurrency)* : garantir la cohérence, tout en conservant des performances acceptables

Atouts supplémentaires

Outils d'administration

- Deux rôles : définir les objets de la BD et veiller à leur bonne utilisation
- Deux caractéristiques : organisationnelle [niveaux externe et conceptuel] (concevoir la BD, gérer les droits d'accès et les vues, etc.) et technique [niveau interne] (correspondance entre le schéma externe et les possibilités du SGBD pour obtenir les meilleures performances, intégrité, sécurité, etc.)
- Installation/désinstallation du SGBD (avec les outils associés)
- Création physique de la base (interne) : emplacement et dimensionnement des fichiers
- Création et suivi de la base (conceptuelle) : schémas, relations, vues
- Gestion des utilisateurs : groupes d'utilisateurs, privilèges
- Utilitaires de surveillance (conformité opérations/droits) pour garantir la sécurité et la cohérence des données suite aux actions des utilisateurs : audit
- Outils de statistiques concernant les données : journalisation
- Utilitaires d'optimisation pour choisir l'implantation optimale des données fournissant les meilleures performances (en prenant en compte les traitements effectués sur les données) : indexation, clustérisation (données contiguës sur le disque pour éviter de multiplier les accès), dénormalisation (cf. les optimisations logiques et physiques de certaines méthodes ... avec le risque de ne plus respecter les FN), réglages *(tuning)* (analyser l'environnement des utilisateurs et la nature des applications, réaliser des observations et des statistiques sur l'activité de la BD, et ensuite régler la BD en exploitant au mieux les ressources matérielles comme le disque, la mémoire cache, etc.)
- Utilitaires de sauvegarde/restauration pour garantir la sécurité et la cohérence des données (suite à des problèmes matériels ou logiciels)
- Outils d'échanges de données entre la BD et le monde extérieur (exemple : intégration depuis ou migration vers d'autres applications ou BD)

Autres outils (Atelier de Génie Logiciel (AGL), générateurs, etc.) livrés avec le SGBD

Performance - Convivialité - Portabilité - etc.

Fonction du système de gestion des transactions

Assurer que toutes les données restent intègres (i. e. cohérentes), quels que soient les évènements qui surviennent (erreur de programmation, défaillance du matériel ou du logiciel, préemption par une application plus prioritaire, etc. ...mais aussi lorsque tout se passe normalement)

Définition

Une transaction est un groupe d'opérations (ordres SQL) indivisible tel que soit l'ensemble est exécuté, soit aucune ne l'est (en cas de rupture)

Exemple

Toute écriture comptable (équilibrée) qui consiste à débiter un ou plusieurs comptes et à en créditer un ou plusieurs autres

Propriétés « ACID » d'une transaction

Atomicité

L'ensemble des opérations d'une transaction est atomique (indivisible i. e. tout est validé ou rien ne l'est)

Remarque : chaque ordre SQL est atomique

Cohérence (ou consistance)

L'exécution d'une transaction fait passer la BD d'un état cohérent à un autre état cohérent

Isolation

Chaque transaction est indépendante des autres transactions (sérialisation des transactions), même si plusieurs transactions peuvent sous certaines conditions travailler sur les mêmes données

Remarque : une mise à jour effectuée par une transaction non encore validée n'est pas (encore) visible par les autres transactions

Durabilité

Les mises à jour des données effectuées par une transaction sont durables et permanentes

Fichier journal (ou fichier de *log*)

Mémorise d'une part le contenu des enregistrements avant et après chaque mise à jour des données (identifiant de la transaction, horodate, fichier et adresse de la page modifiés, valeurs avant et après) et d'autre part des enregistrements indiquant les début, validation et annulation de transactions ainsi que des points de reprise système indiquant un arrêt normal

N. B. : le journal peut être commun ou être séparé en journal des images avant *(before image log)* et journal des images après *(after image log)*

Le journal complet correspond au fichier journal actif (i. e. en ligne) et à ceux archivés. Quand le journal actif arrive à saturation (vers 90 à 95% de taux d'occupation), il est archivé et le suivant (gérés circulairement) est activé

Write-ahead log protocol: écriture d'abord dans le journal puis dans la BD ; ainsi, tout ce qui est dans la BD est aussi dans le journal, mais la réciproque est fausse (en cas d'incident d'instance par exemple)

Démarrage d'une transaction

Implicite au premier ordre du LDD (`CREATE`, `ALTER`, `DROP`), du LMD (`SELECT`, `INSERT`, `UPDATE`, `DELETE`), du LCD (`GRANT`, `REVOKE`), de gestion d'un curseur (`OPEN`, `FETCH`, `CLOSE`) ou de gestion des locateurs (`HOLD LOCATOR`, `FREE LOCATOR`)

Explicite : `START TRANSACTION` ◁ mode ▷ `ISOLATION LEVEL` ◁ niveau d'isolation ▷, `COMMIT AND CHAIN`, `ROLLBACK AND CHAIN`

Le mode indique quelles sont les opérations possibles durant la transaction : `READ ONLY` (lecture seule) ou par défaut `READ WRITE` (lecture et écriture)

Le niveau d'isolation précise le comportement de la transaction relativement aux autres transactions : `READ UNCOMMITED` (niveau 0), `READ COMMITED` (niveau 1), `REPEATABLE READ` (niveau 2) ou par défaut `SERIALIZABLE` (niveau 3). Le niveau d'isolation est bien évidemment inversement proportionnel à la concurrence ; par exemple, plus le niveau est élevé, moins il y a de concurrence

Fin d'une transaction : `COMMIT` et `ROLLBACK`

`COMMIT` [`AND` [`NO`] `CHAIN`]
`ROLLBACK` [`AND` [`NO`] `CHAIN`]

L'option `AND CHAIN` permet d'enchaîner la fin de la transaction en en démarrant une autre aussitôt

Ces deux ordres SQL terminent tous deux une transaction, en relâchant les verrous, et soit en validant (`COMMIT`), soit en défaisant (`ROLLBACK`) toutes les mises à jour de la transaction

Remarque : en l'absence de tels ordres dans un programme d'application, c'est toute son exécution qui est considérée comme une et une seule transaction

Recommandation « forte » : constituer une transaction pour un ensemble minimal d'opérations qui se complètent (et de durée d'exécution la plus courte possible, ce qui tend à essayer de limiter autant que possible l'attente d'une saisie par un utilisateur) ; ainsi, deux opérations indépendantes font l'objet de deux transactions différentes

Attention : certains SGBD ne respectent pas la norme en faisant de l'`AUTOCOMMIT` ce qui engendre automatiquement un `COMMIT` après chaque instruction de mise à jour

Exemple : `COMMIT` et `ROLLBACK`

On considère le schéma relationnel réduit à la seule relation `r ( a )` de clé `a` (un entier) qui de surcroît est vide au départ

Temps	Session : instructions et données		Données visibles par toutes les autres sessions
0		{}	{}
1	`INSERT INTO r VALUES ( 1 )`	{1}	{}
2	`INSERT INTO r VALUES ( 2 )`	{1,2}	{}
3	`DELETE FROM r WHERE a = 1`	{2}	{}
4	`INSERT INTO r VALUES ( 3 )`	{2,3}	{}
5	`COMMIT`	{2,3}	{2,3}
6	`ROLLBACK`	{2,3}	{2,3}
7	`ROLLBACK`	{2,3}	{2,3}
8	`INSERT INTO r VALUES ( 4 )`	{2,3,4}	{2,3}
9	`DELETE FROM r WHERE a = 2`	{3,4}	{2,3}
10	`INSERT INTO r VALUES ( 5 )`	{3,4,5}	{2,3}
11	`ROLLBACK`	{2,3}	{2,3}
12	`COMMIT`	{2,3}	{2,3}
13	`COMMIT`	{2,3}	{2,3}

Vie d'une transaction (2/2)

Point de retour *(savepoint)*

`SAVEPOINT` ◁ point de retour ▷

Marqueur chronologique au sein d'une transaction auquel on peut revenir

Détruits à la fin de la transaction ou par l'ordre `RELEASE SAVEPOINT` ◁ point de retour ▷

Retour possible durant la transaction, sans la terminer, par l'ordre `ROLLBACK TO SAVEPOINT` ◁ point de retour ▷

Exemple : `SAVEPOINT`

On considère le schéma relationnel réduit à la seule relation `r ( a )` de clé `a` (un entier) qui de surcroît est vide au départ

Temps	Session : instructions et données		Données visibles par tous
0		$\{\}$	$\{\}$
1	`INSERT INTO r VALUES ( 1 )`	$\{1\}$	$\{\}$
2	`INSERT INTO r VALUES ( 2 )`	$\{1,2\}$	$\{\}$
3	`COMMIT`	$\{1,2\}$	$\{1,2\}$
4	`INSERT INTO r VALUES ( 3 )`	$\{1,2,3\}$	$\{1,2\}$
5	`INSERT INTO r VALUES ( 4 )`	$\{1,2,3,4\}$	$\{1,2\}$
6	`SAVEPOINT alpha`	$\{1,2,3,4\}$	$\{1,2\}$
7	`INSERT INTO r VALUES ( 5 )`	$\{1,2,3,4,5\}$	$\{1,2\}$
8	`INSERT INTO r VALUES ( 6 )`	$\{1,2,3,4,5,6\}$	$\{1,2\}$
9	`SAVEPOINT beta`	$\{1,2,3,4,5,6\}$	$\{1,2\}$
10	`INSERT INTO r VALUES ( 7 )`	$\{1,2,3,4,5,6,7\}$	$\{1,2\}$
11	`INSERT INTO r VALUES ( 8 )`	$\{1,2,3,4,5,6,7,8\}$	$\{1,2\}$
12	`SAVEPOINT gamma`	$\{1,2,3,4,5,6,7,8\}$	$\{1,2\}$
13	`INSERT INTO r VALUES ( 9 )`	$\{1,2,3,4,5,6,7,8,9\}$	$\{1,2\}$
14	`INSERT INTO r VALUES ( 10 )`	$\{1,2,3,4,5,6,7,8,9,10\}$	$\{1,2\}$
15	`ROLLBACK TO SAVEPOINT gamma`	$\{1,2,3,4,5,6,7,8\}$	$\{1,2\}$
16	`INSERT INTO r VALUES ( 11 )`	$\{1,2,3,4,5,6,7,8,11\}$	$\{1,2\}$
17	`INSERT INTO r VALUES ( 12 )`	$\{1,2,3,4,5,6,7,8,11,12\}$	$\{1,2\}$
18	`ROLLBACK TO SAVEPOINT alpha`	$\{1,2,3,4\}$	$\{1,2\}$
19	`INSERT INTO r VALUES ( 13 )`	$\{1,2,3,4,13\}$	$\{1,2\}$
20	`INSERT INTO r VALUES ( 14 )`	$\{1,2,3,4,13,14\}$	$\{1,2\}$
21	`SAVEPOINT delta`	$\{1,2,3,4,13,14\}$	$\{1,2\}$
22	`DELETE FROM r WHERE a = 3`	$\{1,2,4,13,14\}$	$\{1,2\}$
23	`DELETE FROM r WHERE a = 13`	$\{1,2,4,14\}$	$\{1,2\}$
24	`COMMIT`	$\{1,2,4,14\}$	$\{1,2,4,14\}$
25	`INSERT INTO r VALUES ( 15 )`	$\{1,2,4,14,15\}$	$\{1,2,4,14\}$
26	`INSERT INTO r VALUES ( 16 )`	$\{1,2,4,14,15,16\}$	$\{1,2,4,14\}$
27	`INSERT INTO r VALUES ( 17 )`	$\{1,2,4,14,15,16,17\}$	$\{1,2,4,14\}$
28	`DELETE FROM r WHERE a = 14`	$\{1,2,4,15,16,17\}$	$\{1,2,4,14\}$
29	`DELETE FROM r WHERE a = 15`	$\{1,2,4,16,17\}$	$\{1,2,4,14\}$
30	`SAVEPOINT epsilon`	$\{1,2,4,16,17\}$	$\{1,2,4,14\}$
31	`INSERT INTO r VALUES ( 18 )`	$\{1,2,4,16,17,18\}$	$\{1,2,4,14\}$
32	`INSERT INTO r VALUES ( 19 )`	$\{1,2,4,16,17,18,19\}$	$\{1,2,4,14\}$
33	`DELETE FROM r WHERE a = 17`	$\{1,2,4,16,18,19\}$	$\{1,2,4,14\}$
34	`SAVEPOINT zêta`	$\{1,2,4,16,18,19\}$	$\{1,2,4,14\}$
35	`INSERT INTO r VALUES ( 20 )`	$\{1,2,4,16,18,19,20\}$	$\{1,2,4,14\}$
36	`DELETE FROM r WHERE a = 16`	$\{1,2,4,18,19,20\}$	$\{1,2,4,14\}$
37	`DELETE FROM r WHERE a = 19`	$\{1,2,4,18,20\}$	$\{1,2,4,14\}$
38	`DELETE FROM r WHERE a = 1`	$\{2,4,18,20\}$	$\{1,2,4,14\}$
39	`ROLLBACK TO SAVEPOINT epsilon`	$\{1,2,4,16,17\}$	$\{1,2,4,14\}$
40	`COMMIT`	$\{1,2,4,16,17\}$	$\{1,2,4,16,17\}$

Rôles du système gestion transactions (1/2)

Récupération des pannes

La procédure de reprise (exécutée au redémarrage du SGBD) normale (c.-à-d. après un arrêt normal), détectable par le fait que le dernier enregistrement du journal est un point de reprise système, consiste simplement à restaurer le contexte d'exécution sauvegardé ; sinon, le gestionnaire de transactions doit reconstituer un état cohérent et en perdant le minimum de travaux exécutés avant une panne auquel cas on distingue la reprise à chaud pour un incident d'instance (le SGBD avait perdu ce qu'il avait en mémoire principale, mais sans détérioration de la mémoire auxiliaire) et la reprise à froid pour un incident disque (le SGBD avait perdu des informations de la mémoire auxiliaire)

Incident d'instance (coupure de courant, système d'exploitation défaillant, etc.)

Reprise à partir du dernier point de reprise système du journal et des données correspondantes, puis en lançant (analyse en avant) toutes les transactions et en détectant celles validées dites gagnantes (i. e. commencées et terminées par `COMMIT` ou `ROLLBACK`) et celles non encore validées dites perdantes (i. e. commencées mais non terminées, ni par `COMMIT`, ni par `ROLLBACK`), et ensuite en défaisant (analyse à reculons) toutes les transactions non encore validées

Deux primitives de base : `REDO` ◁ transaction ▷ et `UNDO` ◁ transaction ▷ pour respectivement refaire et défaire des transactions

Incident disque

Restauration de la dernière sauvegarde cohérente (des données et du journal), et utilisation du journal des transactions qui ont suivi pour obtenir l'état le plus récent

Rôles du système gestion transactions (2/2)

Gestion des accès concurrents aux données

Principe : protection active des données au cours d'une transaction (i. e. à tout moment, le SGBD les vérifie), basée sur l'acquisition de verrous sur les tuples traités

Le gestionnaire de transactions (et non le programmeur) positionne automatiquement les verrous

Types de verrous : exclusif ou partagé d'une part, global ou sur un seul tuple d'autre part

Exclusif *(Exclusive lock) vs* Partagé *(Shared lock)*

Un verrou exclusif sert à mettre à jour un ou plusieurs tuples ; les autres transactions n'auront plus accès à ce(s) tuple(s), ni pour le(s) lire ni pour le(s) mettre à jour

Un verrou partagé sert à lire un ou plusieurs tuples ; les autres transactions ne pourront qu'acquérir un verrou partagé sur ce(s) tuple(s) et donc lire le(s) tuple(s) mais pas le(s) mettre à jour

Global *(Global lock)*

Un verrou global sert à bloquer un ensemble de tuples voire une relation

Remarque : un verrou est acquis jusqu'à la fin de la transaction en cours

Problèmes de concurrence

Lecture inconsistante (ou lecture impropre)

Une donnée lue devient caduque

Solution avec un verrou exclusif

Lecture non répétitive (ou lecture non renouvelable)

Une donnée lue diffère de cette même donnée relue

Solution avec un verrou partagé

Lignes fantômes (ou lecture fantôme)

Des données sélectionnées apparaissent, changent ou disparaissent en cours de traitement

Solution avec un verrou global (les solutions avec des verrous partagés et/ou exclusifs conduisent souvent à un inter-blocage)

Inter-blocage

Au moins deux transactions s'attendent l'une l'autre ; elles ne peuvent être débloquées que par une intervention extérieure

Le SGBD détecte un cycle dans le graphe d'attente des transactions (les sommets sont les transactions et les arcs partent de la transaction qui est bloquée vers celle qui la bloque ; ainsi, T' $\longrightarrow$ T" signifie que la transaction T' attend la transaction T", en précisant éventuellement le temps et les tuples concernés) et annule le dernier ordre de la transaction qui : soit a créé l'interblocage, soit a démarré le plus récemment, soit a réalisé le moins de mises à jour, soit mobilise le plus petit nombre de verrous, etc.

Remarque : cela survient fréquemment en situation concurrentielle, ce qui impose, pour une application professionnelle, de programmer correctement les exceptions engendrées par des erreurs (il faut au moins gérer `SQLSTATE` après chaque ordre SQL)

Lecture inconsistante

Problème

T1	T2
	`UPDATE` t
`SELECT` t	
	`ROLLBACK`

Solution

T1	T2	Graphe d'attente
	verrou exclusif sur t	
	`UPDATE` t	T1 T2
attente `SELECT` t		T1 $\xrightarrow[t]{}$ T2
	`ROLLBACK`	T1 T2
`SELECT` t		T1 T2

Commentaire

Une transaction T1 lit un tuple t qui devient inconsistant (ou impropre ou sali) suite à l'annulation de la modification par une autre transaction T2

Remarque : les ordres de mise à jour (`INSERT`, `UPDATE`, `DELETE`) posent automatiquement un verrou exclusif sur les tuples concernés

Lecture non répétitive

Problème

T1	T2
`SELECT` t	
	verrou exclusif sur t
	`UPDATE` t
	`COMMIT`
`SELECT` t	

Solution

T1	T2	Graphe d'attente
verrou partagé sur t		
`SELECT` t `FOR UPDATE`		T1 T2
	attente verrou exclusif sur t	T1 $\xleftarrow[t]{}$ T2
`SELECT` t		
libérer verrou sur t		T1 T2
	verrou exclusif sur t	T1 T2
	`UPDATE` t	T1 T2
	`COMMIT`	T1 T2

Commentaire

Une transaction T1 lit un tuple t qu'elle désire ensuite relire et modifier ; entre-temps, une autre transaction T2 modifie le tuple t de sorte que le critère de sélection initial peut devenir caduc

Remarque : `SELECT ... FOR UPDATE` pose automatiquement un verrou partagé sur les tuples concernés

Problèmes et solutions de concurrence (2/2)

Lignes fantômes

Problème

T1	T2
`SELECT` t1	
	`DELETE` t1
	`DELETE` t2
	`UPDATE` t3
`SELECT` t3	

Solution

T1	T2	Graphe d'attente
verrou global (sur t1, t2, t3)		
`SELECT` t1		T1 T2
	attente verrou exclusif sur t1	T1 $\xleftarrow[t1]{}$ T2
`SELECT` t2		T1 $\xleftarrow[t1]{}$ T2
`SELECT` t3		T1 $\xleftarrow[t1]{}$ T2
libérer verrou (sur t1, t2, t3)		
	verrou exclusif sur t1	
	`DELETE` t1	T1 T2
	verrou exclusif sur t2	
	`DELETE` t2	T1 T2
	verrou exclusif sur t3	
	`UPDATE` t3	T1 T2
	`COMMIT`	T1 T2

Commentaire

Une transaction T1 lit un ensemble de tuples {t1,t2,t3} répondant à un critère de sélection afin d'effectuer un calcul global ; entre-temps, une autre transaction T2 modifie ou supprime les tuples répondant initialement à ce critère (il est même possible que la modification inverse le critère), rendant le calcul incorrect

Inter-blocage

Problème

T1	T2	Graphe d'attente
verrou exclusif sur t1		
(ex. : `UPDATE` t1)		T1 T2
	verrou exclusif sur t2	
	(ex. : `UPDATE` t2)	T1 T2
attente t2 (ex. : `UPDATE` t2)		T1 $\xrightarrow[t2]{}$ T2
	attente t1 (ex. : `UPDATE` t1)	T1 $\xrightarrow[t2]{}$ $\xleftarrow[t1]{}$ T2

« Solution »

Le SGBD détecte un cycle dans le graphe d'attente et détruit le dernier ordre de l'une des deux transactions

Commentaire

Les deux transactions T1 et T2 s'attendent l'une l'autre

Éléments : il y a 144 tuiles, à savoir 36 motifs chacun en 4 exemplaires ; les motifs sont les saisons, les fleurs, les dragons (rouges, verts, blancs), les vents (d'Est, du Sud, d'Ouest, du Nord), les ronds (1 à 9), les bambous (1 à 9), et les caractères (1 à 9)

Règle : les tuiles sont au départ empilées comme sur le dessin ci-dessous ; une tuile peut être libérée si aucune tuile ne la bloque par le dessus et si aucune tuile ne la bloque à gauche ou à droite

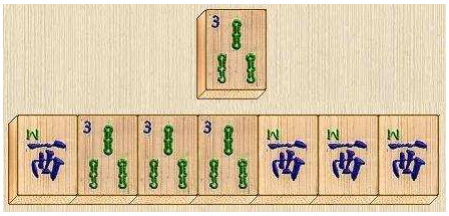
Objectif : enlever les tuiles du même motif deux par deux de sorte à finalement toutes les enlever

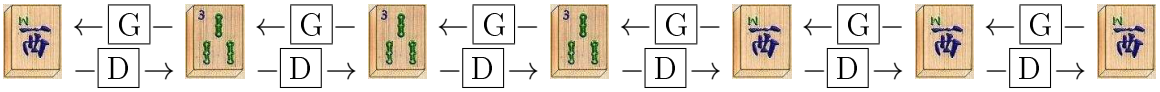
Remarque : la partie ci-dessus ne peut pas être gagnée car un interblocage peut être mis en évidence

Pour démontrer que la partie précédente ne peut pas être gagnée, il suffit de considérer uniquement les quatre  (vents d'Ouest) et les quatre  (bambous 3)

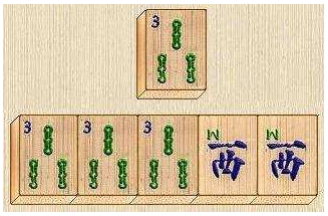
Étudions parallèlement (au jeu en cours) le graphe des blocages (simplifié par le fait qu'il n'y a pas de tuiles superposées) pour lequel $y \ \leftarrow\boxed{\text{G}}- \ x$ (ou $x \ -\boxed{\text{D}}\rightarrow y$) indique que x est bloqué à gauche (droite respectivement) par y et où une tuile x peut être enlevée si et seulement si $\nexists\ y' \ / \ y' \ \leftarrow\boxed{\text{G}}- \ x \vee \nexists\ y'' \ / \ x \ -\boxed{\text{D}}\rightarrow y''$

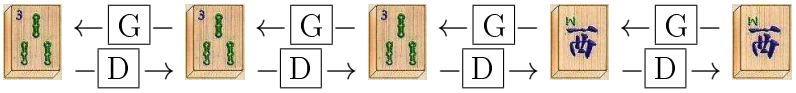
Étape initiale :



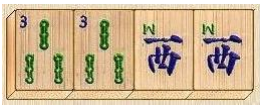


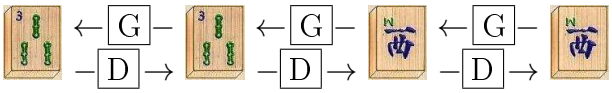
Étape intermédiaire :





Étape finale :





Les tuiles restantes s'interbloquent

Niveaux d'isolation des transactions et verrous

`READ UNCOMMITED` : aucun verrou !

`READ COMMITED` : verrou exclusif sur tout enregistrement à modifier

`REPEATABLE READ` : verrou partagé sur tout enregistrement lu

`SERIALIZABLE` : verrou global

Problèmes de concurrence et niveaux d'isolation des transactions

	`READ UNCOMMITED`	`READ COMMITED`	`REPEATABLE READ`	`SERIALIZABLE`
Lecture inconsistante	peut survenir	*impossible*	*impossible*	*impossible*
Lecture non répétitive	peut survenir	peut survenir	*impossible*	*impossible*
Lignes fantômes	peut survenir	peut survenir	peut survenir	*impossible*
Inter-blocage	peut survenir	peut survenir	peut survenir	peut survenir

Sérialisation (niveau d'isolation `SERIALIZABLE`)

L'exécution concurrente (i. e. simultanée) d'un ensemble de transactions sérialisables est équivalente (i. e. donne le même résultat) à l'exécution séquentielle (i. e. en série) de ces transactions (dans un ordre quelconque)

Cela force toute transaction d'un ensemble de transactions en conflit à attendre la fin d'au moins une autre transaction de l'ensemble

Une telle transaction vérifie : qu'elle ne met pas à jour ni ne lit des données rendues inconsistantes (i. e. impropres ou encore salies) par d'autres transactions, qu'elle ne confirme pas ses mises à jour avant la fin de la transaction, que d'autres transactions ne rendent pas inconsistantes (i. e. impropres ou encore sales) ses données avant qu'elle ne soit terminée

Un exemple de compatibilité des ordres SQL avec les verrous : Oracle (version 7)

Ordre SQL	Verrou	Compatibilité avec				
		`RS`	`RX`	`S`	`SRX`	`X`
`SELECT ... FOR READ ONLY`	aucun	OUI	OUI	OUI	OUI	OUI
`LOCK TABLE ... ROW SHARE ...`	`RS`	OUI	OUI	OUI	OUI	*non*
`SELECT ... FOR UPDATE`	`RS`	+/−	+/−	+/−	+/−	*non*
`LOCK TABLE ... ROW EXCLUSIVE ...`	`RX`	OUI	OUI	*non*	*non*	*non*
`INSERT ...`	`RX`	OUI	OUI	*non*	*non*	*non*
`UPDATE ...`	`RX`	+/−	+/−	*non*	*non*	*non*
`DELETE ...`	`RX`	+/−	+/−	*non*	*non*	*non*
`LOCK TABLE ... SHARE ...`	`S`	OUI	*non*	OUI	*non*	*non*
`LOCK TABLE ... SHARE ROW EXCLUSIVE ...`	`SRX`	OUI	*non*	*non*	*non*	*non*
`LOCK TABLE ... EXCLUSIVE ...`	`X`	*non*	*non*	*non*	*non*	*non*

Une requête d'interrogation (`SELECT`) peut se terminer (après un éventuel `ORDER BY`) par `FOR READ ONLY` pour lire un ou plusieurs tuples que l'on n'envisage pas de relire ou de mettre à jour et qui peut donc être lu ou mis à jour par d'autres transactions ou par `FOR UPDATE` [`OF` ◁ attributs ▷] pour lire un ou plusieurs tuples que l'on envisage de relire ou de mettre à jour

`LOCK TABLE` ◁ relations ▷ `IN` ◁ verrou ▷ `MODE`

Le mode du verrou à appliquer explicitement ◁ verrou ▷ vaut : `ROW SHARE` (`RS`) tuples partagés, `ROW EXCLUSIVE` (`RX`) tuples exclusifs, `SHARE` (`S`) relation partagée, `SHARE ROW EXCLUSIVE` (`SRX`) partage exclusif des tuples, `EXCLUSIVE` (`X`) relation exclusive

Concurrence : exemples (1/4)

On considère pour chacun des quatre exemples ci-dessous le même schéma relationnel réduit à la seule relation `r ( a , b )` de clé `a` contenant initialement les deux tuples $(1,'un')$ et $(2,'deux')$

Blocages et `COMMIT`/`ROLLBACK`

Temps	Première session	Seconde session	Données visibles par les autres sessions
0	$\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$

1	`UPDATE r`
	`SET b = 'ONE'`
	`WHERE a = 1`
	$\{(1,'ONE'),(2,'deux')\}$
			$\{(1,'un'),(2,'deux')\}$

2		`UPDATE r`
		`SET b = 'DOS'`
		`WHERE a = 2`
		$\{(1,'un'),(2,'DOS')\}$
			$\{(1,'un'),(2,'deux')\}$

3	`DELETE FROM r`
	`WHERE a = 2`
	En attente
			$\{(1,'un'),(2,'deux')\}$

On a : Première session $\xrightarrow[\texttt{a=2}]{3}$ Seconde session ...
suivi d'un `ROLLBACK` ou d'un `COMMIT`, dans la seconde ou la première session

4	Débloqué $\{(1,'ONE')\}$	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$
5	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$

4	Débloqué $\{(1,'ONE')\}$	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$
5	`COMMIT` $\{(1,'ONE')\}$	$\{(1,'ONE')\}$	$\{(1,'ONE')\}$

4	Débloqué $\{(1,'ONE')\}$	`COMMIT` $\{(1,'un'),(2,'DOS')\}$	$\{(1,'un'),(2,'DOS')\}$
5	`ROLLBACK` $\{(1,'un'),(2,'DOS')\}$	$\{(1,'un'),(2,'DOS')\}$	$\{(1,'un'),(2,'DOS')\}$

4	Débloqué $\{(1,'ONE')\}$	`COMMIT` $\{(1,'un'),(2,'DOS')\}$	$\{(1,'un'),(2,'DOS')\}$
5	`COMMIT` $\{(1,'ONE')\}$	$\{(1,'ONE')\}$	$\{(1,'ONE')\}$

On considère pour chacun des cinq exemples suivants le même schéma relationnel réduit à la seule relation `r ( a , b )` de clé `a` contenant initialement les deux tuples $(1,'un')$ et $(2,'deux')$

Blocages pour `SELECT`

Temps	Première session	Seconde session
0	$\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$
1	`SELECT * FROM r WHERE a <= 3 FOR READ ONLY` $\{(1,'un'),(2,'deux')\}$	
2		`DELETE FROM r WHERE a = 1` $\{(2,'deux')\}$
3		`UPDATE r SET b = 'ZWEI' WHERE a = 2` $\{(2,'ZWEI')\}$
4		`INSERT INTO r VALUES ( 3 , 'DREI' )` $\{(2,'ZWEI'),(3,'DREI')\}$
5	$\{(2,'ZWEI'),(3,'DREI')\}$	`COMMIT` $\{(2,'ZWEI'),(3,'DREI')\}$

Tps	Première session	Deuxième session	Troisième session	Quatrième session
0	$\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$	$\{(1,'un'),(2,'deux')\}$
1	`SELECT * FROM r WHERE a <= 3 FOR UPDATE` $\{(1,'un'),(2,'deux')\}$			
2		`DELETE FROM r WHERE a = 1` En attente		
3			`UPDATE r SET b = 'TWO' WHERE a = 2` En attente	
4				`INSERT INTO r VALUES ( 3 , 'TRES' )` $\{(1,'un'),(2,'deux'),(3,'TRES')\}$
5	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	Débloqué $\{(2,'deux')\}$	Débloqué $\{(1,'un'),(2,'TWO')\}$	
6	`SELECT * FROM r WHERE a <= 3 FOR UPDATE` En attente			
7		`ROLLBACK` $\{(1,'un'),(2,'deux')\}$		
8	Débloqué $\{(1,'un'),(2,'deux')\}$		`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	
9				`ROLLBACK` $\{(1,'un'),(2,'deux')\}$

avec 2^e session $\xrightarrow[\texttt{a=1}]{2}$ 1^{re} session au temps 2, 2^e session $\xrightarrow[\texttt{a=1}]{2}$ 1^{re} session $\xleftarrow[\texttt{a=2}]{3}$ 3^e session aux temps 3 et 4, 2^e session $\xleftarrow[\texttt{a=1}]{6}$ 1^{re} session $\xrightarrow[\texttt{a=2}]{6}$ 3^e session au temps 6, 1^{re} session $\xrightarrow[\texttt{a=2}]{6}$ 3^e session au temps 7, et aucun blocage aux autres temps (0,1,5,8,9)

Inter-blocages pour `UPDATE` et `DELETE`

Temps	Données initialement visibles par toutes les sessions
0	$\{(1,'un'),(2,'deux')\}$

Temps	Première session	Seconde session
1	`UPDATE r SET b = 'ONE' WHERE a = 1` $\{(1,'ONE'),(2,'deux')\}$	
2		`UPDATE r SET b = 'DOS' WHERE a = 2` $\{(1,'un'),(2,'DOS')\}$
3	`UPDATE r SET b = 'TWO' WHERE a = 2` En attente	
4	Débloqué car inter-blocage / erreur $\{(1,'ONE'),(2,'deux')\}$	`UPDATE r SET b = 'UNO' WHERE a = 1` Attente, détection inter-blocage, attente
5	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	Débloqué $\{(1,'UNO'),(2,'DOS')\}$
6	$\{(1,'un'),(2,'deux')\}$	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$

Temps	Première session	Seconde session
1	`DELETE FROM r WHERE a = 1` $\{(2,'deux')\}$	
2		`DELETE FROM r WHERE a = 2` $\{(1,'un')\}$
3	`DELETE FROM r WHERE a = 2` En attente	
4	Débloqué car inter-blocage / erreur $\{(2,'deux')\}$	`DELETE FROM r WHERE a = 1` Attente, détection inter-blocage, attente
5	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	Débloqué $\{\}$
6	$\{(1,'un'),(2,'deux')\}$	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$

avec (pour la modification et pour la suppression) 1^{re} session $\xrightarrow[\texttt{a=2}]{3}$ 2^{de} session au temps 3, interblocage 1^{re} session $\begin{array}{c}\xrightarrow[\texttt{a=2}]{3}\\ \xleftarrow[\texttt{a=1}]{4}\end{array}$ 2^{de} session détecté et transformé en 1^{re} session $\xleftarrow[\texttt{a=1}]{4}$ 2^{de} session au temps 4, et aucun blocage aux autres temps (0,1,2,5,6)

Inter-blocages pour `INSERT`

Temps	Données initialement visibles par toutes les sessions
0	$\{(1,'un'),(2,'deux')\}$

Temps	Première session	Seconde session

1	`INSERT INTO r`
	`VALUES ( 3 , 'THREE' )`
	$\{(1,'un'),(2,'deux'),(3,'THREE')\}$	

2		`INSERT INTO r`
		`VALUES ( 4 , 'CUATRO' )`
		$\{(1,'un'),(2,'deux'),(4,'CUATRO')\}$

3	`INSERT INTO r`
	`VALUES ( 4 , 'FOUR' )`
	En attente	

4	Débloqué car inter-blocage / erreur
	$\{(1,'un'),(2,'deux'),(3,'THREE')\}$
		`INSERT INTO r`
		`VALUES ( 3 , 'TRES' )`
		Attente, détection inter-blocage, attente

5	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$	Débloqué $\{(1,'un'),(2,'deux'),(3,'TRES'),$ $(4,'CUATRO')\}$
6	$\{(1,'un'),(2,'deux')\}$	`ROLLBACK` $\{(1,'un'),(2,'deux')\}$

avec 1^{re} session $\xrightarrow[\texttt{a=4}]{3}$ 2^{de} session au temps 3, interblocage 1^{re} session $\begin{array}{c}\xrightarrow{3}\\ \texttt{a=4}\\ \xleftarrow{4}\\ \texttt{a=3}\end{array}$ 2^{de} session détecté et transformé en 1^{re} session $\xleftarrow[\texttt{a=3}]{4}$ 2^{de} session au temps 4, et aucun blocage aux autres temps (0,1,2,5,6)

Indexation (1/5)

Objectif

Un index peut augmenter la vitesse d'exécution des requêtes d'interrogation (ainsi que celles de modification et de suppression qui utilisent une sélection) mais sa gestion diminue les performances des requêtes de mise à jour

Remarques

On peut voir un index sur un (ou plusieurs) attribut(s) A d'une relation R de clé primaire C comme une relation $I = \pi_{A,C}(R)$ triée sur A (qui est le critère de recherche)

Un index peut être unique (lorsque toutes les valeurs de(s) l'attribut(s) indexé(s) sont différentes deux à deux) ou non ; ainsi, une contrainte d'unicité peut être définie à l'aide d'un index unique

Règles à respecter pour créer des index

Ne pas indexer de petites relations

Ne pas indexer un (ou plusieurs) attribut(s) ayant peu de valeurs distinctes (sinon, ce serait peu filtrant)

Indexer les attributs intervenant souvent dans les sélections et le tri des requêtes d'interrogation ainsi que dans les sélections des requêtes de modification et de suppression

Indexer les attributs de jointure (clé primaire et clé étrangère) si le SGBD ne les crée pas automatiquement

Syntaxe (hors norme)

La norme SQL ne décrit pas les ordres de création et de suppression d'un index laissant libres les éditeurs de SGBD mais la plupart proposent `CREATE` [`UNIQUE`] `INDEX` ◁ index ▷ `ON` ◁ relation ▷ `(` ◁ attributs ▷ `)`

```
-- créer un index sur le nom des étudiants (en ordre croissant)
CREATE INDEX IndexNomEtudiants ON Etudiants ( NomEtu )
-- créer une contrainte d'unicité via un index unique sur l'année
-- d'obtention des diplômes des étudiants (i. e. un étudiant n'a obtenu
-- qu'un diplôme par année)
CREATE UNIQUE INDEX IndexAnneeDiplomeEtudiant
    ON AvoirObtenu ( NoINE ASC , Annee DESC )
```

Structure interne d'un index

Séquentielle

Les données sont enregistrées séquentiellement dans l'ordre du tri des attributs de l'index

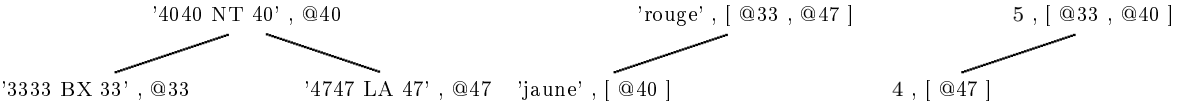
Index hiérarchique

Les plus classiques des index hiérarchiques sont les B-arbres (i. e. arbres équilibrés)

Le nombre d'entrées/sorties est égal à $1 + \log(hauteur(arbre))$

Ils sont présents dans tous les SGBD actuels

Exemples : index sur $(NoImmatChiffres, NoImmatLettres, NoImmatDepart)$ de $Voitures$ (clé primaire), sur $Voitures.Couleur$ (attribut quelconque) et sur $Voitures.NoINE$ (clé étrangère)



où @ désigne l'adresse physique d'un objet

Hachage *(hash-coding)*

La technique de hachage consiste à appliquer une fonction (dite de hachage) à une clé retournant l'emplacement des données (i. e. un numéro de bloc)

Avantages : moins de verrous, une seule entrée/sortie, etc.

Inconvénients : utilisable uniquement pour des recherches d'égalité avec la clé, collisions i. e. $\exists k_1, k_2 / f(k_1) = f(k_2)$ (solutions : chaînage pour parcourir une zone de débordement en cas de conflit, fonction secondaire), etc.

Exemple « simpliste » : la fonction de hachage $f()$ de la clé $(NoImmatChiffres, NoImmatLettres, NoImmatDepart)$ de la relation $Voitures$ retourne une adresse physique correspondant au chiffre des milliers du numéro d'immatriculation en chiffres de sorte que $f(3333,'BX', 33) \neq f(4747,'LA', 47) = f(4040,'NT', 40)$

Cluster

Un *cluster* contient les tuples d'une ou plusieurs relations possédant au moins un attribut en commun (généralement la clé primaire et la clé étrangère), physiquement stockés ensemble (i. e. à côté, dans un même secteur autant que possible) sur le disque

Matrice binaire *(bitmap)*

Pour indexer un attribut A d'une relation R de t tuples, on crée une matrice booléenne ayant t lignes et autant de colonnes que de valeurs distinctes dans A (i. e. $card(\pi_A(R)) = ndist(A)$) et dont la valeur est vraie si et seulement si la valeur dans R est la même que la colonne de la matrice (corollaires : une et une seule valeur est vraie dans chaque ligne ; chaque colonne a au moins une valeur vraie)

Exemples : index sur la couleur des voitures et index sur Etudiants.NomEtu

'rouge'	'jaune'
VRAI	*FAUX*
VRAI	*FAUX*
FAUX	VRAI

'DUPOND'	'DURAND'	'LEROI'	'MARTIN'
FAUX	VRAI	*FAUX*	*FAUX*
FAUX	*FAUX*	VRAI	*FAUX*
FAUX	*FAUX*	*FAUX*	VRAI
FAUX	*FAUX*	VRAI	*FAUX*
VRAI	*FAUX*	*FAUX*	*FAUX*

Esquisse d'algorithmes possibles pour des index stockés sous la forme de B-arbres

Parcours dans l'ordre de la clé primaire `SELECT * FROM Relation ORDER BY CléPrimaire`

Pour chaque sommet s du B-arbre de l'index de la clé primaire
... (parcouru en profondeur, en ordre infixe) faire
Lire le tuple t grâce à son adresse physique $s.@$
Écrire t

Parcours dans l'ordre d'un attribut `SELECT * FROM Relation ORDER BY Attribut`

Pour chaque sommet s du B-arbre de l'index de l'attribut
... (parcouru en profondeur, en ordre infixe) faire
Pour chaque adresse physique @ de s faire
Lire le tuple t grâce à son adresse physique @
Écrire t

Parcours d'une jointure `SELECT * FROM R' JOIN R'' ON R'.PK = R''.FK`

// 1^{er} algorithme
Pour chaque sommet s' du B-arbre de l'index de la clé primaire faire
Rechercher sommet s'' du B-arbre de index de clé étrangère tq $s''.FK = s'.PK$
Si il existe un tel s'' Alors
Lire le tuple t' grâce à son adresse physique $s'.@$
Pour chaque adresse physique @ de s'' faire
Lire le tuple t'' grâce à son adresse physique @
Écrire t', t''

// 2^{e} algorithme
Pour chaque sommet s'' du B-arbre de l'index de la clé étrangère faire
Rechercher sommet s' du B-arbre de index de clé primaire tq $s'.PK = s''.FK$
Lire le tuple t' grâce à son adresse physique $s'.@$
Pour chaque adresse physique @ de s'' faire
Lire le tuple t'' grâce à son adresse physique @
Écrire t', t''

// 3^{e} algorithme
Pour chaque sommet s''' du B-arbre de l'index de la clé primaire
... de la relation contenant la clé étrangère faire
Lire le tuple t'' grâce à son adresse physique $s'''.@$
Rechercher sommet s' du B-arbre de index de clé primaire tq $s'.PK = t''.FK$
Lire le tuple t' grâce à son adresse physique $s'.@$
Écrire t', t''

L-U B-arbres (1/2)

Définition

Arbres équilibrés *(Balanced)* (i. e. deux feuilles (i. e. sommets internes) quelconques sont soit au même niveau, soit ont une différence de niveau en valeur absolue égale à un)

$L, U \in \mathbb{N}^*$ (L *(Lower)* et U *(Upper)* sont des entiers positifs) avec $L \leq U$

Pour tout sommet excepté la racine, on a $L \leq nombre\_de\_clés + 1 \leq U$

Pour tout nœud (i. e. sommet interne), on a $nombre\_de\_fils = nombre\_de\_clés + 1 \leq U$

Les clés sont triées (en ordre infixe)

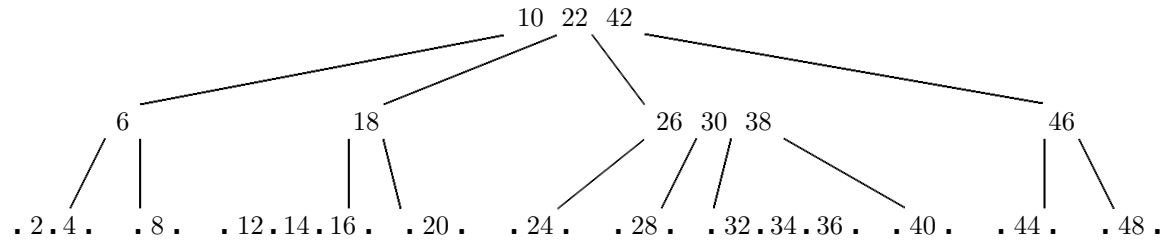
Remarques

Utilisation : implantation des index des BD (et SGF)

Complexité : insertion et suppression en temps d'exécution amorti logarithmique

Ces arbres grandissent par la racine

Exemple avec $L = 2$ et $U = 4$



Algorithme d'insertion

// appelant

Rechercher le sommet Σ où devrait être inséré la clé κ

Insérer(Σ,κ)

// appelé (récursivement)

Action Insérer(Σ,κ)

Si $nombre\_de\_clés(\Sigma) \leq U - 2$ Alors

Ajouter κ à l'ensemble des clés de Σ // on a alors $nombre\_de\_clés(\Sigma) \leq U - 1$

Sinon

// on a $nombre\_de\_clés(\Sigma) = U - 1$ et on suppose $\frac{U}{2} \geq L$

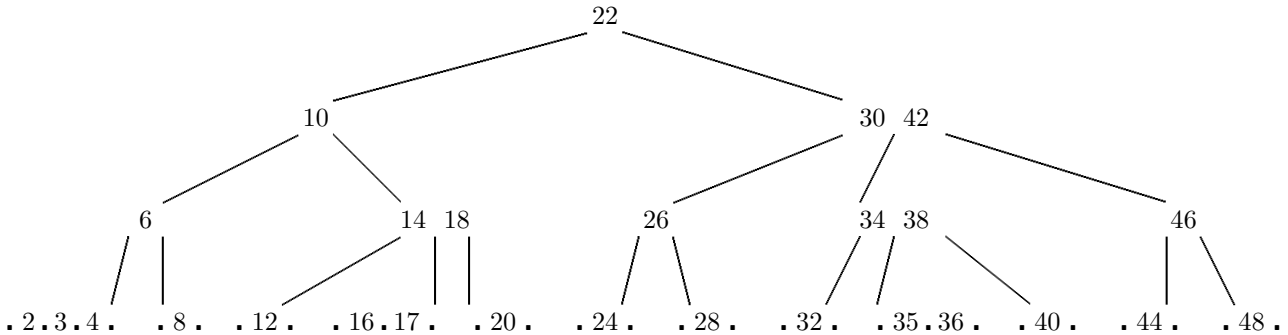
Soit μ la clé médiane de E l'ensemble des clés de $\Sigma \cup \{\kappa\}$ // avec $card(E) = U$

Soient $\Sigma' = \{\lambda \in E : \lambda < \mu\}$ et $\Sigma'' = \{\lambda \in E : \lambda > \mu\}$

// avec $nombre\_de\_clés(\Sigma') = \lfloor \frac{U-1}{2} \rfloor$ et $nombre\_de\_clés(\Sigma'') = \lceil \frac{U-1}{2} \rceil$

Insérer(μ,$père(\Sigma)$)

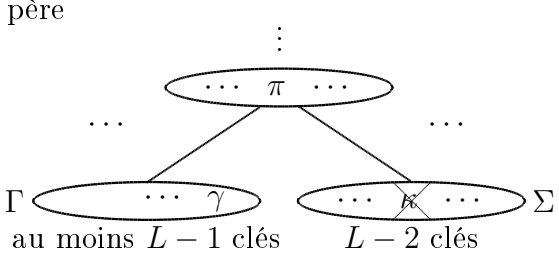
Exemple : après les insertions de 3, de 17 ($\mu = 14$) et de 35 ($\mu = 34, 30, 22$) dans l'exemple



L-U B-arbres (2/2)

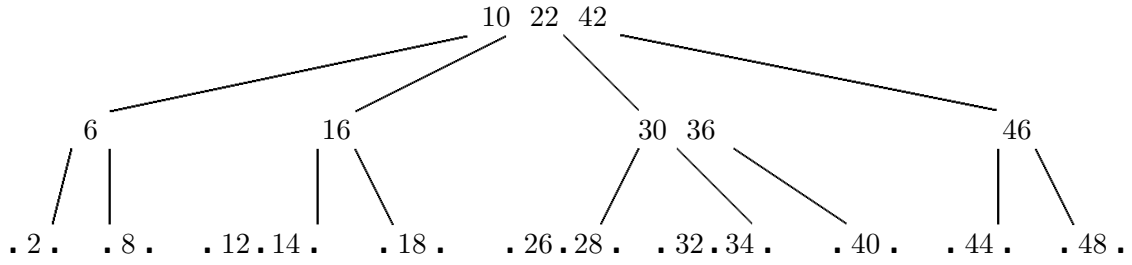
Algorithme de suppression

Rechercher le sommet Σ contenant la clé κ à supprimer
Si Σ est un nœud Alors
Échanger κ avec κ' de feuille Φ la + à droite dans ss-arbre juste à gauche de κ
$\Sigma \longleftarrow \Phi$
// on a Σ qui est maintenant obligatoirement une feuille
Enlever κ de l'ensemble des clés de Σ
Si $nombre\_de\_clés(\Sigma) = L - 2$ Alors
// le nombre de clés est trop faible et il faut donc réorganiser l'arbre
Soit Γ le frère juste à gauche ou juste à droite de Σ
Soit π la clé séparant Γ et Σ dans leur père

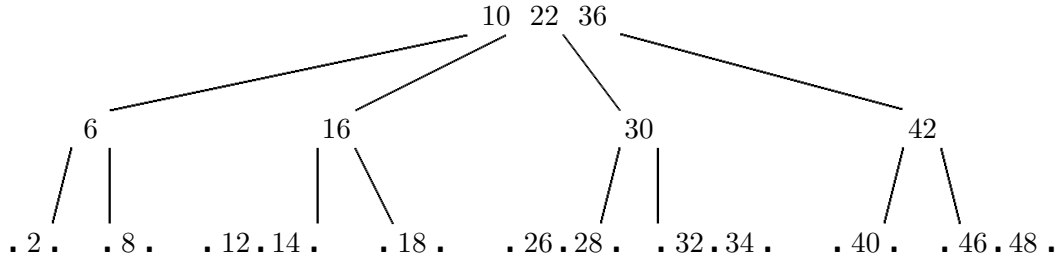


//
Si $nombre\_de\_clés(\Gamma) \geq L$ Alors
Remplacer π par γ la + grande ou la + petite (respectivement) clé de Γ
Placer π dans Σ là où était κ
Sinon
// on a $nombre\_de\_clés(\Gamma) = L - 1$
Fusionner Γ, $\{\pi\}$ et Σ // avec $nombre\_de\_clés(\Gamma \cup \{\pi\} \cup \Sigma) = 2L - 2$
.../... // à itérer pour le père s'il n'a pas suffisamment de clés...

Exemple : après les suppressions de 4, de 38 ($\kappa' = 36$), de 20 ($\pi = 18$) et de 24 (fusion de $\Sigma = \emptyset$,$\pi = 26$ et $\Gamma = \{28\}$) dans l'exemple



...puis après la suppression de 44



Optimisation requête d'interrogation (1/5)

Définitions

Optimiseur

Composant du SGBDR chargé d'élaborer le meilleur plan d'exécution possible pour retourner le résultat (i. e. réponse) d'une requête d'interrogation (i. e. une question)

Remarques

L'optimiseur utilise les index existants (sur les clés primaires, les clés étrangères, les attributs indexés), peut même s'en créer (temporairement) dynamiquement, mais peut aussi décider d'ignorer un index quand il le juge nécessaire (par exemple, lorsque l'attribut indexé est peu filtrant et rend plus de 25% des tuples)

Un optimiseur doit cependant rester simple pour ne pas passer son temps à essayer d'améliorer la vitesse d'exécution de la requête plutôt que d'effectuer réellement le traitement !

Les algorithmes des optimiseurs étaient auparavant, il y a encore une voire deux décennie(s), purement empiriques

Plan d'exécution

Programme optimisé composé d'opérations d'accès à la BD de bas niveau (exemples : recherche d'un tuple par l'adresse physique, recherche d'un tuple par jointure clustérisée, recherche d'un tuple par fonction de hachage, recherche d'un tuple par clé unique, recherche dans un index unique ou non, parcours complet d'un index unique ou non, parcours complet d'une relation (i. e. d'une table), etc.)

Modèle de coût

Ensemble de formules permettant d'estimer le coût d'un plan d'exécution

Objectif

Minimiser le temps nécessaire à l'exécution du plan d'exécution, et donc optimiser simultanément le nombre d'entrées/sorties, le parallélisme entre entrées/sorties, la taille des tampons, le temps de l'unité centrale

Étapes

Compilation

Analyse syntaxique de la requête d'interrogation (en SQL) i. e. d'une chaîne de caractères de la forme `"SELECT ..."`

Contrôle des droits et des contraintes d'intégrité ;
Récupération des relations à partir des vues utilisées

Optimisation de la requête en générant le plan d'exécution, éventuellement stocké pour d'autres utilisations futures

Exécution

Algorithmes d'exécution des opérations d'accès utilisées par le plan d'exécution

Entrées/Sorties des fichiers découpées en pages ;
Gestion de la concurrence (pas de perte d'opération, cohérence entre transactions) ;
Fiabilité (journalisation pour assurer l'atomicité et la durabilité des transactions)

Génération du plan d'exécution (1/2)

Créer l'arbre de la représentation canonique de la requête d'interrogation à l'aide des opérateurs de l'algèbre relationnelle (sélection, jointure, union et différence, regroupement et tri, sélection après regroupement, projection, etc.) dont les feuilles sont les relations et les nœuds les opérateurs, l'évaluation au niveau de la racine correspondant au résultat

Effectuer un premier ordonnancement des opérateurs par restructuration algébrique, l'intérêt de cette heuristique d'optimisation qui consiste à descendre le plus bas possible les opérateurs unaires dans l'arbre étant de diminuer (en ligne et en colonne) le volume de données de la relation (des tuples grâce aux sélections et des attributs grâce aux projections)

Séparer les sélections comportant plusieurs prédicats [$\sigma_{\theta' \wedge \theta''}(R)$ devient $\sigma_{\theta'}(\sigma_{\theta''}(R))$]

Descendre les sélections le plus bas possible dans l'arbre grâce à la commutation de la sélection avec :

la projection [$\sigma(\pi(R)) = \pi(\sigma(R))$ si tous les attributs de la condition sont dans la projection]

la jointure [$\sigma(R \bowtie S) = \sigma(R) \bowtie \sigma(S)$]

l'union [$\sigma(R \cup S) = \sigma(R) \cup \sigma(S)$]

la différence [$\sigma(R \setminus S) = \sigma(R) \setminus \sigma(S)$]

Regrouper les sélections successives portant sur une même relation [$\sigma_{\theta'}(\sigma_{\theta''}(R))$ devient $\sigma_{\theta' \wedge \theta''}(R)$]

Descendre les projections le plus bas possible dans l'arbre grâce à la commutation de la projection avec :

la jointure [$\pi(R \bowtie S) = \pi(R) \bowtie \pi(S)$ si les attributs sont correctement choisis]

l'union [$\pi(R \cup S) = \pi(R) \cup \pi(S)$]

Regrouper les projections successives, voire en éliminer certaines

Exemple : requête d'interrogation SQL, représentation canonique et plan d'exécution après restructuration algébrique

```
-- les numéros et noms des étudiants de numéro 2, 4 ou 8 et
-- les couleurs de leurs voitures non orange
SELECT NoINE , NomEtu , Couleur
FROM Etudiants
NATURAL JOIN Voitures
WHERE NoINE IN (2,4,8) AND Couleur <> 'orange'
```

$\pi_{\text{NoINE,NomEtu,Couleur}}(\sigma_{\text{NoINE}\in\{2,4,8\} \wedge \text{Couleur} \neq \text{'orange'}}(\text{Etudiants} \bowtie \text{Voitures}))$

$\pi_{\text{NoINE,NomEtu}}(\sigma_{\text{NoINE}\in\{2,4,8\}}(\text{Etudiants})) \bowtie$
$\pi_{\text{NoINE,Couleur}}(\sigma_{\text{NoINE}\in\{2,4,8\} \wedge \text{Couleur} \neq \text{'orange'}}(\text{Voitures}))$

Choisir l'un des plans d'exécution (1/2)

Engendrer tous les plans d'exécution candidats (ou adopter une stratégie aléatoire d'exploration de l'espace des plans d'exécution) en permutant les opérateurs binaires (jointure, union, etc.) et regroupements et tris restants en choisissant pour chaque opérateur un algorithme d'exécution

Associer un coût estimé à chaque plan d'exécution basé sur un modèle de coût intégrant les structures et les volumes de données

Génération du plan d'exécution (2/2)

Choisir l'un des plans d'exécution (2/2)

Retenir le plan d'exécution de coût estimé minimal

Le modèle de coût prend généralement comme hypothèse que chaque relation vérifie l'uniformité des distributions des valeurs et l'indépendance des attributs

Supposons connus, pour chaque relation R et pour chaque attribut A :

$card(A)$ le nombre de valeurs de A

$ndist(A)$ le nombre de valeurs distinctes de A (et $\frac{1}{ndist(A)}$ est la densité de A)

$min(A)$ et $max(A)$ la valeur respectivement minimale et maximale de A

$card(R)$ le nombre de tuples de R

Alors :

$p(A = x) = \frac{1}{ndist(A)}$

$p(A \in \{x_1, x_2, \ldots, x_k\}) = \frac{k}{ndist(A)}$

$p(A > x) = \frac{max(A)-x}{max(A)-min(A)}$

$p(A < x) = \frac{x-min(A)}{max(A)-min(A)}$ N. B. : $= 1-p(A > x) \neq 1-p(A = x)-p(A > x)$

$p(A \geq x) = p(A > x) + p(A = x)$

$p(A \leq x) = p(A < x) + p(A = x)$

$p(A \in]x', x''[) = \frac{x''-x'}{max(A)-min(A)}$

$p(A \in [x', x'']) = p(A = x') + p(A \in]x', x''[) + p(A = x'')$

$p(P \wedge Q) = p(P).p(Q)$

$p(P \vee Q) = p(P) + p(Q) - p(P).p(Q)$

$p(\neg P) = 1 - p(P)$

$card(\sigma_\theta(R)) = p(\theta).card(R)$

$card(\pi_{A_1,A_2,\ldots,A_n}(R)) = (1 - d).card(R)$

avec la probabilité de doubles $d \leq \Pi_{i=1}^{n}(1 - \frac{ndist(A_i)}{card(R)})$

$card(R \bowtie S) = card(R \bowtie_{R.Z=S.Z} S) = p.card(R).card(S)$

avec $p \leq \frac{1}{\max\{ndist(R.Z),ndist(S.Z)\}}$

où $p(\triangleleft$ condition $\triangleright)$ est la probabilité que la condition soit vérifiée (facteur de sélectivité)

Exemple : $R = Etudiants$ et $A = Etudiants.NoINE$

$card(A) = 5$, $ndist(A) = 5$, $min(A) = 2$, $max(A) = 7$ et $card(R) = 5$

$p(A = 3) = p(A = 6) = \frac{1}{5}$, $p(A \in \{2, 3\}) = p(A \in \{2, 6\}) = \frac{2}{5}$

$p(A > 5) = \frac{7-5}{7-2} = \frac{2}{5}$ et $p(A > 6) = \frac{7-6}{7-2} = \frac{1}{5}$

Exemple : $R = Etudiants$ et $A = Etudiants.NomEtu$

$card(A) = 5$, $ndist(A) = 4$, $min(A) =' DUPOND'$, $max(A) =' MARTIN'$

$card(R) = 5$

$p(A =' DURAND') = p(A =' LEROI') = \frac{1}{4}$

$p(A \in \{'DUPOND','DURAND'\}) = p(A \in \{'DUPOND','LEROI'\}) = \frac{2}{4}$

Exemples :

$card(\sigma_{NoINE=3}(Etudiants)) = p(NoINE = 3).card(Etudiants) = \frac{1}{5}.5 = 1$

$card(\pi_{NoINE,NomEtu}(Etudiants)) = (1 - d).card(Etudiants) = (1 - d).5$

$d \leq (1 - \frac{ndist(NoINE)}{card(Etudiants)}).(1 - \frac{ndist(NomEtu)}{card(Etudiants)}) = (1 - \frac{5}{5}).(1 - \frac{4}{5}) = 0$

$card(\pi_{NomEtu,DepartNaissEtu}(Etudiants)) = (1-d).card(Etudiants) = (1-d).5$

$d \leq (1 - \frac{ndist(NomEtu)}{card(Etudiants)}).(1 - \frac{ndist(DepartNaissEtu)}{card(Etudiants)}) = (1 - \frac{4}{5}).(1 - \frac{4}{5}) = \frac{1}{25}$

$card(Etudiants \bowtie Voitures) = p.card(Etudiants).card(Voitures) = p.5.3$

$p \leq \frac{1}{\max\{ndist(Etudiants.NoINE),ndist(Voitures.NoINE)\}} = \frac{1}{\max\{5,2\}} = \frac{1}{5}$

Deux exemples (illustrés par les SGBD Oracle et SQL Server respectivement)

```
SELECT COUNT(*) NbDiplomes , AO.IntitAbrege , IntitComplet ,
       MAX(Annee) DernAnnee
FROM AvoirObtenu AO
NATURAL JOIN Diplomes
WHERE IntitAbrege <> 'DEUG' AND NoINE <> 2
GROUP BY AO.IntitAbrege , IntitComplet
HAVING MIN(Annee) < 1983
ORDER BY 4 ASC
```

```
-------------------------------------------------------------------------------
Arbre des opérations                                                      Coût
-------------------------------------------------------------------------------
SELECT                                                                    6 34%
 SORT ORDER BY MAX(Annee)                                                 6 34%
  FILTER MIN(Annee) < 1983
   HASH GROUP BY IntitAbrege , IntitComplet                               6 34%
    TABLE ACCESS BY INDEX ROWID AvoirObtenu , FILTER NoINE <> 2           1  0%
     NESTED LOOPS                                                         4  0%
      TABLE ACCESS FULL Diplomes , FILTER IntitAbrege <> 'DEUG'           3  0%
      INDEX RANGE SCAN FK_AvoirObtenu_REF_Diplomes ,
      ... ACCESS AvoirObtenu.IntitAbrege = Diplomes.IntitAbrege           0  0%
-------------------------------------------------------------------------------
SELECT                                                                      0%
 SORT ORDER BY MAX(Annee)                                                  51%
  FILTER MIN(Annee) < 1983                                                  0%
   NESTED LOOPS , INNER JOIN                                                0%
    COMPUTE SCALAR , OUTPUT COUNT(*)                                        0%
     GROUP BY IntitAbrege , OUTPUT MAX(Annee) MIN(Annee)                    0%
      NESTED LOOPS , INNER JOIN                                             0%
       INDEX SEEK FK_AvoirObtenu_REF_Diplomes ,
       ... FILTER IntitAbrege <> 'DEUG' AND NoINE <> 2                     15%
       "KEY SEEK" PK_AvoirObtenu                                           18%
    CLUSTERED INDEX SCAN PK_Diplomes ,
    ... ACCESS AvoirObtenu.IntitAbrege = Diplomes.IntitAbrege              16%
-------------------------------------------------------------------------------
```

```
SELECT Etudiants.NoINE , NomEtu ,
       NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
EXCEPT
SELECT Etudiants.NoINE , NomEtu ,
       NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
NATURAL JOIN Voitures
NATURAL JOIN AvoirObtenu
WHERE IntitAbrege IN (
    SELECT IntitAbrege
    FROM AvoirObtenu
    WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4) )
ORDER BY Etudiants.NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
```

```
-------------------------------------------------------------------------------------
Arbre des opérations                                                          Coût %
-------------------------------------------------------------------------------------
SELECT                                                                          11 28%
 SORT ORDER BY                                                                  11 28%
  EXCEPT
   SORT UNIQUE                                                                   5 20%
    NESTED LOOPS OUTER                                                           4  0%
     VIEW index$_join$_001                                                       3  0%
      HASH JOIN , ACCESS ROWID=ROWID
       INDEX FAST FULL SCAN PK_Etudiants                                         1  0%
       INDEX FAST FULL SCAN Index_NomEtu                                         1  0%
     TABLE ACCESS BY INDEX ROWID Voitures                                        1  0%
      INDEX RANGE SCAN FK_Voitures_REF_Etudiants ,
      ... ACCESS Etudiants.NoINE = Voitures.NoINE (+)                            0  0%
   SORT UNIQUE                                                                   5 20%
    NESTED LOOPS                                                                 4  0%
     NESTED LOOPS                                                                3  0%
      NESTED LOOPS                                                               2  0%
       INDEX FULL SCAN PK_AvoirObtenu ,
       ... FILTER IntitAbrege <> 'BAC' AND NoINE <> 2 AND NoINE <> 4             1  0%
       TABLE ACCESS BY INDEX ROWID AvoirObtenu                                   1  0%
        INDEX RANGE SCAN FK_AvoirObtenu_REF_Diplomes ,
        ... ACCESS AvoirObtenu.IntitAbrege = Diplomes.IntitAbrege                0  0%
      TABLE ACCESS BY INDEX ROWID Voitures                                       1  0%
       INDEX RANGE SCAN FK_Voitures_REF_Etudiants ,
       ... ACCESS Voitures.NoINE = AvoirObtenu.NoINE                             0  0%
     TABLE ACCESS BY INDEX ROWID Etudiants                                       1  0%
      INDEX UNIQUE SCAN PK_Etudiants , ACCESS Etudiants.NoINE = Voitures.NoINE   0  0%
-------------------------------------------------------------------------------------
SELECT                                                                             0%
 NESTED LOOPS , LEFT ANTI SEMI JOIN , OUTPUT Etudiants.NoINE NomEtu NoImmat        0%
  NESTED LOOPS , LEFT OUTER JOIN Etudiants.NoINE ,
  ... OUTPUT Etudiants.NoINE NomEtu NoImmat                                        0%
   CLUSTERED INDEX SCAN PK_Etudiants , OUTPUT NomEtu                              14%
   INDEX SEEK FK_Voitures_REF_Etudiants ,
   ... FILTER Voitures.NoINE = Etudiants.NoINE , OUTPUT NoImmat                   17%
  TOP 1                                                                            0%
   GROUP BY AvoirObtenu.IntitAbrege                                                0%
    NESTED LOOPS , INNER JOIN , OUTPUT AvoirObtenu.IntitAbrege                     1%
     NESTED LOOPS , INNER JOIN , OUTPUT AvoirObtenu.IntitAbrege                    0%
      NESTED LOOPS , INNER JOIN                                                    0%
       CLUSTERED INDEX SCAN PK_Etudiants , OUTPUT NomEtu                          18%
       CLUSTERED INDEX SCAN PK_Voitures ,
       ... FILTER Voitures.NoINE = Etudiants.NoINE , OUTPUT Voitures.NoINE        18%
      INDEX SEEK FK_AvoirObtenu_REF_Etudiants ,
      ... FILTER AvoirObtenu.NoINE = Etudiants.NoINE ,
      ... OUTPUT AvoirObtenu.IntitAbrege                                         18%
     CLUSTERED INDEX SCAN PK_AvoirObtenu ,
     ... FILTER IntitAbrege <> 'BAC' AND NoINE <> 2 AND NoINE <> 4 ,
     ... OUTPUT IntitAbrege                                                      15%
-------------------------------------------------------------------------------------
```

Le premier exemple

```
SQL> SELECT COUNT(*) NbDiplomes , IntitAbrege , IntitComplet , MAX(Annee) DernAnnee
  2  FROM AvoirObtenu
  3  NATURAL JOIN Diplomes
  4  WHERE IntitAbrege <> 'DEUG' AND NoINE <> 2
  5  GROUP BY IntitAbrege , IntitComplet
  6  HAVING MIN(Annee) < 1983
  7  ORDER BY 4 ASC
  8  ;
```

```
Plan d'exécution
----------------------------------------------------------
Plan hash value: 575000534

-------------------------------------------------------------------------------------
| Id  | Operation             | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |              |     5 |   410 |     9  (34)| 00:00:01 |
|   1 |  SORT ORDER BY        |              |     5 |   410 |     9  (34)| 00:00:01 |
|*  2 |   FILTER              |              |       |       |            |          |
|   3 |    HASH GROUP BY      |              |     5 |   410 |     9  (34)| 00:00:01 |
|*  4 |     HASH JOIN         |              |     5 |   410 |     7  (15)| 00:00:01 |
|*  5 |      TABLE ACCESS FULL| DIPLOMES     |     3 |   147 |     3   (0)| 00:00:01 |
|*  6 |      TABLE ACCESS FULL| AVOIROBTENU  |     7 |   231 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(MIN("AVOIROBTENU"."ANNEE")<1983)
   4 - access("AVOIROBTENU"."INTITABREGE"="DIPLOMES"."INTITABREGE")
   5 - filter("DIPLOMES"."INTITABREGE"<>'DEUG')
   6 - filter("AVOIROBTENU"."NOINE"<>2 AND
              "AVOIROBTENU"."INTITABREGE"<>'DEUG')

Note
-----
   - dynamic sampling used for this statement


Statistiques
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         14  consistent gets
          0  physical reads
          0  redo size
        758  bytes sent via SQL*Net to client
        420  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

The cost is an estimated value proportional to the expected resource use needed to execute the statement with a particular plan. The optimizer calculates the cost of access paths and join orders based on the estimated computer resources, which includes I/O, CPU, and memory. (cf. documentation *Oracle Database Performance Tuning Guide*)

Statistiques (cf. documentation *SQL*Plus User's Guide and Reference*)

recursive calls : nombre de requêtes SQL générées en plus de celles des processus utilisateurs

Exemples : défaut du cache du dictionnaire des données, lancement des déclencheurs, ordres du LDD, déclarations SQL dans les procédures/fonctions/paquetages/blocs PL/SQL anonymes, exécution de contraintes d'intégrité référentielles

db block et *consistent gets* : nombre total de blocs lus (y compris en mémoire principale)

physical reads : nombre de blocs lus sur disque

redo size : nombre d'octets générés par les *redo* (pour le journal)

bytes sent/received : nombre d'octets envoyés/reçus du serveur au client sur le réseau

round-trips to/from : nombre d'allers-retours entre le serveur et le client via le réseau

sorts (memory/disk) : nombre de tris entièrement effectués en mémoire principale/ayant nécessité au moins une écriture sur le disque

rows processed : nombre de lignes manipulées

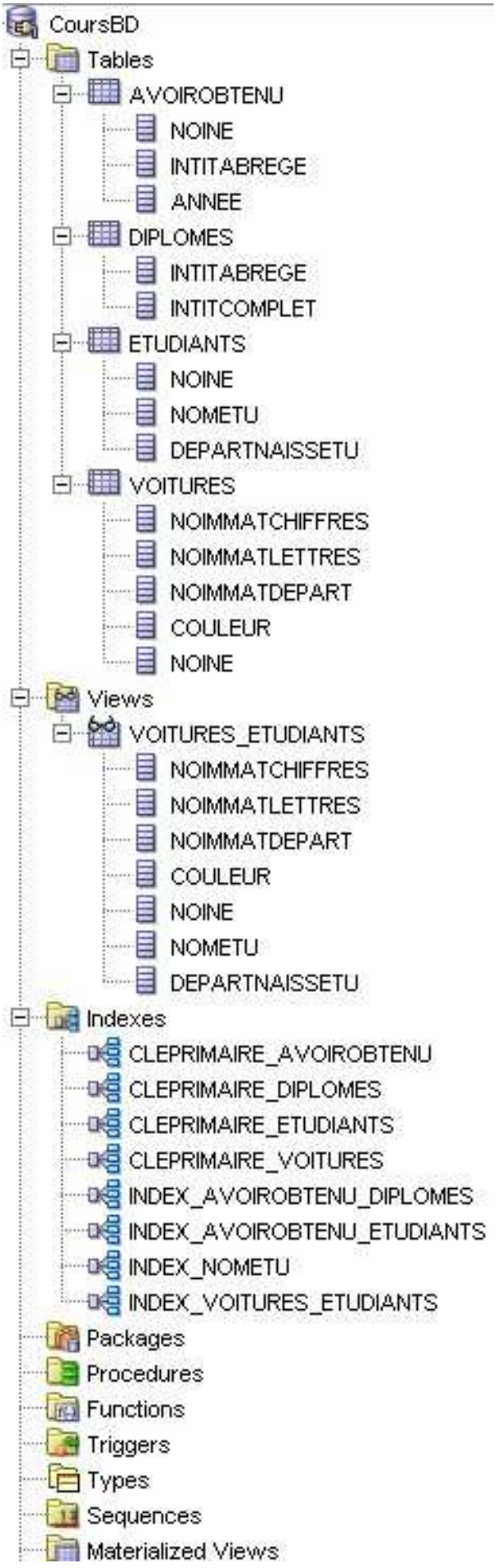
Le second exemple

```
SQL> SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
  2  FROM Etudiants
  3  LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
  4  MINUS
  5  SELECT NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
  6  FROM Etudiants
  7  NATURAL JOIN Voitures
  8  NATURAL JOIN AvoirObtenu
  9  WHERE IntitAbrege IN ( SELECT IntitAbrege
 10                         FROM AvoirObtenu
 11                         WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4 ) )
 12  ORDER BY NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
 13  ;
```

```
Plan d'exécution
----------------------------------------------------------
Plan hash value: 147499272

-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                           |     5 |   672 |     19  (22)| 00:00:01 |
|   1 |  SORT ORDER BY                 |                           |     5 |   672 |     19  (22)| 00:00:01 |
|   2 |   MINUS                        |                           |       |       |             |          |
|   3 |    SORT UNIQUE                 |                           |     5 |   345 |      8  (25)| 00:00:01 |
|*  4 |     HASH JOIN OUTER            |                           |     5 |   345 |      7  (15)| 00:00:01 |
|   5 |      TABLE ACCESS FULL         | ETUDIANTS                 |     5 |   125 |      3   (0)| 00:00:01 |
|   6 |      TABLE ACCESS FULL         | VOITURES                  |     3 |   132 |      3   (0)| 00:00:01 |
|   7 |    SORT UNIQUE                 |                           |     3 |   327 |     10  (10)| 00:00:01 |
|   8 |     NESTED LOOPS               |                           |     3 |   327 |      9   (0)| 00:00:01 |
|   9 |      MERGE JOIN CARTESIAN      |                           |     9 |   801 |      9   (0)| 00:00:01 |
|  10 |       NESTED LOOPS             |                           |       |       |             |          |
|  11 |        NESTED LOOPS            |                           |     3 |   207 |      6   (0)| 00:00:01 |
|  12 |         TABLE ACCESS FULL      | VOITURES                  |     3 |   132 |      3   (0)| 00:00:01 |
|* 13 |         INDEX UNIQUE SCAN      | CLEPRIMAIRE_ETUDIANTS     |     1 |       |      0   (0)| 00:00:01 |
|  14 |        TABLE ACCESS BY INDEX ROWID| ETUDIANTS                 |     1 |    25 |      1   (0)| 00:00:01 |
|  15 |       BUFFER SORT              |                           |     3 |    60 |      8   (0)| 00:00:01 |
|* 16 |        INDEX FAST FULL SCAN    | CLEPRIMAIRE_AVOIROBTENU   |     3 |    60 |      1   (0)| 00:00:01 |
|* 17 |      INDEX UNIQUE SCAN         | CLEPRIMAIRE_AVOIROBTENU   |     1 |    20 |      0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("ETUDIANTS"."NOINE"="VOITURES"."NOINE"(+))
  13 - access("ETUDIANTS"."NOINE"="VOITURES"."NOINE")
  16 - filter("NOINE"<>2 AND "NOINE"<>4 AND "INTITABREGE"<>'BAC')
  17 - access("VOITURES"."NOINE"="AVOIROBTENU"."NOINE" AND "AVOIROBTENU"."INTITABREGE"="INTITABREGE")
       filter("AVOIROBTENU"."INTITABREGE"<>'BAC')

Note
-----
   - dynamic sampling used for this statement


Statistiques
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         33  consistent gets
          0  physical reads
          0  redo size
        740  bytes sent via SQL*Net to client
        420  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          4  sorts (memory)
          0  sorts (disk)
          3  rows processed
```

Plan exécution Oracle [SQL Developer] (1/3) 220

La base de données



Plan exécution Oracle [SQL Developer] (2/3) 221

Le premier exemple

```
SELECT COUNT(*) NbDiplomes , IntitAbrege , IntitComplet , MAX(Annee) DernAnnee
FROM AvoirObtenu
NATURAL JOIN Diplomes
WHERE IntitAbrege <> 'DEUG' AND NoINE <> 2
GROUP BY IntitAbrege , IntitComplet
HAVING MIN(Annee) < 1983
ORDER BY 4 ASC
```

Operation	Optimizer	Cost	Cardinality	...	...	...	...
SELECT STATEMENT	ALL_ROWS	9	5	...			
SORT(ORDER BY)		9	5	...			
FILTER							
HASH(GROUP BY)		9	5	...			
HASH JOIN		7	5	...			
TABLE ACCESS(FULL) GUIBERT.DIPLOMES		3	3	...			
TABLE ACCESS(FULL) GUIBERT.AVOIROBTENU		3	7	...			

ACCESS PREDICATES	FILTER PREDICATES
	MIN("AVOIROBTENU"."ANNEE")<1983
"AVOIROBTENU"."INTITABREGE"="DIPLOMES"."INTITABREGE"	
	"DIPLOMES"."INTITABREGE"<>'DEUG'
	"AVOIROBTENU"."NOINE"<>2 AND "AVOIROBTENU"."INTITABREGE"<>'DEUG'

Plan exécution Oracle [SQL Developer] (3/3) 222

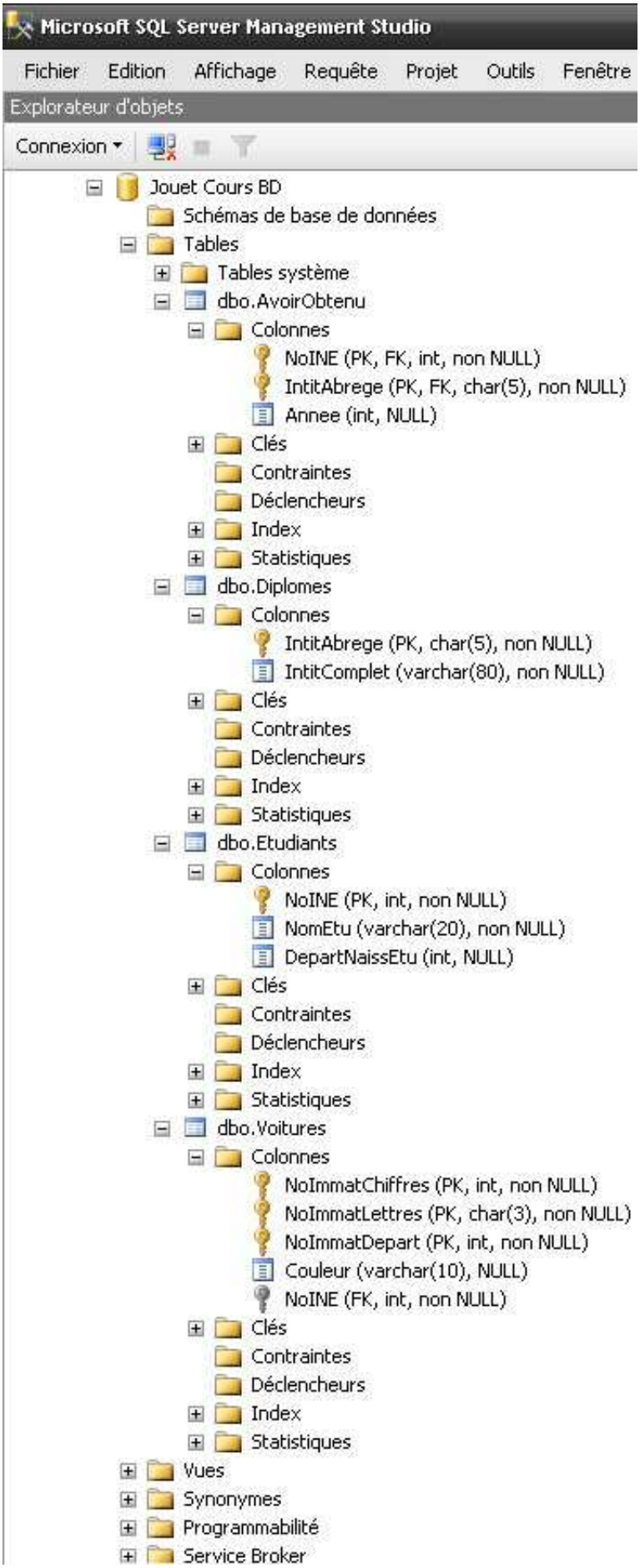
Le second exemple

```
SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
MINUS
SELECT NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
NATURAL JOIN Voitures
NATURAL JOIN AvoirObtenu
WHERE IntitAbrege IN ( SELECT IntitAbrege
                       FROM AvoirObtenu
                       WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4 ) )
ORDER BY NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
```

Operation	Optimizer	Cost	Cardinality	Bytes	Bytes
SELECT STATEMENT	ALL_ROWS	19	5	672	672
SORT(ORDER BY)		19	5	672	672
MINUS					
SORT(UNIQUE)		8	5	345	345
HASH JOIN(OUTER)		7	5	345	345
TABLE ACCESS(FULL) GUIBERT.ETUDIANTS		3	5	125	125
TABLE ACCESS(FULL) GUIBERT.VOITURES		3	3	132	132
SORT(UNIQUE)		10	3	327	327
NESTED LOOPS		9	3	327	327
MERGE JOIN(CARTESIAN)		9	9	801	801
NESTED LOOPS					
NESTED LOOPS		6	3	207	207
TABLE ACCESS(FULL) GUIBERT.VOITURES		3	3	132	132
INDEX(UNIQUE SCAN) GUIBERT.CLEPRIMAIRE_ETUDIANTS		0	1		
TABLE ACCESS(BY INDEX ROWID) GUIBERT.ETUDIANTS		1	1	25	25
BUFFER(SORT)		8	3	60	60
INDEX(FAST FULL SCAN) GUIBERT.CLEPRIMAIRE_AVOIROBTENU		1	3	60	60
INDEX(UNIQUE SCAN) GUIBERT.CLEPRIMAIRE_AVOIROBTENU		0	1	20	20

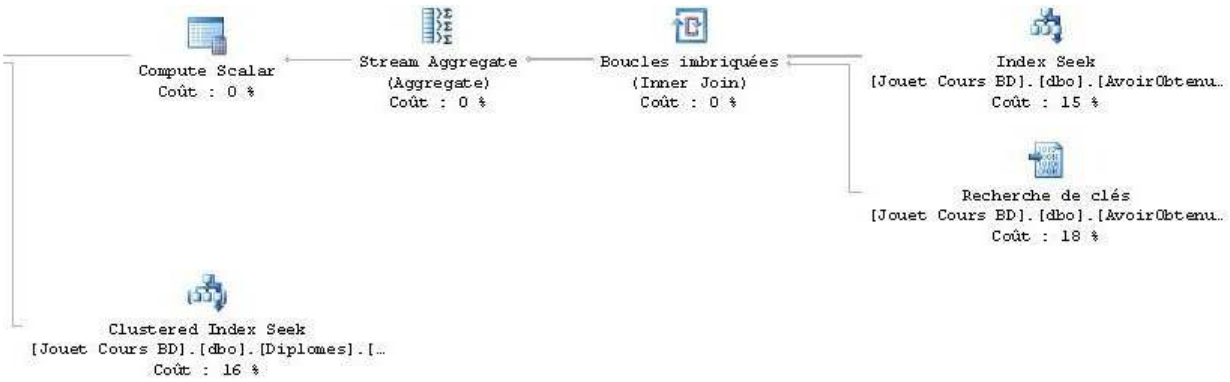
ACCESS PREDICATES	FILTER PREDICATES
"ETUDIANTS"."NOINE"="VOITURES"."NOINE"(+)	
"ETUDIANTS"."NOINE"="VOITURES"."NOINE"	"NOINE"<>2 AND "NOINE"<>4 AND "INTITABREGE"<>'BAC'
"VOITURES"."NOINE"="AVOIROBTENU"."NOINE" AND "AVOIROBTENU"."INTITABREGE"="INTITABREGE"	"AVOIROBTENU"."INTITABREGE"<>'BAC'

La base de données



Le premier exemple (1/3)

```
SELECT COUNT(*) NbDiplomes , AO.IntitAbrege , IntitComplet , MAX(Annee) DernAnnee
FROM AvoirObtenu AO
JOIN Diplomes ON AO.IntitAbrege = Diplomes.IntitAbrege
WHERE AO.IntitAbrege <> 'DEUG' AND NoINE <> 2
GROUP BY AO.IntitAbrege , IntitComplet
HAVING MIN(Annee) < 1983
ORDER BY 4 ASC
```



SELECT

Cached plan size	26 O
Estimated Operator Cost	0 (0 %)
Estimated Subtree Cost	0,0224469
Estimated Number of Rows	2,5929

Instruction

SELECT COUNT(*) NbDiplomes ,
AO.IntitAbrege , IntitComplet , MAX(Annee)
DernAnnee
FROM AvoirObtenu AO
JOIN Diplomes ON AO.IntitAbrege =
Diplomes.IntitAbrege
WHERE AO.IntitAbrege <> 'DEUG' AND
NoINE <> 2
GROUP BY AO.IntitAbrege , IntitComplet
HAVING MIN(Annee) < 1983
ORDER BY 4 ASC

Trier

Trie l'entrée.

Physical Operation	Trier
Opération logique	Trier
Estimated I/O Cost	0,0112613
Estimated CPU Cost	0,0001056
Estimated Operator Cost	0,0113668 (51 %)
Estimated Subtree Cost	0,0224469
Estimated Number of Rows	2,5929
Estimated Row Size	64 O
ID du nœud	0

Output List

[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege;
[Jouet Cours BD].[dbo].[Diplomes].IntitComplet;
Expr1006; Expr1007

Order By

Expr1007 Croissant

Le premier exemple (2/3)

Filtre

Restriction du jeu de lignes fondée sur un prédicat.

Physical Operation	Filtre
Opération logique	Filtre
Estimated I/O Cost	0
Estimated CPU Cost	0,0000014
Estimated Operator Cost	0,0000015 (0 %)
Estimated Subtree Cost	0,0110801
Estimated Number of Rows	2,5929
Estimated Row Size	64 O
ID du nœud	1

Predicate

[Expr1005]<(1983)

Output List

[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege; [Jouet Cours BD].[dbo].[Diplomes].IntitComplet; Expr1006; Expr1007

Boucles imbriquées

Pour chaque ligne dans l'entrée supérieure (extérieure), cette fonction analyse l'entrée inférieure (intérieure) et génère les lignes correspondantes.

Physical Operation	Boucles imbriquées
Opération logique	Inner Join
Estimated I/O Cost	0
Estimated CPU Cost	0,0000125
Estimated Operator Cost	0,0000125 (0 %)
Estimated Subtree Cost	0,0110786
Estimated Number of Rows	3
Estimated Row Size	68 O
ID du nœud	3

Output List

[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege; [Jouet Cours BD].[dbo].[Diplomes].IntitComplet; Expr1005; Expr1006; Expr1007

Outer References

[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

Compute Scalar

Calcul de nouvelles valeurs à partir de valeurs existantes dans une ligne.

Physical Operation	Compute Scalar
Opération logique	Compute Scalar
Estimated I/O Cost	0
Estimated CPU Cost	0,0000053
Estimated Operator Cost	0 (0 %)
Estimated Subtree Cost	0,0074668
Estimated Number of Rows	3
Estimated Row Size	24 O
ID du nœud	4

Output List

[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege; Expr1005; Expr1006; Expr1007

Stream Aggregate

Calcule les valeurs résumées pour les groupes de lignes dans un flux trié.

Physical Operation	Stream Aggregate
Opération logique	Aggregate
Estimated I/O Cost	0
Estimated CPU Cost	0,0000053
Estimated Operator Cost	0,0000053 (0 %)
Estimated Subtree Cost	0,0074668
Estimated Number of Rows	3
Estimated Row Size	24 O
ID du nœud	5

Output List

[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege; Expr1005; Expr1007; Expr1012

Regrouper par

[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

Le premier exemple (3/3)

Boucles imbriquées

Pour chaque ligne dans l'entrée supérieure (extérieure), cette fonction analyse l'entrée inférieure (intérieure) et génère les lignes correspondantes.

Physical Operation	Boucles imbriquées
Opération logique	Inner Join
Estimated I/O Cost	0
Estimated CPU Cost	0,0000268
Estimated Operator Cost	0,0000338 (0 %)
Estimated Subtree Cost	0,0074614
Estimated Number of Rows	6,4
Estimated Row Size	16 O
ID du nœud	6

Output List
[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege; [Jouet Cours BD].[dbo].[AvoirObtenu].Annee

Outer References
[Jouet Cours BD].[dbo].[AvoirObtenu].NoINE; [Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

Index Seek

Analyse une plage de lignes particulière à partir d'un index non cluster.

Physical Operation	Index Seek
Opération logique	Index Seek
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001658
Estimated Operator Cost	0,0032908 (15 %)
Estimated Subtree Cost	0,0032908
Estimated Number of Rows	6,4
Estimated Row Size	16 O
Ordered	True
ID du nœud	7

Predicate
[Jouet Cours BD].[dbo].[AvoirObtenu].[NoINE] as [AO].[NoINE]<(2) OR [Jouet Cours BD].[dbo].[AvoirObtenu].[NoINE] as [AO].[NoINE]>(2)

Objet
[Jouet Cours BD].[dbo].[AvoirObtenu].[Index_AvoirObtenu_Diplomes] [AO]

Output List
[Jouet Cours BD].[dbo].[AvoirObtenu].NoINE; [Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

Seek Predicates
Plage de fin : [Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege < Opérateur scalaire ('DEUG'); Plage de début : [Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege > Opérateur scalaire ('DEUG')

Recherche de clés

Utilise une clé de clustering fournie pour faire une recherche sur une table qui possède un index cluster.

Physical Operation	Recherche de clés
Opération logique	Recherche de clés
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001581
Estimated Operator Cost	0,0041368 (18 %)
Estimated Subtree Cost	0,0041368
Estimated Number of Rows	1
Estimated Row Size	11 O
Ordered	True
ID du nœud	9

Objet
[Jouet Cours BD].[dbo].[AvoirObtenu].[ClePrimaire_AvoirObtenu] [AO]

Output List
[Jouet Cours BD].[dbo].[AvoirObtenu].Annee

Seek Predicates
Préfixe : [Jouet Cours BD].[dbo].[AvoirObtenu].NoINE; [Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege = Opérateur scalaire ([Jouet Cours BD].[dbo].[AvoirObtenu].[NoINE] as [AO].[NoINE]); Opérateur scalaire([Jouet Cours BD].[dbo].[AvoirObtenu].[IntitAbrege] as [AO].[IntitAbrege])

Clustered Index Seek

Analyse d'une plage de lignes particulière à partir d'un index cluster.

Physical Operation	Clustered Index Seek
Opération logique	Clustered Index Seek
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001581
Estimated Operator Cost	0,0035993 (16 %)
Estimated Subtree Cost	0,0035993
Estimated Number of Rows	1
Estimated Row Size	51 O
Ordered	True
ID du nœud	21

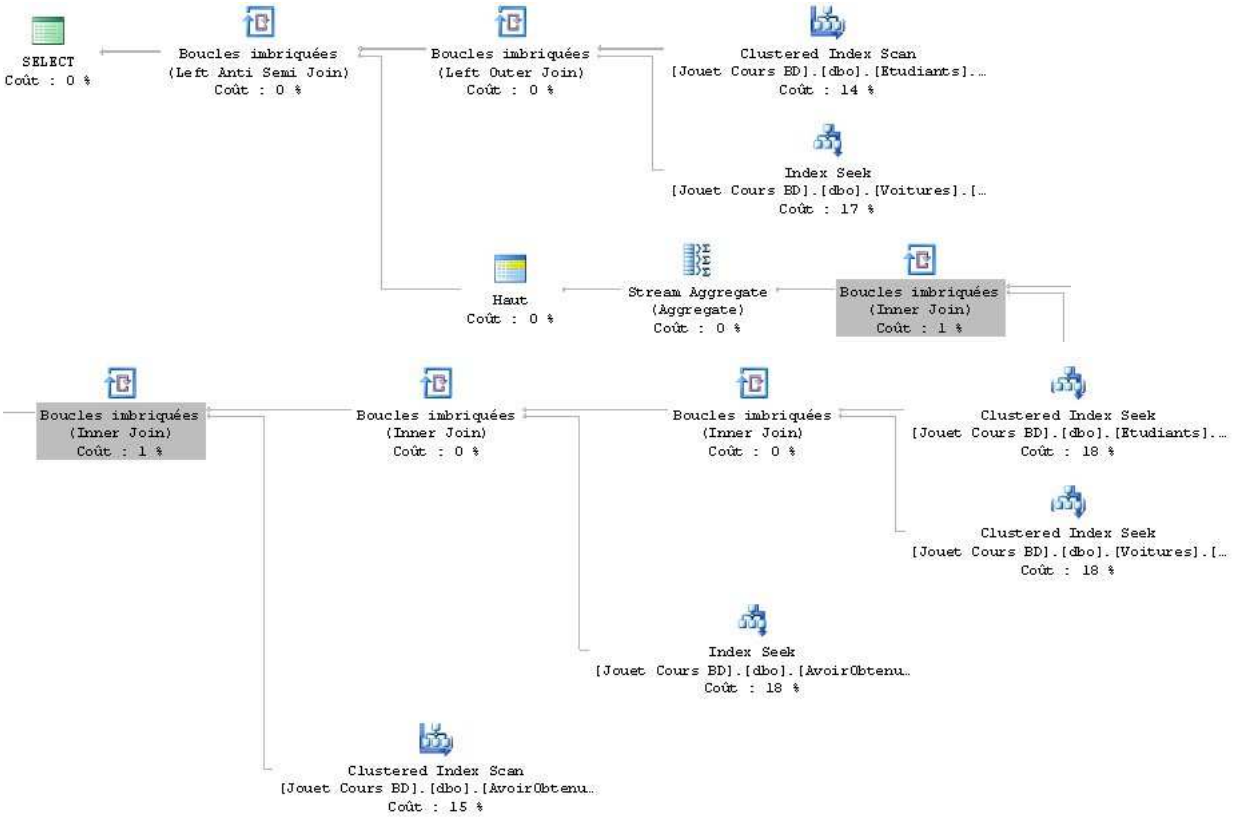
Objet
[Jouet Cours BD].[dbo].[Diplomes].[ClePrimaire_Diplomes]

Output List
[Jouet Cours BD].[dbo].[Diplomes].IntitComplet

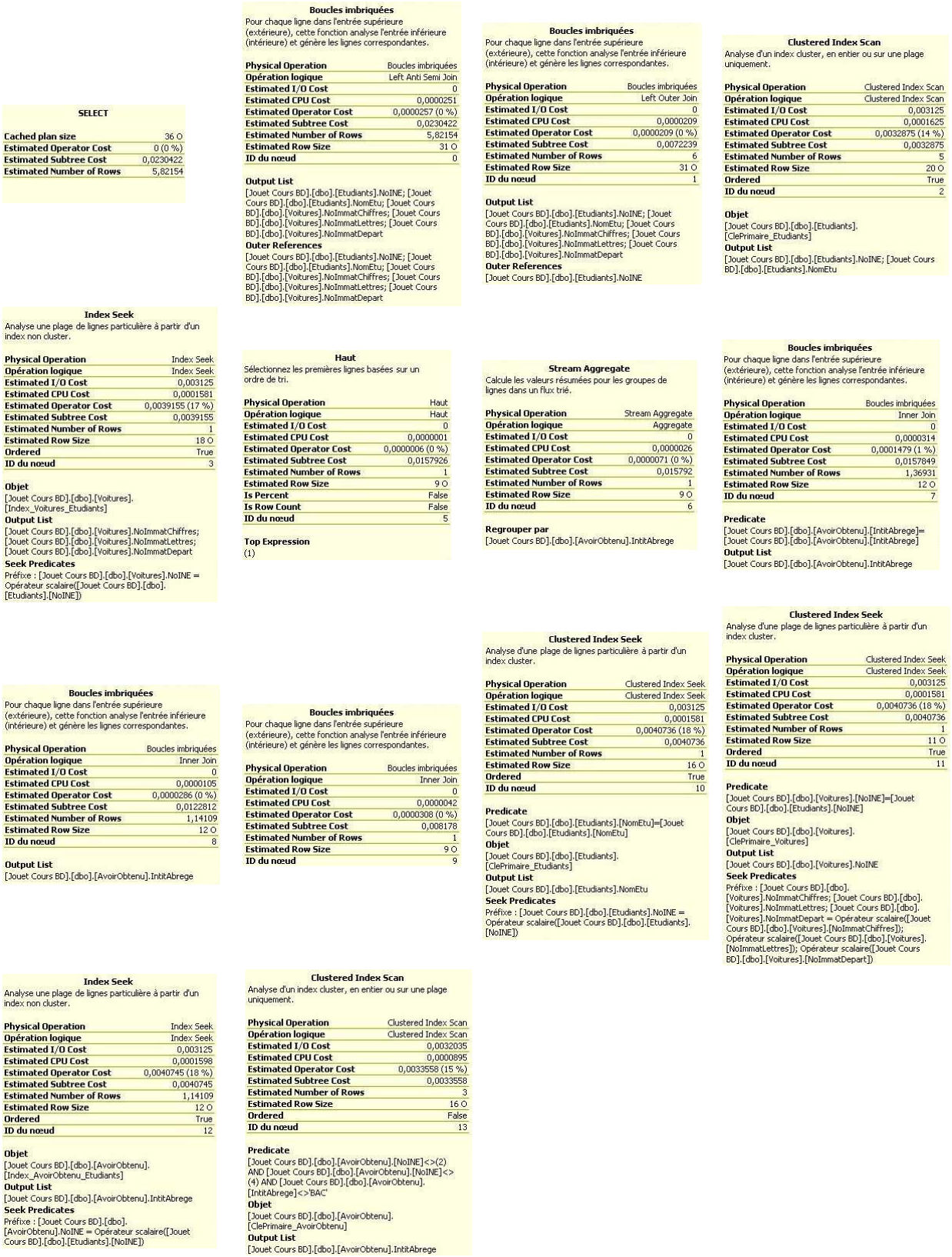
Seek Predicates
Préfixe : [Jouet Cours BD].[dbo].[Diplomes].IntitAbrege = Opérateur scalaire([Jouet Cours BD].[dbo].[AvoirObtenu].[IntitAbrege] as [AO].[IntitAbrege])

Le second exemple (1/2)

```
SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
LEFT OUTER JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
EXCEPT
SELECT Etudiants.NoINE , NomEtu , NoImmatChiffres , NoImmatLettres , NoImmatDepart
FROM Etudiants
JOIN Voitures ON Etudiants.NoINE = Voitures.NoINE
JOIN AvoirObtenu ON Etudiants.NoINE = AvoirObtenu.NoINE
WHERE IntitAbrege IN ( SELECT IntitAbrege
                       FROM AvoirObtenu
                       WHERE IntitAbrege <> 'BAC' AND NoINE NOT IN ( 2 , 4) )
ORDER BY Etudiants.NoINE , NoImmatChiffres , NoImmatLettres , NoImmatDepart
```



Le second exemple (2/2)



SELECT

Cached plan size	36 O
Estimated Operator Cost	0 (0 %)
Estimated Subtree Cost	0,0230422
Estimated Number of Rows	5,82154

Boucles imbriquées
Pour chaque ligne dans l'entrée supérieure (extérieure), cette fonction analyse l'entrée inférieure (intérieure) et génère les lignes correspondantes.

Physical Operation	Boucles imbriquées
Opération logique	Left Anti Semi Join
Estimated I/O Cost	0
Estimated CPU Cost	0,0000251
Estimated Operator Cost	0,0000257 (0 %)
Estimated Subtree Cost	0,0230422
Estimated Number of Rows	5,82154
Estimated Row Size	31 O
ID du nœud	0

Output List
[Jouet Cours BD].[dbo].[Etudiants].NoINE; [Jouet Cours BD].[dbo].[Etudiants].NomEtu; [Jouet Cours BD].[dbo].[Voitures].NoImmatChiffres; [Jouet Cours BD].[dbo].[Voitures].NoImmatLettres; [Jouet Cours BD].[dbo].[Voitures].NoImmatDepart
Outer References
[Jouet Cours BD].[dbo].[Etudiants].NoINE; [Jouet Cours BD].[dbo].[Etudiants].NomEtu; [Jouet Cours BD].[dbo].[Voitures].NoImmatChiffres; [Jouet Cours BD].[dbo].[Voitures].NoImmatLettres; [Jouet Cours BD].[dbo].[Voitures].NoImmatDepart

Boucles imbriquées
Pour chaque ligne dans l'entrée supérieure (extérieure), cette fonction analyse l'entrée inférieure (intérieure) et génère les lignes correspondantes.

Physical Operation	Boucles imbriquées
Opération logique	Left Outer Join
Estimated I/O Cost	0
Estimated CPU Cost	0,0000209
Estimated Operator Cost	0,0000209 (0 %)
Estimated Subtree Cost	0,0072239
Estimated Number of Rows	6
Estimated Row Size	31 O
ID du nœud	1

Output List
[Jouet Cours BD].[dbo].[Etudiants].NoINE; [Jouet Cours BD].[dbo].[Etudiants].NomEtu; [Jouet Cours BD].[dbo].[Voitures].NoImmatChiffres; [Jouet Cours BD].[dbo].[Voitures].NoImmatLettres; [Jouet Cours BD].[dbo].[Voitures].NoImmatDepart
Outer References
[Jouet Cours BD].[dbo].[Etudiants].NoINE

Clustered Index Scan
Analyse d'un index cluster, en entier ou sur une plage uniquement.

Physical Operation	Clustered Index Scan
Opération logique	Clustered Index Scan
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001625
Estimated Operator Cost	0,0032875 (14 %)
Estimated Subtree Cost	0,0032875
Estimated Number of Rows	5
Estimated Row Size	20 O
Ordered	True
ID du nœud	2

Objet
[Jouet Cours BD].[dbo].[Etudiants].[ClePrimaire_Etudiants]
Output List
[Jouet Cours BD].[dbo].[Etudiants].NoINE; [Jouet Cours BD].[dbo].[Etudiants].NomEtu

Index Seek
Analyse une plage de lignes particulière à partir d'un index non cluster.

Physical Operation	Index Seek
Opération logique	Index Seek
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001581
Estimated Operator Cost	0,0039155 (17 %)
Estimated Subtree Cost	0,0039155
Estimated Number of Rows	1
Estimated Row Size	18 O
Ordered	True
ID du nœud	3

Objet
[Jouet Cours BD].[dbo].[Voitures].[Index_Voitures_Etudiants]
Output List
[Jouet Cours BD].[dbo].[Voitures].NoImmatChiffres; [Jouet Cours BD].[dbo].[Voitures].NoImmatLettres; [Jouet Cours BD].[dbo].[Voitures].NoImmatDepart
Seek Predicates
Préfixe : [Jouet Cours BD].[dbo].[Voitures].NoINE = Opérateur scalaire([Jouet Cours BD].[dbo].[Etudiants].[NoINE])

Haut
Sélectionnez les premières lignes basées sur un ordre de tri.

Physical Operation	Haut
Opération logique	Haut
Estimated I/O Cost	0
Estimated CPU Cost	0,0000001
Estimated Operator Cost	0,0000006 (0 %)
Estimated Subtree Cost	0,0157926
Estimated Number of Rows	1
Estimated Row Size	9 O
Is Percent	False
Is Row Count	False
ID du nœud	5

Top Expression
(1)

Stream Aggregate
Calcule les valeurs résumées pour les groupes de lignes dans un flux trié.

Physical Operation	Stream Aggregate
Opération logique	Aggregate
Estimated I/O Cost	0
Estimated CPU Cost	0,0000026
Estimated Operator Cost	0,0000071 (0 %)
Estimated Subtree Cost	0,015792
Estimated Number of Rows	1
Estimated Row Size	9 O
ID du nœud	6

Regrouper par
[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

Boucles imbriquées
Pour chaque ligne dans l'entrée supérieure (extérieure), cette fonction analyse l'entrée inférieure (intérieure) et génère les lignes correspondantes.

Physical Operation	Boucles imbriquées
Opération logique	Inner Join
Estimated I/O Cost	0
Estimated CPU Cost	0,0000314
Estimated Operator Cost	0,0001479 (1 %)
Estimated Subtree Cost	0,0157849
Estimated Number of Rows	1,36931
Estimated Row Size	12 O
ID du nœud	7

Predicate
[Jouet Cours BD].[dbo].[AvoirObtenu].[IntitAbrege]= [Jouet Cours BD].[dbo].[AvoirObtenu].[IntitAbrege]
Output List
[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

Boucles imbriquées
Pour chaque ligne dans l'entrée supérieure (extérieure), cette fonction analyse l'entrée inférieure (intérieure) et génère les lignes correspondantes.

Physical Operation	Boucles imbriquées
Opération logique	Inner Join
Estimated I/O Cost	0
Estimated CPU Cost	0,0000105
Estimated Operator Cost	0,0000286 (0 %)
Estimated Subtree Cost	0,0122812
Estimated Number of Rows	1,14109
Estimated Row Size	12 O
ID du nœud	8

Output List
[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

Boucles imbriquées
Pour chaque ligne dans l'entrée supérieure (extérieure), cette fonction analyse l'entrée inférieure (intérieure) et génère les lignes correspondantes.

Physical Operation	Boucles imbriquées
Opération logique	Inner Join
Estimated I/O Cost	0
Estimated CPU Cost	0,0000042
Estimated Operator Cost	0,0000308 (0 %)
Estimated Subtree Cost	0,008178
Estimated Number of Rows	1
Estimated Row Size	9 O
ID du nœud	9

Clustered Index Seek
Analyse d'une plage de lignes particulière à partir d'un index cluster.

Physical Operation	Clustered Index Seek
Opération logique	Clustered Index Seek
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001581
Estimated Operator Cost	0,0040736 (18 %)
Estimated Subtree Cost	0,0040736
Estimated Number of Rows	1
Estimated Row Size	16 O
Ordered	True
ID du nœud	10

Predicate
[Jouet Cours BD].[dbo].[Etudiants].[NomEtu]=[Jouet Cours BD].[dbo].[Etudiants].[NomEtu]
Objet
[Jouet Cours BD].[dbo].[Etudiants].[ClePrimaire_Etudiants]
Output List
[Jouet Cours BD].[dbo].[Etudiants].NomEtu
Seek Predicates
Préfixe : [Jouet Cours BD].[dbo].[Etudiants].NoINE = Opérateur scalaire([Jouet Cours BD].[dbo].[Etudiants].[NoINE])

Clustered Index Seek
Analyse d'une plage de lignes particulière à partir d'un index cluster.

Physical Operation	Clustered Index Seek
Opération logique	Clustered Index Seek
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001581
Estimated Operator Cost	0,0040736 (18 %)
Estimated Subtree Cost	0,0040736
Estimated Number of Rows	1
Estimated Row Size	11 O
Ordered	True
ID du nœud	11

Predicate
[Jouet Cours BD].[dbo].[Voitures].[NoINE]=[Jouet Cours BD].[dbo].[Etudiants].[NoINE]
Objet
[Jouet Cours BD].[dbo].[Voitures].[ClePrimaire_Voitures]
Output List
[Jouet Cours BD].[dbo].[Voitures].NoINE
Seek Predicates
Préfixe : [Jouet Cours BD].[dbo].[Voitures].NoImmatChiffres; [Jouet Cours BD].[dbo].[Voitures].NoImmatLettres; [Jouet Cours BD].[dbo].[Voitures].NoImmatDepart = Opérateur scalaire([Jouet Cours BD].[dbo].[Voitures].[NoImmatChiffres]); Opérateur scalaire([Jouet Cours BD].[dbo].[Voitures].[NoImmatLettres]); Opérateur scalaire([Jouet Cours BD].[dbo].[Voitures].[NoImmatDepart])

Index Seek
Analyse une plage de lignes particulière à partir d'un index non cluster.

Physical Operation	Index Seek
Opération logique	Index Seek
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001598
Estimated Operator Cost	0,0040745 (18 %)
Estimated Subtree Cost	0,0040745
Estimated Number of Rows	1,14109
Estimated Row Size	12 O
Ordered	True
ID du nœud	12

Objet
[Jouet Cours BD].[dbo].[AvoirObtenu].[Index_AvoirObtenu_Etudiants]
Output List
[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege
Seek Predicates
Préfixe : [Jouet Cours BD].[dbo].[AvoirObtenu].NoINE = Opérateur scalaire([Jouet Cours BD].[dbo].[Etudiants].[NoINE])

Clustered Index Scan
Analyse d'un index cluster, en entier ou sur une plage uniquement.

Physical Operation	Clustered Index Scan
Opération logique	Clustered Index Scan
Estimated I/O Cost	0,0032035
Estimated CPU Cost	0,0000895
Estimated Operator Cost	0,0033558 (15 %)
Estimated Subtree Cost	0,0033558
Estimated Number of Rows	3
Estimated Row Size	16 O
Ordered	False
ID du nœud	13

Predicate
[Jouet Cours BD].[dbo].[AvoirObtenu].[NoINE]<>(2) AND [Jouet Cours BD].[dbo].[AvoirObtenu].[NoINE]<>(4) AND [Jouet Cours BD].[dbo].[AvoirObtenu].[IntitAbrege]<>'BAC'
Objet
[Jouet Cours BD].[dbo].[AvoirObtenu].[ClePrimaire_AvoirObtenu]
Output List
[Jouet Cours BD].[dbo].[AvoirObtenu].IntitAbrege

SGBD⋆ (1/12)

Définitions

BD répartie (BD⋆) *(distributed database)*

Ensemble de BD gérées sur des sites différents et apparaissant à l'utilisateur comme une BD unique (i. e. ensemble structuré et cohérent de données, mémorisées sur des machines distinctes autonomes, et géré par un SGBD⋆)

Schéma global

Schéma conceptuel de la BD⋆

SGBD réparti (SGBD⋆) *(distributed DBMS)*

Système qui gère des collections de BD logiquement reliées, distribuées sur un réseau, en fournissant un mécanisme d'accès qui rend la répartition transparente aux utilisateurs

Client de SGBD⋆

Application qui accède aux informations distribuées par les interfaces du SGBD⋆

Serveur de SGBD⋆

SGBD gérant une BD locale intégrée dans une BD⋆

Nœud (ou site) du SGBD⋆

Calculateur dans le réseau participant à la gestion d'une BD⋆

BD interopérables

BD capables d'échanger des données en comprenant mutuellement ce qu'elles représentent

Multibase

BD hétérogènes (SGBD d'éditeurs différents ; SGBD de types différents : relationnel, réseau, hiérarchique, objet voire SGF ; données de natures différentes : alphanumériques, multimédia, géographiques, etc.) capables d'interopérer avec une application *via* un langage commun (mais sans modèle commun)

BD fédérées

BD hétérogènes accédées comme une seule BD *via* une vue commune (i. e. un modèle commun)

Générations

1^{re} génération (1975-1982)

Un SGBDR réparti

Inconvénients : degré d'hétérogénéité limité, objets complexes rarement gérés (exceptés les BLOB)

Exemples : Oracle*s-Stars, Ingres/Star

2^{e} génération (1980-1987)

Plusieurs SGBDR fédérés

3^{e} génération (depuis 1994)

SGBD basés sur le modèle orienté-objet

Avantages et inconvénients

Avantages

Prise en compte de la répartition géographique des données (car les données sont naturellement décentralisées et l'évolutivité de la BD⋆ est liée à celle de l'organisation)

Prise en compte de la distribution fonctionnelle des données (car le système d'information se répartit de lui-même dans l'organisation ; adaptation du système d'information à la structure organisationnelle)

Atténuation des conséquences d'une erreur humaine

Disponibilité des données en cas de dysfonctionnement des applications, sureté de fonctionnement en cas de panne (matérielle ou logicielle d'un nœud (ou site) du SGBD⋆) en travaillant en mode dégradé (sur les seules données accessibles)

Flexibilité pour assurer le partage des données hétérogènes et réparties (exemple : utiliser des SGBD spécialisés pour gérer certaines informations)

Performances attendues supérieures grâce à l'éclatement des données sur plusieurs bases gérées par des serveurs différents et spécialisés

Intégration du réseau

Inconvénients

Complexité du SGBD⋆

Peu d'expérience

Performances dépendent de la vitesse et de la charge du réseau, de la judicieuse répartition des données et des traitements

Distribution du contrôle des données entre plusieurs sites engendrant des problèmes de cohérence non triviaux (cf. algorithmes d'achèvement en deux (voire trois) phases sur deux ou plusieurs niveaux, de période de doute, de *logical unit of work*) pour la concurrence (accès simultané à une information dupliquée et répartie) et la sécurité (reprise après panne)

Hétérogénéité des matériels et logiciels actuels (ODBC, *de facto* l'un des standards actuels, ne résolvant pas tous les problèmes)

Difficulté de changement (par exemple, pour intégrer un nouveau type de SGBD)

Objectifs

Multiclients multiserveurs

Mécanismes de contrôle de la concurrence adaptés garantissant que l'effet de l'exécution simultanée des transactions est le même que celui d'une exécution en série (cf. l'isolation des propriétés des transactions)

Exécution de requêtes distribuées (i. e. émise par un client, dont l'exécution nécessite l'exécution de plusieurs sous-requêtes, chacune sur son propre serveur)
N. B. : une transaction distribuée est une transaction mettant en œuvre plusieurs serveurs

Transparence à la localisation des données

Propriété d'un SGBD⋆ permettant d'écrire des requêtes avec des noms d'objets référencés ne contenant pas la localisation des données (schéma global couplé à des schémas de fragmentation ; schéma de localisation pour gérer l'interconnexion entre sites)

Avantages

Transparence pour l'utilisateur et l'écriture des requêtes (gestion automatique de la décomposition/recomposition des requêtes/résultats)

Possibilité (pour l'administrateur réseau) de déplacer les objets sans modifier les requêtes

Inconvénients

Le SGBD⋆ doit rechercher les sites capables de répondre à la question pour l'exécuter

Compromis

`CREATE VIEW ... AS SELECT ... FROM` ◁ relation ▷`@`◁ base ▷ `...`

Meilleure disponibilité des données

Ne dépend pas d'un site unique (centralisé)

Optimise les performances

Permet les copies pouvant même évoluer indépendamment pour converger ultérieurement (cf. réplication)

Garantit l'atomicité des transactions et la cohérence des mises à jour réparties pour les transactions distribuées

Autonomie locale des sites

Pas d'administration centralisée mais administrations locales indépendantes et séparées pour chaque serveur de la BD⋆

Gestion du schéma global et gestion des schémas des objets ou relations distants utilisés dans les requêtes par adjonction d'une extension du dictionnaire (exemple : remplie lors du premier accès puis mise à niveau des schémas importés automatiquement *via* une gestion des versions)

Support de l'hétérogénéité

Utilisation d'un langage pivot (SQL) pour accéder aux BD hétérogènes

Intégration de schémas i. e. rapprocher des schémas sémantiquement différents afin de constituer un schéma unique (exemples : une même information peut avoir un nom différent ou une représentation différente ; inversement, deux données de même nom peuvent avoir deux sens différents)

Conception

Descendante

Élaboration du schéma conceptuel global puis des schémas locaux

Ascendante

Intégration des BD locales existantes dans une BD fédérée (après réconciliation sémantique des schémas locaux participant au schéma global)

Partition ou duplication (avec $BD\star = \bigcup_s BD_s$ où BD_s est la BD du site s)

BD⋆ partitionnée : $BD_i \bigcap BD_j = \emptyset, \forall i \neq j$

BD⋆ totalement dupliquée : $BD_s = BD\star, \forall s$

BD⋆ partiellement dupliquée : $BD_i \bigcap BD_j$ quelconque

Fragment (ou partition) : sous-relation obtenue par sélection et/ou projection d'une relation, localisée sur un site unique

Fragment horizontal

Fragment obtenu uniquement par sélection

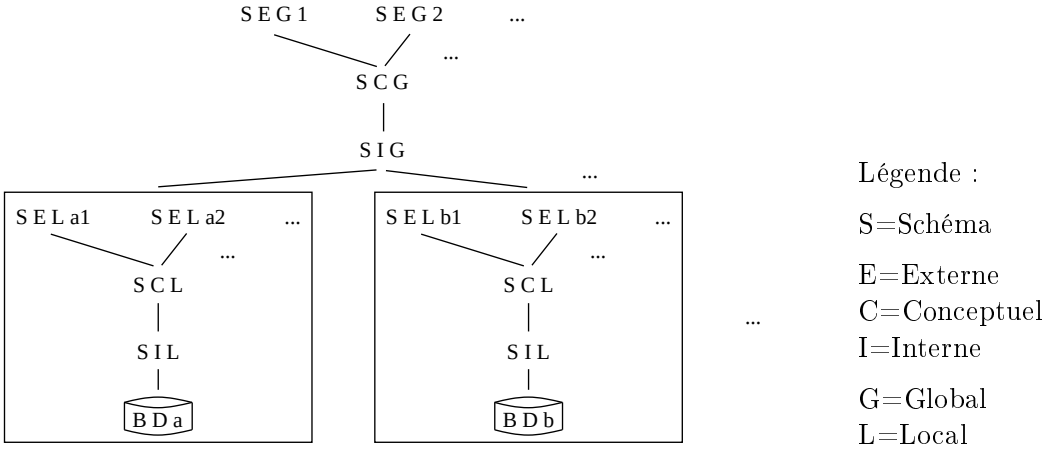
Exemple : $\sigma_{NoImmatDepart=33}(Voitures)$ sur le site bordelais et $\sigma_{NoImmatDepart<>33}(Voitures)$ sur un autre site ; tous les sites ont le même schéma global

Fragment vertical

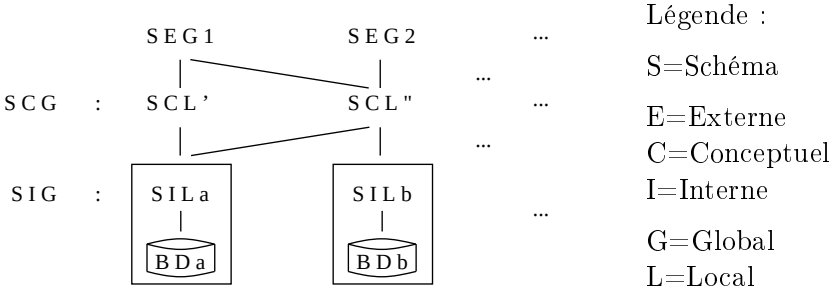
Fragment obtenu uniquement par projection

Exemple : $\pi_{NoINE,NomEtu}(Etudiants)$ sur le site gérant les noms des étudiants et $\pi_{NoINE,DepartNaissEtu}(Etudiants)$ sur le site gérant les départements de naissance des étudiants ; décomposition du schéma global entre les sites, avec la seule répétition des clés primaires sur chaque site

Répartition interne (un seul schéma conceptuel ; plusieurs schémas internes)



Répartition interne et conceptuelle (plusieurs schémas conceptuels (multi-bases) ; plusieurs schémas internes)



Réplication : . . . /. . .

Achèvement en deux (voire trois) phases *(two-phases commit protocol)* (1/3)

Objectif

Garantir l'atomicité des transactions

Le mécanisme de validation (`COMMIT`) ou d'invalidation (`ROLLBACK`) d'une transaction est plus complexe dans un SGBD⋆ (ressources distinctes connectées *via* un réseau de communication dans lequel les liens peuvent spontanément être interrompus)

Protocole

Phase 1

Le coordinateur demande à tous les participants de se mettre dans un état où ils peuvent choisir un achèvement pour la transaction (`PREPARE`)

Chaque participant force l'écriture du journal, libère les ressources, et envoie `OK` (si tout s'est bien passé) ou `NOT OK` (sinon) au coordinateur

Phase 2

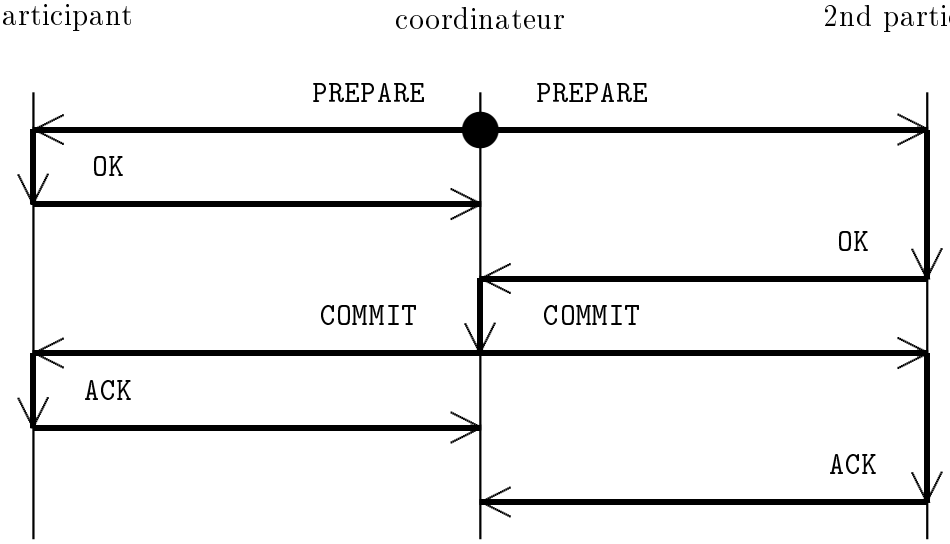
Si toutes les réponses des participants sont `OK`

Alors le coordinateur valide les commandes d'achèvement à tous les participants (`COMMIT`) et chaque participant effectue un achèvement local (en forçant l'écriture du `COMMIT` dans le journal)

Sinon le coordinateur renvoie un ordre d'invalidation (`ROLLBACK`) à tous les participants (`ABORT`) et chaque participant doit défaire l'effet local de la transaction (`ROLLBACK` local)

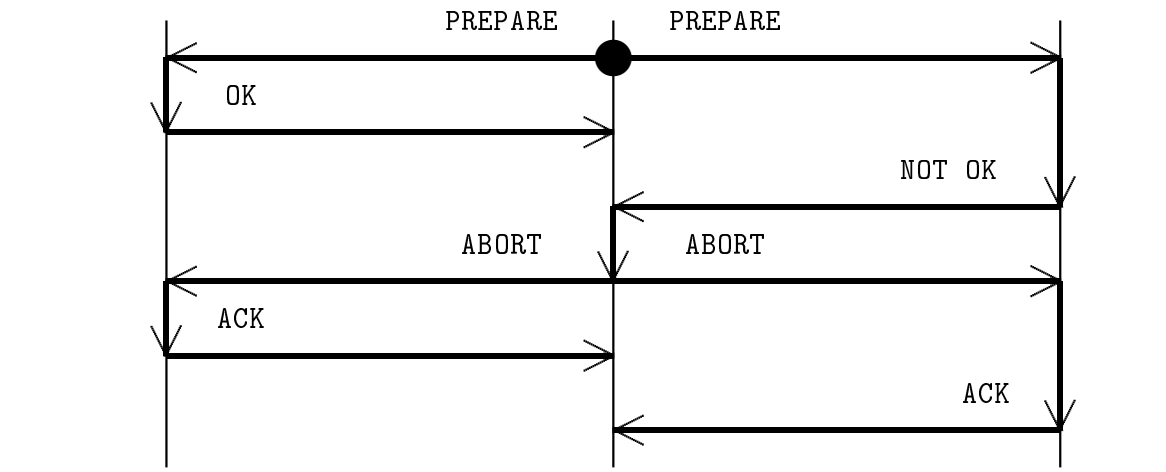
Cas principaux

Premier cas : validation

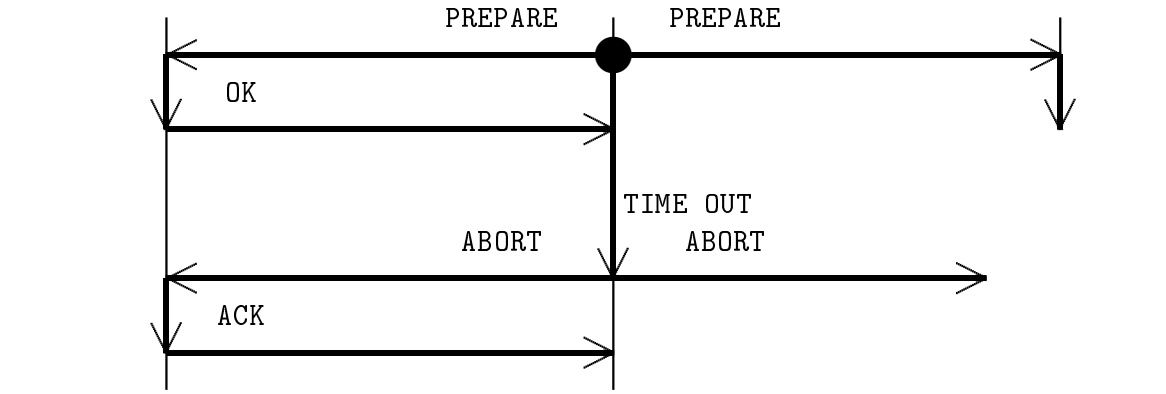


Achèvement en deux (voire trois) phases (2/3)

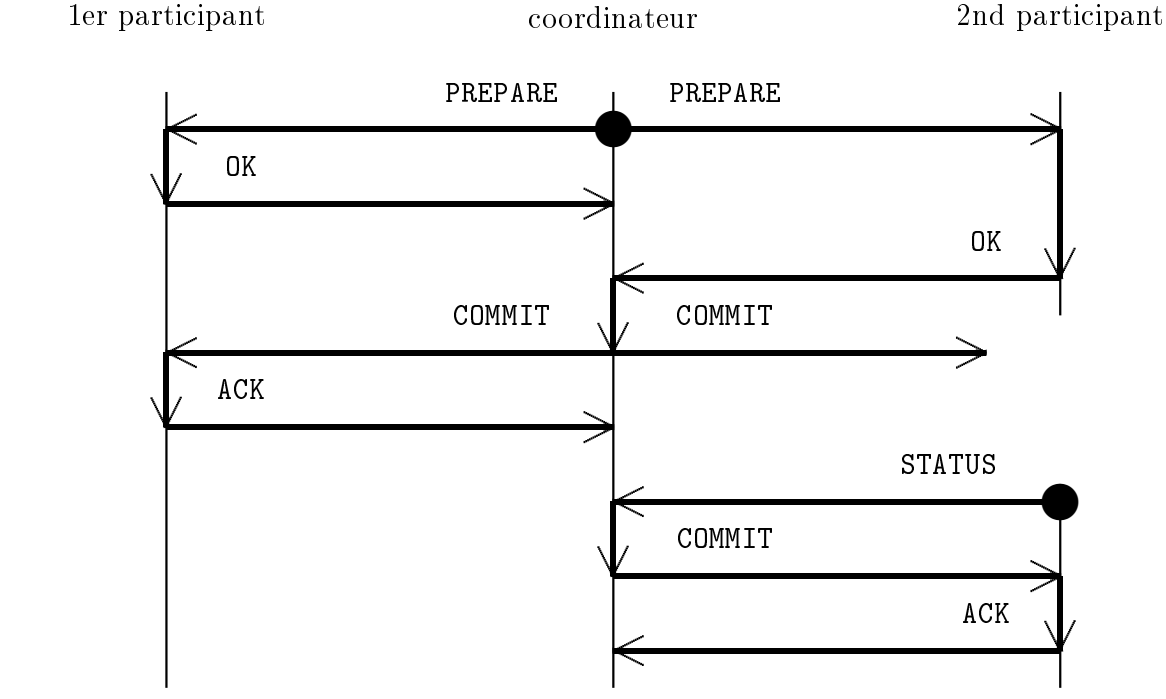
Deuxième cas : invalidation



Troisième cas : invalidation suite à une interruption

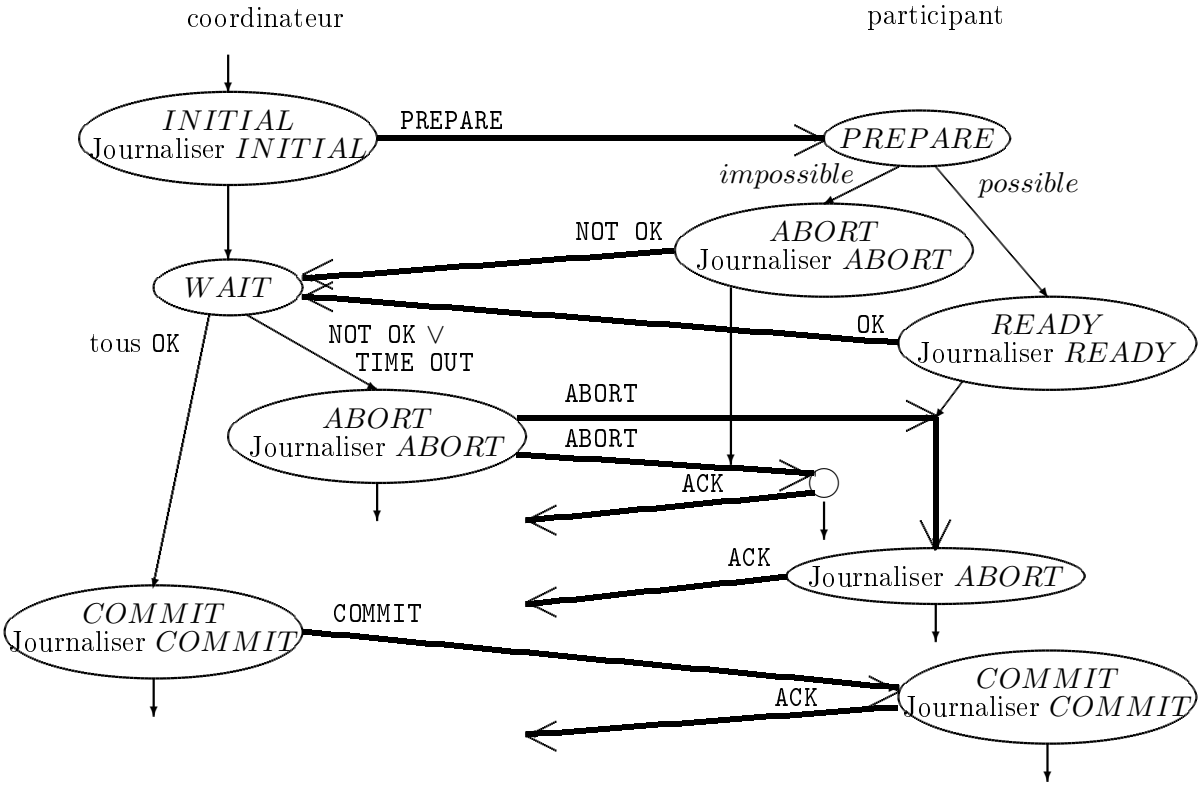


Quatrième cas : validation malgré une interruption



Achèvement en deux (voire trois) phases (3/3)

Automate



Problème : un participant dans l'état *READY* doit attendre soit que tous les autres participants aient répondu `OK`, soit le premier qui a répondu `NOT OK`, soit le `TIME OUT`

Solution : protocole d'achèvement en trois phases qui introduit un état supplémentaire *PRECOMMIT* pour les participants (entre les états *READY* et *COMMIT*) et le coordinateur (entre les états *WAIT* et *COMMIT*)

Moniteur transactionnel

Implante le protocole d'achèvement en deux phases dans le cadre de systèmes hétérogènes

Accès continu aux données et reprise rapide du système réparti en cas de panne

Accroissement de la sécurité des accès *via* leurs propres contrôles

Amélioration des performances (*via* la gestion de caches) et support multifilières aux requêtes

Exemple : modèle *Distributed Transaction Processing* (DTP) de l'X/Open met en œuvre le protocole d'achèvement (hiérarchique) en deux phases *(Transaction Processing)* (TP) (qui est lui-même un protocole standard proposé par l'ISO dans le cadre de l'architecture *Open Systems Interconnection* (OSI) (modèle ISO 7498-1 composé des sept couches physique, liaison, réseau, transport, session, présentation et application) afin d'assurer une validation cohérente des transactions dans un système distribué)

Contrôle de la concurrence : verrouillage en deux phases (1/3)

Objectif du contrôle de la concurrence

Garantir la cohérence et l'isolation des transactions afin d'éviter :

les pertes de mise à jour (exemple : une première transaction lit un entier, puis une seconde transaction double cet entier, puis la première transaction incrémente l'entier)

les incohérences violant les contraintes d'intégrité (exemple : considérons la contrainte selon laquelle deux entiers doivent être égaux ; une première transaction incrémente le premier entier, puis une seconde transaction double les deux entiers, puis la première transaction incrémente le second entier)

les lectures non répétitives

Précédence (et graphe de précédence)

Propriété stipulant qu'une transaction a accompli une opération sur une donnée avant qu'une autre transaction n'accomplisse une opération

Deux opérations : lire, écrire

$T1$ précède $T2$ (que l'on peut noter $T1 \text{—} \{t\} \longrightarrow T2$) lorsque :

$T1$ lit t puis $T2$ écrit t

$T1$ écrit t puis $T2$ écrit t

$T1$ écrit t puis $T2$ lit t

Théorème : une condition suffisante de sériabilité (cf. la propriété d'isolation des transactions) est que le graphe de précédence reste sans circuit

Principe du verrouillage en deux phases

Technique de contrôle des accès concurrents consistant à verrouiller les objets (en lecture ou en écriture) au fur et à mesure des accès par les transactions (phase d'acquisition des verrous) et à relâcher les verrous (phase de relâchement) seulement après l'obtention de tous les verrous

Corollaire : ordre des transactions identique à celui sur les objets accédés en mode incompatible

Compatibilité des opérations :

	lecture	écriture
lecture	compatible	*non compatible*
écriture	*non compatible*	*non compatible*

Protocole (du verrouillage en deux phases)

Lors d'une demande de verrouillage, si l'objet désiré est verrouillé, la transaction en demandant l'accès est mise en attente jusqu'à sa libération

Corollaire : toute transaction attend la fin des transactions incompatibles (et le graphe de précédence est ainsi sans circuit)

Choix de l'unité de verrouillage : relation, page, tuple

Verrouillage implicite des nœuds et explicite des feuilles du graphe d'imbrication des granules : *relation* $\longrightarrow$ *page* $\longrightarrow$ *tuple* (où *tuple* est ici la seule feuille)

Nouvelle compatibilité des opérations :

	lecture	écriture	lecture implicite	écriture implicite
lecture	compatible	*non compatible*	compatible	*non compatible*
écriture	*non compatible*	*non compatible*	*non compatible*	*non compatible*
lecture implicite	compatible	*non compatible*	compatible	compatible
écriture implicite	*non compatible*	*non compatible*	compatible	compatible

Contrôle de la concurrence : verrouillage en deux phases (2/3)

Inter-blocages inter-sites (1/2)

Problème

Dans un système réparti, chaque serveur gère les verrous de ses données (verrouillage décentralisé) mais il faut aussi résoudre les inter-blocages inter-sites

Première solution : la prévention i. e. empêcher que les inter-blocages surviennent (1/2)

Deux techniques de contrôle des accès concurrents : les verrouillages synchronisent l'exécution imbriquée d'un ensemble de transactions (en les sérialisant) et les estampilles imposent une exécution en série spécifique (définie par l'ordre chronologique des estampilles)

Premier algorithme *Wait-Die* : quand une transaction $T1$ demande de verrouiller un tuple déjà verrouillé par une autre transaction $T2$, soit $T1$ attend si elle est plus vieille que $T2$ (i. e. $T1$ a démarré avant $T2$) soit $T1$ meurt (i. e. est annulée et redémarrée) si elle est plus jeune que $T2$ (i. e. $T1$ a démarré après $T2$)

Second algorithme *Wound-Wait* : quand une transaction $T1$ demande de verrouiller un tuple déjà verrouillé par une autre transaction $T2$, soit $T1$ attend si elle est plus jeune que $T2$ (i. e. $T1$ a démarré après $T2$) soit $T1$ perturbe $T2$ (i. e. $T2$ est annulée et redémarrée) si elle est plus vieille que $T2$ (i. e. $T1$ a démarré avant $T2$)

Les deux algorithmes garantissent que toutes les transactions en attente attendent en respectant l'ordre chronologique (les transactions les plus vieilles attendent les plus jeunes pour *Wait-Die*, et inversement pour *Wound-Wait*)

Corollaire : pas d'interblocage possible

N. B. : les deux algorithmes consistent à annuler (en libérant tous ses verrous) puis redémarrer une transaction qui créerait un cycle dans le graphe d'attente

Remarque : *Wound-Wait* provoque en général moins de reprises de transactions

Exemple : problème de l'inter-blocage (cf. les problèmes et solutions de concurrence des transactions) appliqué avec ces deux algorithmes

Wait-Die :

T'	T''
démarrage	
	démarrage
verrou $t1$	
	verrou $t2$
demande verrou $t2$; attente $t2$	
débloqué ; verrou $t2$	demande verrou $t1$; T'' meurt (libère verrou $t2$, redémarre) demande verrou $t2$; T'' meurt (redémarre) demande verrou $t2$; T'' meurt (redémarre) ⋮

Wound-Wait :

T'	T''
démarrage	
	démarrage
verrou $t1$	
	verrou $t2$
demande verrou $t2$; T' perturbe T'' verrou $t2$	T'' perturbée (libère verrou $t2$, redémarre) demande verrou $t2$; attente $t2$

Contrôle de la concurrence : verrouillage en deux phases (3/3)

Inter-blocages inter-sites (2/2)

Première solution : la prévention (2/2)

Une estampille de transaction est un numéro unique attribué à une transaction lors de son démarrage permettant de l'ordonner strictement parmi les autres transactions (exemple : concaténation de l'horodate et d'un numéro de site)

Les conflits (survenant lorsqu'une transaction veut lire un tuple déjà lu ou mis à jour par une transaction plus jeune) sont résolus en annulant et en redémarrant la transaction à laquelle est associée une nouvelle estampille

Remarque : l'inter-blocage est impossible

N. B. : aucune mise à jour non validée n'est réalisée

Algorithmes de lecture et de mise à jour *via* les estampilles

T : transaction
e : estampille de T
t : tuple
e_dern_lect : estampille de la dernière (i. e. la plus récente) lecture de t
$e\_dern\_màj$: estampille de la dernière (i. e. la plus récente) mise à jour validée de t

Action Lire(t,T)
  Si $e \geq e\_dern\_màj$ Alors
    $e\_dern\_lect \longleftarrow \max\{e, e\_dern\_lect\}$
    `SELECT`
  Sinon
    AnnulerRedémarrer(T)

Action MettreÀJour(t,T)
  Si $e \geq e\_dern\_lect$ et $e \geq e\_dern\_màj$ Alors
    $e\_dern\_màj \longleftarrow e$
    `INSERT` ou `UPDATE` ou `DELETE`
  Sinon
    AnnulerRedémarrer(T)

Seconde solution : la détection i. e. supprimer les inter-blocages par reprise des transactions afin de rompre les circuits d'attentes

Deux algorithmes : centralisé (collecte des états afin de construire le graphe d'attente) ou distribué (propagation d'enquêtes auprès des contrôleurs de concurrence)

Réplication (1/2)

Définitions

Réplication : technique (de répartition) permettant de gérer des copies multiples pouvant diverger à un moment donné mais convergeant à terme vers la même valeur (si on arrête la production de transactions de mises à jour)

Cliché *(snapshot)* : copie dérivée asynchrone, donc mise à jour périodiquement selon le choix de l'administrateur

Copie dérivée : copie de sous-ensembles intégrés de plusieurs relations maîtresses définies par une question portant sur ces relations

Avantages

Augmentation des performances (lecture/écriture des informations sur le site le plus proche du client)

Disponibilité des données accrue (en cas de panne d'un serveur par exemple)

Problèmes

Convergence des copies (possiblement longues mais devant impérativement survenir lors de l'arrêt des transactions)

Transparence de cette gestion pour les utilisateurs

Techniques de diffusion des mises à jour

Déclencheurs

Files persistantes : mémorisation d'une transaction reportée jusqu'à l'émission par une tâche de fond

Appels périodiques : les sites gérant les copies appellent le (ou les) site(s) centralisant les mises à jour

Mise à jour synchrone ou asynchrone

Mise à jour synchrone : mode de distribution dans lequel toutes les sous-opérations locales effectuées suite à une mise à jour globale sont accomplies pour le compte de la même transaction

Mise à jour asynchrone : mode de distribution dans lequel certaines des sous-opérations locales effectuées suite à une mise à jour globale sont accomplies dans des transactions indépendantes, en temps différé

Réplication asymétrique ou symétrique

Réplication asymétrique : technique de gestion de copies basée sur un site primaire seul autorisé à mettre à jour la donnée et chargé de diffuser les mises à jour aux autres copies dites secondaires

Réplication symétrique : technique de gestion de copies où chaque copie peut être mise à jour à tout instant et assure la diffusion des mises à jour aux autres copies

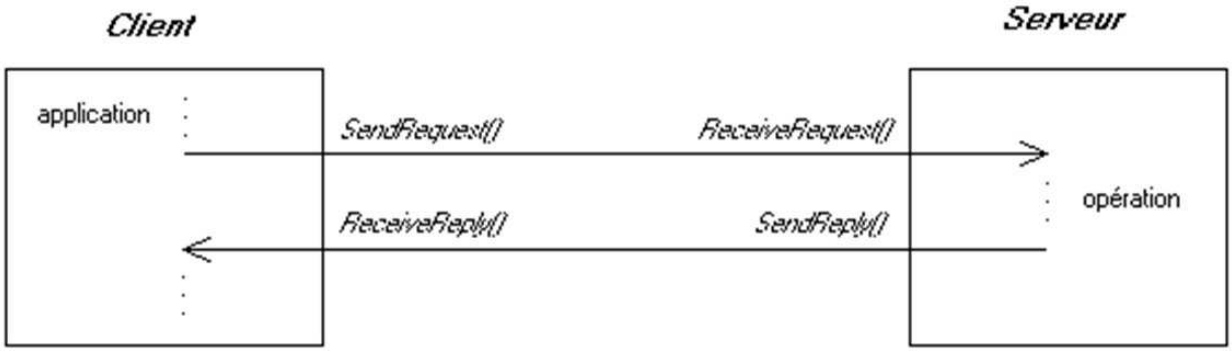
Remarque : toutes les combinaisons de mise à jour (synchrone ou asynchrone) et de réplication (asymétrique ou symétrique) sont possibles

Réplication (2/2)

Problème de la convergence de copies pour la réplication symétrique

- Risque de divergence car la sériabilité de l'exécution globale n'est pas forcément respectée
- Deux techniques applicables dans le cas des mises à jour synchrones
  - Prévention des conflits par verrouillage des copies (toute transaction mettant à jour la donnée pose un verrou en écriture sur chaque copie)
  - Détection des conflits par ordonnancement des mises à jour (*via* un mécanisme d'estampillage (synchronisé) ou de certification)
- Il est impossible de défaire des transactions validées dans le cas de mises à jour asynchrones !
  - Il faut alors détecter les conflits (*via* les estampilles ou la vérification des valeurs avant les mises à jour) pour les soumettre ensuite aux utilisateurs qui doivent eux-mêmes réconcilier les copies

Définitions



Architecture Client/Serveur *(client-server architecture)* (C/S)

Modèle d'architecture applicative où les programmes sont répartis entre processus clients et serveurs communiquant par des requêtes avec réponses

Client *(client)*

Processus demandant l'exécution d'une opération à un autre processus (serveur) par l'envoi d'un message contenant son descriptif et attendant la réponse par un message en retour

Serveur *(server)*

Processus accomplissant une opération sur demande d'un client et lui transmettant la réponse

Requête *(request)*

Message transmis par un client à un serveur décrivant l'opération à exécuter

Réponse *(reply)*

Message transmis par un serveur à un client suite à l'exécution d'une opération contenant les paramètres de retour de l'opération

Fonctions de base

Procédure de connexion *(connection procedure)*

Opération consistant à ouvrir un chemin depuis un client vers un serveur (après authentification)

Procédure de déconnexion *(deconnection procedure)*

Opération consistant à fermer le chemin (ouvert lors de la connexion) allant du client vers le serveur

Préparation de requête *(request preparation)*

Opération consistant à envoyer une requête avec des paramètres non instanciés à un serveur afin qu'il prépare son exécution (« compilation » et définition du plan d'exécution)

Exécution de requête *(request execution)*

Opération consistant à envoyer une demande d'exécution d'une requête précédemment préparée à un serveur, en fournissant les valeurs des paramètres

Récupération des résultats *(result fetching)*

Opération consistant à rapporter tout ou partie du résultat d'une requête sur le client

Cache des requêtes *(request caching)*

Technique permettant de conserver des requêtes « compilées » (avec leurs plans d'exécution) sur le serveur afin de les réutiliser pour répondre à des requêtes similaires

Cache des résultats *(result caching)*

Technique permettant de transférer les résultats par blocs et de les conserver sur le client et/ou le serveur afin de les réutiliser pour répondre à des requêtes

Composants

Système ouvert : solution non propriétaire basée sur des standards de l'ISO, de l'ANSI, de l'*Institute of Electrical and Electronic Engineers* (IEEE), de l'X/Open, de l'*Object Management Group* (OMG) et de l'*Open Software Foundation* (OSF)

SGBDR + SQL sur le serveur

Stations de travail personnelles (interface graphique, connexion au réseau)

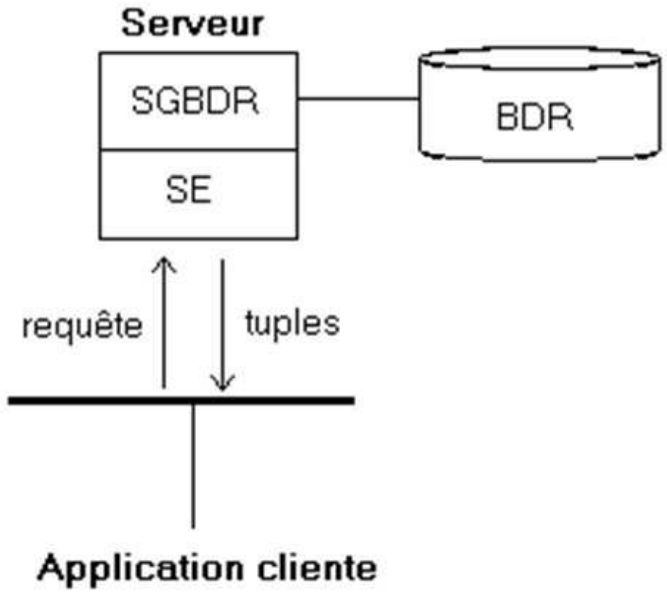
Outils de développement d'applications variés

Logiciels de transport de requêtes et de réponses

Outils de conception, de déploiement et de maintenance

Générations

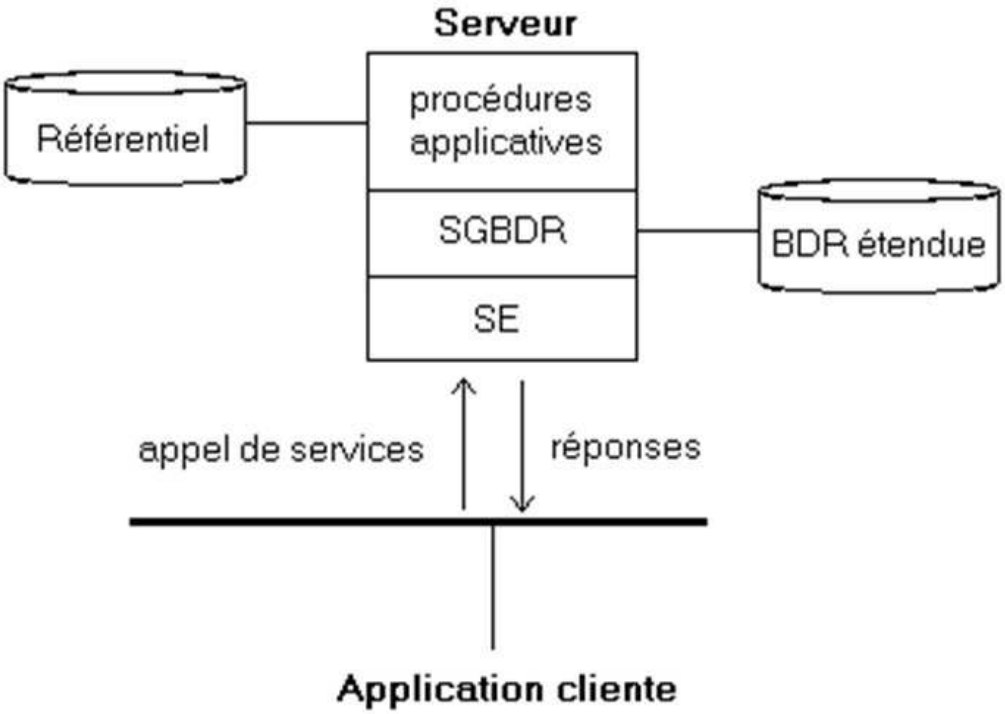
1^{re} génération (début des années 80)



Développement en SQL sur le serveur géré par un SGBDR (au dessus d'un système d'exploitation)

Développement des applications en L4G + SQL + interface graphique sur le client

2^{de} génération (milieu des années 90)

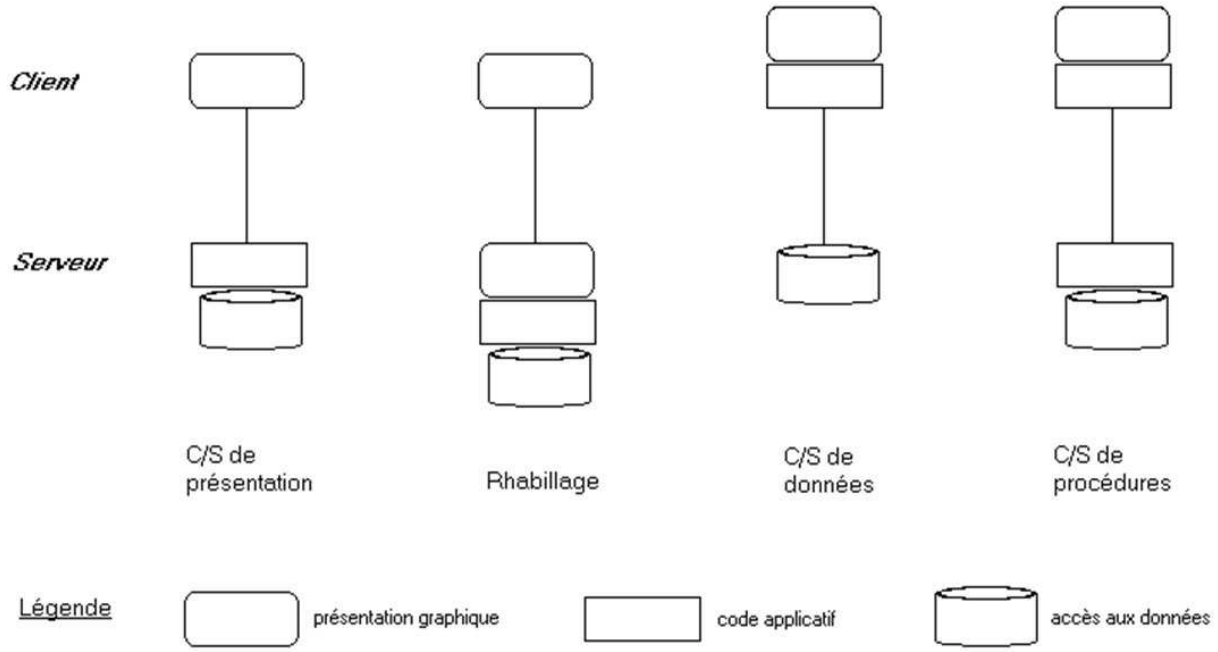


Traitements applicatifs au sein du serveur (procédures stockées notamment)

Approche orientée objet (interface graphique, modélisation des données)

Déploiement de l'applicatif facilité (partitionnement automatique du code de l'applicatif entre client et serveur) par la gestion d'un référentiel des objets de l'application au sein du serveur

Niveaux de C/S



C/S de présentation *(presentation client-server)*

Un processus exécute uniquement les fonctions de dialogue avec l'utilisateur, l'autre gérant les données et exécutant le code applicatif

Rhabillage *(revamping)*

Un processus exécute les fonctions de dialogue sophistiquées avec l'utilisateur, l'autre gérant les données et exécutant le code applicatif et assurant les dialogues simplifiés avec le client

C/S de données *(data client-server)*

Un programme applicatif contrôlé par une interface de présentation sur une machine cliente accède à des données sur une machine serveur par des requêtes de manipulation de données

C/S de procédures *(procedure client-server)*

Un programme applicatif contrôlé par une interface de présentation sur une machine cliente sous-traite l'exécution de procédures applicatives à une machine serveur (ces procédures encapsulent le plus souvent la BD)

N. B. : C/S de données et de procédures

Clients

Possèdent du code de l'application non directement lié aux données : dialogues interactifs avec l'utilisateur, traitements spécialisés des messages, affichage des résultats

Serveur (SGBDR et procédures stockées)

Stockage, distribution, sécurité des données

Accès transactionnels et décisionnels

Réseau

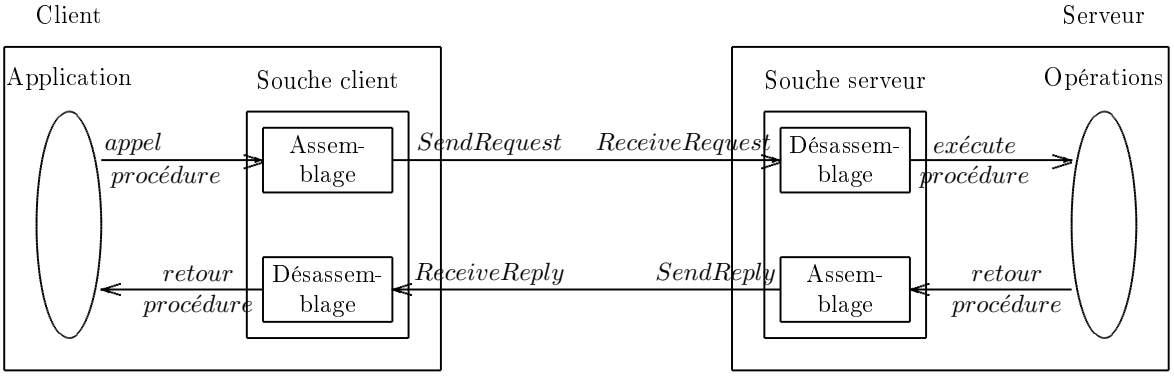
Transfert des demandes et des résultats

Connectabilité du client au serveur

Appel de procédure à distance *(Remote Procedure Call)* (RPC)

Définition

Technique permettant d'appeler une procédure distante comme s'il s'agissait d'une procédure locale, en rendant transparent les messages échangés et les assemblages/désassemblages de paramètres



Assemblage *(marshalling)*

Procédé consistant à prendre une collection de paramètres, à les arranger et à les coder en format externe pour constituer un message à émettre

Désassemblage *(unmarshalling)*

Procédé consistant à prendre un message en format externe et à reconstituer la collection des paramètres qu'il représente en format interne

Souche *(stub)*

Représentant d'une procédure sur un site client ou serveur capable de recevoir un appel de procédure du client et de le transmettre au format adapté à l'implantation ou à son représentant

N. B. : deux types de dialogues

Synchrone (dialogue sans file d'attente ; les commandes d'émission et de réception sont bloquantes)

Asynchrone (dialogue avec file d'attente ; au moins l'une des commandes d'émission ou de réception n'est pas bloquante)

Médiateur *(middleware)*

Définition

Ensemble des services logiciels construits au dessus d'un protocole de transport afin de permettre l'échange des requêtes et des réponses associées entre client et serveur de manière transparente (afin de cacher l'hétérogénéité des composants mis en œuvre, ce qui suppose un format d'échange standard pour les différents codages internes)

Types de médiateurs

Manipulation de données

Transactionnels

D'invocation d'objets (exemples : CORBA [OMG], OLE2 [Microsoft])

Niveaux de médiateurs de manipulation de données

Transporteur de requêtes

Accès multi-bases

Coordinateurs de systèmes distribués

Objectifs d'un médiateur de manipulation de données

Transport de requêtes et de réponses

Simplification de la programmation (API, intégration uniforme aux langages) : transparence aux réseaux (établissement d'une session i. e. au dessus de la couche transport de l'OSI), aux serveurs (malgré des dialectes différents) et aux langages (par les fonctions)

Harmonisation des types de données

Performance (gestion de caches clients et serveurs, parcours des résultats complexes tels les ensembles, objets longs, etc.)

Fiabilité (gestionnaire de transaction intégré)

Types de médiateurs de manipulation de données

Interfaces applicatives et transporteur de requêtes

Interface applicative i. e. interface de programmation d'application *(Application Programming Interface)* (API) : ensemble de fonctions standards pour accéder à un service local ou distant (exemple : CLI pour l'accès à des BD relationnelles distantes)

Transporteur de requêtes : met en œuvre des protocoles de niveau session ou application, en s'appuyant sur des protocoles de niveau transport, pour expédier des requêtes à des services distants et récupérer les réponses

Architectures multi-bases

Intégration des API et transporteurs de requêtes au sein d'un même langage (SQL en général) pour accéder à des BD distantes hétérogènes

Résolution des problèmes de transactions multi-sites

Architectures pour applications distribuées

Cadre général (BD ou autre) pour exécuter des services distribués

Exemples : gestion de données uniformes, services de dénomination globaux, services de synchronisation entre processus distants, services de gestion de la sécurité, service de transactions distribuées

L'API standard *Client Interface* (CLI) (1/2)

L'X/Open (organisation à but non lucratif née en 1984 à l'initiative de constructeurs et d'utilisateurs) coordonne le développement d'API dans le cadre du *Common Application Environment* (CAE)

Objectifs du CAE : portabilité des logiciels applicatifs *via* des interfaces utilisateur communes, interfaces de programmation communes, modèles d'interconnexion communs

L'X/Open a repris les travaux du SAG sur les médiateurs et a proposé l'API CLI basée sur SQL standardisant des appels d'un client vers un serveur (de BD) SQL dans un langage quelconque

Exemples d'implantation de CLI : ODBC, JDBC

Fonctions *(routines)* (1/2)

N. B. : toutes les fonctions retournent un `SMALLINT`

Identification de contexte *(handle)*

`AllocHandle` (E/ ◁ type contexte ▷ , E/ ◁ contexte ▷ , S/ ◁ contexte ▷)

`FreeHandle` (E/ ◁ type contexte ▷ , E/ ◁ contexte ▷)

Identification du contexte de l'environnement de BD SQL *(SQL environment)*

`AllocEnv` (S/ ◁ contexte environnement ▷)

`SetEnvAttr` (E/ ◁ contexte environnement ▷ , E/ ◁ caractéristique ▷ , E/ ◁ valeur ▷ , E/ ◁ longueur chaîne ▷)

`FreeEnv` (E/ ◁ contexte environnement ▷)

Identification du contexte de la connexion SQL *(SQL connection)*

`AllocConnect` (E/ ◁ contexte environnement ▷ , S/ ◁ contexte connexion ▷)

`SetConnectAttr` (E/ ◁ contexte connexion ▷ , E/ ◁ caractéristique ▷ , E/ ◁ valeur ▷ , E/ ◁ longueur chaîne ▷)

`FreeConnect` (E/ ◁ contexte connexion ▷)

Gestion de la connexion

`Connect` (E/ ◁ contexte connexion ▷ , E/ ◁ serveur ▷ , E/ ◁ longueur chaîne serveur ▷ , E/ ◁ compte ▷ , E/ ◁ longueur chaîne compte ▷ , E/ ◁ mot de passe ▷ , E/ ◁ longueur chaîne mot de passe ▷)

`Disconnect` (E/ ◁ contexte connexion ▷)

Identification du contexte d'ordre SQL *(SQL statement)*

`AllocStmt` (E/ ◁ contexte connexion ▷ , S/ ◁ contexte ordre SQL ▷)

`SetStmtAttr` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ caractéristique ▷ , E/ ◁ valeur ▷ , E/ ◁ longueur chaîne ▷)

`FreeStmt` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ option ▷)

Envoi d'une requête pour une exécution immédiate

`ExecDirect` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ ordre SQL ▷ , E/ ◁ longueur chaîne ordre SQL ▷)

Envoi d'une requête préparée

`Prepare` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ ordre SQL ▷ , E/ ◁ longueur chaîne ordre SQL ▷)

`BindParam` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ numéro paramètre ▷ , E/ ◁ type valeur ▷ , E/ ◁ type paramètre ▷ , E/ ◁ taille attribut ▷ , E/ ◁ nombre décimales ▷ , [E/] ◁ valeur paramètre ▷ , [S/] ◁ longueur ▷)

`Execute` (E/ ◁ contexte ordre SQL ▷)

CLI (2/2)

Fonctions *(routines)* (2/2)

Réception, pour une requête d'interrogation, du résultat et des informations sur sa structure

`NumResultCols` (E/ ◁ contexte ordre SQL ▷ , S/ ◁ nombre attributs ▷)

`DescribeCol` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ numéro attribut ▷ , S/ ◁ nom attribut ▷ , E/ ◁ longueur tampon ▷ , S/ ◁ longueur nom ▷ , S/ ◁ type ▷ , S/ ◁ taille attribut ▷ , S/ ◁ nombre décimales ▷ , S/ ◁ attribut facultatif ? ▷)

`ColAttribute` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ numéro attribut ▷ , E/ ◁ identifiant champ ▷ , S/ ◁ caractéristique caractère ▷ , E/ ◁ longueur tampon ▷ , S/ ◁ longueur chaîne ▷ , S/ ◁ caractéristique numérique ▷)

`BindCol` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ numéro attribut ▷ , E/ ◁ type cible ▷ , [S/] ◁ valeur cible ▷ , E/ ◁ longueur tampon ▷ , [S/] ◁ longueur ▷)

`Fetch` (E/ ◁ contexte ordre SQL ▷)

`FetchScroll` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ orientation ▷ , E/ ◁ valeur déplacement ▷)

Gestion de curseur

`SetCursorName` (E/ ◁ contexte ordre SQL ▷ , E/ ◁ curseur ▷ , E/ ◁ longueur nom curseur ▷)

`GetCursorName` (E/ ◁ contexte ordre SQL ▷ , S/ ◁ curseur ▷ , E/ ◁ longueur maximale nom curseur ▷ , S/ ◁ longueur nom curseur ▷)

`CloseCursor` (E/ ◁ contexte ordre SQL ▷)

Réception du résultat d'une requête de mise à jour

`RowCount` (E/ ◁ contexte ordre SQL ▷ , S/ ◁ nombre lignes ▷)

Gestion des transactions

`EndTran` (E/ ◁ type contexte ▷ , E/ ◁ contexte ▷ , E/ ◁ type terminaison ▷)

Annulation

`Cancel` (E/ ◁ contexte ordre SQL ▷)

Gestion des erreurs

`Error` (E/ ◁ contexte environnement ▷ , E/ ◁ contexte connexion ▷ , E/ ◁ contexte ordre SQL ▷ , S/ ◁ état erreur ▷ , S/ ◁ erreur initiale ▷ , S/ ◁ message erreur ▷ , E/ ◁ longueur tampon ▷ , S/ ◁ longueur message ▷)

`GetDiagRec` (E/ ◁ type contexte ▷ , E/ ◁ contexte ▷ , E/ ◁ numéro enregistrement ▷ , S/ ◁ état erreur ▷ , S/ ◁ erreur initiale ▷ , S/ ◁ message erreur ▷ , E/ ◁ longueur tampon ▷ , S/ ◁ longueur message ▷)

Copie de descripteur

`CopyDesc` (E/ ◁ contexte descripteur source ▷ , E/ ◁ contexte descripteur destination ▷)

...

`GetFunctions`, `GetInfo` et `GetTypeInfo`

`DataSources`

`GetEnvAttr`, `GetConnectAttr` et `GetStmtAttr`

`ParamData` et `PutData`

`GetData`

`GetDescField` et `SetDescField`

`GetDescRec` et `SetDescRec`

`GetDiagField`

Open Database Connectivity (ODBC)

Bibliothèque de fonctions (API), sans pré-compilation, permettant d'accéder à tout SGBD, en C/S

Trois niveaux de conformité

Niveau noyau : fonctions de bas niveau

Niveau 1 : fonctions fournissant plus d'informations, paramétrage des différents identificateurs

Niveau 2 : fonctions de plus haut niveau (dont curseurs sophistiqués)

Exemples de fonctions du niveau noyau

`SQLAllocEnv` : définit un identificateur d'environnement pour l'application

`SQLAllocConnect` : définit un identificateur de connexion (i. e. un lien logique entre l'application et le SGBD)

`SQLConnect` et `SQLDisconnect` : démarre et arrête une connexion physique (liée à un identificateur de connexion)

`SQLAllocStmt` : définit un identificateur d'ordre SQL (lié à un identificateur de connexion)

`SQLExecDirect` : exécute une requête sans paramètre

`SQLPrepare`, `SQLSetParam` et `SQLExecute` : prépare, définit les paramètres et exécute une requête paramétrée

`SQLDescribeCol`, `SQLColAttributes` et `SQLNumResultCols` : analyse la structure du résultat d'une requête d'interrogation

`SQLBindCol` : associe les attributs du résultat et les variables hôtes ; gère le curseur

`SQLFetch` : retourne un tuple du résultat

`SQLRowCount` : retourne le nombre de tuples affectés par une requête de mise à jour

`SQLTransact` : gère les transactions

`SQLError` : gère les erreurs

Java Database Connectivity (JDBC)

Ensemble de classes (API) fourni avec Java permettant d'accéder à tout SGBD, en C/S

Quatre types de pilotes JDBC

Type 1 : utilisation d'une autre technologie (exemple : passerelle JDBC-ODBC *via* un pilote ODBC)

Type 2 : les appels JDBC sont convertis en appels natifs pour le SGBD

Type 3 : les appels JDBC sont convertis en un protocole indépendant du SGBD de sorte qu'un serveur les convertit ensuite dans le protocole du SGBD

Type 4 : les appels JDBC sont convertis directement en un protocole réseau exploité par le SGBD ; ces pilotes encapsulent directement l'interface cliente du SGBD (fournis par les éditeurs des SGBD)

Exemple (passerelle JDBC-ODBC)

```
// charge le pilote et crée une instance
Class.forName("jdbc.odbc.JDBC-ODBC_Driver").newInstance();
    // N. B. : pilote à rechercher
// stipule le DSN ODBC de la BD
String url = "jdbc:odbc:DSN_BD"; // N. B. : à remplacer
// se connecte
Connection connex = DriverManager.getConnection(url,compte,mot_passe);
// crée "statement" pour préparation et envoi requête statique ou dynamique
Statement stmt = connex.createStatement();
// précise la requête
String req = "SELECT NoINE , NomEtu , DepartNaissEtu FROM Etudiants";
// déclare le résultat de la requête
ResultSet res; // tuples dans un tableau de lignes de tuples résultats
// récupère tous les tuples du résultat
res = stmt.executeQuery(req);
// déclare les variables
int numero_etu; // numéro de l'étudiant
String nom_etu[]; // nom de l'étudiant
int depart_naiss_etu ; // département de naissance de l'étudiant
// parcourt les tuples du résultat
while (res.next()) {
    // récupère les informations du tuple
    numero_etu = res.getInt("NoINE"); // numéro de l'étudiant
    nom_etu = res.getString("NomEtu"); // nom de l'étudiant
    depart_naiss_etu = res.getInt("DepartNaissEtu");
        // département de naissance de l'étudiant
     // traite le tuple
    System.out.println("N° :" + numero_etu + ", Nom : " + nom_etu +
                       ", Département :" + depart_naiss_etu);
        // affiche les informations sur l'étudiant
    }
// libère le "statement"
stmt.close();
// se déconnecte
connex.close();
```

Le standard *Remote Data Access* (RDA)

Protocole de communication pour accéder depuis un client à des BD distantes

Standard ISO (1993), IEC et ANSI

Deux parties

- Générique
  - Couche C/S
  - Services nécessaires
  - Protocole de connexion à une BD quelconque
- Spécifique au langage SQL
  - Requêtes possibles pour une BD SQL
- ... ainsi qu'un langage de navigation dans des bases réseaux *(Network Data Language)* (NDL)

Construit au dessus des couches présentation et session de l'OSI

S'appuie sur un mode session conversationnelle i. e. de traitement transactionnel *(Transaction Processing)* (TP) de l'OSI et sur la forme canonique ASN.1 de représentation des données de l'ISO

Suppose un service d'association programme à programme de type OSI *Association Control Service Element* (ACSE) basé sur trois commandes

- `Associate` : demande d'association
- `ARelease` : relâchement d'association
- `Abort` : abandon d'association

Cinq types de services

- Gestion de dialogues : débuter/terminer des dialogues
- Gestion de transaction : débuter/terminer des transactions
- Contrôle : rapporter sur l'état ou arrêter une opération en cours
- Gestion de ressources : permettre/interdire l'accès aux ressources
- Langage de BD : accéder/modifier des ressources

Types de messages

- `Initialize` : initialisation d'une requête
- `Terminate` : terminaison d'une requête
- `BeginTransaction` : début de transaction
- `Commit` : validation de transaction
- `Rollback` : reprise de transaction
- `Cancel` : annulation de requête
- `Status` : demande d'état d'une requête
- `Open` : ouverture d'une ressource
- `Close` : fermeture d'une ressource
- `ExecuteDBL` : exécution d'une requête BD
- `DefineDBL` : préparation d'une requête BD
- `InvokeDBL` : invocation d'une requête BD préparée
- `DropDBL` : suppression d'une requête BD

Distributed Computing Environment (DCE)

Standard d'architecture distribuée de l'OSF autour de POSIX afin de faciliter le développement de grandes applications C/S autour de réseaux locaux, nationaux et internationaux

Utilise le RPC pour faire communiquer client et serveur

Fonctionnalités proposées

Dialogue C/S

Standards de services

Service de sécurité (compte et mot de passe)

Service de répertoires des ressources (annuaire)

Gestion du temps global synchronisé

Gestion des fichiers distribués

Standard pour la gestion et appel de serveurs multifilières *(multi-threads)* (i. e. des serveurs capables de traiter plusieurs requêtes quasi-simultanément au sein d'un même processus multiplexé)

Cellule

Unité d'administration et d'opération pour un groupe d'utilisateurs dans un réseau DCE

Contient au moins le service de sécurité, le service de répertoires des ressources et la gestion du temps global synchronisé

RPC

Repose sur un langage de définition d'interface *(Interface Definition Language)* (IDL) étendant les déclarations de variables et fonctions en langage C

On obtient, en partant de la définition d'interface, et après compilation, trois éléments :

Souche client : récupère les appels et les transmet au serveur (une fois compilé avec le client)

Souche serveur : récupère les appels et les transmet aux implantations des fonctions (une fois compilé avec le serveur)

Entête pour le code client et serveur : uniformisation des déclarations de type C/S

Système de fichiers distribués *(Distributed File System)* (DFS)

Gère des ensembles de fichiers connus globalement dans le réseau (grâce au système de dénomination et de répertoires)

Autres fonctionnalités

Rend possible le déplacement transparent pour les utilisateurs des fichiers d'une cellule à l'autre

Gère les sauvegardes automatiques

Protège les fichiers *via* un système étendu de permissions (*read*, *write*, *execute*, *control*, *insert*, *delete*)

Optimise les accès C/S *via* des caches sur le client

Offre des commandes standards POSIX sur les fichiers distribués (exemples : `cd`, `mkdir`, `ls`, `fopen`, ...)

SGBD déductif (1/2)

SGBD déductif ou Système de Gestion de Bases de Connaissances (SGBC)

À la fois un SGBDR et un Système Expert

Intelligence Artificielle (IA)

Objectif

Comprendre et imiter l'intelligence humaine

Mécanismes de l'intelligence humaine

Abstraction (capacité à raisonner sans s'appuyer sur la réalité)

Généralisation (raisonner, conclure du particulier au général)

Spécialisation (mise en œuvre de concepts dans un domaine particulier)

Analogie (adopter par ressemblance un raisonnement d'une situation connue à une situation nouvelle)

Correction d'erreur (i. e. ne pas reproduire deux fois la même erreur ... comme le font les algorithmes déterministes)

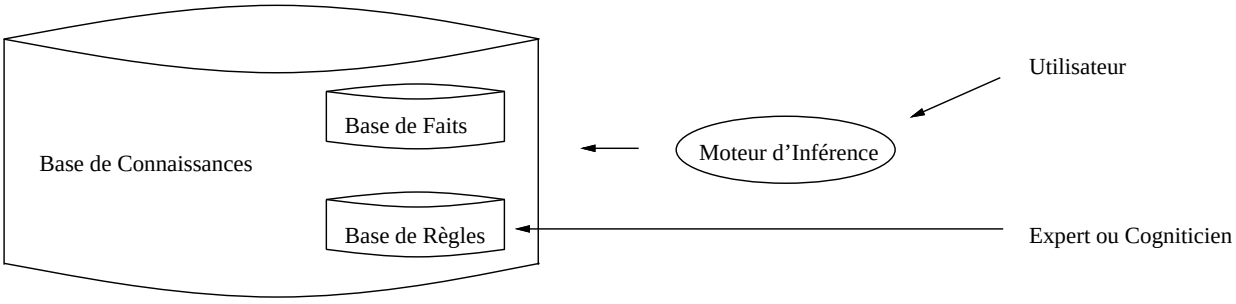
Cas d'informatisation : combinatoire vaste ; science non exacte ; traitement symbolique ; traitement non algorithmique ; données incomplètes ; mimer l'homme

Applications : reconnaissance des formes ; linguistique ; systèmes experts ; etc.

Système Expert (1/2)

Objectif

Aide au raisonnement humain et à la prise de décision dans un domaine spécifique



Base de faits (i. e. la BD : tuples des relations)

Connaissances assertionnelles (i. e. informations toujours vraies)

Implantation par une BD (relationnelle) : données, dictionnaire des données, intégrité des données, sécurité des données, évaluation des requêtes, etc.

Base de règles

Connaissances opératoires (i. e. mode d'emploi des faits)

Règle de production ($\triangleleft$ prémisse $\triangleright \Longrightarrow \triangleleft$ conclusion $\triangleright$) : `si` $\triangleleft$ condition $\triangleright$ `alors` $\triangleleft$ buts $\triangleright$

Exemple : `si` il pleut `alors` le temps est nuageux (calcul des propositions (sans variable))

Exemple : `si` x est un homme `alors` x est mortel
(logique du premier ordre i. e. calcul des prédicats (avec variables))

Acteurs

Expert : personne ayant la connaissance et/ou l'expérience du domaine

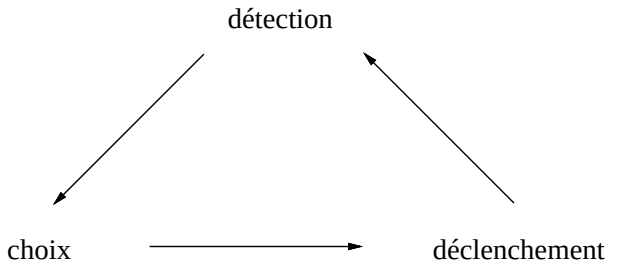
Cogniticien : spécialiste en IA

Implantation : intégration des règles (définition, limites car la récursivité est possible, évaluation au préalable ou à la demande), moteur d'inférence (associé au SGBDR)

Système Expert (2/2)

Moteur d'inférence (i. e. le logiciel qui infère)

Cycle d'inférence



Détection des règles applicables, relativement aux conditions (en chaînage avant) ou aux buts (en chaînage arrière)

Choix d'une (seule) règle à appliquer (parmi celles applicables)

Déclenchement de la règle à appliquer : déduction (en chaînage avant) ou régression (en chaînage arrière)

Chaînage

Avant : question → cycles d'inférences (i. e. déductions) → faits déduits

Arrière : hypothèse à vérifier → cycles d'inférences (i. e. régressions) → réponse

Mixte

Régime

Irrévocable (sans retour en arrière sur les choix)

Par tentatives

Arrêt

Absence de règle applicable (pour un chaînage en avant)

Succès ou échec (pour un chaînage en arrière)

Ordre

0 (sans variable)

1 (avec variables)

Fonctions d'un SGBDR classique

cf. calcul relationnel ou encore l'utilisation du calcul des prédicats en BD

Création d'un tuple

Exemple : *créer l'étudiant de numéro 5*
$Etudiants(5,'DURAND',33)$

Contrainte d'intégrité

Exemple : *toute voiture est possédée par un étudiant*
$\forall v\ \exists e / Voitures(v,,e) \rightarrow Etudiants(e,,)$

Expression d'une requête

Exemple : *les voitures des étudiants nés en Gironde*
$\{(v) / Voitures(v,,e) \wedge Etudiants(e,,33)\}$
avec projection (v), sélection (33) et jointure (même e pour $Voitures$ et $Etudiants$)

Création d'une vue à partir d'autres relations

Exemple : *les noms des étudiants*
$NomsEtudiants(n) \leftarrow Etudiants(e,n,d)$

Entrepôt de données *(data warehouse)* (1/5)

Définitions

Entrepôt de données (ou BD multidimensionnelle) : serveur [de données] (souvent parallèle) gérant des données :

Avec historique (avec ainsi la possibilité de voir l'évolution au cours du temps ; N. B. : les suppressions sont effectuées en fonction de la durée définie par la fenêtre de rémanence)

Organisées par sujets spécifiques (exemple : tous les comptes d'un client pour une banque)

Consolidées à partir de bases applicatives hétérogènes dans une BD unique (i. e. les informations disséminées dans les bases sources sont regroupées dans une base synthétisant ces informations)

Aidant à la prise de décision dans l'organisation (les responsables ont ainsi une vision globale du système d'information *via* des histogrammes, agrégats, fonctions statistiques) grâce à des modèles mathématiques permettant de décrire les évolutions constatées voire prédire les évolutions futures selon différentes hypothèses

Magasin de données *(datamart)*

Entrepôt de données ciblé sur un seul sujet

Infocentre

Technique consistant à recopier tout ou partie des BD pour les traitements décisionnels

N. B. : l'informatique décisionnelle *(Business Intelligence)* (BI) ou le management du système d'information *(Decision Support System)* (DSS) désigne les moyens, outils et méthodes qui permettent de collecter, consolider, modéliser et restituer les données d'une organisation afin d'offrir une aide à la décision et de permettre aux responsables de la stratégie d'entreprise d'avoir une vue d'ensemble de l'activité traitée

Logiciels : Business Objects [société éponyme ... SAP AG], Cognos [... IBM], Hyperion [... Oracle Corp.]

Problème

Cohabitation entre le transactionnel (données brutes éclatées dans de nombreuses relations suite à la normalisation) i. e. *On-Line Transaction Processing* (OLTP) et le décisionnel (données de synthèse élaborées) i. e. *On-Line Analysis Processing* (OLAP)

Caractéristiques	OLTP	OLAP	
Systèmes dits ...	Opérationnels	Stratégiques	
Niveau de gestion des données	Information	Connaissance	
Période faste d'utilisation	Jusqu'au milieu années 90	Depuis années 80	
Opérations typiques	Mises à jour	Analyse	
Type d'accès	Lecture/Écriture	Lecture	
Niveau d'analyse	Élémentaire	Global	
Écran	Fixe	Variable	
Quantité d'information échangée	Faible	Importante	
Orientation	Ligne	Multidimensionnelle	
Taille de la BD	100 Mo à 1 Go	1 Go à 1 To	(2000)
Ancienneté des données	Récente	Historique	
Dépenses (logiciels et services)		1,45 G€ +12% (2006)	

Entrepôt de données (2/5)

Conception : niveaux de schémas de données

Schéma d'intégration : schéma des données (souvent relationnel) de l'entrepôt de données, cible de l'intégration des bases sources

Schéma d'échange : schéma des données (souvent relationnel) spécifique à chaque base source, permettant l'échange des données de la base source (schéma exporté) vers l'entrepôt de données (schéma importé)

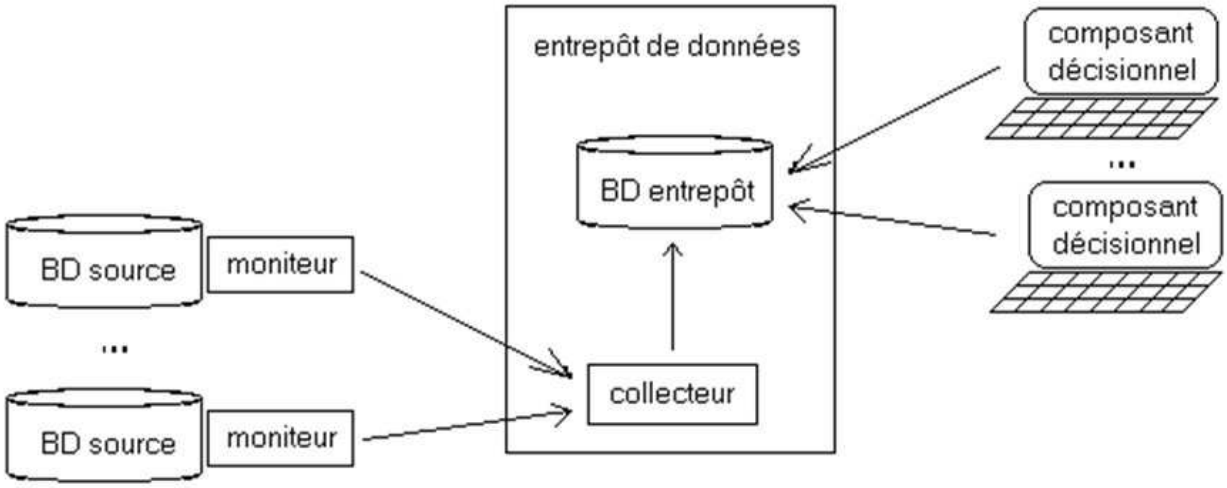
Schéma local : schéma des données d'une base source

Étapes pour constituer l'entrepôt de données

Chargement initial (souvent décomposé par lots de chargement partitionnés et triés compte-tenu des importants volumes de données à traiter et du souhait de ne pas perturber les applications gérant les bases sources)

Rafraîchissements (pour intégrer les mises à jour des bases sources en évitant notamment d'avoir à tout recalculer)

Fonctions



Extraction de données

Un moniteur détecte les mises à jour sur les bases sources (BD relationnelles ou autres, voire des fichiers) dites légataires (i. e. héritées du passé) afin de les envoyer vers l'entrepôt de données (approche *push*)

Collecteur

Intégration des mises à jour des différentes bases sources dans l'entrepôt de données (en respectant l'organisation par sujet)

Exploitation (analyse et fouille) des données

Recherche des informations, analyse en tendance (courbe d'évolution des données), aide à la décision (extrapolation), découverte de connaissances (règles, contraintes, tendances)

Formulation à l'aide de requêtes et d'une présentation graphique

Composants

Médiateur *(middleware)* d'extraction et de collecte

Composants du kit d'alimentation

Un moniteur de source qui détecte les mises à jour effectuées sur la base source et repère les données à envoyer à l'entrepôt de données pour son intégration ultérieure

Un adaptateur de source qui traduit les requêtes et les données de la base source vers l'entrepôt de données, et inversement

Un médiateur *(mediator)* qui donne une vision intégrée des différentes bases sources et en extrait des parties par des requêtes

Exemple de technique pour mettre à jour les données de l'entrepôt de données : utilisation des déclencheurs des SGBD des bases sources après chaque opération de mise à jour

Problème : les bases sources peuvent être gérées par un SGBDR ou un SGBD d'un autre type voire être des fichiers, et il faut alors interroger périodiquement les bases sources ou leurs journaux (approche *pull*) ; chaque opération de mise à jour est transformée en une suite de suppressions suivie d'une suite d'insertions pour chacune des relations impliquées, ces opérations élémentaires étant gérées dans des files d'attente persistantes (une par relation exportée) envoyées périodiquement à l'entrepôt de données pour y être importées

Architecture

Choix du couplage matériel/logiciel

ROLAP *(Relational OLAP)* pour un SGBDR : les données sont exploitées par des fonctions OLAP (exemple : `CUBE` du `GROUP BY`)

Schéma en étoile : une relation centrale contient k attributs sur lesquels sont effectués les calculs des fonctions d'agrégation ainsi qu'une référence (i. e. une clé étrangère) pour chacune des $d - k$ autres relations (une par dimension) qui quant à elles contiennent les attributs à différents niveaux de granularité

Exemple :

RelCentr (IdRelCentr , A_1 , ... , A_k , IdRelDimGéo , ...)
RelDimGéo (IdRelDimGéo , Pays , Département)
IdRelCentr est la clé de la relation RelCentr
IdRelDimGéo est la clé de la relation RelDimGéo
RelCentr.IdRelDimGéo référence RelDimGéo.IdRelDimGéo

Schéma en flocon : schéma en étoile normalisant chacune des $d - k$ autres informations des dimensions par autant de relations que nécessaire (souvent une chaîne de relations, chacune correspondant à un niveau de granularité)

Exemple :

RelCentr (IdRelCentr , A_1 , ... , A_k , IdRelDimGéoDpt , ...)
RelDimGéoDpt (IdRelDimGéoDpt , Département , IdRelDimGéoPays)
RelDimGéoPays (IdRelDimGéoPays , Pays)
IdRelCentr est la clé de la relation RelCentr
IdRelDimGéoDpt est la clé de la relation RelDimGéoDpt
IdRelDimGéoPays est la clé de la relation RelDimGéoPays
RelCentr.IdRelDimGéoDpt référence RelDimGéoDpt.IdRelDimGéoDpt
RelDimGéoDpt.IdRelDimGéoPays référence RelDimGéoPays.IdRelDimGéoPays

MOLAP *(Multidimensional OLAP)* ou SGBD multidimensionnel : technique implantant directement les fonctions OLAP en mémoire principale, avec des structures de données persistantes adaptées

MROLAP *(Multidimensional Relational OLAP)* : couche multidimensionnelle apportée à un SGBDR

Serveur parallèle de BD

À mémoire partagée (jusqu'à dix processeurs à cause de la contention mémoire) ou à mémoire distribuée *(share nothing)* (pour du parallélisme massif : des centaines de processeurs)

Exemple : architecture de type grappe *(cluster)*, composée d'un réseau en forme d'hypercube dont chaque élément est une machine multiprocesseurs à mémoire ; il existe de telles machines capables de supporter les versions parallèles d'Oracle, d'Informix ou de DB2

Doit supporter du parallélisme inter-requêtes (en faisant prendre en charge des requêtes simultanément issues de clients différents par des processeurs différents) et intra-requêtes (en découpant une requête en unités fonctionnelles (projection, sélection, jointure, tri, etc.), chaque unité pouvant être prise en charge par un ou plusieurs processeurs différents travaillant en parallèle)

Exploitation

Analyse (de données) multidimensionnelle

En plus de toutes les dimensions de l'analyse, il faut ajouter une dimension temporelle (historisation des données, gestion des versions)

Problème : manipulation (et visualisation) des données

Outils OLAP pour l'analyse de données historisées

Cube de données *(datacube)*

Cube de données (de dimension d) : représentation de d attributs sous forme d'un cube multidimensionnel, ayant $d-k$ attributs sur lesquels les agrégations sont possibles, les k autres attributs étant les résultats des calculs des fonctions d'agrégation (souvent, $k=1$)

Vue d'un cube de données : vue définie à partir d'un cube de données par agrégation des quantités selon l'un des 2^{d-k} sous-ensembles d'i attributs (avec $0 \leq i \leq d-k$) choisis parmi les $d-k$ possibles

Les données sont organisées pour mettre en évidence le sujet analysé et les axes de l'analyse ; une donnée est un point dans un espace à plusieurs dimensions

Granularité des dimensions : visibilité selon différents niveaux d'agrégation (exemples : axe temporel agrégé par jour ou semaine ou mois ou trimestre ou année, axe géographique agrégé par ville ou département ou région ou pays ou continent, axe produit agrégé par numéro ou type ou gamme ou marque, etc.)

Opérateurs permettant de passer d'une vue à une autre

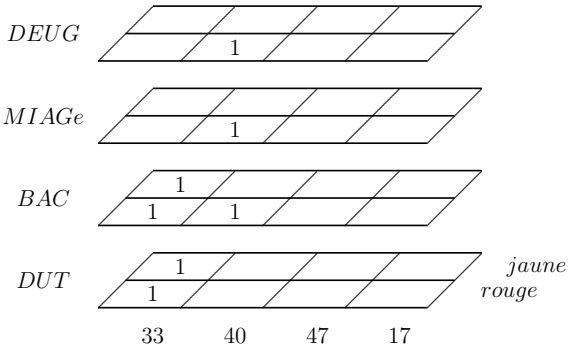
Dépliage *(drilldown)* i. e. l'extension d'une dimension du cube par une dimension à grain plus fin ou pliage *(rollup)* i. e. la réduction d'une dimension du cube par une dimension à grain plus large

Coupe *(slice)* : sélection (par prédicat) de tranches du cube selon une dimension

Exemple : distribution du nombre d'étudiants selon le département de naissance (quatre valeurs : 33, 40, 47 et 17), l'intitulé abrégé des diplômes (quatre valeurs : *'DUT'*, *'BAC'*, *'MIAGe'* et *'DEUG'*) et la couleur des voitures (deux valeurs : *'rouge'* et *'jaune'*)

```
SELECT DepartNaissEtu , IntitAbrege , Couleur , COUNT(*)
FROM Etudiants NATURAL JOIN AvoirObtenu NATURAL JOIN Voitures
GROUP BY DepartNaissEtu , IntitAbrege , Couleur
```

DepartNaissEtu	IntitAbrege	Couleur	COUNT(*)
33	'DUT'	'rouge'	1
33	'BAC'	'rouge'	1
40	'DEUG'	'rouge'	1
40	'MIAGe'	'rouge'	1
40	'BAC'	'rouge'	1
33	'DUT'	'jaune'	1
33	'BAC'	'jaune'	1



Généralités

Fouille (ou forage ou exploitation) de données *(datamining)*

Ensemble de techniques d'exploitation de larges BD afin d'en tirer les liens sémantiques significatifs et plus généralement des règles et des modèles pour la compréhension et l'aide à la décision

Exemple

Une chaîne de détail, les magasins Walmart des États-Unis d'Amérique, a su expliquer, en analysant les tickets de caisse, la corrélation entre la vente de couches-culottes pour bébés et de bière en fin de journée : l'achat des lourds paquets de couches était délégué au mari à son retour du travail

Domaines d'application

Analyse de risque, marketing direct, grande distribution, gestion des stocks, maintenance, contrôle qualité, médical, analyse financière, etc.

Approche

Découverte de modèles (non exacts et donc avec des coefficients de vraisemblance et des probabilités) automatiquement extraites de grandes BD historisées (souvent depuis les entrepôts de données) de sorte à induire des règles et obtenir de nouvelles connaissances (compréhension, prédiction)

Techniques issues de l'intelligence artificielle et de l'analyse de données

Exemples : apprentissage, visualisation, statistiques, réseaux neuronaux, systèmes à base de règles, BD, analyse de données, etc.

Méthode

Modèles type

Modèle fonctionnel

Technique permettant d'approcher une dépendance entre n variables d'entrées et une variable de sortie par une fonction, avec un éventuel indicateur de confiance i. e. $y = f(x_1, \ldots, x_n)$

Exemples : régression linéaire, réseaux neuronaux

Modèle logique

Technique permettant d'exprimer des relations entre données par un ensemble de règles du type `si` ◁ prémisse ▷ `alors` ◁ conclusion ▷, éventuellement pondérable par un indicateur de confiance

Exemples : arbres de décision, règles associatives

Démarche générale

Identifier le problème : cerner les objectifs

Préparer les données d'entrée : trouver les sources, collecter les données, nettoyer les données, transformer les données, intégrer les données

Explorer plusieurs modèles : choisir une technique, échantillonner sur un groupe, valider sur le reste de la population, calculer le pourcentage d'erreurs

Utiliser le modèle sur le réel : observer la réalité, recommander des actions

Suivre et améliorer le modèle : bâtir des estimateurs, corriger et affiner le modèle

Techniques principales (1/3)

Analyse statistique ou probabiliste

Régression linéaire : approximation par une droite d'une relation entre plusieurs quantités

Régression logistique : cas discret de la régression linéaire, avec des variables binaires

Test du χ^2 : détermination de l'existence d'une dépendance entre deux variables

Réseaux bayésiens : prédiction du futur à partir du passé, en supposant la reproductibilité des probabilités

Réseaux neuronaux

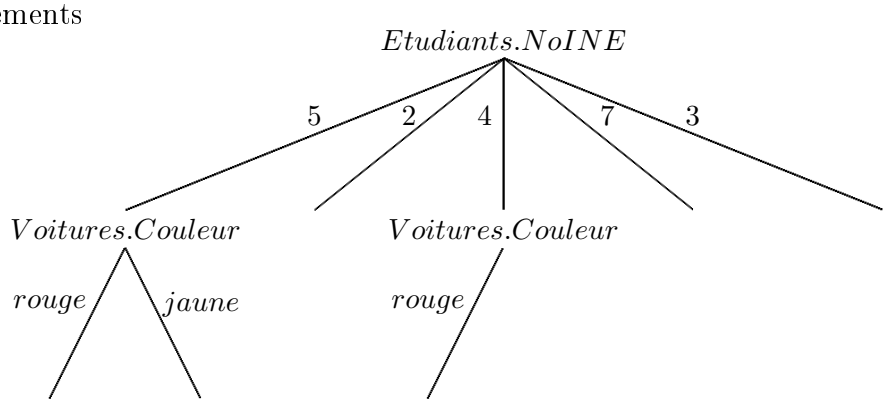
Réseau à plusieurs entrées et plusieurs sorties, composé de cellules interconnectées généralement organisées en couches, chaque cellule obtenant une valeur de sortie en appliquant une fonction de transfert (souvent une fonction symoïde i. e. une fonction de seuil) à une combinaison linéaire de ses entrées i. e. $s = f(\sum_{i=1}^{n} \omega_i e_i)$

Remarque : souvent trois couches (codage, intelligence, préparation des sorties)

Segmentation des données

Préambule : l'objectif de la segmentation est de classer les éléments

Arbre de décision : arbre classant les éléments en sous-classes par division hiérarchique où un nœud est une sous-classe du père et un arc est la valeur d'un prédicat de placement des éléments



Exemple :

La construction de l'arbre est effectuée par division récursive des nœuds en cherchant à chaque étape l'attribut le plus discriminant, par exemple en minimisant l'évolution de l'entropie ($E(X) = -\sum_i p(x_i) \log p(x_i)$ qui note l'incertitude sur la valeur de la variable X i. e. mesure ici le désordre introduit par un découpage en classes) à chaque niveau

Un élagage, basé sur un modèle de coût pour déterminer l'utilité d'une division et décider s'il faut l'éliminer, est ensuite effectué récursivement depuis les feuilles

Réseaux d'agents

Techniques principales (2/3)

Groupage des données

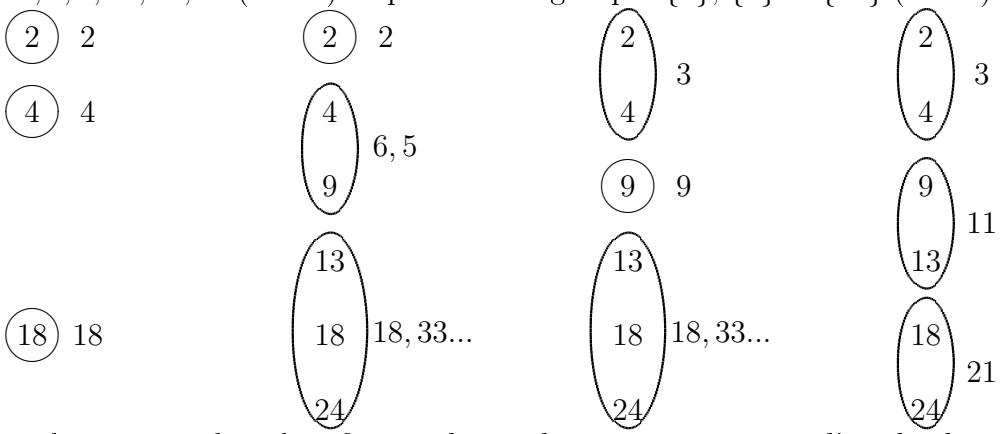
Préambule : un groupage est une méthode consistant à former des groupes d'éléments similaires à partir d'une fonction de distance i. e. une fonction d telle que $d(x,x)=0$, $d(x,y)>0$ si $x \neq y$, $d(x,y)=d(y,x)$ et $d(x,y) \leq d(x,z)+d(z,y)$

Méthode des k-moyennes *(k-means)* dite des centres mobiles : regroupement de n éléments dans k groupes tel que chaque élément appartient au groupe dont il est à une distance minimale du centre, le centre d'un groupe étant la moyenne des éléments du groupe

Algorithme :

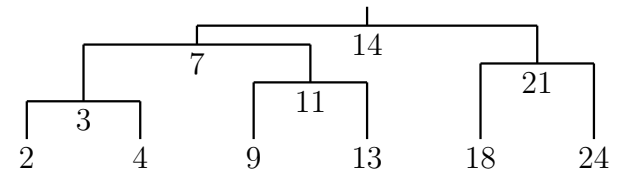
Choisir k des n éléments pour former les k groupes (d'1! élément) de départ
Affecter chacun des $n-k$ autres éléments dans groupe de centre le + proche
Calculer le centre de chaque groupe
Tant qu'il existe un élément n'appartenant pas au bon groupe faire
// i. e. $\exists e \in G_i$ (i^e groupe) , $j \neq i$ / $d(e,C_i) > d(e,C_j)$
// où C_l est le centre de G_l
Affecter chaque élément dans le groupe de centre le plus proche
Calculer le centre de chaque groupe

Exemple : $2,4,9,13,18,24$ ($n=6$) en partant des groupes $\{2\}$, $\{4\}$ et $\{18\}$ ($k=3$)



Méthode par agglomération dite classification hiérarchique : constitution d'un dendrogramme i. e. d'un arbre par extraction récursive des éléments les plus proches

Exemple : $2,4,9,13,18,24$ donne



Méthode par voisinage dense : création du voisinage d'un élément en commençant par ajouter tous les éléments qui sont à une distance inférieure à un seuil fixé et en itérant pour les éléments obtenus ; un voisinage est dense s'il contient plus d'éléments qu'un nombre fixé

Algorithme des k plus proches voisins

Techniques principales (3/3)

Découverte des règles associatives

Définitions

Règle associative : règle du type `si` X `alors` Y i. e. $X \Longrightarrow Y$ où X et Y sont des ensembles traduisant le fait que si X appartient à une transaction alors Y appartient à la même transaction avec une certaine probabilité

Support d'une règle associative : mesure indiquant le pourcentage de transactions vérifiant une règle associative ; $Support(X \Longrightarrow Y) = \frac{nombre\ de\ transactions(X \cup Y)}{nombre\ de\ transactions\ de\ la\ BD}$

Confiance en une règle associative : mesure indiquant le pourcentage de transactions vérifiant la conclusion parmi celles vérifiant la prémisse ; $Confiance(X \Longrightarrow Y) = \frac{nombre\ de\ transactions(X \cup Y)}{nombre\ de\ transactions(X)}$

Objectif

Rechercher les règles associatives ayant un support supérieur à un minimum (dit support fréquent) et une confiance supérieure à un minimum lui-même supérieur à celui du support

Principe

Calculer les supports pour tous les ensembles possibles en ne retenant que ceux de support fréquent

Comme tous les sous-ensembles d'un ensemble fréquent sont fréquents, l'algorithme dit Apriori [Agrawal, 1994] consiste à rechercher successivement des ensembles de taille 1, 2, ... en essayant de fusionner deux ensembles ne différant que d'un seul élément

D'autres algorithmes rendent ce traitement plus efficace : Apriori partitionné, comptage dynamique, basé sur des index *bitmap*

Améliorations

Conviction d'une règle associative : mesure la dépendance entre la prémisse et la négation de la conclusion (de 1 à ∞ i. e. du plus indépendant au plus dépendant) ; $Conviction(X \Longrightarrow Y) = \frac{P(X).P(\neg Y)}{P(X \bigwedge \neg Y)}$

Élimination des règles redondantes

BD du *World Wide Web*

Le couplage entre les SGBD et Internet est nécessaire pour (au moins) les types d'applications suivantes :

- Architecture à trois niveaux *(three-tiered architecture)* ou C/S Web
  - Un client (navigateur Web)
  - Un serveur d'application (serveur Web et services applications)
  - Un serveur de données (SGBD)
  - Remarque : le serveur *HyperText Transfer Protocol* (HTTP), protocole C/S définit pour le Web au dessus de TCP/IP, sert de passerelle entre le Web et la BD
- Génération de sites Web pour gérer le schéma (dynamique) des fichiers *HyperText Markup Language* (HTML) liés
- Commerce électronique

Les données échangées sur Internet sont de nature différente de celles gérées par les SGBD (on parle d'hypermédia)

BD multimédia (cf. norme ISO/IEC SQL/MM)

Définitions

Multimédia : au moins deux médias parmi texte, image, son, vidéo voire géographique ou géométrique, etc.

Hypermédia : mot-valise obtenu à partir de hypertexte et de multimédia

Objectifs

Stocker efficacement des objets multimédias

Effectuer efficacement des recherches par le contenu

Stockages d'objets multimédias

Champ binaire long *(Binary Large Object)* (BLOB) ou champ chaîne de caractères long *(Character Large Object)* (CLOB)

Raster i. e. matrice binaire (par exemple, pour une image)

Vecteur : description du contenu des objets par des éléments géométriques (point dans 2 ou 3 dimensions, vecteur i. e. un point de départ et un point d'arrivée, ligne i. e. une liste de points, polygone i. e. une ligne fermée, surface à trous i. e. liste de polygones positifs ou négatifs) et d'une structure topologique voire des relations sémantiques (inclusion, intersection, adjacence, élimination des redondances)

Balise où tout objet complexe est représenté comme une agrégation d'objets élémentaires par un graphe orienté, les sommets pour les composants et les arcs pour les compositions

Dimensions d'un système multimédia

Contenu : BLOB et CLOB, intégrant éventuellement leur organisation sous forme de collections imbriquées par exemple

Espace (dimension spatiale) : pour les données géographiques voire les images décomposées en éléments plus ou moins rapprochés

Scénario : description des interactions avec l'utilisateur

Structure : langages à balise *(markup languages)* pour mémoriser les objets multimédias

Les techniques utilisées relèvent souvent de l'IA et de la reconnaissance de formes et considèrent par exemple : les extensions des B-arbres à des données multidimensionnelles (points et volumes à plusieurs dimensions), des algorithmes géométriques, les algorithmes d'extraction de formes ou de contours ou de propriétés caractéristiques, des recherches exactes ou approximatives, la structuration de documents, etc.

Indexation d'objets multimédias

Arbre quadrant *(quadtree)* : image décomposée récursivement en quatre blocs de même taille si elle n'est pas homogène

Arbre R *(R-tree)* : inclusion de rectangles (avec chevauchements possibles au niveau des nœuds), les feuilles correspondant au plus petit rectangle englobant l'objet géométrique

Arbre R* *(R*-tree)* : arbre R dont les sommets, lors d'une insertion ou d'un éclatement, sont choisis de façon à minimiser les surfaces de recouvrement

Arbre k-dimensionnel *(K-D-tree)* : arbre binaire stockant des points de dimension d, où la dimension $n \mod d$ est le critère de branchement gauche ou droit d'un sommet de niveau n

Arbre SS *(SS-tree)* : arbre R* avec des sphères (et non des rectangles) englobantes, avec une stratégie d'éclatement assurant un voisinage isotrope (i. e. qui a les mêmes propriétés, dans toutes les directions) ; utilisation, par exemple, pour la recherche de points similaires à un point donné

BD géographique (cf. norme ISO/IEC Geographic information & SQL) et spatiale

Définitions

Objet spatial : objet 3D (voire 2,5D i. e. objet 2D muni d'une 3^e coordonnée)

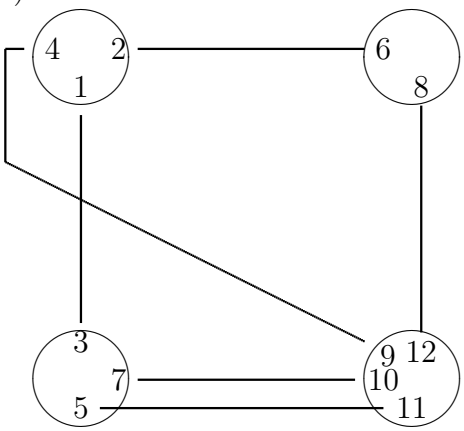
Système d'Information Géographique (SIG) : système spécialisé de gestion de cartes

Aspects géométriques : définition des éléments spatiaux (point, ligne, surface, volume)

Aspects topologiques : description des liens entre les éléments spatiaux (intérieur/extérieur, coupe, adjacence ... ou toute relation invariante par translation, rotation, changement d'échelle)

Codage d'une carte par une permutation (codant les sommets) et une involution sans point fixe (codant les arêtes) transitives (i. e. que tout élément peut atteindre tout élément)

Exemple : $\sigma = (1,2,4)(3,5,7)(6,8)(9,10,11,12)$ et
$\alpha = (1,3)(2,6)(4,9)(5,11)(7,10)(8,12)$



Contre-exemple : $\sigma = (1,2)(3)(4)$ et $\alpha = (1,2)(3,4)$

Interrogation

Algèbre spatiale : algèbre relationnelle disposant de prédicats sur les fonctions spatiales

Fonctions géométriques : union, intersection, inclusion, frontière

Fonctions géographiques : différence, intersection, adjacence

Fonctions extractives : sous-région, ligne, point

Fonctions de mesure : longueur, surface, diamètre

Projection spatiale : fonctions extractives et de mesure

Sélection spatiale (exemple : clipage i. e. sélection des objets appartenant à une fenêtre rectangulaire définie par l'utilisateur)

Jointure spatiale (par comparaison de deux attributs géométriques) dont la jointure de chevauchement

Requêtes

Requête factuelle (sur les données alphanumériques)

Requête géométrique (exemples : intersection, distance au plus égale à, etc.)

Requête topologique (éventuellement *via* des requêtes géométriques plus complexes)

Requête récursive (exemple : plus court des chemins allant du département Informatique de l'IUT Bordeaux 1 au Restaurant Universitaire n° 3 du campus bordelais)

BD textuelle

Objectif

Interroger par mots-clés un ensemble de documents (textuels)

Principe

Gestion d'un thésaurus i. e. d'un dictionnaire des mots significatifs, synonymes, polysèmes (mots ayant plusieurs sens différents), et les différentes formes (genre et nombre, conjugaisons, etc.) d'un mot

Interrogation (i. e. recherche documentaire)

Utilisation de connecteurs (et, ou, puis, sans), pondération des mots selon leur importance, mots exacts ou approchants, synonymes ou homonymes/homophones/homographes/polysèmes ou etc.

Il faut maximiser le nombre de réponses correctes ($\frac{nombre\ de\ réponses\ correctes}{nombre\ de\ documents\ pertinents}$) et minimiser le bruit ($1 - \frac{nombre\ de\ réponses\ correctes}{nombre\ de\ réponses}$)

Techniques

Liste inverse

Principe : liste des mots significatifs, avec pour chacun la liste des documents le contenant (et la fréquence d'apparition)

Signature

Principe : la recherche de mots-clés, après avoir appliqué la fonction de hachage, correspond à un filtre binaire dans le fichier des signatures contenant la signature de chaque document

La signature d'un document est une chaîne binaire de longueur fixe correspondant à l'union des chaînes obtenues après avoir appliqué une fonction de hachage sur les mots significatifs appartenant au document

Matrice des fréquences relatives

Matrice à deux dimensions : nombre de documents et nombre de mots significatifs

Objectif : les fréquences relatives tiennent compte de l'importance des mots

Calcul d'une distance de similarité pour rechercher les d documents les plus proches (matrice) d'une question (vecteur car la question est équivalente à un document)

Problème : matrice volumineuse

BD d'images (voire de vidéos)

Objectif

Recherche les i images les plus proches d'une image de référence

Principes

Indexer (pour effectuer des recherches sur le contenu)

Description manuelle : mots-clés et légendes

Indexation automatique : signature, distribution des couleurs, texture des images (exemple : extraction des granularités répétitives *via* des transformées de Fourier), extraction de sous-objets, détection des contours, reconnaissance de formes

Déterminer les classes de thèmes

Thème : caractéristique d'une image

Classe de thème : ensemble de thèmes formant une interprétation d'une image dans un domaine particulier

Exemples : couleur (globale ou locale i. e. proportion de chaque couleur dans l'image ou dans une localisation de l'image respectivement), texture (i. e. canevas répétitifs), forme (angles, contours), etc.

N. B. : à chaque image correspond un (ou plusieurs) vecteur(s) de thèmes

Techniques

Calcul d'une distance entre les vecteurs de thèmes

Calcul du coût de transformation d'un vecteur de thèmes en un autre

Fonctions

Manipulations de base : tourner, translater, « cliper » un rectangle d'une image

Amélioration des couleurs : améliorer le contraste, peindre avec une nouvelle palette

Composition d'images : superposer, soustraire, intersecter

Extraction de thèmes : histogramme des couleurs ou des angles, contours par lignes brisées, spectre caractéristique de la texture

Mesure de similarité

Techniques de transformations

Division en rectangles homogènes i. e. recherche des plus grandes sous-matrices homogènes d'une matrice

Transformation de Fourier discrète (pour calculer des coefficients caractéristiques à partir d'une matrice)

Transformation cosinus discrète i. e. transformation de Fourier discrète dans $\mathbb{R}$ (et non dans $\mathbb{C}$) (pour calculer un spectre)

Transformation rotationnelle (pour comparer des polygones)

Données semi-structurées

Données complexes, hétérogènes, distribuées, à structure variable voire incomplète (les modèles de données n'imposent pas de structure *a priori* mais la définissent dans les données elles-mêmes *a posteriori*)

Données représentées en *eXtensible Markup Language* (XML) (cf. norme ISO/IEC SQL/XML), standard d'échange de données sur le Web, permettant une modélisation flexible (exemple : balises non prédéfinies) de documents à structure irrégulière (autorisant cependant la définition de règles d'intégrité sur la structure ou encore des méta-informations)

Exemples de langages d'accès aux données semi-structurées (i. e. d'interrogation d'un document XML) : *XML Query Language* (XQL) qui est un sur-ensemble de XPath, XQuery qui a une syntaxe proche de SQL, etc.

XQL

Le document XML est composé de deux parties : la structure logique (la *Document Type Definition* (DTD) définissant la structure du document) suivie du document physique (l'instance, composée d'éléments imbriqués)

Quelques éléments du langage :

`/` : exploration du niveau depuis la racine ou du nœud courant

`//` : exploration de tous les niveaux depuis la racine ou du nœud courant

`*` : étiquette quelconque

`@` : attribut (d'une feuille)

`.` : élément courant

`[...]` : filtre

comparaisons

opérations booléennes (négation, et logique, ou logique)

opérations ensemblistes (union, intersection)

`{...}` ou `(...)` : restructuration du résultat *(grouping operator)*

séquence (pour redéfinir l'ordre des éléments)

fonction prédéfinie ou définie par l'utilisateur

jointure

produit cartésien

`->` : renommage

etc.

XQL : l'exemple « jouet » (1/5)

La DTD

```
<?xml version="1.0"?>
<!DOCTYPE DOCUMENT [
<!ELEMENT DOCUMENT (Etudiants,Diplomes)*>
    <!ELEMENT Etudiants (NoINE,NomEtu,DepartNaissEtu,VoituresPossedees,
                         DiplomesObtenus)>
        <!ELEMENT NoINE (#PCDATA)>
        <!ELEMENT NomEtu (#PCDATA)>
        <!ELEMENT DepartNaissEtu (#PCDATA)>
        <!ELEMENT VoituresPossedees (Voitures)*>
            <!ELEMENT Voitures (NoImmatChiffres,NoImmatLettres,
                                NoImmatDepart,Couleur)>
                <!ELEMENT NoImmatChiffres (#PCDATA)>
                <!ELEMENT NoImmatLettres (#PCDATA)>
                <!ELEMENT NoImmatDepart (#PCDATA)>
                <!ELEMENT Couleur (#PCDATA)>
        <!ELEMENT DiplomesObtenus (AvoirObtenu)*>
            <!ELEMENT AvoirObtenu (IntitAbrege,Annee)>
                <!ELEMENT IntitAbrege (#PCDATA)>
                <!ELEMENT Annee (#PCDATA)>
    <!ELEMENT Diplomes (IntitAbrege,IntitComplet)>
        <!ELEMENT IntitAbrege (#PCDATA)>
        <!ELEMENT IntitComplet (#PCDATA)>
]>
```

Le document physique (1/3)

```
<DOCUMENT>
<Etudiants>
    <NoINE> 7 </NoINE>
    <NomEtu> LEROI </NomEtu>
    <DepartNaissEtu> 33 </DepartNaissEtu>
</Etudiants>
<Etudiants>
    <NoINE> 3 </NoINE>
    <NomEtu> DUPOND </NomEtu>
    <DepartNaissEtu> 17 </DepartNaissEtu>
    <DiplomesObtenus>
        <AvoirObtenu>
            <IntitAbrege> DUT </IntitAbrege>
            <Annee> 1983 </Annee>
        </AvoirObtenu>
        <AvoirObtenu>
            <IntitAbrege> MIAGe </IntitAbrege>
            <Annee> 1985 </Annee>
        </AvoirObtenu>
        <AvoirObtenu>
            <IntitAbrege> BAC </IntitAbrege>
            <Annee> 1981 </Annee>
        </AvoirObtenu>
    </DiplomesObtenus>
</Etudiants>
```

XQL : l'exemple « jouet » (2/5)

Le document physique (2/3)

```
<Etudiants>
    <NoINE> 5 </NoINE>
    <NomEtu> DURAND </NomEtu>
    <DepartNaissEtu> 33 </DepartNaissEtu>
    <VoituresPossedees>
        <Voitures>
            <NoImmatChiffres> 3333 </NoImmatChiffres>
            <NoImmatLettres> BX </NoImmatLettres>
            <NoImmatDepart> 33 </NoImmatDepart>
            <Couleur> rouge </Couleur>
        </Voitures>
        <Voitures>
            <NoImmatChiffres> 4040 </NoImmatChiffres>
            <NoImmatLettres> NT </NoImmatLettres>
            <NoImmatDepart> 40 </NoImmatDepart>
            <Couleur> jaune </Couleur>
        </Voitures>
    </VoituresPossedees>
    <DiplomesObtenus>
        <AvoirObtenu>
            <IntitAbrege> DUT  </IntitAbrege>
            <Annee> 1983 </Annee>
        </AvoirObtenu>
        <AvoirObtenu>
            <IntitAbrege> BAC </IntitAbrege>
            <Annee> 1981 </Annee>
        </AvoirObtenu>
    </DiplomesObtenus>
</Etudiants>
<Etudiants>
    <NoINE> 2 </NoINE>
    <NomEtu> LEROI </NomEtu>
    <DepartNaissEtu> 40 </DepartNaissEtu>
    <DiplomesObtenus>
        <AvoirObtenu>
            <IntitAbrege> DEUG </IntitAbrege>
            <Annee> 1982 </Annee>
        </AvoirObtenu>
        <AvoirObtenu>
            <IntitAbrege> BAC </IntitAbrege>
            <Annee> 1980 </Annee>
        </AvoirObtenu>
    </DiplomesObtenus>
</Etudiants>
```

XQL : l'exemple « jouet » (3/5)

Le document physique (3/3)

```
<Etudiants>
    <NoINE> 4 </NoINE>
    <NomEtu> MARTIN </NomEtu>
    <DepartNaissEtu> 47 </DepartNaissEtu>
    <VoituresPossedees>
        <Voitures>
            <NoImmatChiffres> 4747 </NoImmatChiffres>
            <NoImmatLettres> LA </NoImmatLettres>
            <NoImmatDepart> 47 </NoImmatDepart>
            <Couleur> rouge </Couleur>
        </Voitures>
    </VoituresPossedees>
    <DiplomesObtenus>
        <AvoirObtenu>
            <IntitAbrege> DEUG </IntitAbrege>
            <Annee> 1980 </Annee>
        </AvoirObtenu>
        <AvoirObtenu>
            <IntitAbrege> MIAGe </IntitAbrege>
            <Annee> 1982 </Annee>
        </AvoirObtenu>
        <AvoirObtenu>
            <IntitAbrege> BAC </IntitAbrege>
            <Annee> 1977 </Annee>
        </AvoirObtenu>
    </DiplomesObtenus>
</Etudiants>
<Diplomes>
    <IntitAbrege> DUT </IntitAbrege>
    <IntitComplet> Diplôme Universitaire de Technologie </IntitComplet>
</Diplomes>
<Diplomes>
    <IntitAbrege> BAC </IntitAbrege>
    <IntitComplet> Baccalauréat </IntitComplet>
</Diplomes>
<Diplomes>
    <IntitAbrege> MIAGe </IntitAbrege>
    <IntitComplet> Maîtrise des Méthodes Informatiques Appliquées
                   à la Gestion des Entreprises </IntitComplet>
</Diplomes>
<Diplomes>
    <IntitAbrege> DEUG </IntitAbrege>
    <IntitComplet> Diplôme d'Études Universitaires Générales </IntitComplet>
</Diplomes>
</DOCUMENT>
```

XQL : l'exemple « jouet » (4/5)

Interrogations (1/2)

Les couleurs des voitures (projection)

`/Etudiants/VoituresPossedees/Voitures/Couleur` ou `//Couleur`

```
<xql:result>
    <Couleur> rouge </Couleur>
    <Couleur> jaune </Couleur>
    <Couleur> rouge </Couleur>
</xql:result>
```

Les voitures rouges (sélection)

```
//Voitures[@Couleur='rouge']
```

```
<xql:result>
    <Voitures>
        <NoImmatChiffres> 3333 </NoImmatChiffres>
        <NoImmatLettres> BX </NoImmatLettres>
        <NoImmatDepart> 33 </NoImmatDepart>
        <Couleur> rouge </Couleur>
    </Voitures>
    <Voitures>
        <NoImmatChiffres> 4747 </NoImmatChiffres>
        <NoImmatLettres> LA </NoImmatLettres>
        <NoImmatDepart> 47 </NoImmatDepart>
        <Couleur> rouge </Couleur>
    </Voitures>
</xql:result>
```

Les immatriculations des voitures (projection)

```
//Voitures { NoImmatChiffres | NoImmatLettres | NoImmatDepart }
```

```
<xql:result>
    <Voitures>
        <NoImmatChiffres> 3333 </NoImmatChiffres>
        <NoImmatLettres> BX </NoImmatLettres>
        <NoImmatDepart> 33 </NoImmatDepart>
    </Voitures>
    <Voitures>
        <NoImmatChiffres> 4040 </NoImmatChiffres>
        <NoImmatLettres> NT </NoImmatLettres>
        <NoImmatDepart> 40 </NoImmatDepart>
    </Voitures>
    <Voitures>
        <NoImmatChiffres> 4747 </NoImmatChiffres>
        <NoImmatLettres> LA </NoImmatLettres>
        <NoImmatDepart> 47 </NoImmatDepart>
    </Voitures>
</xql:result>
```

XQL : l'exemple « jouet » (5/5)

Interrogations (2/2)

Les étudiants et leurs diplômes (jointure)

```
/Etudiants[$d:=DiplomesObtenus/AvoirObtenu/IntitAbrege] {
    NoINE | NomEtu | DepartNaissEtu |
    DiplomesObtenus/AvoirObtenu/IntitAbrege |
    DiplomesObtenus/AvoirObtenu/Annee |
    //Diplomes[IntitAbrege=$d] { IntitAbrege | IntitComplet } }
```

Remarque : la jointure est effectuée en affectant (`:=`) l'intitulé abrégé du diplôme obtenu par les étudiants dans la variable `d` qui est ensuite comparée (`=`) à l'intitulé abrégé du diplôme des diplômes

Tous les étudiants et tous les diplômes (produit cartésien)

```
/Etudiants {
    NoINE | NomEtu | DepartNaissEtu |
    DiplomesObtenus/AvoirObtenu/IntitAbrege |
    DiplomesObtenus/AvoirObtenu/Annee |
    //Diplomes { IntitAbrege | IntitComplet } }
```

Introduction aux objets (1/3)

Historique

1966 : Simula (premier langage de programmation à objets)

1977 : *Third Generation of Database System Manifest* (première référence de SGBDO)

1989 : création de l'*Object Management Group* (OMG)

1991 : décollage de l'approche objet et création de l'*Object Database Management Group* (ODMG), groupe de standardisation regroupant cinq éditeurs de SGBDO (Object Design, O2 Technology, Objectivity, Ontos et Versant)

1994 : l'OMG accepte ODMG-93 comme service standard de persistance basé sur un modèle objet, des langages de définition/manipulation/interrogation d'objets *(Object Definition/Manipulation/Query Language)* (ODL/OML/OQL), et l'intégration de ces langages en Smalltalk et en C++

1996 : plus de 80 % des entreprises utilisent l'approche objet pour un chiffre d'affaires d'1G$

Différentes approches en programmation

- Approche objet
  - Héritage (et polymorphisme)
  - Approche à base d'objets
    - Type abstrait de données
    - Approche modulaire
      - Modules
      - Approche structurée

Objectif

Gain de productivité en développement (programmation modulaire et extensible), et en qualité et maintenance des applications

Avantages

Modularité (découpage en composants modélisant simultanément l'état et le comportement)

Réutilisabilité

Extensibilité/Évolutivité (bibliothèques)

Modélisation d'applications complexes

Représentation directe des entités du monde réel dans l'environnement informatique (sans la déformation inhérente à la normalisation)

Environnements de développement riches

Outils interactifs pour la création rapide d'interface homme-machine (IHM) graphique

Prototypage rapide d'applications

Exploitation du parallélisme (multi-processeurs ou distribué) lors de l'implantation

Inconvénients

Absence de théorie complète, voire de consensus sur une méthodologie purement objet

Technologie finalement assez récente, riche mais complexe (implantation difficile de certains mécanismes comme l'héritage multiple, les liaisons dynamiques, le ramasse-miettes, etc.)

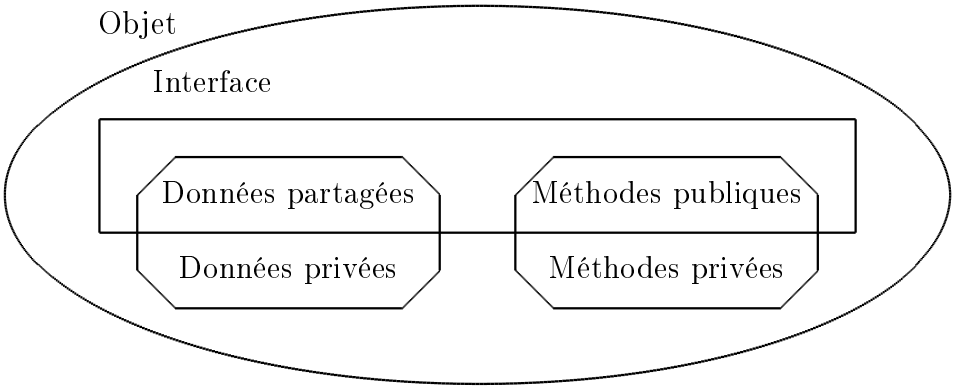
Compromis généralité/performances

Introduction aux objets (2/3)

Outils de développement orientés objets (part de marché)

- Langage de programmation orientés objets (pur objet [Smalltalk 20 %] ou hybride [C++ 75 %]) 40 %
- Ateliers de développement (applications C/S autour de BD relationnelles) : L4G, ADE 30 %
- SGBDO (SGBDOO ou SGBDRO) 15 %
- Outils méthodologiques de conception (AGL) 7 %
- Bibliothèques de classes 5 %
- Médiateur orienté-objet (pour systèmes distribués) 3 %

Principes



Abstraction — Classe ; Objet

Permet de s'intéresser au comportement des objets (par les opérations visibles définissant l'interface externe) indépendamment de leur représentation interne (l'implantation des structures de données associées aux opérations est cachée)

Encapsulation des données et des opérations (qui manipulent ces données) dans les objets ; la partie spécification d'un objet (données partagées et méthodes publiques) est accessible de l'extérieur *via* son interface contrairement à la partie implantation (données et méthodes privées)

Décomposition — Généralisation/Spécialisation ; Agrégation ; Héritage

Objets complexes décomposés en objets plus simples

Connexion — Association ; Lien

Relations ou interactions entre objets

Trois points de vue ont conduit à la notion d'objet

Structurel

L'objet est une instance d'un type de données (abstrait) caractérisé par une structure cachée (accessible par des opérations)

Conceptuel

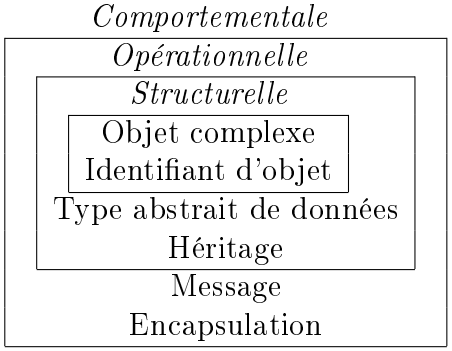
L'objet est un concept du monde réel pouvant être spécialisé

Acteur

L'objet est une entité autonome et active (qui intègre donc simultanément des aspects statiques et dynamiques) qui communique par échange de messages (qui constitue l'unique moyen pour assurer la communication entre les objets)

Introduction aux objets (3/3)

Différentes orientations



Ex. : ORION (1985), O2 (1986), Gemstone (1988)
Ex. : PostGres
Ex. : AIM-P

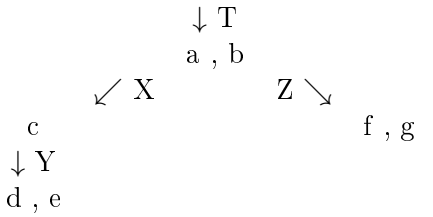
Objets complexes, en *Non First Normal Form* (NF^2)

Représentations

Tabulaire

T						
		X			Z	
			Y			
a	b	c	d	e	f	g

Arborescente



Relationnelle

T (a , b , X , Z)
X (c , Y)
Y (d , e)
Z (f , g)

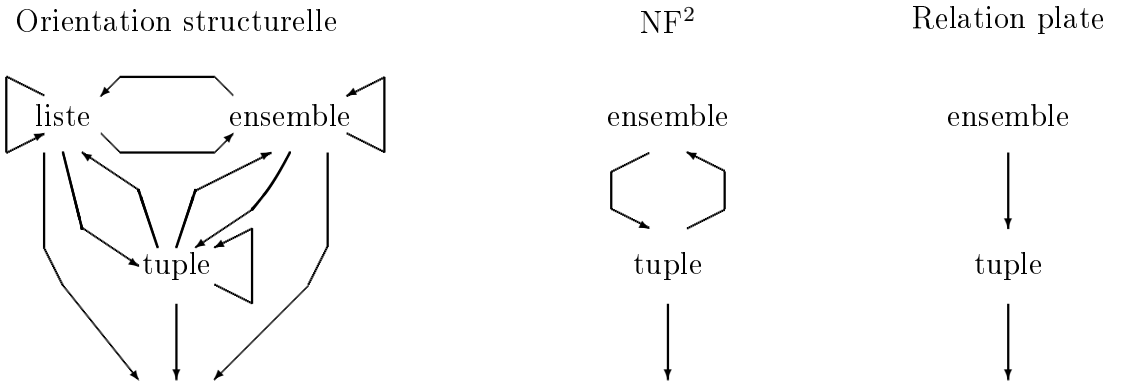
Relation complexe *vs* relation plate

Les objets sont conçus tels qu'ils sont (pour NF^2)

Absence de redondance (NF^2 relativement à 1FN)

Absence de jointure (NF^2 relativement à 4FN)

Définition des objets



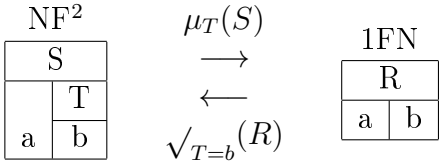
Récursivement par rapport à des objets atomique (ou type élémentaire), tuple ou ensemble voire liste

Algèbre relationnelle étendue

Projections et sélections sur des relations

Exemple : $\pi_{\sigma_{R=\emptyset}(S)}(T)$

Opération de restructuration



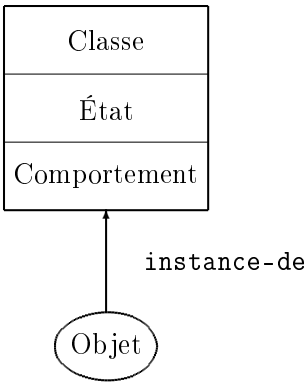
Jointure sur clé interne

Définitions sur les objets (1/3)

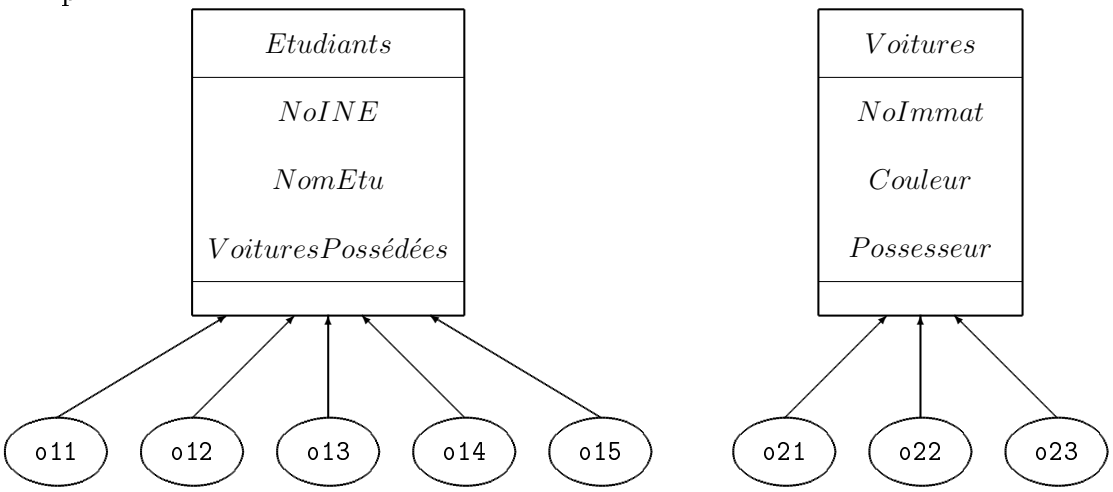
Instanciation

Relation entre un objet et sa classe d'appartenance (qui a permis de le créer)

Graphe d'instanciation



Exemples :



Objet

Abstraction d'une donnée caractérisée par :

Un identifiant *(object identifier)* unique et invariant

Un état représenté par une valeur simple ou structurée (composée de valeurs simples, de références ou de valeurs structurées)

Une classe d'appartenance définissant son comportement par un ensemble d'opérations applicables à l'objet

Instance d'une classe

Rôles : recevoir et analyser les messages extérieurs, envoyer des messages et créer de nouveaux objets

Exemples (représentation sous la forme `<oid,classe,état>`) ;
N. B. : on a ici introduit de la redondance entre les liens de possession des voitures par les étudiants

```
< o11 , Etudiants , ( 5 , 'DURAND' , 33 , [ o21 , o23 ] ) >
< o12 , Etudiants , ( 2 , 'LEROI' , 40 , [ ] ) >
< o13 , Etudiants , ( 4 , 'MARTIN' , 47 , [ o22 ] ) >
< o14 , Etudiants , ( 7 , 'LEROI' , 33 , [ ] ) >
< o15 , Etudiants , ( 3 , 'DUPOND' , 17 , [ ] ) >
< o21 , Voitures , ( '3333 BX 33' , 'rouge' , o11 ) >
< o22 , Voitures , ( '4747 LA 47' , 'rouge' , o13 ) >
< o23 , Voitures , ( '4040 NT 40' , 'jaune' , o11 ) >
```

Définitions sur les objets (2/3)

Attribut (ou variable d'instance)

Définit l'état des objets (aspect statique)

Caractérisé par un nom et un type (type de base ou classe)

Collection : valeur d'un attribut multivalué (tel que ensemble, multi-ensemble, tableau, liste, etc.)

Opération

Définit le comportement des objets (aspect dynamique)

Applicable à un objet de la classe, pouvant modifier tout ou partie de l'état de l'objet et retourner des valeurs calculées à partir de cet état

Publique (interface de la classe, accessible à tout utilisateur de la classe) ou privée (accessible qu'au programmeur de la classe)

Signature

Définit le type des arguments d'entrée et de sortie de l'opération

Envoi de message

Protocole de communication inter-objet par invocation d'opération publique par un message contenant l'identifiant de l'objet receveur, le nom de l'opération et ses arguments

Type abstrait de données

Définit un ensemble de fonctions manipulant un ensemble de sortes et spécifie formellement ces fonctions et leurs relations par des axiomes

Classe

Type abstrait de données caractérisé par des propriétés (attributs et opérations) communes à tous ses objets et un mécanisme (l'instanciation) permettant de créer des objets ayant ces propriétés (nouvel identifiant d'objet et structure de données correspondante)

L'ensemble des objets de la classe forme l'extension de la classe

Métaclasse

Classe générateur de classes, disposant d'opérations applicables aux classes (création et destruction d'instances, ajout ou suppression d'attribut, etc.)

Classe collection

Classe générique permettant d'imposer une structure (ensemble *(set)*, multi-ensemble *(bag)*, tableau *(array)*, liste *(list)*, etc.) à une collection d'objets et de les manipuler par des opérations associées (par exemple, pour un ensemble, l'union, l'appartenance, etc.)

Association

Relation entre plusieurs classes décrivant les liens entre les objets de ces classes

Cardinalité

Caractéristique d'une association permettant de contraindre le nombre d'objets maximum connectés à chaque objet

Agrégation

Relation entre deux classes spécifiant que des objets d'une classe (cible) sont les composants de l'autre classe (source)

Association `partie-de`

Définitions sur les objets (3/3)

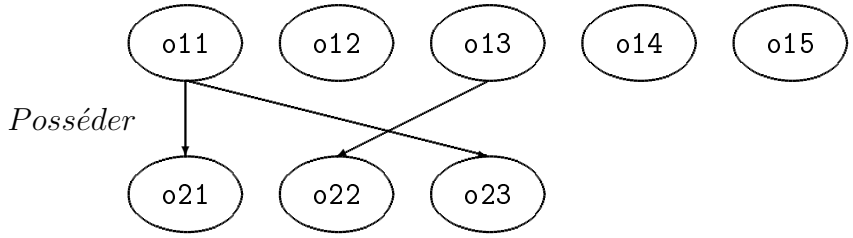
Lien

Instance d'une association

Monovalué (entre un objet source et un objet cible) ou multivalué (entre un objet source et plusieurs objets cibles)

Lien inverse (par exemple, il faut assurer la cohérence entre le fait qu'un étudiant possède 0, 1 ou plusieurs voitures et qu'une voiture est possédée par un étudiant)

Graphe d'objets : représentation graphique des liens entre objets



Expression de chemin (i. e. parcours d'un graphe d'objets) : $o.A_1.A_2.\ldots.A_i.\ldots.A_n$ (de l'identifiant d'objet o à la valeur atomique en passant par les attributs des objets $o.A_1.A_2.\ldots.A_i$ pour $1 \leq i < n$)

Exemples : `o11.NomEtu` retourne `'DURAND'`
`o21.Possesseur` retourne `o11`
`o21.Possesseur.NomEtu` retourne `'DURAND'`

Généralisation (inverse de la spécialisation)

Fonction qui à une classe d'origine (ou sous-classe ou classe inférieure ou classe dérivée) fait correspondre une classe plus générale (ou superclasse ou classe supérieure ou classe de base)

Association `est-un`

Spécialisation (inverse de la généralisation)

Fonction qui à une classe fait correspondre toutes ses sous-classes

Héritage

Mécanisme de transmission des propriétés d'une classe vers une sous-classe (les propriétés d'une sous-classe sont ses propriétés propres et celles de toutes les superclasses dont elle dépend directement ou par transitivité)

Simple (toute classe ne peut posséder qu'une superclasse directe) ou multiple (mécanisme par lequel une classe hérite des propriétés de plus d'une superclasse)

Polymorphisme (ou surcharge dynamique)

Faculté d'une opération à pouvoir s'appliquer à des objets de classes différentes (i. e. prendre plusieurs implantations en fonction du type d'objet auquel elle s'applique)

Surcharge

Redéfinition d'une opération héritée avec un code différent (mais de même signature, excepté pour l'argument de contrôle désignant l'objet)

Liaison dynamique

Mécanisme de sélection de code d'une opération à l'exécution en fonction de la classe d'appartenance de l'objet receveur

Mécanisme de ramasse-miettes

Destruction automatique des objets non référencés (i. e. objets créés sans avoir été libérés) par les variables du programme en cours d'exécution

Applications visées par les SGBDO

Gestions non classiques (et plus techniques) telles que l'ingénierie (Conception Assistée par Ordinateur (CAO), Fabrication Assistée par Ordinateur (FAO), etc.), bureautique, administration de réseau, etc.

Gestion d'applications hiérarchiques, de BD très volumineuses, etc.

BD Objet (BDO)

Organisation cohérente d'objets persistants (objets survivant à la fin de l'exécution du programme les ayant créés) et partagés par des utilisateurs concurrents

Objectifs

Accès (création, lecture, écriture, destruction) transparent aux objets temporaires et persistants

Réduction des dysfonctionnements causés par les conversions des types du langage de programmation et du SGBD

Schéma composé de classes et d'associations permettant de définir la structure (attributs, types) et le comportement (opérations) des objets

Concepts susceptibles d'être couverts par un modèle BDO (venant des langages à objets et des modèles de données sémantiques) : objet, classe, attribut, identité d'objet, héritage, classe dépendante, association, opération, encapsulation

SGBDO : SGBDRO et SGBDOO (systèmes à objets persistants et SGBDOO purs)

Un SGBDO est composé d'un SGBD classique (intégrité des données, durabilité des données, gestion des transactions, indépendance physique des données, possibilité de requêtes) et de caractéristiques objets (objet, héritage [multiple], polymorphisme, [envoi de messages])

Utilisation d'identité permanente d'objet, des opérations associées aux objets, d'objets atomiques volumineux, d'objets de structures complexes (hiérarchie, graphes, etc.) et non plus limité aux trois seuls types de données initiaux (relation, tuple, type élémentaire), des types de base (booléen, entier, réel, caractère, chaînes de caractères) et multimédias (texte, image, son, vidéo)

SGBD Relationnel-Objet (SGBDRO)

SGBDO issu de l'approche consistant à étendre le modèle relationnel et son langage (SQL) avec les concepts objets (langage de types et notion d'identité) i. e. SQL est muni d'un langage de types (types abstraits définis par les développeurs étendant les types de base ou structurés fournis par le constructeur tels que relation (i. e. table), tuple, ensemble, multi-ensemble, liste, tableau) par définition d'objets complexes avec sous-typage, et incorpore la notion d'identité (et donc instanciable)

Quelques notions

Un type représente la classe en compréhension

Une relation représente la classe en extension

Une sous-relation indique l'inclusion des éléments de la sous-relation dans la super-relation

Un sous-type spécialise un type avec héritage de propriétés

Création de type simple ou structuré, et des opérations associées

Type générique relation spécialisé en types de collections (ensemble, multi-ensemble, liste, tableau) avec des opérations associées (inclusion d'un ensemble dans un autre, position d'un élément dans une liste, etc.)

Expression d'un type d'associations du modèle entité-association de lien 1:n entre les deux relations correspondant aux types d'entités reliées

Une référence directe aux objets remplace la clé étrangère (une expression de chemin correspond à une suite de jointures implicites)

SGBD Orienté-Objet (SGBDOO)

SGBDO issu de l'approche consistant à étendre un modèle à objet par la définition et la manipulation d'objets persistants, ayant engendré deux courants : systèmes à objets persistants et SGBDOO purs

Système à objets persistants

À partir d'un langage de programmation à objets, permet de définir le schéma de la BDO et d'écrire des programmes d'applications intégrant les appels au SGBDOO

Exemples : Gemstone (avec Smalltalk), ObjectStore (avec C++)

SGBDOO pur

Part d'un modèle à objet qui intègre d'emblée les concepts des modèles de données sémantiques et des langages de programmation à objets

Exemples : O2, Orion, Versant

La persistance des objets

Gestion de la persistance

Remarque : doit éviter les conversions excessives de format et de structure (exemple : linéaire comme un fichier et complexe comme un objet)

Persistance manuelle

Opérations mises à la disposition du programmeur permettant de créer un objet persistant à partir d'un objet temporaire, de lire un objet persistant devenant après allocation un objet temporaire (activation), d'écrire et de libérer un objet temporaire devenant un objet persistant (désactivation), de détruire un objet persistant

Persistance par héritage

Technique permettant de définir la persistance d'un objet par appartenance à une classe (héritée d'une classe persistante)

Persistance par référence

Technique permettant de définir la persistance d'un objet par création explicite d'une racine de persistance ou par rattachement à un objet persistant

Support des collections

Le langage de programmation à objets peut combiner des manipulations navigationnelles (au sein de la BD et/ou dans l'espace des objets temporaires, de manière uniforme, en utilisant des expressions de chemins) et ensemblistes (algèbre relationnelle étendue à la manipulation d'objets complexes grâce à la classe collection)

Évolution du schéma de la BDO

Les différentes évolutions, changeant la définition des classes et la structure d'héritage dans le graphe de classes

Mise à jour d'un attribut d'une classe : ajout ou suppression d'un attribut ; modification du nom, type, héritage d'un attribut

Mise à jour d'une opération d'une classe : ajout ou suppression d'une opération ; modification du nom, contenu, héritage d'une opération

Mise à jour de l'héritage d'une classe : ajout ou suppression d'une classe dans une liste de superclasses

Mise à jour du graphe de classes : ajout ou suppression d'une classe ; modification du nom d'une classe

Objectif

Évolutions à prendre en compte à tout moment, sans impact sur les utilisateurs (concurrents), et visibles immédiatement

Il faut donc préserver la cohérence du schéma, des données, des programmes d'applications, et des objets

Cohérence des objets selon trois approches

Manuelle : interdiction de mettre à jour un type si des instances de ce type existent

Mise à jour immédiate : propagation automatique et immédiate des mises à jour de schéma sur les objets concernés

Mise à jour paresseuse : mises à jour effectuées lors du prochain accès mémoire

Respect d'invariants (avant et après évolution)

Graphe de classes acyclique et connexe

Nom distinct pour toutes les propriétés d'une classe (héritées ou non)

Origine unique pour toutes les propriétés d'une classe

Héritage complet : une classe hérite de toutes les propriétés de ses superclasses

L'approche orienté-objet

Modèle objet

Objets : état et comportement, littéraux ou mutables, identité, durée de vie (procédure, processus ou persistant), opération sur le type objet (création, suppression, nommé ?, type de l'objet, etc.)

Type : regroupement des objets de mêmes caractéristiques, hiérarchie des types (de base, structuré, collection, exception, itérateur) avec héritage multiple

Classe : implantation d'un type

Association : liens (1:1, 1:n et n:n), création ou suppression, ajout ou suppression d'un membre, recherche des objets cibles d'une association à partir d'un objet source, création d'un itérateur parcourant une association multivaluée

Attribut : opération d'affectation et d'obtention d'une valeur, multivalué (structuré ou collection)

Opération : signature, exception, retour normal ou non

ODL et OML

Définition des types d'objets (avec leurs propriétés et les signatures d'opérations associées) indépendamment du langage de programmation à objets

Écriture du contenu des opérations et des programmes d'applications

Une interface de type ODL spécifie une classe ODMG (nom du type, spécification de l'héritage, nom de l'extension, liste des attributs clés, propriétés des instances, signatures des opérations avec exceptions)

Object Query Language (OQL)

Langage (déclaratif, non uniquement ensembliste) de requête, interactif ou intégré (dans un langage de programmation à objets, sans dysfonctionnement)

Requête : fonction qui, appliquée à une expression désignant des objets, renvoie un objet résultat

L'expression est indépendante du type d'objet (atome, littéral, structure, collection)

Expression : élémentaire, arithmétique, de définition de requête, de construction (objet, structure, collection), de manipulation de collection (quantifications existentielle et universelle, sélection, regroupement et fonction d'agrégation, tri, opération ensembliste, accès indexé, conversion)

Les objets désignés sont ceux atteignables à partir des noms définis comme racine de persistance

L'objet résultat est un objet existant ou un nouvel objet

Évolution « naturelle » de l'approche relationnelle vers l'approche orientée objet

Intégration (au relationnel) des classes et des méthodes

Introduction de types étendus (*Set*, *Tuple*)

Intégration de l'héritage multiple

Adjonction des champs longs *(Large Object)* (LOB)

Développement d'extensions multimédias

Évolution de SQL à OQL