Introduction aux bases de données

*

Objectifs

- Connaître les principales caractéristiques des bases de données
- Être capable d'effectuer une modélisation de données puis de créer une base de données équivalente
- Maîtriser l'essentiel du langage SQL

*

Michel Cartereau - Octobre 2014

AgroParisTech - UFR d'informatique - 16, rue Claude Bernard - F 75231 PARIS CEDEX 05

michel.cartereau à agroparistech.fr - Téléphone : +33 1 44 08 16 80

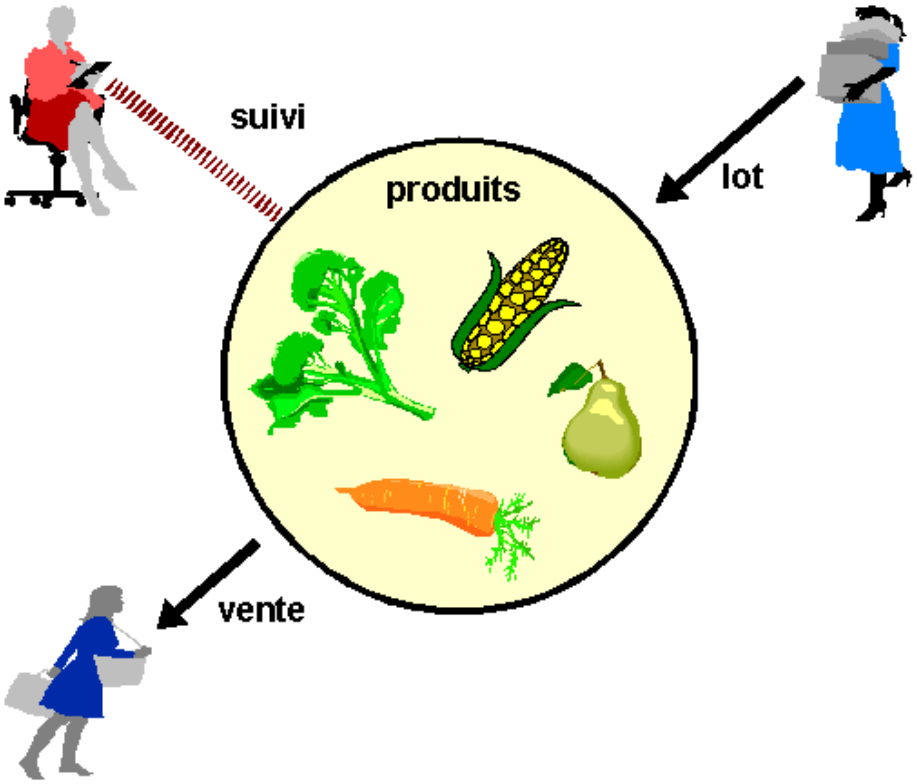
* http://www.agroparistech.fr/mmip/mc/

INTRODUCTION

SYSTÈME D'INFORMATION

PRINCIPALES CARACTÉRISTIQUES D'UNE BASE DE DONNÉES

SYSTÈME DE GESTION DE BASE DE DONNÉES (SGBD)



- EXEMPLE SIMPLE D'ORGANISATION
  - Vente de produits (fruits, légumes) par une coopérative

    Produits achetés à des producteurs et revendus à des clients
  - Différentes activités

    Achat de lot à un producteur, vente de produit à un client, suivi de l'évolution des stocks, etc.
  - Données

    Identification des produits avec prix de vente, d'achat et quantités, coordonnées des producteurs et des acheteurs, etc.

- LE SYSTÈME D'INFORMATION
  - Système opérationnel adapté au fonctionnement de l'organisation

    support de ses activités bâti autour des données manipulées

SYSTÈME D'INFORMATION ESSENTIEL AUX ORGANISATIONS

DÉFINITION DU SYSTÈME D'INFORMATION

OBJECTIF : ASSURER LE BON FONCTIONNEMENT D'UNE ORGANISATION

- UNE MODÈLISATION DE L'ORGANISATION

  - Principaux éléments caractéristiques

    Tâches des différents acteurs (les « métiers »)

    Procédures régissant les activités et les interactions entre acteurs

    Entités caractéristiques des activités (les « objets métiers »)

  - Exemple dans le cas de la coopérative

    Le personnel (vendeurs, directeur, etc.)

    Les activités essentielles (achats, ventes, etc.)

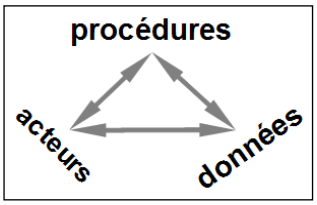
    L'aide à la décision (suivi des stocks, statistiques de ventes, etc.)

    Les entités des produits, lots et personnes

- DÉFINITIONS

  - Système d'information

    un ensemble organisé de ressources (humaines, matérielles) et de procédures, permettant l'exécution des processus internes à une organisation via les manipulations de ses données intrinsèques



  - Donnée

    un fait brut, impossible à interpréter tel que

    exemple : un stock à 0

  - Information

    une donnée prenant du sens dans un contexte d'interprétation

    exemple : manque de carottes en stock

  - Connaissance

    une information analysée dans un contexte d'action

    exemple : un stock à reconstituer par l'achat de lots de carottes

DIMENSION HUMAINE CAPITALE DANS UN SYSTÈME D'INFORMATION

INFORMATISATION DU SYSTÈME D'INFORMATION

INFORMATIQUE INCONTOURNABLE POUR LES SYSTÈMES D'INFORMATION

● OUTIL INFORMATIQUE

- Principaux éléments constitutifs
  organisation considérée comme un « domaine d'application »
  base(s) des données utilisées dans les processus de l'organisation
  applications informatiques pour l'exécution des tâches par les acteurs
- Systèmes difficiles à concevoir
  complexité liée à celle de l'organisation et des processus humains
  difficulté de traduire fidèlement la réalité sous forme informatique
  évolution naturelle du fonctionnement et donc des besoins
- Deux grands types de réalisation
  création à façon du système d'information par des spécialistes,
  démarche souvent coûteuse et délicate pour un domaine conséquent
  utilisation et-ou adaptation de progiciels spécialisés (produits logiciels)
  dans tous les cas, prise en compte obligatoire de nouveaux besoins

● QUELQUES FAMILLES DE PROGICIELS

- Progiciel de gestion intégré (*enterprise resource planning*, *ERP*)
  fonctionnalités principales d'une entreprise : gestion de production,
  gestion commerciale, ressources humaines, comptabilité, etc.
  exemples : SAP Business one, Microsoft Dynamics AX, OpenERP
- Gestion des ressources humaines (*human resource management*, *HRM*)
  paye, prestations sociales, carrières et compétences, etc.
  exemples : HR Access, Adequasys, OpenPortal
- Gestion de la relation avec le client (*customer relationship management*, *CRM*)
  base de clients et de prospects, suivi individualisé, marketing, etc.
  exemples : CRM SalesForce, SugarCRM
- Gestion de la chaîne logistique (*supply chain management*, *SCM*)
  planification, approvisionnement, stockage, transport, etc.
  exemples : Oracle SCM, JDA Software

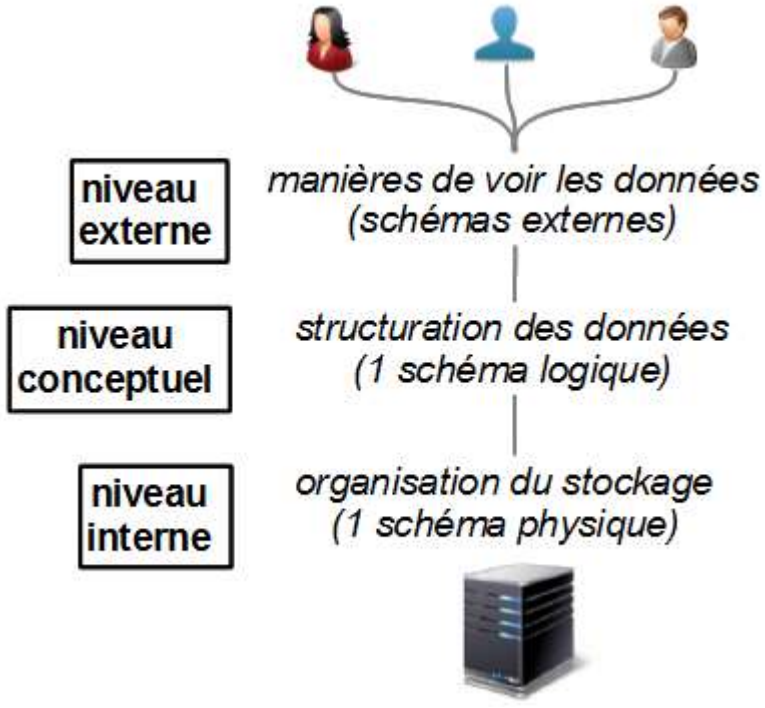
TOUT UN UNIVERS DE PRODUITS ET DE MÉTIERS INFORMATIQUES,
UN ÉLÉMENT PRIMORDIAL AU SEIN DES ENTREPRISES

CARACTÉRISTIQUES D'UNE BASE DE DONNÉES

ENSEMBLE STRUCTURÉ DE DONNÉES COHÉRENTES ET PÉRENNES

- RÔLES ESSENTIELS D'UNE BASE DE DONNÉES [1]
  - Assurer le stockage informatisé
    organisation de l'enregistrement sur la mémoire secondaire (disques)
    garantie de pérennité des données même en cas de panne technique
  - Prendre en compte la structure des données
    données stockées avec et selon leur schéma de structuration
    garantie de cohérence des données
  - Permettre des utilisations simultanées et autorisées
    contrôle d'accès et gestion de la concurrence des opérations
    garantie de confidentialité et d'intégrité des données

- MODÈLE ANSI SPARC [2]
  - Modèle défini par un comité de l'ANSI en 1975
    architecture de référence pour les bases de données



Ensemble de schémas formant le « schéma de la base »

LA BASE DE DONNÉES EST AU CŒUR DU SYSTÈME D'INFORMATION

[1] En anglais : *database*
[2] *American national standards institute*, *standards planning and requirements committee*

SYSTÈMES POUR LES BASES DE DONNÉES

APERÇU DE L'EXISTANT

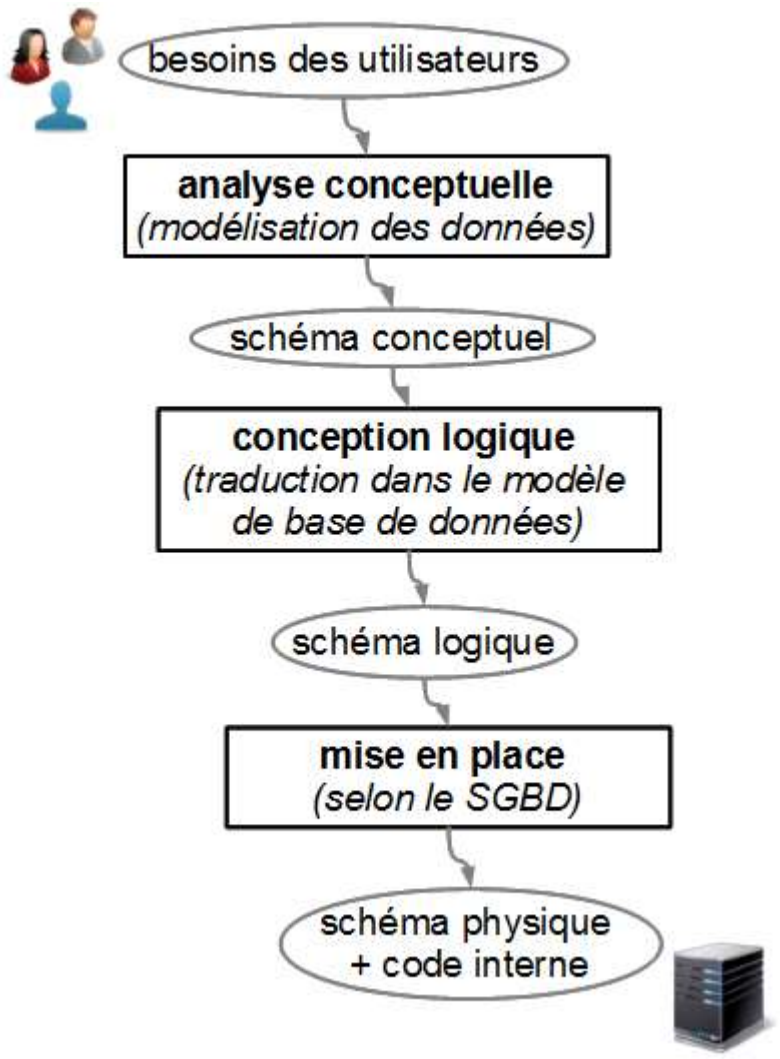
- PRINCIPALES FAMILLES
  - Modèle relationnel dominant
    organisation des données par tables
  - Autres modèles
    objet : stockage persistant d'objets (avec ou sans l'héritage)
    XML : documents semi-structurés
    spatial : systèmes d'information géographique
    noSQL (*not only SQL*) : gros volumes de données hétérogènes
    hiérarchique/réseau : anciens modèles (*Codasyl data model*)
- SYSTÈMES DE GESTION DE BASE DE DONNÉES (SGBD [1])
  - Principales fonctionnalités
    stockage sur disques des données, de leur structure et du code associé
    manipulations des données (enregistrement, mises à jour, recherches)
    gestion des accès (confidentialité, concurrence)
    administration (droits, sauvegarde/restauration, optimisation, répartition)
  - Langages de manipulations
    relationnel : SQL (*structured query language*), QBE (*query by example*)
    Objet : OQL (*object query language*)
    XML : XQuery (*XML Query*)
  - Principaux SGBD existants
    relationnel : Oracle, DB2, SQL server, Sybase, MySQL, PostgreSQL
    objet : Caché, db4o - XML : BaseX - noSQL : HBase, BigTable

UN MARCHÉ TRÈS IMPORTANT EN VALEUR

[1] En anglais : *database management system (DBMS)*

CONSTRUCTION D'UNE BASE DE DONNÉES

PARTIE INTÉGRANTE DE LA CONCEPTION D'UN SYSTÈME D'INFORMATION



PRINCIPALES ÉTAPES DE LA CONSTRUCTION

1) Analyse conceptuelle

modélisation des données à partir des besoins et de l'existant, sous la forme d'un schéma conceptuel (étape d'abstraction)

2) Conception logique

traduction du schéma conceptuel selon le modèle de base de données sous la forme d'un schéma logique (étape automatisable)

3) Mise en place

optimisation du schéma logique en un schéma physique mis en place dans un SGBD avec production de code complémentaire

MÉTHODOLOGIES DÉFINIES POUR LES SYSTÈMES D'INFORMATION

MODÉLISATION

MODÈLE ENTITÉ-ASSOCIATION

UML

MODÈLE RELATIONNEL

MODÈLE ENTITÉ-ASSOCIATION

MODÉLISATION DES DONNÉES

● OBJECTIFS

- Description de la structuration des données
  modélisation abstraite au niveau conceptuel par un diagramme
  indépendance vis-à-vis de tout modèle de base de donnée
- Simplicité des concepts
  ne demande pas de connaissances avancées en modélisation
  facilite le dialogue entre les concepteurs et les utilisateurs
  traduction facilitée vers un schéma logique de base de données

● HISTORIQUE

- Formalisation en 1976 [1] (*entity-relationship model*, *ER model*)
  version initiale de base, utilisée avec différentes formes graphiques
  très répandu en France entre 1980 et 2000 via la méthode Merise [2]
- Modèle entité-relation étendu (*enhanced entity-relationship, EER*)
  enrichissement par ajouts pour adaptation à des domaines complexes

● OUTILS D'AIDE À LA MODÉLISATION

- Atelier de génie logiciel (AGL)
  édition de diagrammes, conversions entre différents types de schémas
  production automatique de code
- Exemples de logiciels utilisant le modèle entité-association
  AMC*Designor/PowerAMC, DB-Main, MySQL Workbench

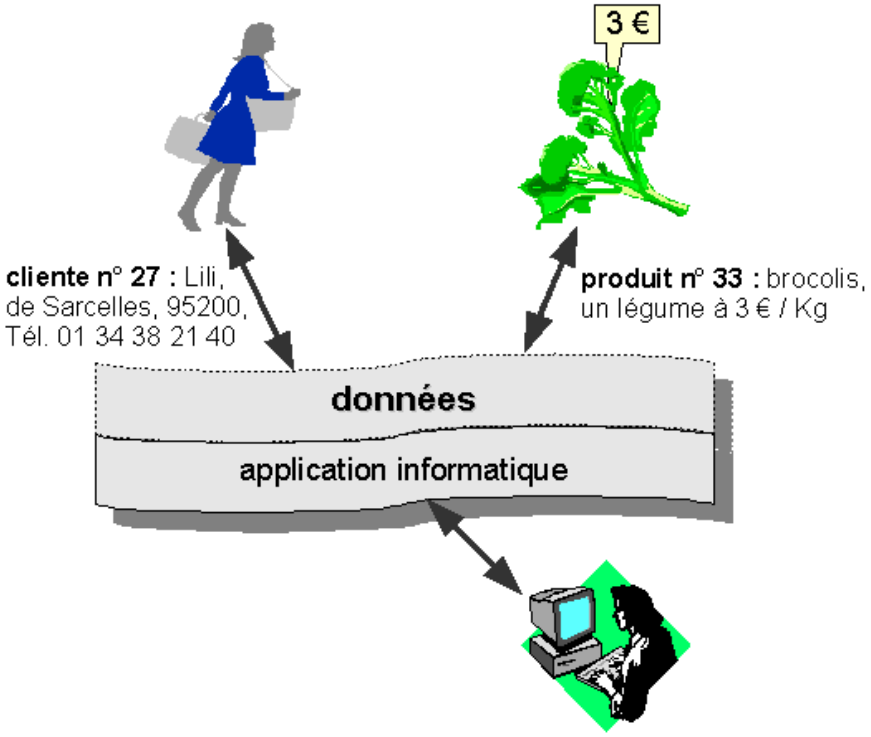
UN FORMALISME ENCORE TRÈS UTILISÉ

[1] Peter CHEN. *The entity-relationship model - Toward a unified view of data*. ACM Transactions on Database Systems, vol. 1, n° 1, pages 9–36, mars 1976.

[2] Merise : méthode d'analyse, de conception et de gestion de projet informatique développée en France vers 1975 ; nom du fruit du merisier « *qui ne peut porter de beaux fruits que si on lui greffe une branche de cerisier : ainsi en va-t-il des méthodes informatiques bien conçues, qui ne produisent de bons résultats que si la greffe sur l'organisation réussit* »

EXEMPLE SIMPLE DE MODÉLISATION DE DONNÉES

QUELLES SONT LES INFORMATIONS MANIPULÉES DANS LA COOPÉRATIVE ?



- MODÉLISATION
  - Identification des éléments (« entités ») du domaine d'application
    exemples : un client, un produit, un fournisseur, un lot, une vente
  - Description de la structure de chaque entité [1]
    liste des données élémentaires [2] rattachées (« attributs » [3])
    exemple : produit avec numéro, nom, type (légume ou fruit), prix
  - Représentation de chaque information (« format »)
    utilisation de types de données élémentaires (texte, nombre, etc.)
    exemple : le nom d'un produit est un texte
  - Identification de dépendances entre des entités (« associations »)
    exemple : une vente s'applique à un seul produit et un seul client, avec une date et un prix associés
    description de la nature de chaque association

[1] Cela correspond en fait formellement à un « type d'entité »
[2] Une donnée élémentaire ou « atomique», correspond à une valeur simple (nombre, texte, date, etc.) et non pas à une valeur multiple (liste, ensemble, etc.)
[3] Comme pour le type d'entité, cela correspond en fait formellement à un « type d'attribut »

DONNÉES DU PRODUIT

MODÉLISATION DE L'ENTITÉ « PRODUIT » [1]

- COMMENT REPRÉSENTER UN PRODUIT ?

  - De quoi a-t-on besoin ?

    Le nom du produit, exemple : carotte

    Connaître son type, légume ou fruit, exemple : légume

    Connaître son prix de vente (1 kilogramme), exemple : 1 euro

  - Comment identifier un produit de manière unique ?

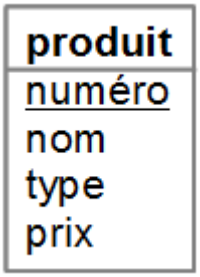
    Identification possible avec son nom mais risque d'ambiguïté, donc plus fiable avec un numéro d'ordre associé

  - Représentation par les attributs

    1) **numéro** : nombre entier, l'« identifiant » [2]
    2) **nom** : texte
    3) **type** : texte, soit `LEGUME`, soit `FRUIT`
    4) **prix** : nombre décimal en euros, exemple : 1,5 (1 euro 50)

  - Modélisation

    Représentation de l'entité « produit » par un rectangle avec son nom en entête de la liste de ses attributs (identifiant en premier)

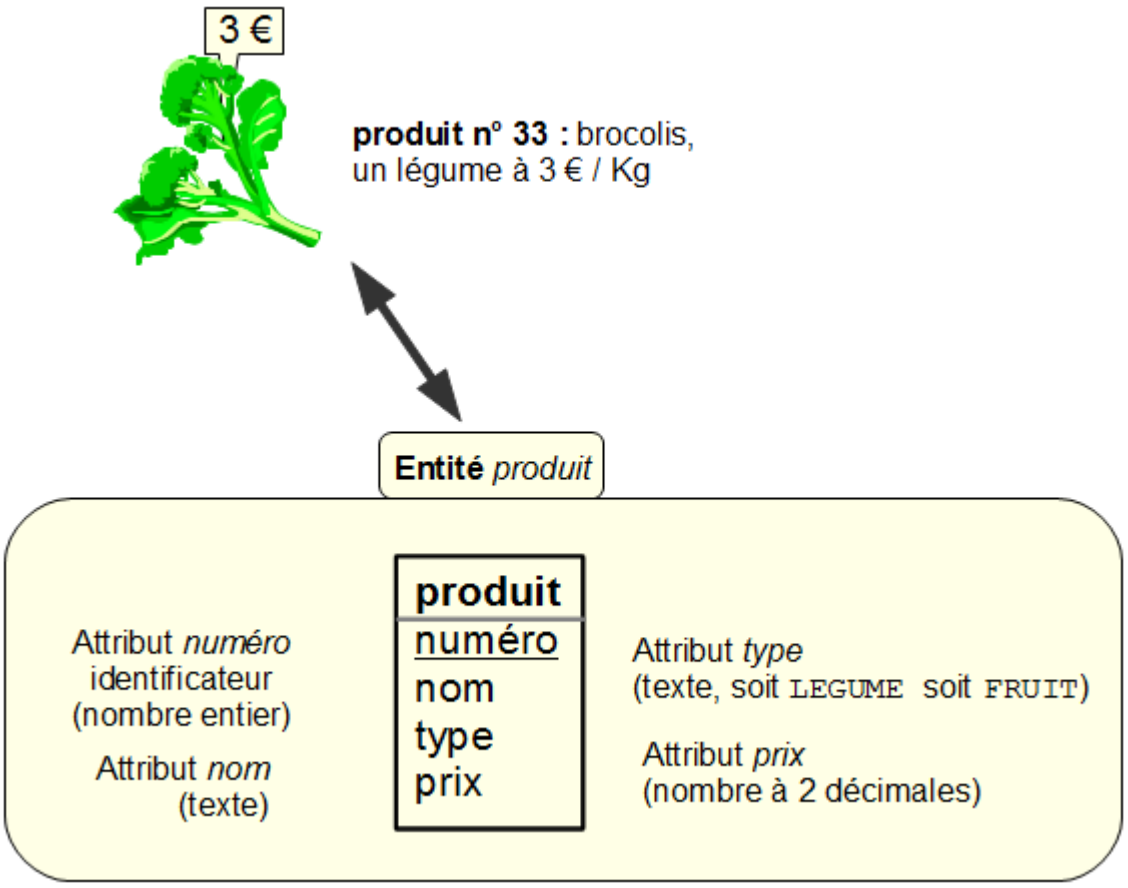


    Par convention, l'identifiant est souligné

[1] La représentation est bien sûr simplifiée ici, dans le cadre de cette présentation.
[2] Dans le modèle « entité-relation », l'« identifiant » identifie de manière unique une entité ; il correspond à un ou plusieurs attributs ; sa valeur est invariable.

MODÉLISATION D'UNE ENTITÉ

EXEMPLE DU PRODUIT



- LISTE DES ATTRIBUTS

  Ensemble des données élémentaires caractéristiques de l'entité

  Principaux types de donnée possibles :

  - une valeur numérique avec ou sans partie décimale, éventuellement une valeur monétaire
  - un texte libre (taille maximale fixée en nombre de caractères), ou un libellé parmi quelques valeurs possibles
  - une date et-ou une heure

- DÉTERMINATION DE L'IDENTIFIANT

  Il sert à identifier de manière unique un exemplaire de l'entité, il est toujours connu et il ne peut pas être modifié (invariable)

  Cet identifiant consiste préférentiellement en un code ou plusieurs attributs combinés, ou sinon, de manière générale, on considère un numéro d'ordre

DONNÉE DE LA PERSONNE

UNE PERSONNE EST SOIT UN PRODUCTEUR, SOIT UN ACHETEUR

- COMMENT REPRÉSENTER UNE PERSONNE ?

  - De quoi a-t-on besoin ?

    La nom de la personne
    exemple : De La Rue

    Connaître son adresse dont notamment le code postal et la ville,
    exemple : 80, bd. J. Jaurès - 92110 - Clichy

    Connaître son numéro de téléphone
    exemple : 01.47.15.30.00

  - Comment identifier une personne de manière unique ?

    Identification avec un numéro d'ordre associé

  - Représentation par les attributs

    1) **numéro** : nombre entier, l'« identifiant »
    2) **nom** : texte
    3) **adresse** : texte
    4) **code postal** : texte
    5) **ville** : texte
    6) **téléphone** : texte

  - Modélisation

    Représentation de l'entité « personne »

personne
numéro
nom
adresse
code postal
ville
téléphone

DONNÉE DE LA VENTE

LA VENTE D'UN PRODUIT À UNE PERSONNE

- COMMENT REPRÉSENTER UNE VENTE ?

  - Caractérisation d'une vente

    Un produit est vendu à une personne, à une date, avec une quantité et à un prix éventuellement différent du prix du produit (exemple : réduction)

    Entité pour la vente, avec associations à un produit et à une personne, et identification impossible de manière unique avec ses informations :
    → identification par un numéro d'ordre

  - Représentation

    Entité « vente » :

    1) **numéro** : nombre entier, l'« identifiant »
    2) **quantité** : nombre entier de kilogrammes achetés
    3) **prix** : nombre à 2 décimales du prix payé
    4) **date** : de la vente (sous forme internationale : *année–mois–jour*)

    et associations avec produit (« vendre ») et personne (« acheter »)

  - Modélisation d'une vente

    Détermination du nombre d'associations possibles (« cardinalité») pour chacune des entités, sous la forme « minimum**,** maximum »

    1 vente concerne exactement un produit → cardinalité `1,1`

    1 produit peut ne pas être vendu ou l'être N fois [1] → cardinalité `0,N`

    1 vente concerne exactement une personne → cardinalité `1,1`

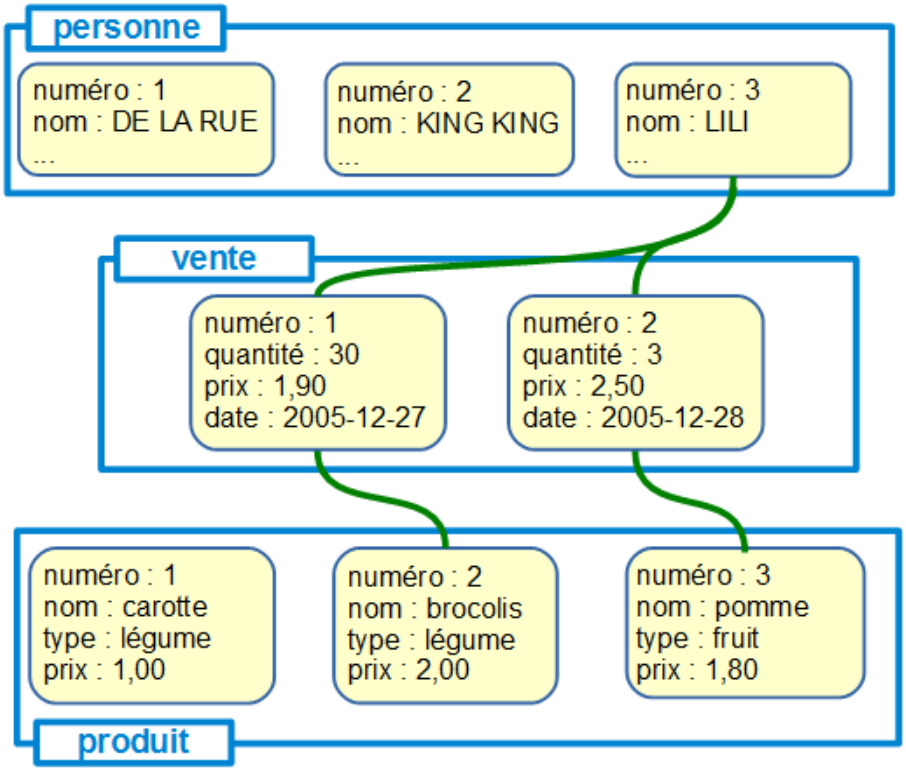
    1 personne peut ne pas acheter ou le faire N fois → cardinalité `0,N`

[1] N désigne ici un nombre non limité a priori, c'est-à-dire plusieurs fois.

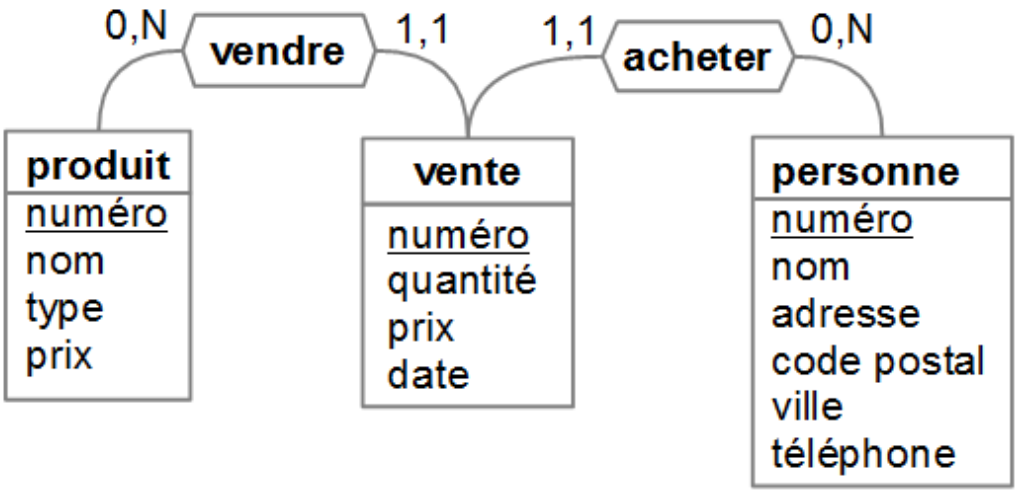
DONNÉE DE LA VENTE (SUITE)

LA VENTE D'UN PRODUIT À UNE PERSONNE

- Exemple de ventes



- Représentation des associations



Association représentée par un polygone avec son nom à l'intérieur, et les cardinalités notées sur les branches vers les entités

DONNÉE DU LOT

UN LOT CORRESPOND ICI À UN SEUL PRODUIT

- COMMENT REPRÉSENTER UN LOT ?

  - Caractérisation d'un lot

    Une entité en double association avec produit et personne : un lot correspond à un produit, et il est fourni par une personne (producteur)

    Il faut aussi connaître la taille initiale du lot, la taille du reste (pas encore vendu), la date de la fourniture et le prix d'achat (pour 1 kg)

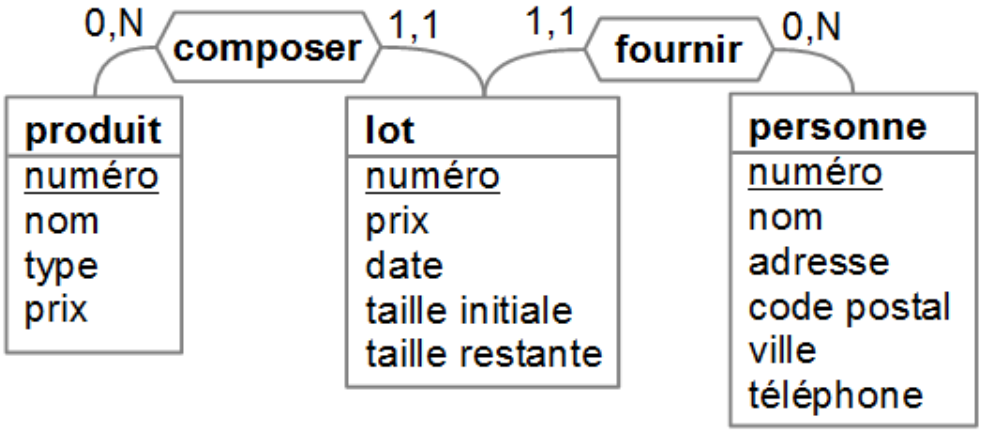
    Exemple : lot de 450 Kg de carottes entièrement vendus, fourni par De La Rue le 29 décembre 2005 au prix de 0,70 € le kg

  - Comment identifier un lot de manière unique ?

    Impossible de manière unique avec les informations
    → identification par un numéro d'ordre

  - Modélisation

    Association « composer » entre produit et lot
    1 produit peut ne pas composer ou le faire N fois → cardinalité `0,N`
    1 lot est composé d'exactement un produit → cardinalité `1,1`
    Association « fournir » entre personne et lot
    1 personne peut ne pas fournir ou le faire N fois → cardinalité `0,N`
    1 lot est fourni par exactement une personne → cardinalité `1,1`



CAS D'ASSOCIATIONS « 1 À PLUSIEURS » OU « 1 : N »
(1 PRODUIT COMPOSE EXCLUSIVEMENT PLUSIEURS LOTS,
1 PERSONNE FOURNIT EXCLUSIVEMENT PLUSIEURS LOTS)

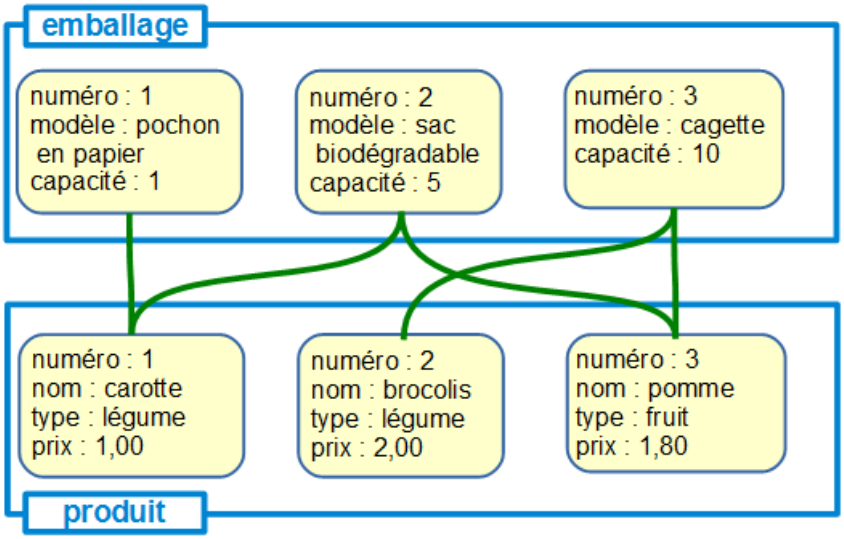
EMBALLAGES DE PRODUIT

CAS DE PRODUITS CONDITIONNÉS AVEC DES TYPES D'EMBALLAGE

- EXEMPLE DE L'EMBALLAGE DE PRODUIT

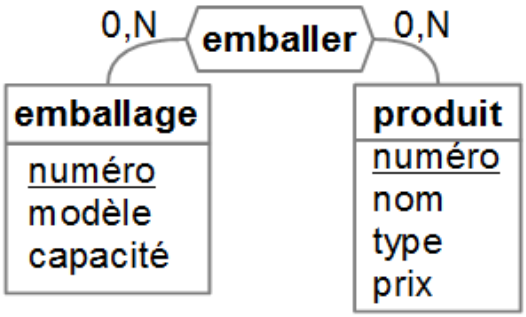
  - Caractérisation d'un emballage

    le modèle, défini par un intitulé (exemple : cagette)
    la capacité, correspondant au volume contenu exprimé en litre
    identification par un numéro d'ordre



  - Association entre un produit et un type d'emballage

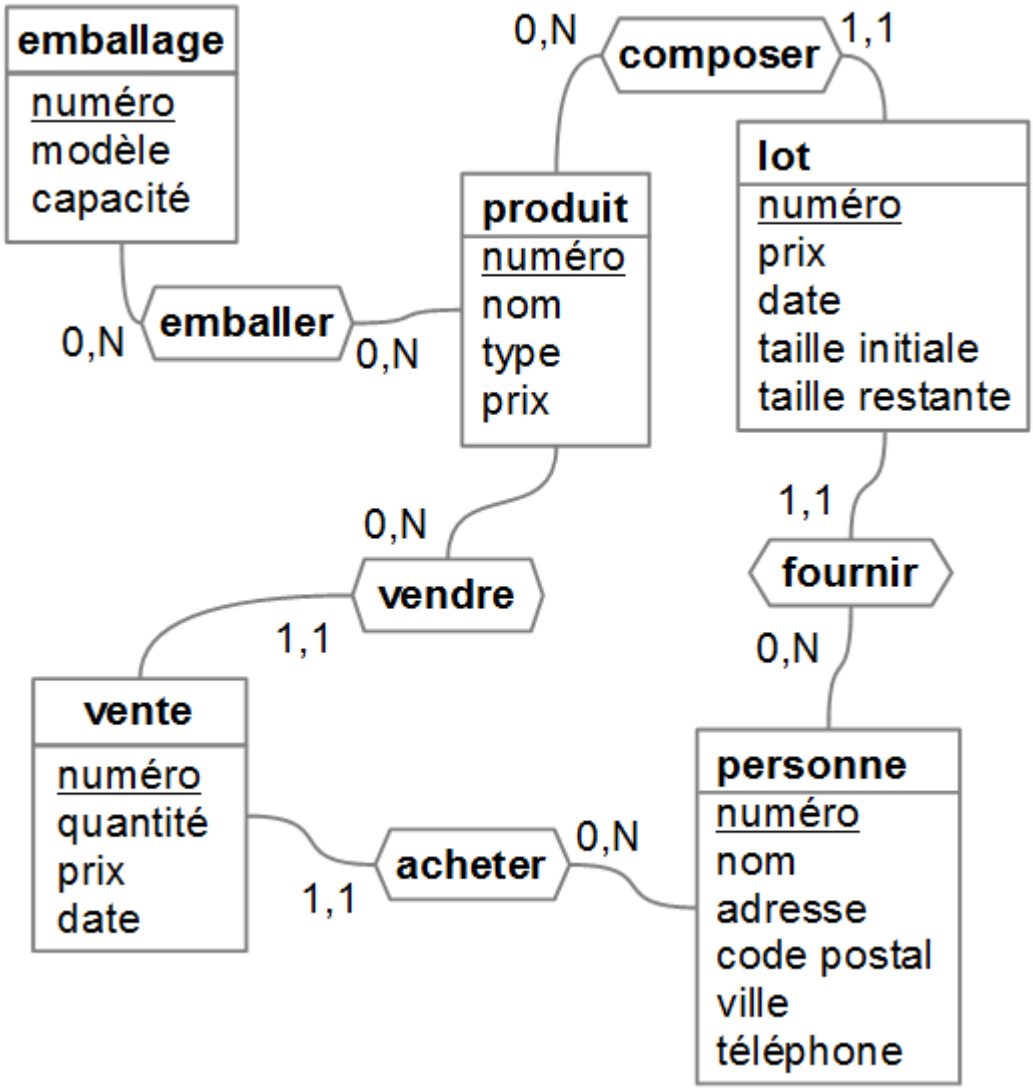
    Association « emballer » : un produit est conditionné selon un type d'emballage
    1 type d'emballage peut ne pas être utilisé ou conditionner N produits → cardinalité `0,N`
    1 produit peut ne pas être emballé ou être conditionné selon N types d'emballage → cardinalité `0,N`



CAS D'ASSOCIATION « PLUSIEURS À PLUSIEURS » OU « N : N »

MODÉLISATION DE LA COOPÉRATIVE

MODÈLE DE DONNÉES GLOBAL



POUR UNE RÉALITÉ DONNÉE, PLUSIEURS MODÈLES SONT POSSIBLES
MAIS ILS SONT PLUS OU MOINS EXPRESSIFS
ET ILS PEUVENT ÊTRE PARFOIS INEFFICACES

EXERCICES D'EXTENSION DU MODÈLE

1) Un lot peut être vérifié une ou plusieurs fois par un employé (éventuellement différent à chaque fois) afin de déterminer son état général (bon, dégradé ou perdu) ; un employé est caractérisé par sa fonction et son poste téléphonique.

   Exemple : le lot n° 1, vérifié le 30/12/2005 comme bon par Joyeux puis dégradé le 3/1/2006 par Lapin.

   Proposer une modélisation de l'employé et puis de la vérification

2) La coopérative offre des promotions sous la forme d'un panier de 2 produits, avec un prix attractif au kilogramme

   Exemple : le panier « purée hivernale », à 3,2 euros,
   composé de 1 kg de carottes et de 1 kg de panais

   a) Proposer une modélisation de ce panier

   b) Quelles sont les conséquences sur les autres entités ?

3) Imaginer une association « un à un » (1:1)

REDONDANCE DES DONNÉES

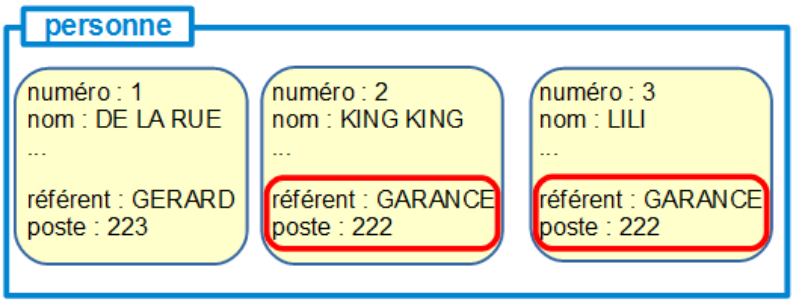
DUPLICATION INUTILE DE DONNÉES

- CAS DE REDONDANCE

  - Exemple

    On définit un référent unique pour toute personne enregistrée, qui correspond à un employé de la coopérative en contact avec elle, identifié par le nom de cet employé et son poste téléphonique

    Si on rajoute ces attributs dans l'entité « personne » :



    on voit apparaître rapidement une duplication d'information inutile et dangereuse car si par exemple le poste téléphonique d'un référent change, il faut alors répercuter la modification à chaque répétition

  - Conséquences

    Gâchis de mémoire en cas de répétitions nombreuses

    Risques élevés d'incohérence lors de mises à jour incomplètes

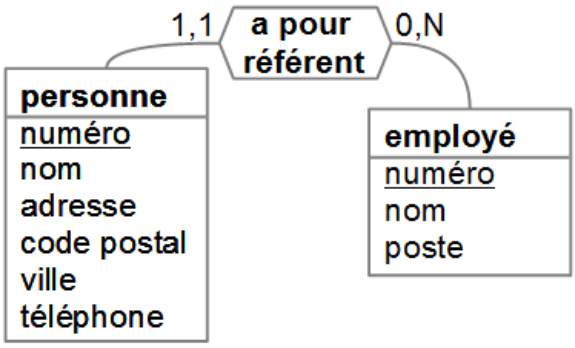
  - Analyse

    Cas général de « **dépendance fonctionnelle** » entre 2 attributs d'un enregistrement : la connaissance de la valeur du premier attribut entraîne la détermination du second attribut de manière unique

    Exemple : si on connaît le nom du référent, on peut trouver le poste

  - Solution

    Elimination des dépendances fonctionnelles via la décomposition de l'entité en plusieurs entités ; exemple :



LA REDONDANCE EST A PRIORI UN DÉFAUT DANS UNE BASE DE DONNÉES

MODÉLISATION CONCEPTUELLE : RÉCAPITULATIF

PROCÉDURE D'ANALYSE ET DE REPRÉSENTATION DES DONNÉES

- DÉCOMPOSITION EN ENTITÉS
  - Inventaire des éléments de la réalité
    identification d'éléments indépendants
  - Détermination des attributs pour chaque entité
    liste des données atomiques, avec leurs formats de valeurs
  - Choix de l'identifiant
    un ou plusieurs des attributs

- IDENTIFICATION DES ASSOCIATIONS
  - Associations existantes
    associations entre deux entités
  - Détermination des caractéristiques de chaque association
    ses cardinalités

- VALIDATION DU MODÈLE CONCEPTUEL
  - Elimination des redondances
    redécomposition d'entité si besoin
  - Détermination des contraintes
    valeurs possibles, obligatoire ou facultative
  - Confrontation aux besoins
    vérification de l'adéquation vis-a-vis de tous les besoins recensés

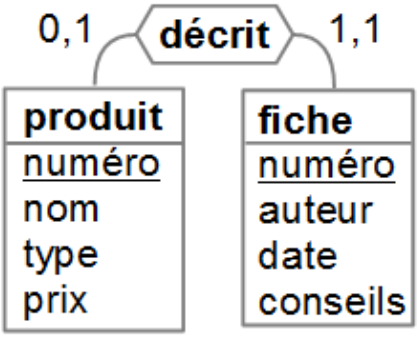
PREMIÈRE DESCRIPTION INFORMELLE D'UNE MODÉLISATION CONCEPTUELLE

ASSOCIATION « UN À UN »

TROISIÈME TYPE D'ASSOCIATION (1 : 1)

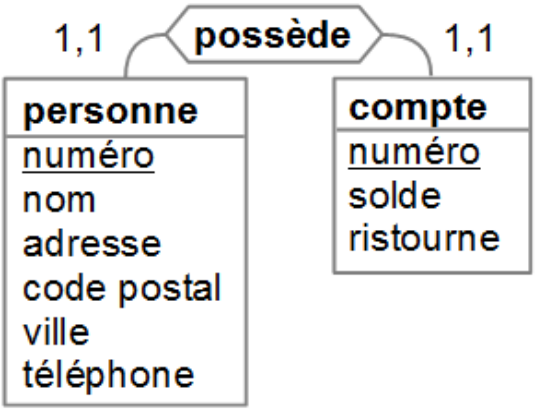
- EXEMPLE D'UNE FICHE DESCRIPTIVE

  - Conseils pour la vente d'un produit
    Une fiche optionnelle pour un produit, avec son propre numéro d'identification
  - Nouvelle association « décrit »
    Liaisons avec chacune une cardinalité maximale égale à 1



- EXEMPLE D'UN COMPTE DE CLIENT

  - Compte interne crédité par le client
    Tout client possède un seul compte et tout compte est attribué à un client
  - Nouvelle association « possède »
    Liaisons avec chacune une cardinalité égale à 1,1



- ASSOCIATION « UN À UN »

  - Cas d'association avec des cardinalités maximales de 1
    Chaque entité liée participe au plus une fois à l'association

VEILLER À LA PERTINENCE D'UNE ASSOCIATION 1 : 1
VIS-À-VIS D'UNE FUSION DES DEUX ENTITÉS LIÉES

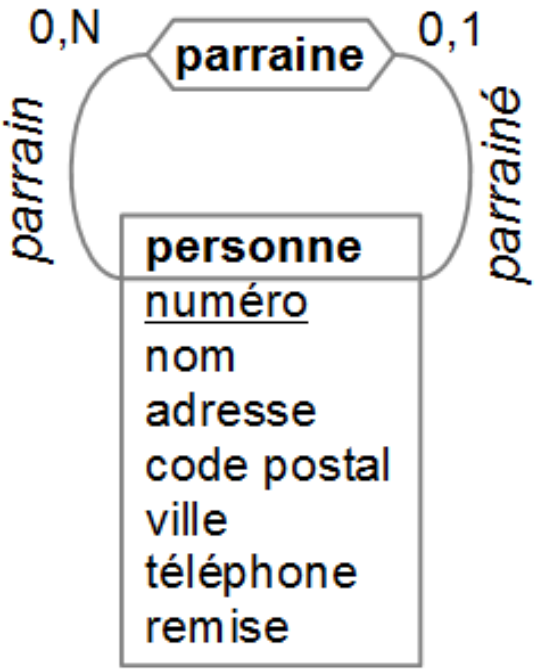
ASSOCIATION CYCLIQUE ET RÔLES

CAS PARTICULIER D'ASSOCIATION À UNE SEULE ENTITÉ

- EXEMPLE D'UN PARRAINAGE
  - Mise en place d'un parrainage avec un système de récompense

    Un client peut faire venir une ou plusieurs personnes, qui deviennent de nouveaux clients, et il se voir alors attribuer un taux de remise
  - Nouvelle association « parraine »

    Liaisons vers la même entité « personne » avec des rôles à distinguer



- ASSOCIATION CYCLIQUE
  - Cas d'association entre une entité et elle-même

    Distinction de chaque liaison par son rôle dans l'association

    Nom du rôle noté sur la branche de la liaison dans le diagramme

APPELLATIONS INCORRECTES : « RÉCURSIVE », « UNAIRE » OU « RÉFLEXIVE »

IDENTIFICATION

CARACTÉRISTIQUES GÉNÉRALES

● IDENTIFICATION D'UNE ENTITÉ

- L'identifiant (*identifier*)

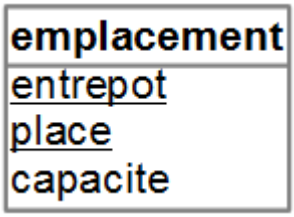
  Il sert à identifier de manière unique un exemplaire de l'entité, autrement dit il permet de distinguer deux exemplaires d'une même entité

  Il existe toujours [1] pour une entité, et possède obligatoirement une valeur qui doit rester invariable en permanence

- Composition de l'identifiant

  A priori un ou plusieurs attributs de l'entité forment l'identifiant

  Exemple : un emplacement de stockage identifié par son entrepôt et sa place à l'intérieur de cet entrepôt



  En cas d'absence d'identifiant naturel, ajout d'un attribut artificiel créé exprès, sans signification dans le domaine d'application mais avec une valeur fixée automatiquement [2], comme par exemple un numéro d'ordre à incrémentation automatique

● PLUSIEURS IDENTIFICATIONS POTENTIELLES POUR UNE ENTITÉ

- Cas d'identification avec plusieurs composants

  Identification nécessairement « minimale » (aucune partie superflue)

  Contre-exemple : l'emplacement de stockage identifié par son entrepôt, sa place au sein de cet entrepôt et aussi sa capacité

- Cas où plusieurs formes d'identification minimales possibles

  Choix d'une forme (a priori c'est la plus simple), appellée « primaire », les autres sont alors dites « secondaires » (ou « candidates »)

EVITER DE PRENDRE UN NOM COMME IDENTIFICATION (ORTHOGRAPHE VARIABLE, PEUT AUSSI CHANGER AVEC LE TEMPS)

[1] Dans le cadre de cette présentation, on exclut l'absence d'identificateur dans une entité.
[2] Les SGBD peuvent gérer ce type de donnée à valeur garantie unique (en anglais : *surrogate key*)

ENTITÉ FAIBLE

CAS D'ENTITÉ À IDENTIFICATION PARTICULIÈRE

- ENTITÉ DÉPENDANTE D'UNE AUTRE ENTITÉ

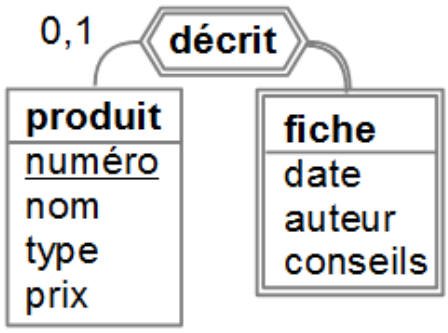
  Cas d'entité « faible » sans identifiant intrinsèque, qui ne peut pas exister sans une autre entité à laquelle elle est associée

  Association dite alors « identifiante » [1]

- IDENTIFICATION « IMPLICITE »

  Identifiant composé par l'identifiant de l'entité à laquelle elle est liée

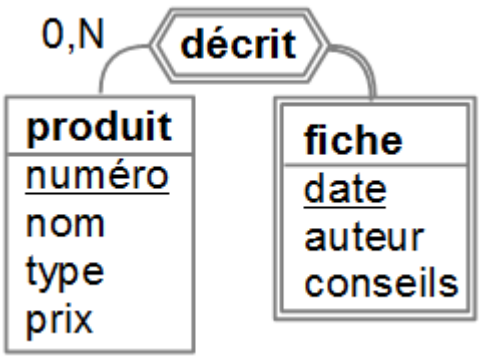
  Exemple : cas de la fiche descriptive d'un produit, sans numéro spécifique, identifiée alors par le numéro du produit décrit



- IDENTIFICATION « HYBRIDE »

  Identifiant composé par l'identifiant de l'entité à laquelle elle est liée, combiné avec un ou plusieurs attributs de l'entité identifiée

  Exemple : cas de versions d'une fiche descriptive d'un produit, sans numéro spécifique, identifiées alors par le numéro du produit décrit et par la date de rédaction



[1] Indication dans le diagramme par un trait doublé pour l'entité faible, l'association identifiante et la branche associée.

RÉCAPITULATIF DU MODÈLE ENTITÉ-ASSOCIATION

CAS DU MODÈLE DE BASE SOUS SA FORME PRÉSENTÉE ICI

- COMPOSANTS
  - Entité

    Un élément autonome du domaine d'application ou dépendant d'une autre entité dans le cas d'une entité faible, identifié par un nom
  - Attribut

    Une propriété d'une entité, à valeur atomique, facultative ou obligatoire [1]
  - Association

    Liaison entre deux entités identifiée par un nom, avec une cardinalité associée à chaque rôle (minimum-maximum d'associations possibles pour un exemplaire de l'entité remplissant le rôle) ; une association cyclique relie une entité à elle-même ; 3 familles : un à un (1:1), un à plusieurs (1:N) et plusieurs à plusieurs (N:N)
  - Identifiant

    Identification d'une entité, composé d'attribut(s) et-ou de l'identifiant de l'entité liée en cas d'entité faible

- CONTRAINTES
  - Noms distincts

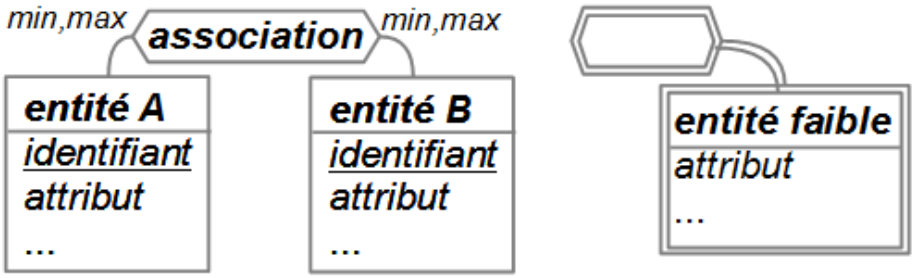
    parmi les entités, parmi les associations, parmi les attributs d'une entité
  - Attributs

    Déterminé si obligatoire, valeur dans le domaine de définition

    Identifiant obligatoire pour toute entité avec des valeurs uniques
  - Cardinalités

    minimum 0 ou 1 et maximum 1 ou N, soit : 0,1 ou 1,1 ou 0,N ou 1,N

- NOTATION GRAPHIQUE [2]



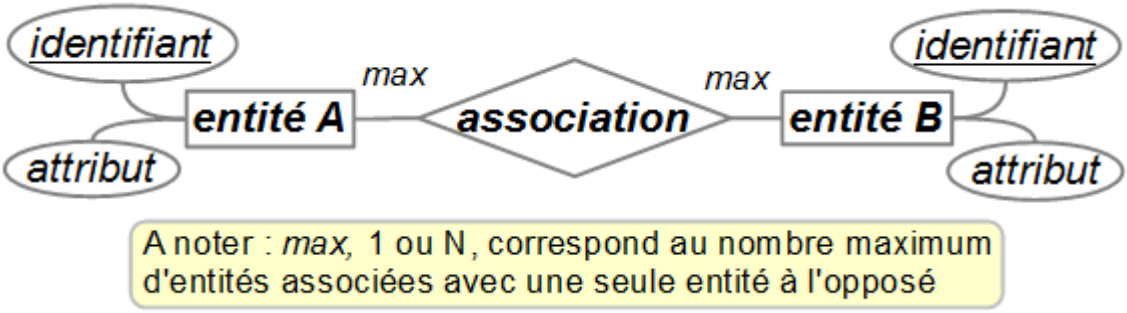
[1] Il n'y a pas de règle a priori pour la notation du caractère obligatoire/facultatif d'un attribut.
[2] Schéma appelé en anglais *entity-relationship diagram* (*ERD*)

AUTRES NOTATIONS DU MODÈLE ENTITÉ-ASSOCIATION

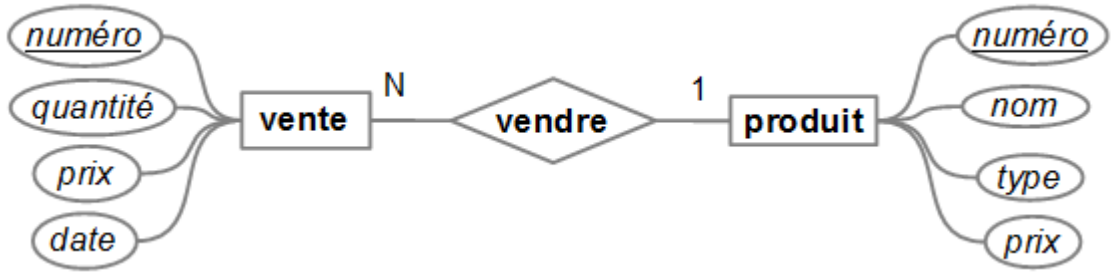
PRÉSENTATION DES PRINCIPALES NOTATIONS

● NOTATION DE CHEN

Notation du concepteur du modèle entité-association, Peter Chen, 1971

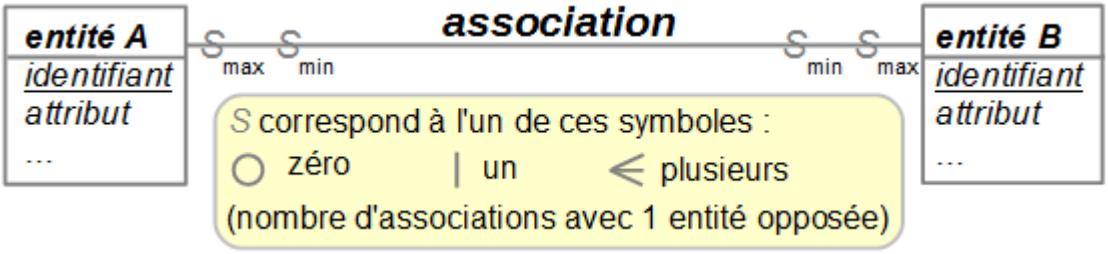


Exemple (relation « vendre » du type 1 à plusieurs)

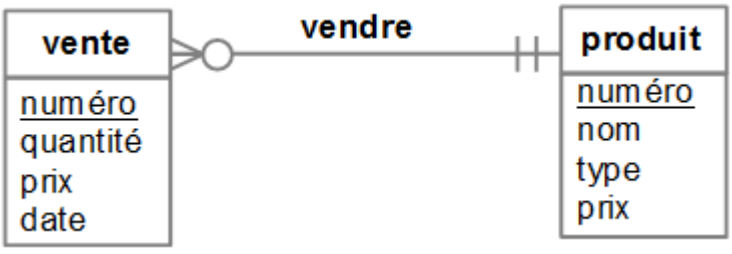


● NOTATION DE LA PATTE DE CORBEAU (*CROW'S FOOT*)

Notation développée dans les années 1980



Exemple (relation « vendre » du type 1 à plusieurs)



NOTATIONS AYANT ÉVOLUÉ AU COURS DU TEMPS

EXTENSIONS AU MODÈLE ENTITÉ-ASSOCIATION

PRÉSENTATION SYNTHÉTIQUE DES PRINCIPAUX AJOUTS

- ATTRIBUTS
  - Attribut composé

    Décomposition en plusieurs sous-attributs sur plusieurs niveaux
  - Attribut multivalué

    Valeur sous la forme d'un ensemble
  - Contraintes sur les valeurs

    Domaine de valeurs plus restreint (intervalle, liste de constantes, etc.), contrainte entre valeurs (condition sous forme de comparaison) et condition d'existence d'attributs facultatifs (coexistence, exclusion, etc.)
- ASSOCIATIONS
  - Association n-aire

    Liaisons entre plus de 2 entités (généralement 3)
  - Attribut d'association

    Attribut spécifique à une association
  - Cardinalité généralisée

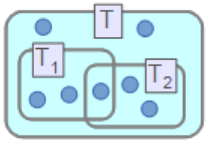
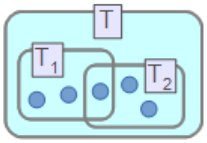
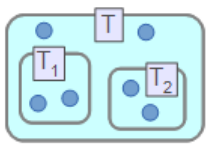
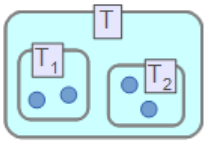
    Minimum/maximum possible avec une valeur entière supérieure à 1

- ENTITÉS
  - Généralisation-spécialisation (« est un », *is a*)

    Attributs supplémentaires d'une sous-entité par rapport à une sur-entité plus générale, avec un mécanisme d'héritage des attributs

    Contrainte de disjonction/recouvrement de la spécialisation :
    un exemplaire d'entité dans une seule/plusieurs sous-entités spécialisées

    Contrainte de couverture totale/partielle de la généralisation :
    un exemplaire d'entité obligatoirement/optionnellement dans une sous-entité spécialisée

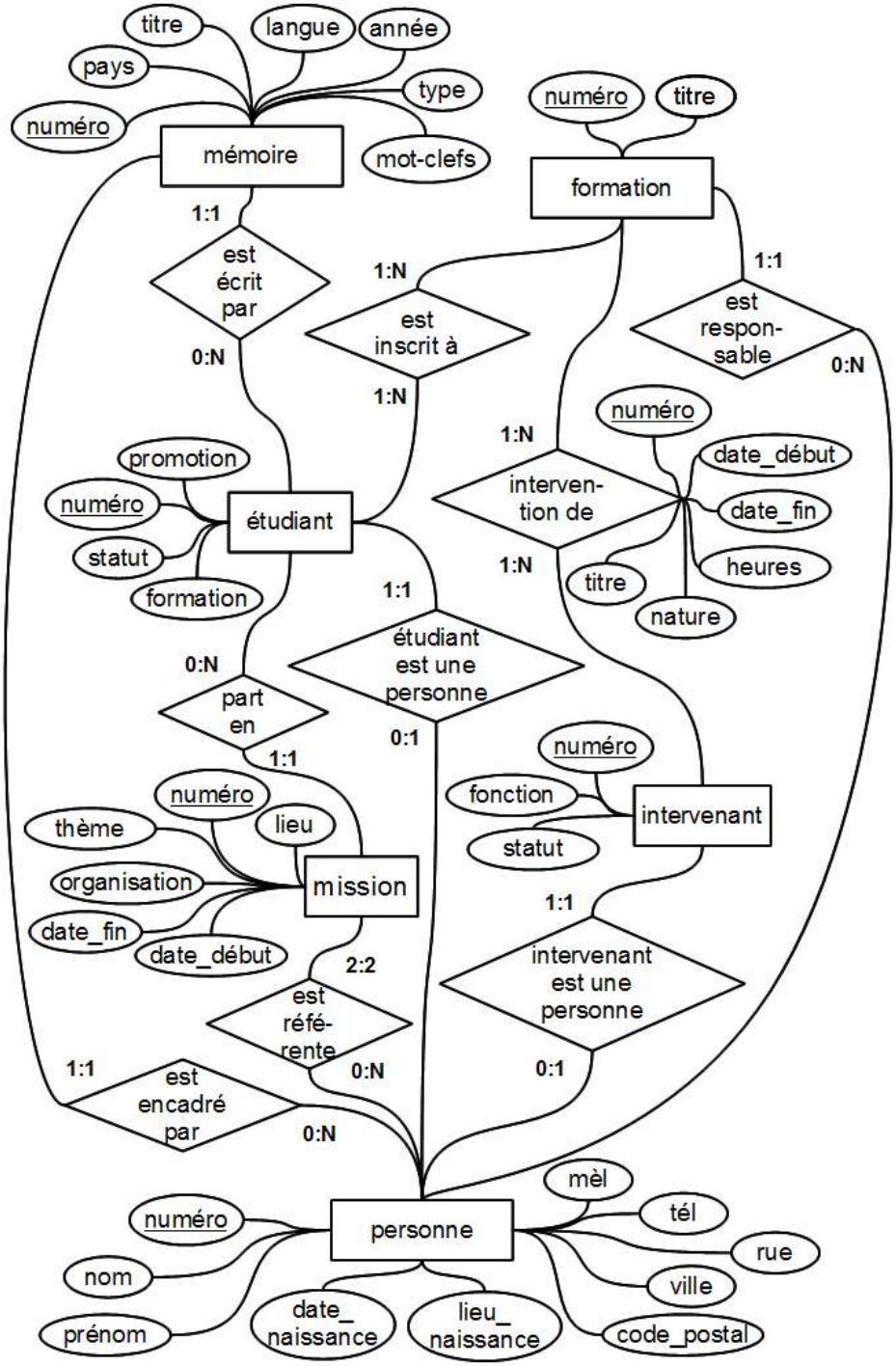
sur-entité T / sous-entités T_1, T_2	**couverture partielle**	**couverture totale**
recouvrement		
disjonction		

MODÈLE PLUS RICHE MAIS TRADUCTION PLUS COMPLEXE

EXERCICES SUR LE MODÈLE ENTITÉ-ASSOCIATION

1. On considère le modèle suivant décrit selon une notation de Chen avec les cardinalités (*min*:*max*). Vérifier sa concordance avec le modèle de base étudié. Indiquer si les assertions suivantes sont valides :

a) On peut déterminer si une personne est étudiante.
b) L'adresse de mèl pour chaque intervenant d'une formation est connue.
c) Il est possible de dresser la liste de tous les intervenants vus en formation par un étudiant donné.
d) Un intervenant peut encadrer le mémoire d'un étudiant qui n'a pas suivi sa(es) intervention(s).



EXERCICES (SUITE)

2. On considère la cantine d'un hôpital avec le suivi diététique des repas de chaque patient et la traçabilité des produits utilisés. Tous les plats sont confectionnés selon des recettes répertoriées avec l'indication des calories et des matières grasses au niveau de chaque ingrédient. Il faut connaître l'origine de chaque ingrédient au niveau de la réalisation d'un plat par rapport aux approvisionnements caractérisés notamment par le fournisseur et la date d'achat. Les cuisiniers sont aussi répertoriés et on doit pouvoir identifier le(s) auteur(s) de la réalisation d'un plat. De plus, on distingue les cuisiniers apprentis avec l'identification de leur cuisinier tuteur. Créer le schéma conceptuel de ce domaine d'application selon le modèle entité-association de base.

3. Un étudiant obtient un rang et une note en résultat d'un ou plusieurs concours. Proposer une modélisation dans le modèle entité-association de base, puis étendu.

4. Modéliser le modèle entité-association de base par lui-même.

FORMALISME D'UML

UML (*UNIFIED MODELING LANGUAGE*)

● PRINCIPALES CARACTÉRISTIQUES

- Langage pour la modélisation et le développement de logiciel
  Fondé sur le modèle orienté-objet (classes d'objets)
  Utilisé pour spécification, visualisation, création et documentation
- Résultat d'une évolution des méthodes de modélisation
  Unification dans les années 1990 principalement de trois méthodes :
  OMT (*object-modeling technique*) de J. Rumbaugh,
  OOD (*object-oriented design*) de G. Booch,
  OOSE (*object-oriented software engineering*) de I. Jacobson
  Création au sein de l'OMG (*object management group*) en 1997 puis normalisation en 2000 par l'ISO
- Toute une famille de diagrammes
  Présentation ici du diagramme de classe, adapté au schéma conceptuel

● DIAGRAMME DE CLASSE

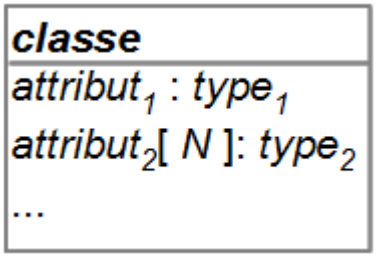
- Entités
  Assimilation à une classe d'objet, **sans** notion d'identifiant
  Attributs avec un type, une répétition (« multiplicité ») éventuelle et la possibilité d'exprimer des contraintes [1] entre les attributs de l'entité
- Association
  Représentation avec l'indication optionnelle d'un sens (rôle) et avec la possibilité de branches multiples (association n-aire)
  Placement de cardinalité à l'inverse du modèle entité-association
  Notations d'une cardinalité : *minimum*`..`*maximum*, `*` pour plusieurs avec les cas particuliers : `1` pour `1..1`, `*` pour `0..*`
  Attributs d'une association noté par une classe, qui est reliée à la branche de l'association
  Cas particuliers d'association
  - Généralisation : la classe B hérite de la classe A
  - Composition : la classe A contient la classe B (dépendante de A)
  - Agrégation : la classe A inclut la classe B (indépendante de A)

PRÉSENTATION ICI SIMPLIFIÉE D'UML

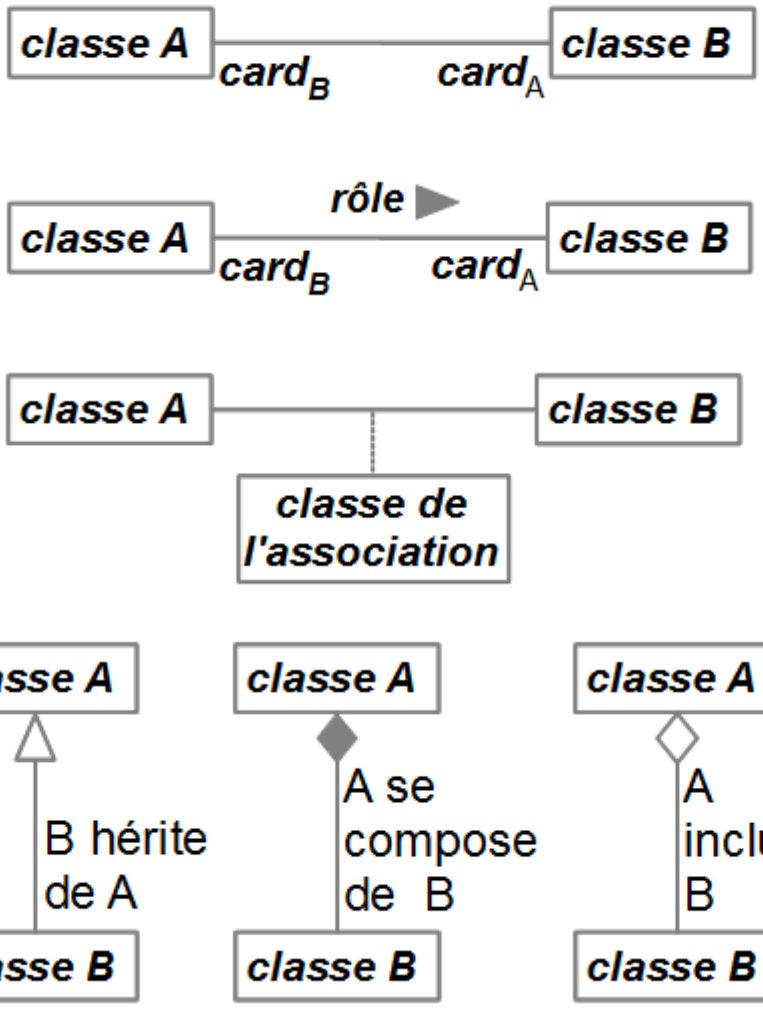
[1] UML dispose d'un langage d'expression de contraintes (*object constraint language*, OCL)

NOTATIONS DE DIAGRAMME DE CLASSE EN UML

- ENTITÉ



- ASSOCIATION

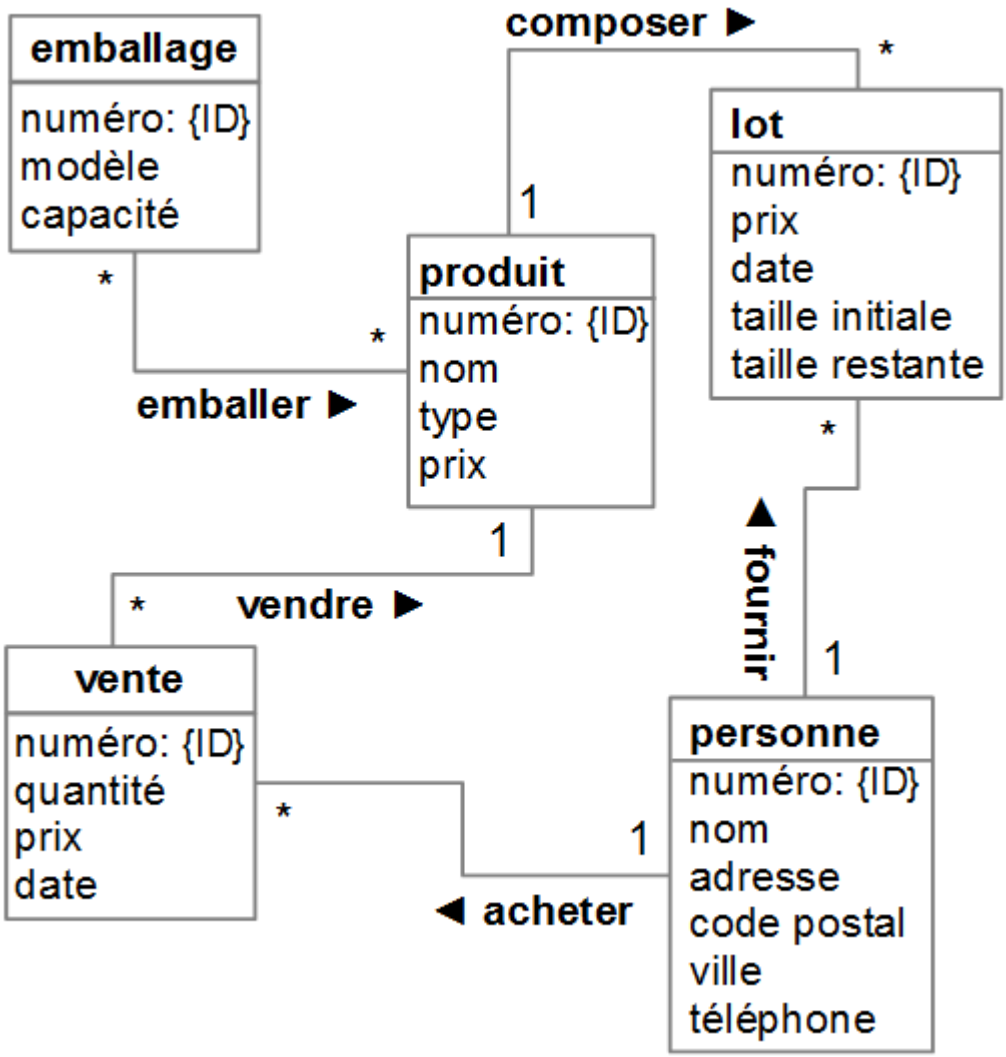


OUTILS SPÉCIALISÉS COMME L'ÉDITEUR ARGOUML [1]

[1] Outil gratuit et libre disponible en http://argouml.tigris.org/

EXEMPLE DE DIAGRAMME DE CLASSE EN UML

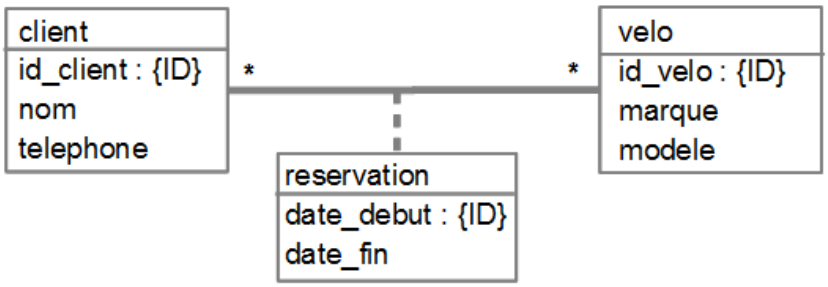
CAS DE LA COOPÉRATIVE



Nota bene : comme la notion d'attribut identifiant n'existe pas dans la notation d'UML, on utilise ici la notation « : {ID} » afin de l'indiquer.

EXERCICES SUR UML

1. Créer le schéma en modèle entité-association de base équivalent au diagramme suivant en UML de la réservation d'un vélo (où la notation « : {ID} » indique un identifiant) :



2. Représenter en UML le modèle de la scolarité d'un étudiant vu à l'exercice n° 1 page 30

3. Représenter en UML le modèle de la cantine de l'hôpital vu à l'exercice n° 2 page 31

MODÈLE RELATIONNEL

UN MODÈLE D'ORGANISATION DES DONNÉES

● CARACTÉRISTIQUES GÉNÉRALES

- Conçu en 1970 par E. CODD [1]
  objectifs d'indépendance des applications vis-à-vis de la représentation interne des données, de cohérence et de non redondance des données
- Un modèle mathématique
  « algèbre relationnelle » basée sur la théorie des ensembles
- Concepts fondateurs des bases de données relationnelles
  à l'origine des tables avec leurs clefs, et du langage SQL

● CONCEPTS DE BASE

- Le domaine
  un domaine D est un ensemble nommé de valeurs
  exemples : domaine `chiffres` des entiers de 0 à 9 `(0, 1, 2, ..., 9)`
  domaine `couleurs` de couleurs fondamentales `(rouge, vert, bleu)`
- La relation
  une relation R sur les domaines D_1, D_2, ..., D_n est un sous-ensemble du produit cartésien des domaines D_1 x D_2 x ... x D_n
  autrement dit, c'est un ensemble de n-uplets [2] dont chaque élément appartient respectivement au domaine D_1, D_2, ..., D_n
  degré de la relation : le nombre de ses domaines (`n` ici)
  attribut A_i : désigne les valeurs de même rang `i` dans la relation
  schéma de la relation : R(A_1 : D_1, A_2 : D_2, ..., A_n : D_n) ou R(A_1, A_2, ..., A_n)
  exemple : relation `symbole(sens : chiffres, apparence : couleurs)` caractérisée par l'ensemble [3] des n-uplets `{ (0, vert), (1, bleu), (1, vert), (2, rouge) }`, et de degré 2 (binaire)
- Correspondances entre les modèles relationnel et entité-association
  attribut ↔ attribut, relation ↔ entité, n-uplet ↔ exemplaire d'entité,

UNE RELATION VA CORRESPONDRE À UNE TABLE DE BASE RELATIONNELLE

[1] Edgar CODD. *A relational model of data for large shared data banks*. Communications of the ACM, vol. 13, n° 6, pages 377-387, juin 1970
[2] Un n-uplet s'appelle en anglais *tuple* ; chacun de ses éléments est bien défini dans son domaine respectif
[3] Cet ensemble de n-uplets est une définition par « extension » de la relation, à l'opposé de la définition par « intention » constituée avec le schéma complet de la relation : R(A_1 : D_1, A_2 : D_2, ..., A_n : D_n)

CLEF

CONCEPT ÉQUIVALENT À L'IDENTIFIANT DU MODÈLE ENTITÉ-ASSOCIATION

- CONCEPT DE CLEF
  - Définition d'une clef de relation

    un ou plusieurs attributs identifiant chaque n-uplet de façon unique

    chaque n-uplet est unique car la relation est un ensemble, autrement dit la combinaison de tous les attributs forment une clef (la plus grande)
  - Clef « candidate », « primaire », « secondaire » [1]

    soit les clefs candidates dont aucune sous-partie forme une clef (il en existe au-moins une), on choisit parmi elles celle composée du nombre minimum d'attributs (ou la plus pertinente) et appelée la clef primaire (autre clef candidate que la primaire, dite secondaire ou alternative)

    notation par soulignement de son ou ses attributs : $R(\underline{A_1, A_2}, ..., A_n)$

    exemple : `passage(date, contrôleur, rapport)`

- CLEF ÉTRANGÈRE
  - Définition d'une clef « étrangère » ou « externe »

    soit E un groupe d'attributs d'une relation R, E est une clef étrangère dans R si tous ses éléments correspondent à ceux d'une clef primaire F dans une relation S (non nécessairement différente de R)
  - Exemple de la coopérative (voir page 19)

    représentation dans le modèle relationnel par les relations :

    `produit(numéro, nom, type, prix)`

    `personne(numéro, nom, adresse, code_postal, ville, téléphone)`

    `lot(numéro, ref_produit, ref_producteur, prix, date, taille_initiale, taille_restante)`

    `vente(numéro, ref_produit, ref_acheteur, quantité, prix, date)`

    avec les clefs étrangères : *ref_produit*, *ref_producteur*, *ref_acheteur*
  - Correspondances avec les associations du modèle entité-association

    association du type 1 à plusieurs ↔ clef étrangère ajoutée dans la relation issue de l'entité au rôle à cardinalité maximale 1

    association du type 1 à 1 ↔ clef étrangère ajoutée dans l'une des relations issue des entités liées

    association du type plusieurs à plusieurs ↔ relation composée des clefs étrangères issues de chaque entité liée

MODÈLE ENTITÉ-ASSOCIATION PLUS EXPRESSIF QU'UN ENSEMBLE DE RELATIONS

[1] En anglais : *superkey* (clef de relation), *candidate* (candidate), *primary* (primaire), *secondary* (secondaire), *alternate* (alternative), *foreign* (étrangère)

EXEMPLE : ENTITÉ EN RELATIONNEL

CAS DE L'ENTITÉ PERSONNE

- Traduction d'une entité vers la base relationnelle

  Définition d'une « table » [1] pour une entité, avec une colonne [2] par attribut et une ligne par élément de donnée (« n-uplet » [3])

  L'identifiant est appelé la « clef » ; il est automatiquement créé un « index » associé dans la base pour la gestion des données [4]

  Le « schéma de la table » indique le nom de la table et la liste des attributs, avec la spécification des caractéristiques de chacun :
  - indication si c'est la clef [5]
  - type de ses valeurs (« domaine »)
  - donnée obligatoire [6] ou facultative

- Schéma de la table « personne »

1) **numero** : nombre entier (clef)
2) **nom** : texte, d'au plus 40 caractères
3) **adresse** : texte, d'au plus 60 caractères
4) **code_postal** : texte, de 5 caractères
5) **ville** : texte, d'au plus 40 caractères
6) **telephone** : texte, de 14 caractères, facultatif [7]

personne

numero	nom	adresse	code_postal	ville	telephone
1	DE LA RUE	9, rue Convention	93100	MONTREUIL	01.48.70.60.00
2	KING KING	1, place d'Italie	75013	PARIS	01.44.08.13.13
3	LILI	3, rue Résistance	95200	SARCELLES	01.34.38.20.00
4	CESAR	80, bd. J. Jaurès	92110	CLICHY	

UNE BASE DE DONNÉE RELATIONNELLE EST UN ENSEMBLE DE TABLES

[1] Une table est aussi appelée une « relation » (vocabulaire du modèle relationnel)
[2] Une colonne s'appelle aussi un « champ » ; ici, un nom de colonne est par précaution noté sans accents et en remplaçant tout espace par un trait de souligné (« _ ») afin d'éviter des anomalies dans les traitements informatiques
[3] L'élément de la table ou « n-uplet » s'appelle aussi un enregistrement, et « *tuple* » en anglais.
[4] Cet index permet de retrouver rapidement un enregistrement à partir de sa clef
[5] Si l'identifiant de l'entité est composé de plusieurs attributs, chacun de ses attributs est alors indiqué comme définissant la clef.
[6] La valeur d'une clef est toujours obligatoire ; on peut aussi indiquer la valeur par défaut en cas d'absence de la donnée lors de l'enregistrement.
[7] Dans cette présentation, seul le cas facultatif est explicitement indiqué (sinon la valeur est obligatoire)

EXEMPLE : ASSOCIATION EN RELATIONNEL

CAS DE L'ASSOCIATION DE LA VENTE

- Traduction dans la base de données relationnelle

  Définition d'une table pour la vente, où chaque entité impliquée dans une association est représentée par sa clef ; ici, clefs d'entités associées : numéro du produit vendu et numéro de la personne acheteuse

  Schéma de la table « vente » :

  1) **numero** : nombre entier (clef)
  2) **ref_produit** : numéro d'identification dans la table « produit »
  3) **ref_acheteur** : numéro d'identification dans la table « personne »
  4) **quantite** : nombre entier de kilogrammes achetés
  5) **prix** : nombre à 2 décimales du prix payé en euros
  6) **date** : de la vente

- Exemple de données

  Exemple : vente n° 1 de 30 kg de brocolis à Lili le 27/12/2005 au prix de 1 € 90 le kilogramme

vente

numero	ref_produit	ref_acheteur	quantite	prix	date
1	**2**	**3**	**30**	**1,90**	**2005-12-27**
2	3	3	3	2,50	2005-12-28

produit

numero	nom	type		prix
		LEGUME	FRUIT	
1	CAROTTE	X		1,00
2	**BROCOLIS**	**X**		**2,00**

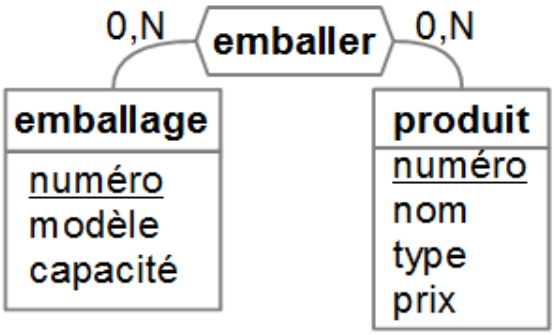
personne

numero	nom	adresse	code_postal	ville	telephone
1	DE LA RUE	9, rue Convention	93100	MONTREUIL	01.48.70.60.00
2	KING KING	1, place d'Italie	75013	PARIS	01.44.08.13.13
3	**LILI**	**3, rue Résistance**	**95200**	**SARCELLES**	**01.34.38.20.00**

EXEMPLE : ASSOCIATION EN RELATIONNEL (FIN)

CAS DE PRODUITS CONDITIONNÉS AVEC DES TYPES D'EMBALLAGE

- EXEMPLE DE L'EMBALLAGE DE PRODUIT
  - Cas d'association « plusieurs à plusieurs » ou « N à N »



  - Représentation dans la base de données

Association représentée par une table de correspondance constituée des clefs de chaque entité associée, avec identification par ces clefs

Schéma de la table « emballer » :

1) **ref_produit** : numéro d'identification dans la table « produit »
2) **ref_emballage** : numéro d'identification dans la table « emballage »

Exemple : carottes conditionnées dans un pochon en papier

emballer

ref_produit	ref_emballage
1	**1**
1	2
2	3

produit

numero	nom	type		prix
		LEGUME	FRUIT	
1	**CAROTTE**	**X**		**1,00**
2	BROCOLIS	X		2,00

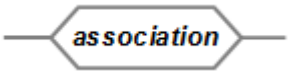
emballage

numero	modele	capacite
1	**pochon en papier**	**1**
2	sac biodégradable	5
3	cagette	10

REPRÉSENTATION PAR UNE TABLE DE DOUBLES RÉFÉRENCES

ASSOCIATION EN RELATIONNEL

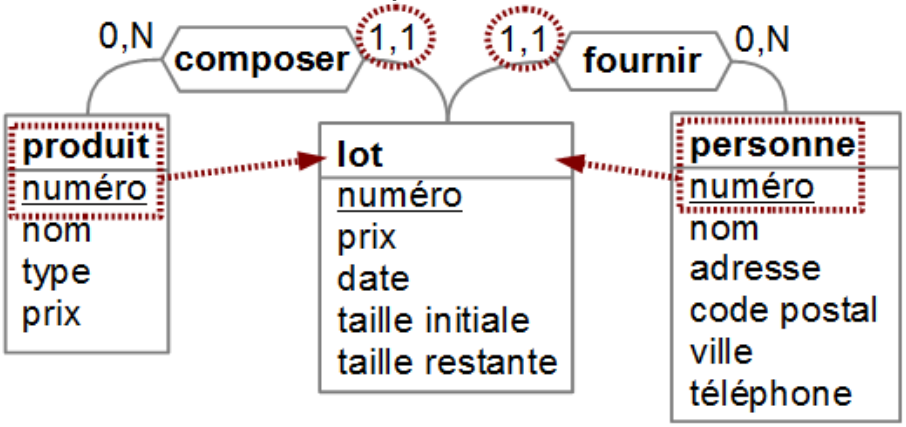
TRADUCTION DANS LE MODÈLE RELATIONNEL



- Cas particulier d'association « 1 à N » avec une cardinalité 1,1 [1]

  Association représentée dans l'entité à cardinalité 1,1, par l'identifiant de l'autre entité liée en tant qu'attribut

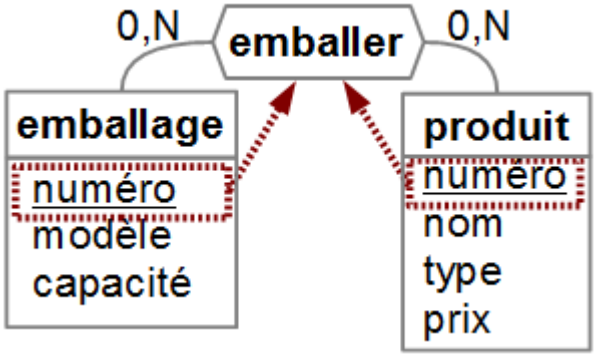
  Exemple : associations « composer » et « fournir »



- Cas général d'une association « N à N » sans cardinalité 1,1

  Association représentée par une table, où chaque entité impliquée dans l'association y est représentée par sa clef en tant qu'attribut

  Exemple : association « emballer »



- Clef primaire et clef étrangère

  La clef représentant une entité dans la table d'une association est appelée « **clef étrangère** » par opposition à la « **clef primaire** » servant d'identifiant dans la table de l'entité [2]

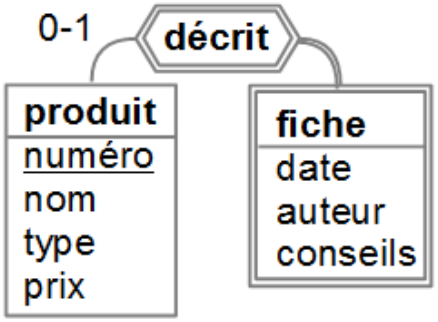
  Exemple : `ref_acheteur` est une clef étrangère dans la table `vente`, `numéro` est la clef primaire dans la table `personne`

UNE ASSOCIATION EST REPRÉSENTÉE EN GÉNÉRAL PAR UNE TABLE, SAUF DANS LE CAS OU UNE DES BRANCHES PORTE UNE CARDINALITÉ 1,1

[1] En fait, il s'agit d'une cardinalité maximale de 1 ; ainsi, cela peut aussi être 0,1
[2] Une clef étrangère est aussi appelée « clef externe » et une clef primaire « clef de relation »

ASSOCIATIONS PARTICULIÈRES EN RELATIONNEL

- ASSOCIATION « UN À UN »



  - Cas d'association avec des cardinalités maximales de 1

    Chaque entité liée participe au plus une fois à l'association
  - Traduction dans le modèle relationnel

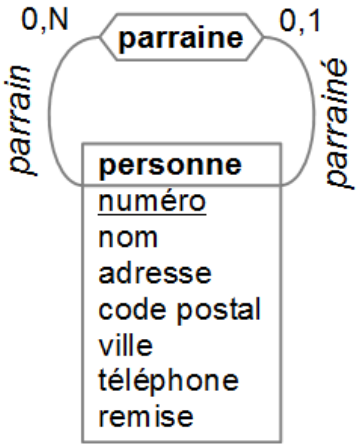
    Si cardinalités 0,1 et 1,1, clef étrangère placée dans la table représentant l'entité de la branche 1,1

    Si cardinalités 0,1 et 0,1, clef étrangère placée dans l'une des tables et déclarée comme facultative

    Si cardinalités 1,1 et 1,1, clef étrangère placée dans l'une des tables et déclarée comme obligatoire

    Exemple : ajout d'un attribut « ref_fiche » dans la table « produit », qui identifie sa fiche descriptive

- ASSOCIATION CYCLIQUE



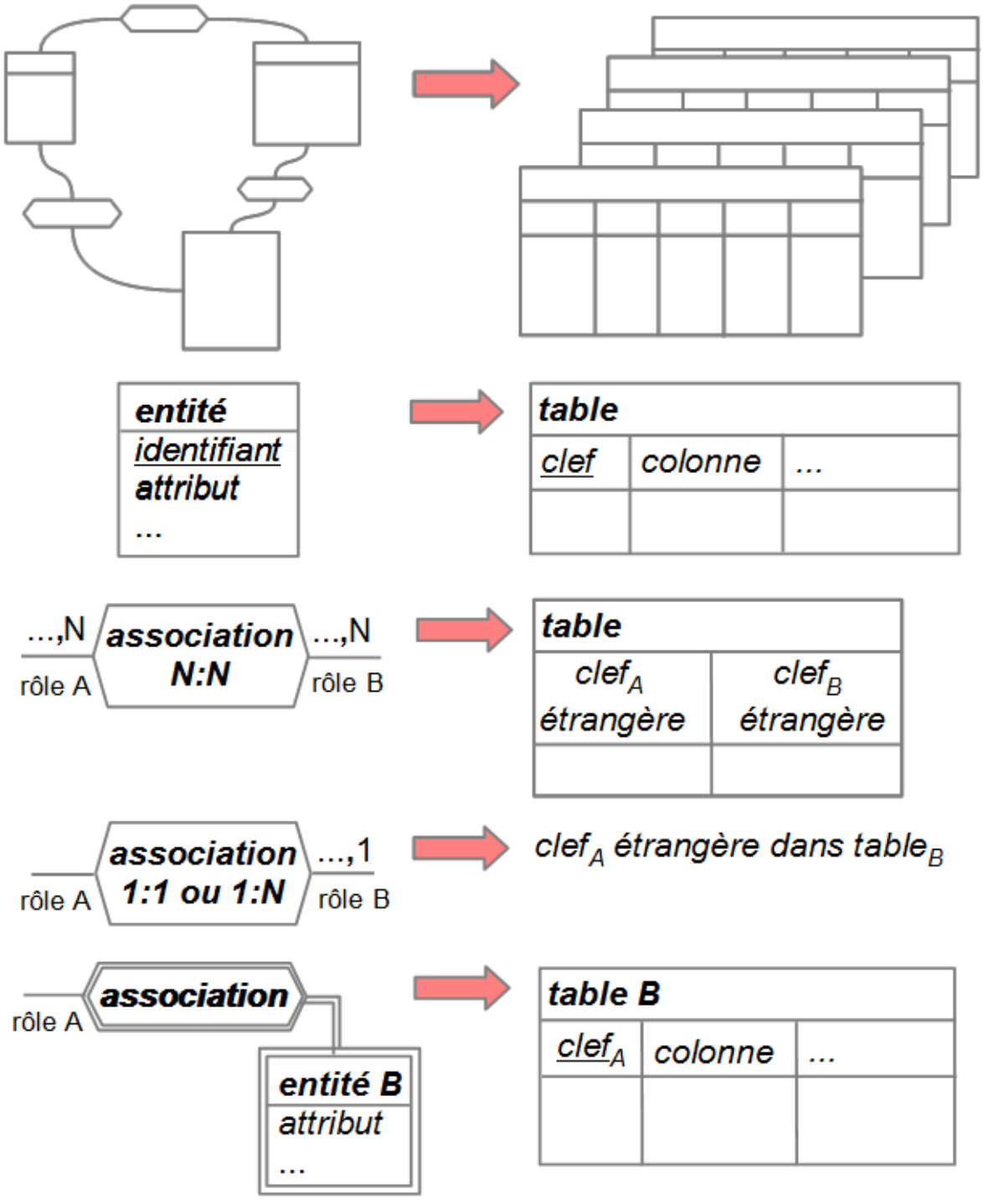
  - Traduction dans le modèle relationnel

    Selon le type d'association, via l'ajout de clef(s) étrangère(s)

    Exemple : ajout d'un attribut « parrain » dans la table personne, qui identifie la personne qui parraine un nouveau client

PASSAGE À LA BASE DE DONNÉES RELATIONNELLE

TRADUCTION [1] DES ENTITÉS-ASSOCIATIONS DU MODÈLE CONCEPTUEL VERS LE SCHÉMA LOGIQUE DE LA BASE DE DONNÉES RELATIONNELLE



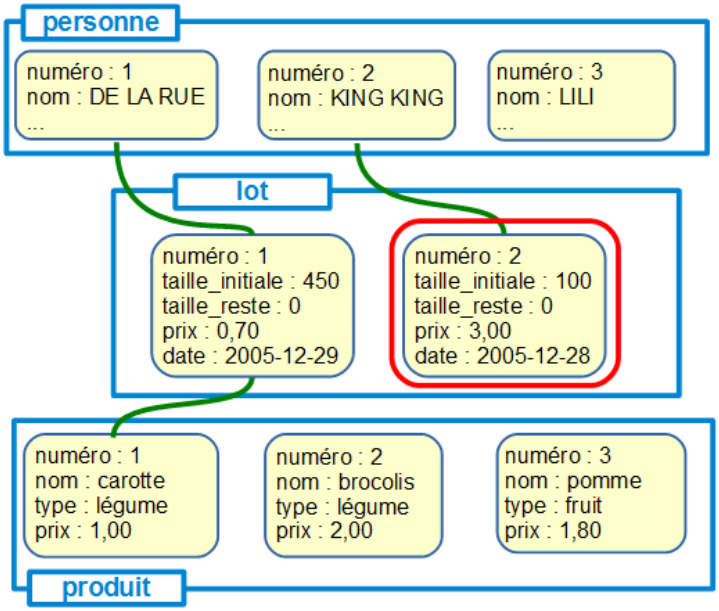
OPÉRATION DE TRADUCTION AUTOMATISABLE

[1] Opération aussi appelée « dérivation » du modèle conceptuel vers le schéma logique de la base

INCOHÉRENCE DES DONNÉES

CAS OÙ LES DONNÉES ENREGISTRÉES SONT INCORRECTES

- CAS D'INCOHÉRENCE
  - Exemple de référence incorrecte : incohérence référentielle
    Cas où un lot fait référence à un produit non enregistré



    Cause possible de cette situation incohérente
    suppression brutale du produit correspondant
  - Autres cas : incohérence de valeur
    Valeur obligatoire absente, ou en dehors du domaine de la colonne

- CONTRAINTES D'INTÉGRITÉ DANS UNE BASE RELATIONNELLE
  - Contrainte d'unicité
    Toute valeur d'une clef est unique
  - Contrainte de domaine
    Toute valeur d'une colonne doit correspondre à son domaine et elle doit être effectivement présente si obligatoire
    Exemple : la valeur d'un type de produit est soit `LEGUME`, soit `FRUIT`
  - Contrainte d'intégrité référentielle
    Toute valeur d'une clef étrangère doit exister comme clef primaire
    Exemple : toute référence à un produit doit exister dans la table *produit*

LA COHÉRENCE EST UNE QUALITÉ INDISPENSABLE DES BASES DE DONNÉES

EXERCICE SUR LA COHÉRENCE

1) DANS LE CADRE DU MODÈLE DE LA COOPÉRATIVE, INDIQUER LES CONTRAINTES D'INTÉGRITÉ RELATIONNELLE

2) EXISTE-T-IL D'AUTRES CONTRAINTES DE COHÉRENCE ? LESQUELLES ?

ALGÈBRE RELATIONNELLE

BASES FORMELLES DES REQUÊTES SUR LES TABLES

● GÉNÉRALITÉS

- Fondement théorique
  formalisation d'opérateurs pour manipuler les tables
- Opérations sur une ou plusieurs relations
  le résultat d'une opération constitue une relation (« fermeture »)

● PROJECTION DE R

- Relation composée avec un sous-ensemble des attributs d'une relation

  notations : R [*liste d'attributs*] ou $\pi_{\textit{liste d'attributs}}$ (R)

  degré de la projection = nombre d'attributs de la liste

  élimination des éventuels doublons de n-uplets en résultat (ensemble)

  exemple : `produit[nom, prix]`

● RESTRICTION (OU SÉLECTION) DE R

- Sous-ensemble des n-uplets d'une relation vérifiant une condition
  condition exprimée par comparaison (=, ≠, <, ≤, >, ≥), appartenance à une liste de valeurs (∈), ou opération logique (et ∧, ou ∨, non ¬)

  notations : R (*condition*) ou $\sigma_{\textit{condition}}$ (R)

  degré de la restriction = degré de R

  exemple : `produit(type = "fruit")`

● PRODUIT CARTÉSIEN DE R PAR S

- Ensemble obtenu en complétant chaque n-uplet de R par un n-uplet de S
  notation : R x S

  degré du produit cartésien = degré de R + degré de S

  exemple : `lot x produit`

ALGÈBRE RELATIONNELLE (SUITE)

- UNION DE R_1 ET R_2 (SCHÉMAS IDENTIQUES)
  - Ensemble [1] des n-uplets dans R_1 ou dans R_2
    notation : $R_1 \cup R_2$
    exemple : `vente(prix > 10)` $\cup$ `vente(quantité > 50)`

- INTERSECTION DE R_1 ET R_2 (SCHÉMAS IDENTIQUES)
  - Ensemble des n-uplets à la fois dans R_1 et dans R_2
    notation : $R_1 \cap R_2$
    exemple : `vente(prix > 10)` $\cap$ `vente(quantité > 100)`

- DIFFÉRENCE DE R_1 PAR R_2 (SCHÉMAS IDENTIQUES)
  - Ensemble des n-uplets dans R_1 mais pas dans R_2
    notation : $R_1 - R_2$
    exemple : `vente(prix > 10)` - `vente(quantité < 5)`

- JOINTURE DE R ET S
  - Ensemble de n-uplets obtenus en complétant chaque n-uplet de R par un n-uplet de S qui vérifient ensemble une condition
    condition exprimée par comparaison d'attributs ($=, \neq, <, \leq, >, \geq$)
    notations : R Θ (*condition*) S ou R $\bowtie$(*condition*) S
    degré de la jointure = degré de R + degré de S
    « thétajointure » dans le cas général, « équijointure » en cas d'égalité
    jointure « naturelle » en cas d'égalité entre attribut identique : R * S
    (attribut commun à R et S présent alors une seule fois en résultat)
    exemple : `lot` $\bowtie$`(ref_produit = numero) produit`

L'UNION, L'INTERSECTION ET LA DIFFÉRENCE SONT DES OPÉRATIONS ENSEMBLISTES

EXERCICES SUR L'ALGÈBRE RELATIONNELLE

1. Exprimer en algèbre relationnelle les extractions suivantes dans le cadre de l'exemple de la coopérative :

   a) Les prix des fruits

   b) Les fournisseurs

   c) Les produits encore en stock

   d) Les fournisseurs qui ne sont pas aussi acheteurs

   e) Les produits fournis par un producteur de Thiverval-Grignon et qui ont été vendus à des clients d'autres communes

2. Réécrire la jointure de deux relations à l'aide d'autres opérations

3. Peut-on transformer l'expression $R(condition_1) \cap R(condition_2)$ en une opération unique ?

4. On définit l'opération relationnelle de division d'une relation R selon une relation R' (composées d'attributs de R), notée R / R', par l'ensemble des n-uplets qui complétés par ceux de R' sont dans R. Construire une exemple de division puis réécrire cette opération à l'aide du produit cartésien, de la différence et de la projection.

DÉPENDANCE FONCTIONNELLE

CONCEPT LIÉ À LA RECHERCHE DE REDONDANCES ET DE CLEFS PRIMAIRES

- CONCEPT DE DÉPENDANCE FONCTIONNELLE
  - Exemple illustratif

    soit la relation `client(numero, nom, département, agent, téléphone)` correspondant à un client identifié par un numéro, décrit par ses nom et département de résidence, et auquel est affecté le conseiller pour le département, décrit par son nom d'agent et son n° téléphonique

    ici la valeur du téléphone est liée à celle de l'agent quelque soit le client
  - Définition d'une dépendance fonctionnelle

    soit E_1 et E_2 deux ensembles d'attributs d'une relation R, E_1 dépend fonctionnellement de E_2 si tous les n-uplets de R avec la même valeur pour E_1 ont aussi à chaque instant la même valeur pour E_2

    expression « E_1 détermine E_2 » dans la relation R, notée : $E_1 \rightarrow E_2$ ou aussi E_1 est le déterminant, E_2 est le déterminé, E_2 dépend de E_1

    exemples : `{département}` → `{agent}` ; `{numero}` → `{nom, département}`
  - Dépendances fonctionnelles associée à une clef

    une clef de relation R détermine l'ensemble de tous les attributs de R ou tout sous-ensemble de ses attributs

    exemples : `{numero}`→ `{nom, département, agent, téléphone}` ; `{numero}`→ `{nom, département}` ; `{numero}`→ `{nom}`

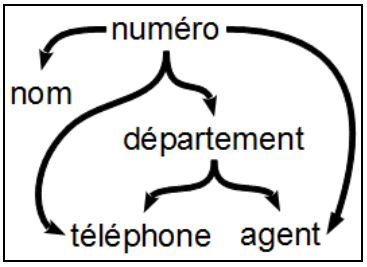
- GRAPHE DES ATTRIBUTS ET DÉPENDANCES FONCTIONNELLES (« GRAPHE ADF »)
  - Dépendance fonctionnelle minimale

    une dépendance fonctionnelle est dite minimale quand le déterminé se réduit à un seul attribut

    exemple : `{département}` → `{agent}` dans la relation `client`
  - Représentation par le graphe des attributs et dépendances fonctionnelles

    arc orienté pour chaque dépendance fonctionnelle minimale de la relation

    exemple :



LA DÉPENDANCE FONCTIONNELLE EST UNE PROPRIÉTÉ DU DOMAINE D'APPLICATION

PROPRIÉTÉS DES DÉPENDANCES FONCTIONNELLES

SOIT UNE RELATION R ET DES ENSEMBLES E_1, E_2, E_3, E_4 D'ATTRIBUTS DE R

- AXIOMES D'AMSTRONG [1]

  1) Réflexivité

  si E_2 est inclus dans E_1, alors $E_1 \rightarrow E_2$

  2) Augmentation

  si $E_1 \rightarrow E_2$ alors $E_1 \cup E_3 \rightarrow E_2 \cup E_3$

  3) Transitivité

  si $E_1 \rightarrow E_2$ et $E_2 \rightarrow E_3$ alors $E_1 \rightarrow E_3$

  4) Décomposition

  si $E_1 \rightarrow E_2 \cup E_3$ alors $E_1 \rightarrow E_2$ et $E_1 \rightarrow E_3$

  5) Union

  si $E_1 \rightarrow E_2$ et $E_1 \rightarrow E_3$ alors $E_1 \rightarrow E_2 \cup E_3$

  6) Pseudo-transitivité

  si $E_1 \rightarrow E_2$ et $E_2 \cup E_3 \rightarrow E_4$ alors $E_1 \cup E_3 \rightarrow E_4$

- TYPES DE DÉPENDANCE FONCTIONNELLE
  - Dépendance fonctionnelle triviale

    $E_1 \rightarrow E_2$ est dite « triviale » quand E_2 est inclus dans E_1 (réflexivité)
  - Dépendance fonctionnelle partielle

    $E \rightarrow \{A\}$ est dite « partielle » quand A n'est pas un attribut de E (non trivialité) et quand il existe un sous-ensemble E' de E tel que $E' \rightarrow \{A\}$
  - Dépendance fonctionnelle élémentaire [2] ou totale ou irréductible à gauche

    $E \rightarrow \{A\}$ est dite « élémentaire » quand elle n'est ni triviale, ni partielle ; autrement dit il n'existe pas de sous-ensemble E' de E tel que $E' \rightarrow \{A\}$
  - Dépendances fonctionnelles et clefs candidates

    tous les attributs d'une relation sont déterminés par une dépendance fonctionnelle élémentaire dont le déterminant est une clef candidate

LES CLEFS SONT LIÉES AUX DÉPENDANCES FONCTIONNELLES

[1] William AMSTRONG. *Dependency structures of database relationships*. IFIP World Congress, North-Holland Ed., pages 580-583, 1974.

[2] Edgar CODD. *Further normalization of the data base relational model*. IBM Research Report RJ909, 1971.

DÉRIVATION ET COUVERTURE MINIMALE

MINIMISATION DU NOMBRE DE DÉPENDANCES FONCTIONNELLES

- ENSEMBLE DE DÉPENDANCES FONCTIONNELLES
  - Dérivation d'une dépendance fonctionnelle

    production d'une ou plusieurs dépendances fonctionnelles « dérivées », par application d'un axiome d'Amstrong à partir de celle considérée

    exemple : `{numero}→ {nom, département}` se dérive par décomposition en `{numero}→ {nom}` et `{numero}→ {département}`

    une dépendance fonctionnelle « de base » n'est pas issue d'une dérivation
  - Fermeture

    soit un ensemble D de dépendances fonctionnelles d'une relation, D augmenté de l'ensemble des dérivations obtenues à partir de ses dépendances fonctionnelles, constitue la fermeture de D notée D^+

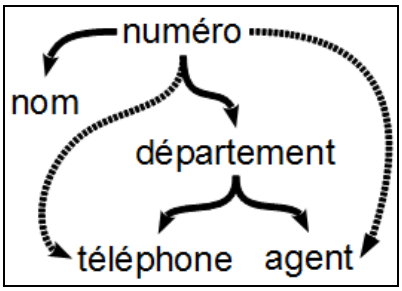
- COUVERTURE MINIMALE (OU IRRÉDUCTIBLE)
  - Définition

    la couverture minimale d'un ensemble D de dépendances fonctionnelles, est un plus petit sous-ensemble C de dépendances fonctionnelles élémentaires dont la fermeture est identique à celle de D ($C^+ \equiv D^+$)
  - Propriétés

    les dépendances fonctionnelles de la couverture minimale permettent de produire par dérivation toutes les autres qui existent dans la relation, et elles sont de base (pas issues d'une dérivation)

    tout ensemble de dépendances fonctionnelles possède au-moins une couverture minimale
  - Exemple pour la relation `client`

    couverture minimale : { `{numero}→ {nom}`, `{numero}→ {département}`, `{département}→ {téléphone}`, `{département}→ {agent}` }



IL EXISTE DES MÉTHODES POUR DÉTERMINER LA COUVERTURE MINIMALE

DÉTERMINATION D'UNE CLEF CANDIDATE

PROCÉDURE DÉFINIE PAR JEAN-LUC HAINAUT [1]

- ATTRIBUTS INTERNES ET EXTERNES
  - Attribut externe

    une dépendance fonctionnelle « externe » détermine un attribut qui ne figure pas en déterminant dans une autre dépendance fonctionnelle, appelé attribut « externe »

    un attribut externe correspond à un sommet terminal du graphe (orienté) des attributs et dépendances fonctionnelles (« graphe ADF »)
  - Attribut interne

    une dépendance fonctionnelle « interne » détermine un attribut qui figure en déterminant dans une autre dépendance fonctionnelle, appelé attribut « interne »

- ALGORITHME
  - Détermination d'une clef candidate pour la relation R

    a) on constitue C avec l'ensemble des attributs de R

    b) on détermine G le graphe ADF de la couverture minimale de C

    c) itérer

    tant qu'il existe un attribut A de C externe dans G

    retirer A de C et mettre à jour G

    recommencer

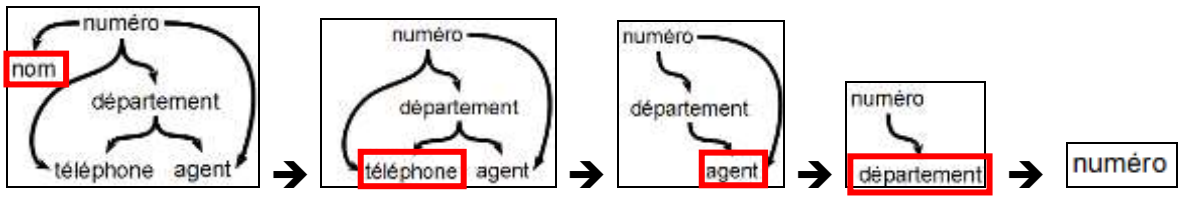
    sortir si G ne contient plus d'arc

    ici, G contient au-moins un circuit

    retirer un attribut impliqué dans un circuit et mettre à jour G

    recommencer

    d) C constitue une clef candidate de R
  - Exemple avec la relation client



EN L'ABSENCE DE CIRCUIT, IL N'Y A QU'UNE SEULE CLEF CANDIDATE

[1] Jean-Luc HAINAUT. Bases de données, concept, utilisation et développement. Dunod, 2012, pages 86-89.

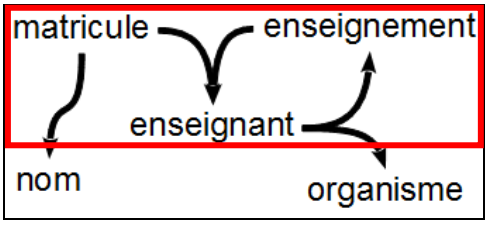
EXEMPLE DE CIRCUIT DANS LE GRAPHE ADF

CAS PARTICULIER DE DÉTERMINATION DE CLEF CANDIDATE

- EXEMPLE DE RELATION

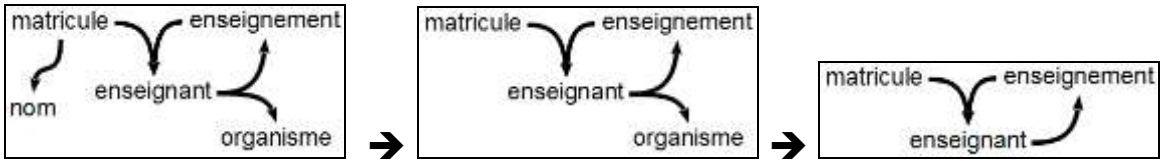
  - Relation `formation`
    un étudiant décrit par son matricule et son nom, suit des enseignements
    un enseignement est assuré par un ou plusieurs enseignants spécialisés dans un enseignement et qui suivent chacun un groupe d'étudiants
    un enseignant est rattaché à un organisme
    `formation(matricule, nom, enseignement, enseignant, organisme)`

  - Circuit dans le graphe ADF
    en effet il y a : `{enseignant}→ {enseignement}`
    et `{matricule, enseignement}→ {enseignant}`



- DÉTERMINATION DE CLEF CANDIDATE

  - Application de l'algorithme (voir page 52)
    on aboutit au bout de 2 étapes au circuit



  - Premier cas de suppression du circuit
    suppression de l'attribut `enseignant`
    d'où la clef candidate `(matricule, enseignement)`

matricule enseignement

  - Second cas de suppression du circuit
    suppression de l'attribut enseignement
    d'où la clef candidate `(matricule, enseignant)`

matricule
enseignant

DANS LE CAS D'UN CIRCUIT, IL Y A PLUSIEURS CLEFS CANDIDATES

EXEMPLE D'ÉLIMINATION DE REDONDANCE

« NORMALISATION » DE RELATION »

- CAS DE LA RELATION `client(numero, nom, département, agent, téléphone)`

client				
numero	**nom**	**departement**	**agent**	**telephone**
1	DE LA RUE	93	PAUL	01.48.70.60.00
2	KING KING	75	DU JARDIN	01.44.08.13.13
3	LILI	95	GLBERTE	01.34.38.20.00
4	MARIUS	75	DU JARDIN	01.44.08.13.13
5	FOU	75	DU JARDIN	01.44.08.13.13

- Redondance au niveau de l'agent affecté à un département

  en fait le département détermine l'agent et son téléphone indépendamment du client

  dépendance fonctionnelle `{département}→ {agent, téléphone}` où le déterminant n'est pas une clef de la relation

- Isolation des attributs de la dépendance fonctionnelle dans une relation

  `conseiller(département, agent, téléphone)`

conseiller		
departement	**agent**	**telephone**
93	PAUL	01.48.70.60.00
75	DU JARDIN	01.44.08.13.13
95	GLBERTE	01.34.38.20.00

- Réduction de la relation `client`

  retrait des attributs déterminés par la dépendance fonctionnelle

  `client(numero, nom, département)`

client		
numero	**nom**	**departement**
1	DE LA RUE	93
2	KING KING	75
3	LILI	95
4	MARIUS	75
5	FOU	75

  le déterminant de la dépendance fonctionnelle devient une clef étrangère

ELIMINATION DE REDONDANCE PAR DÉCOMPOSITION EN SOUS-RELATIONS

UNE MÉTHODE DE NORMALISATION DE RELATION

NORMALISATION SELON LES DÉPENDANCES FONCTIONNELLES

- PRINCIPE DE NORMALISATION

  - Théorème de Heath

    Soit la relation R, soit A, B et C des ensembles d'attributs de R avec la dépendance fonctionnelle $A \rightarrow B$, la jointure de ses projections sur $(A \cup B)$ et $(A \cup C)$ est équivalente à R (schémas et n-uplets identiques)

  - Décomposition de la relation R

    Soit la relation R, soit A, B et C des ensembles d'attributs de R avec la dépendance fonctionnelle $A \rightarrow B$, on peut décomposer R en $R [A \cup B]$ et $R [A \cup C]$ sans perte (la décomposition « préserve les données »)

- NORMALISATION D'UNE RELATION R

  - Principe de l'algorithme de décomposition

    On considère les dépendances fonctionnelles de base et non triviales, dont le déterminant ne constitue pas une clef candidate de la relation (ces dépendances fonctionnelles sont dites « anormales »)

    Pour chacune de ces dépendances fonctionnelles anormales, on décompose progressivement la relation selon le théorème de Heath

  - Algorithme simplifié

    a) Construire le graphe ADF de la couverture minimale de R

    b) Déterminer les clefs candidates de R

    c) Identifier les dépendances fonctionnelles anormales

    d) tant qu'il y a dans R une dépendance fonctionnelle anormale $A \rightarrow B$

    créer une sous-relation S_i avec les attributs de A et de B

    retirer dans R les attributs de B

    recommencer

    e) Regrouper les éventuelles sous-relations créées avec le même déterminant, qui sont équivalentes à la relation de leur regroupement

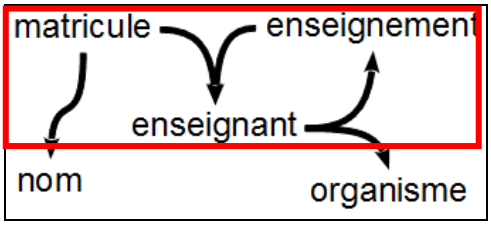
NORMALISATION DITE DE BOYCE-CODD

CAS PARTICULIER DE NORMALISATION

CAS DE PRÉSENCE D'UN CIRCUIT AVEC UNE DÉPENDANCE ANORMALE

- NORMALISATION DE LA RELATION `formation`
  - Circuit dans le graphe ADF de la couverture minimale (voir page 53)

    `formation(matricule, nom, enseignement, enseignant, organisme)`



  - Application de l'algorithme (voir page 55)

    clefs candidates : `(matricule, enseignement), (matricule, enseignant)`

    dépendances fonctionnelles anormales : `{matricule}→ {nom}, {enseignant}→ {organisme}, {enseignant}→ {enseignement}`

    étape 1 : sous-relation `etudiant(matricule, nom)`
    et `formation(matricule, enseignement, enseignant, organisme)`

    étape 2 : sous-relation `intervenant(enseignant, organisme)`
    et `formation(matricule, enseignement, enseignant)`

    étape 3 : sous-relation `specialite(enseignant, enseignement)`
    et `formation(matricule, enseignant)`
  - Décomposition finale en 4 relations

    `etudiant(matricule, nom)`
    `intervenant(enseignant, organisme)`
    `specialite(enseignant, enseignement)`
    `formation`^{bis}`(matricule, enseignant)`

    élimination de redondances mais perte de dépendance fonctionnelle :
    `{matricule, enseignement}→ {enseignant}`

- NORMALISATION ALTERNATIVE POTENTIELLE
  - Conservation du circuit dans la relation

    choix de la clef candidate servant de déterminant

    `etudiant(matricule, nom)`
    `intervenant(enseignant, organisme)`
    `formation`^{ter}`(matricule, enseignement, enseignant)`

    préservation des dépendances fonctionnelles mais redondance :
    `{enseignant}→ {enseignement}`

NORMALISATION ALTERNATIVE DITE EN 3ÈME FORME NORMALE

NORMALISATIONS D'UNE RELATION

PRÉSENTATION ICI SIMPLIFIÉE

- PRINCIPAUX TYPES DE FORME NORMALE
  - Première forme normale (1FN)

    relation dont toutes les valeurs d'attributs sont atomiques

    autrement dit, il ne peut pas y avoir de valeur multiple ou composée
  - Deuxième forme normale (2FN)

    relation en 1ère forme normale et tout attribut n'appartenant à aucune clef candidate ne dépend pas que d'une partie d'une clef candidate
  - Troisième forme normale (3FN)

    relation en 2ème forme normale et tout attribut A n'appartenant à aucune clef candidate ne dépend **directement** que des clefs candidates [Codd]

    (un attribut dépend « directement » d'un ensemble d'attributs X s'il n'existe pas un ensemble d'attributs Y tel que $X \rightarrow Y$ et $Y \rightarrow \{A\}$ avec A n'est pas dans Y, Y n'est pas dans X, Y ne dépend pas de X)

    ou aussi : pour chaque dépendance fonctionnelle $X \rightarrow \{A\}$ où A est un attribut, soit A est un élément de X, soit X est un clef, soit A appartient à une clef candidate [Zaniolo]
  - Forme normale de Boyce-Codd (FNBC)

    relation dont les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clef candidate détermine un attribut

    toute relation en FNBC est aussi en 3FN (mais pas l'inverse)

- PRINCIPE GÉNÉRAL DE NORMALISATION
  - Construction du graphe ADF

    avec les dépendances fonctionnelles d'une couverture minimale
  - Normalisation de la relation

    décomposition en sous-relations selon l'algorithme (voir page 55)

    choix d'une décomposition finale dans le cas d'un circuit avec présence d'une dépendance anormale, en FNBC ou en 3FN

PLUSIEURS NORMALISATIONS POTENTIELLES POUR UNE RELATION

EXERCICES SUR LA NORMALISATION

1. Démontrer que les trois derniers axiomes d'Amstrong dérivent logiquement des trois premiers.

2. On considère la représentation à un instant donné de pièces de théâtre dans les salles de différentes villes par des compagnies de théâtre disposant d'un régisseur attitré ; chaque salle est identifiée globalement ; une pièce peut être jouée dans une ville par plusieurs compagnies mais dans des salles distinctes ; voici la relation correspondante :
   `représentation(pièce, ville, salle, compagnie, régisseur)`

   a) Construire une couverture minimale des dépendances fonctionnelles et représenter son graphe ADF

   b) Déterminer la ou les clefs candidates de la relation

   c) La relation est-elle normalisée ? Si non, en trouver les décompositions normalisées possibles

   d) On considère maintenant une variante de la relation précédente avec l'extension du programme des représentations sur une soirée, le partage du régisseur entre plusieurs compagnies, et la possibilité de succession de plusieurs pièces différentes dans une salle pendant une soirée. Déterminer la couverture minimale des dépendances fonctionnelles, les clefs candidates et normaliser la relation si besoin.

LOGICIEL DE MODÉLISATION DB-MAIN

PRÉSENTATION DE LA VERSION 9.2B DE DB-MAIN [1]

- PRINCIPALES FONCTIONNALITÉS
  - Outil gratuit d'aide à la conception de bases de données
    Création de schéma conceptuel et traduction vers une base relationnelle
    Issu du monde universitaire (Université de Namur, Belgique)
    Disponible pour un environnement Windows ou Linux (avec Java)
  - Plusieurs formalismes disponibles
    Entité-association étendu (ERA dans DB-MAIN), UML, modèle relationnel
    Conversion possible d'un schéma entre entité-association étendu et UML
  - Production d'un schéma physique pour plusieurs SBD relationnels
    En SQL 92, Microsoft Access, Firebird, MySQL, PostgreSQL et SQLite
- PRINCIPES D'UTILISATION
  - Organisation en « projet »
    Regroupement de plusieurs schémas conceptuels avec la trace des traductions réalisées
  - Procédure élémentaire type
    1) Création d'un schéma conceptuel dans un formalisme (ERA ou UML)
    2) Traduction du schéma conceptuel en schéma logique relationnel
    3) Production du code pour la création du schéma physique
- PRINCIPALES EXTENSIONS AU MODÈLE ENTITÉ-ASSOCIATION DE BASE
  - Attribut
    Attribut multivalué, contraintes d'existence
  - Association
    ordre supérieur à 2 (ternaire, ...), attributs propres, cardinalité généralisée
  - Entité
    Généralisation-spécialisation (« est un »)

[1] Outil gratuit disponible en http://db-main.eu/ ; une simple procédure d'inscription donne accès à une licence d'utilisation pour une période de 12 mois renouvelable.

DB-MAIN : SCHÉMA ENTITE-ASSOCIATION

DÉMARRAGE

1) CRÉATION D'UN NOUVEAU PROJET

Commande FILE NEW PROJECT

Attribution d'un nom au projet

Après validation, apparition de l'onglet dédié au projet

Sauvegarde via la commande FILE SAVE PROJECT

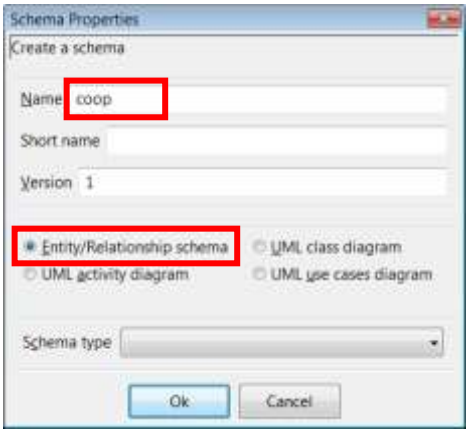
2) CRÉATION D'UN NOUVEAU SCHÉMA

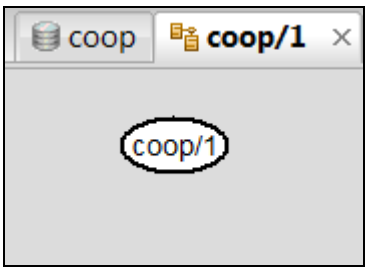
Commande PRODUCT NEW SCHEMA

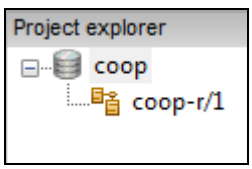
Attribution d'un nom au schéma et choix du formalisme

Après validation, apparition de l'onglet dédié au schéma

Autre vue dans le panneau de l'explorateur de projet (commande WINDOW PROJECT EXPLORER)







DB-MAIN : ENTITÉ

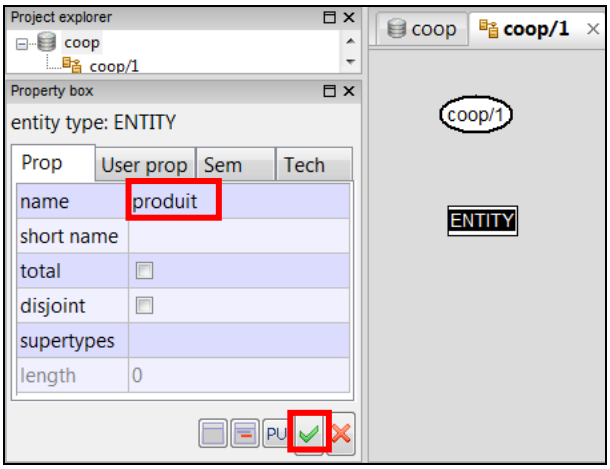
CONSTRUCTION D'UN SCHÉMA CONCEPTUEL SELON LE MODÈLE ENTITÉ-ASSOCIATION

a) DÉFINITION D'UNE ENTITÉ

Ajout d'une entité via la commande NEW ENTITY TYPE ou le bouton 

Indication du nom de l'entité dans la boîte de dialogue des propriétés associée puis validation

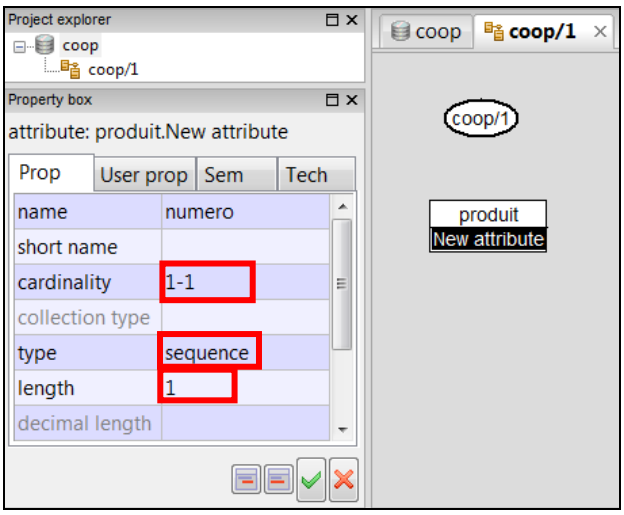
Astuce : recliquer sur le bouton  pour retrouver l'usage normal du pointeur de la souris (sélection)



Ajout du premier attribut via la commande NEW ATTRIBUTE FIRST ATT... ou le bouton 

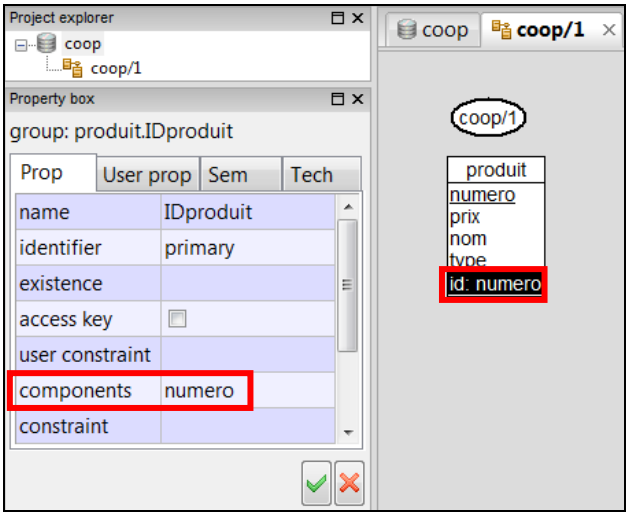
Indication de nom, type [1] et taille [2], caractère obligatoire (cardinalité 1-1) ou facultatif (0-1) de l'attribut dans la boîte de dialogue des propriétés puis validation

Idem pour les autres attributs avec la commande NEW ATTRIBUTE NEXT ATT... ou le bouton 



Indication de l'identifiant par sélection de l'attribut puis clic sur le bouton ID (soulignement de l'attribut et ligne tout en bas débutant par `id:` suivi de l'attribut)

En cas de composition de l'identifiant par plusieurs attributs, soit effectuer une sélection multiple de tous les attributs avant de cliquer sur ID, soit créer l'identifiant avec le 1er attribut puis y ajouter les attributs suivants via sa propriété COMPONENTS



[1] Type `sequence` pour un numéro d'identification automatique, `compound` pour un attribut composé

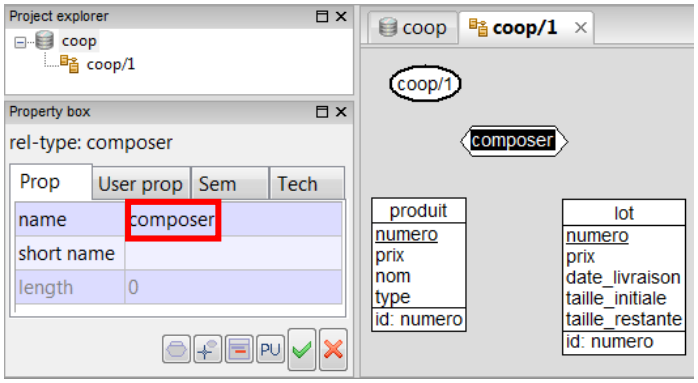
[2] La propriété de taille (`length`) sert en cas d'un texte (nombre maximal de caractères) ou d'un nombre réel à précision fixe (nombre global de chiffres, y compris ceux décimaux, fixé par `decimal length`)

b) DÉFINITION D'UNE ASSOCIATION

Ajout de l'association via NEW REL-TYPE ou bouton

Indication du nom de l'association dans la boîte de dialogue des propriétés associée puis validation

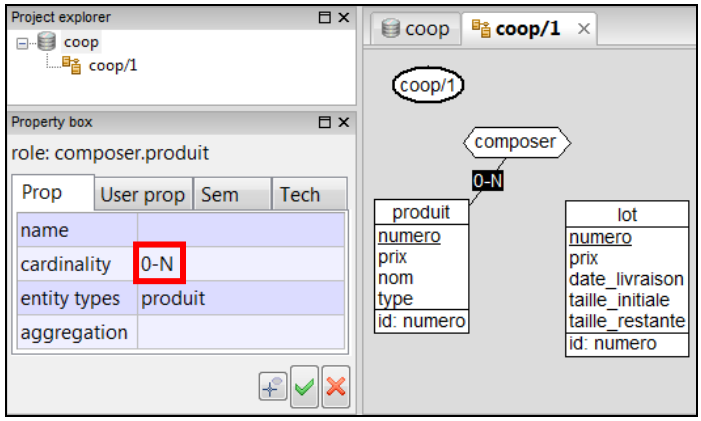
Astuce : pour déplacer un élément du schéma avec le pointeur de la souris (sélection activée), le faire glisser en cliquant sur son nom



Ajout d'une branche via NEW ROLE/REL-TYPE/LINK... ou bouton

Cliquer sur l'association puis faire glisser afin d'étendre la branche apparue jusque sur l'entité liée

Indication de la cardinalité et si besoin du rôle (NAME) dans la boîte de dialogue des propriétés puis validation

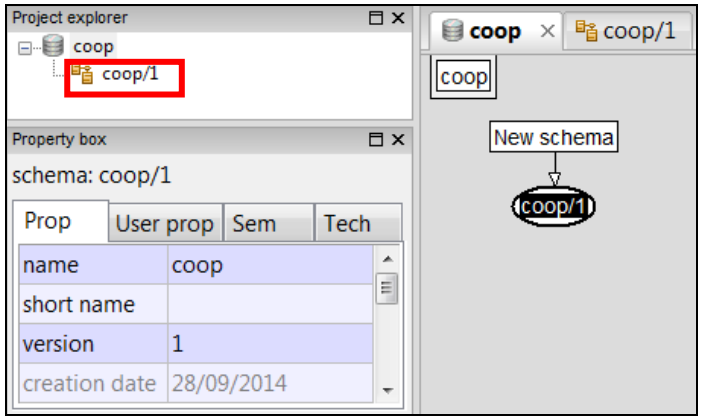


c) FIN DE CRÉATION DU SCHÉMA

Ne pas oublier d'enregistrer le projet (FILE SAVE PROJECT)

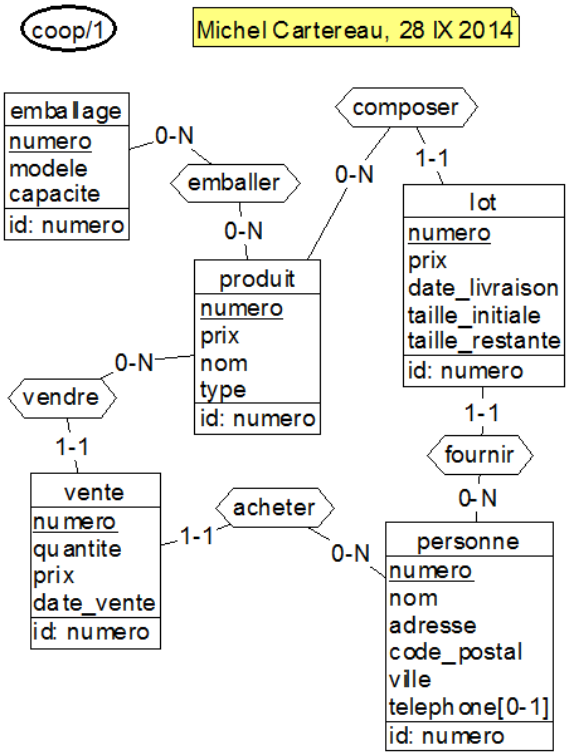
Retour au schéma via son onglet ou si absent, par double-clic sur son icône dans le panneau d'exploration de projet

Indication des phases de la conception dans le panneau du projet

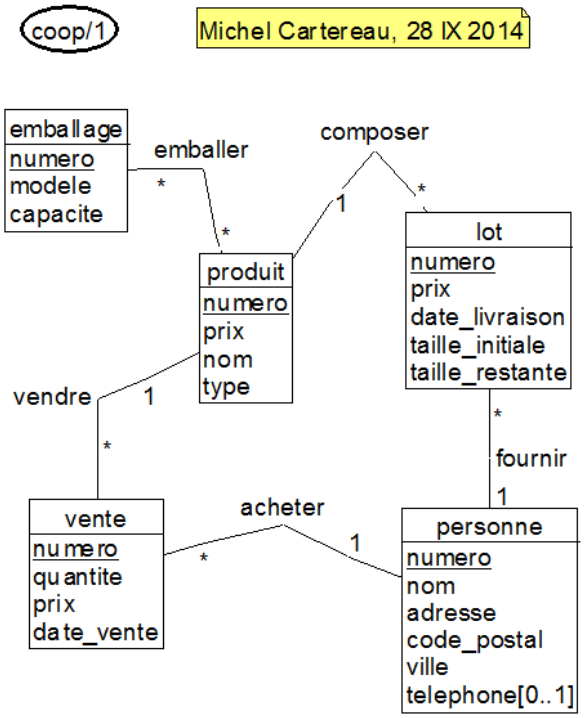


DB-MAIN : TRADUCTION EN UML

- SCHÉMA CONCEPTUEL FINAL DE LA COOPÉRATIVE EN ENTITÉ-ASSOCIATION



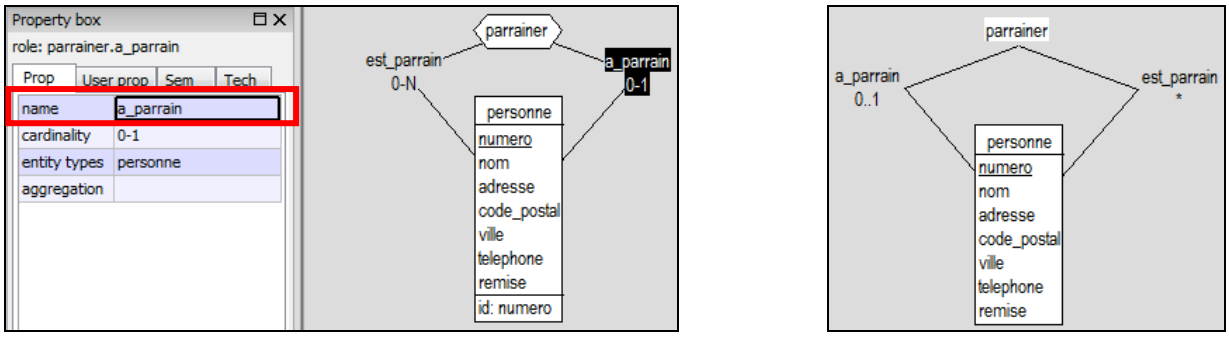
- Traduction selon le formalisme d'UML (et vice-versa)
  commande TRANSFORM ERA -> UML CLASS



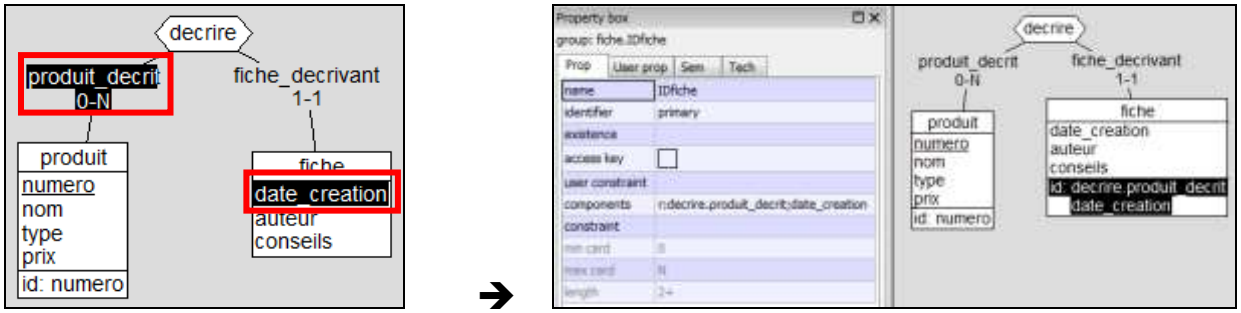
- Insertion d'une note dans le diagramme
  commande NEW NOTE

COMPLÉMENTS SUR LES ÉLÉMENTS D'UN MODÈLE CONCEPTUEL

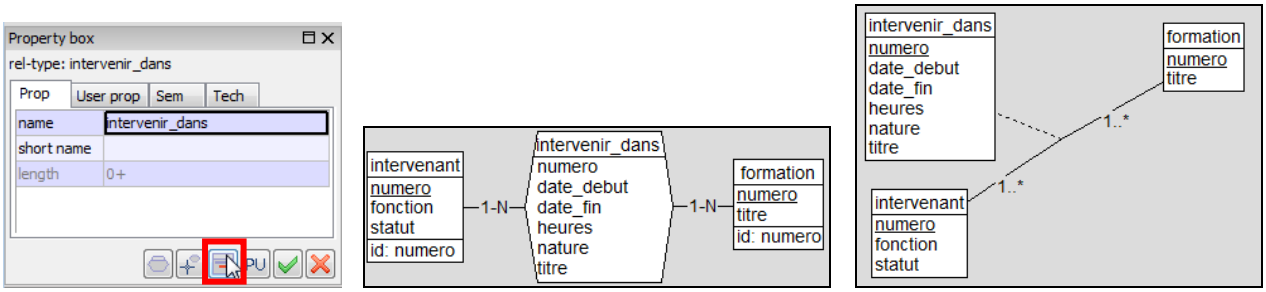
- Association cyclique : donner un nom à chacun de ses rôles



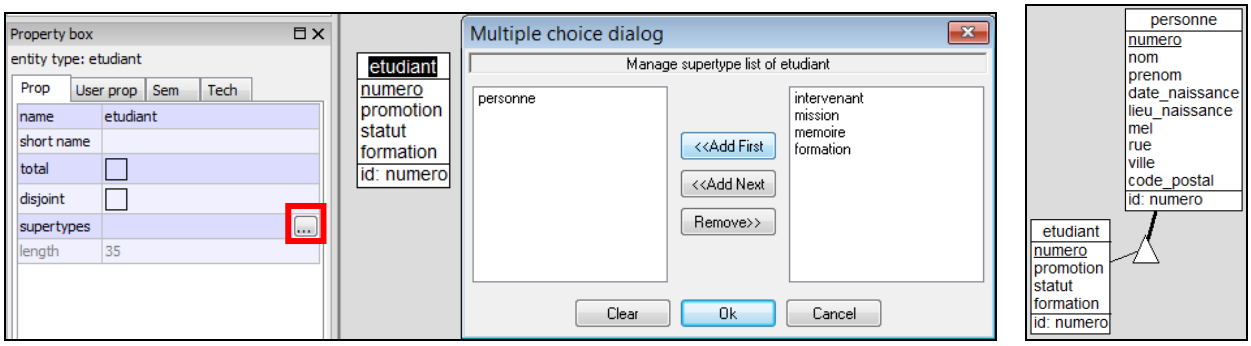
- Identification implicite d'entité faible reconnue automatiquement, avec une clef primaire attribuée automatiquement lors de la traduction en relationnel
- Identication hybride d'entité faible : sélection du rôle et de l'attribut puis ID



- Attributs d'association : créer l'association puis définir ses attributs

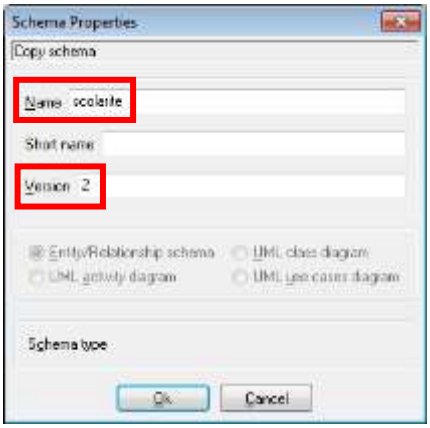


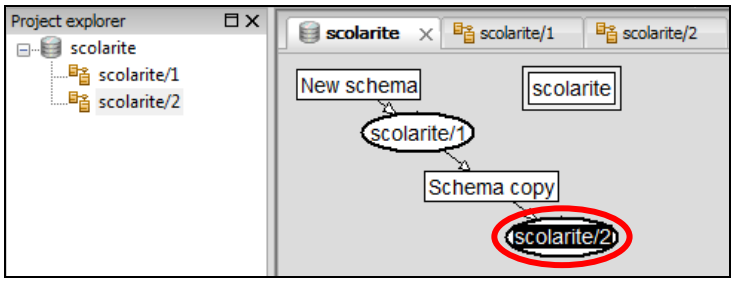
- Généralisation/spécialisation : propriété `supertypes` de la sous-entité



DB-MAIN : MANIPULATIONS GÉNÉRALES

- Sélection multiple : par le lasso, ou sélections avec la touche CTRL enfoncée
- Suppression : sélection et touche SUPPR, ou clic-droit puis DELETE
- Duplication d'un schéma : PRODUCT COPY PRODUCT puis indiquer nom et-ou version du nouveau schéma (opération indiquée dans l'onglet du projet [1])





- Duplication d'un élément : EDIT COPY/PASTE ou via le clic droit (au sein d'un schéma, ou entre les schémas d'un projet)
- Barre d'outils de zoom et alignement graphique : WINDOW GRAPHICAL TOOLS

- Barre d'outils pour l'alignement d'un rôle sur sa branche en UML : WINDOW UML ROLE POSITION TOOLS

- Format du texte (global au schéma) : EDIT CHANGE FONT
- Couleur de texte : soit via la sélection d'éléments puis EDIT CHANGE COLOR, soit par choix d'une couleur (EDIT CHANGE COLOR), sélection des éléments puis EDIT COLOR SELECTED ; annulation via EDIT REMOVE COLOR
- Présentation des entités et associations : VIEW GRAPHICAL SETTINGS arrondi des formes, visualisation du domaine des attributs
- Transformer un schéma en image : sélection des éléments puis EDIT COPY GRAPHIC (image créée dans le presse-papiers)

CONSULTER AUSSI LA DOCUMENTATION DE DB-MAIN [2]

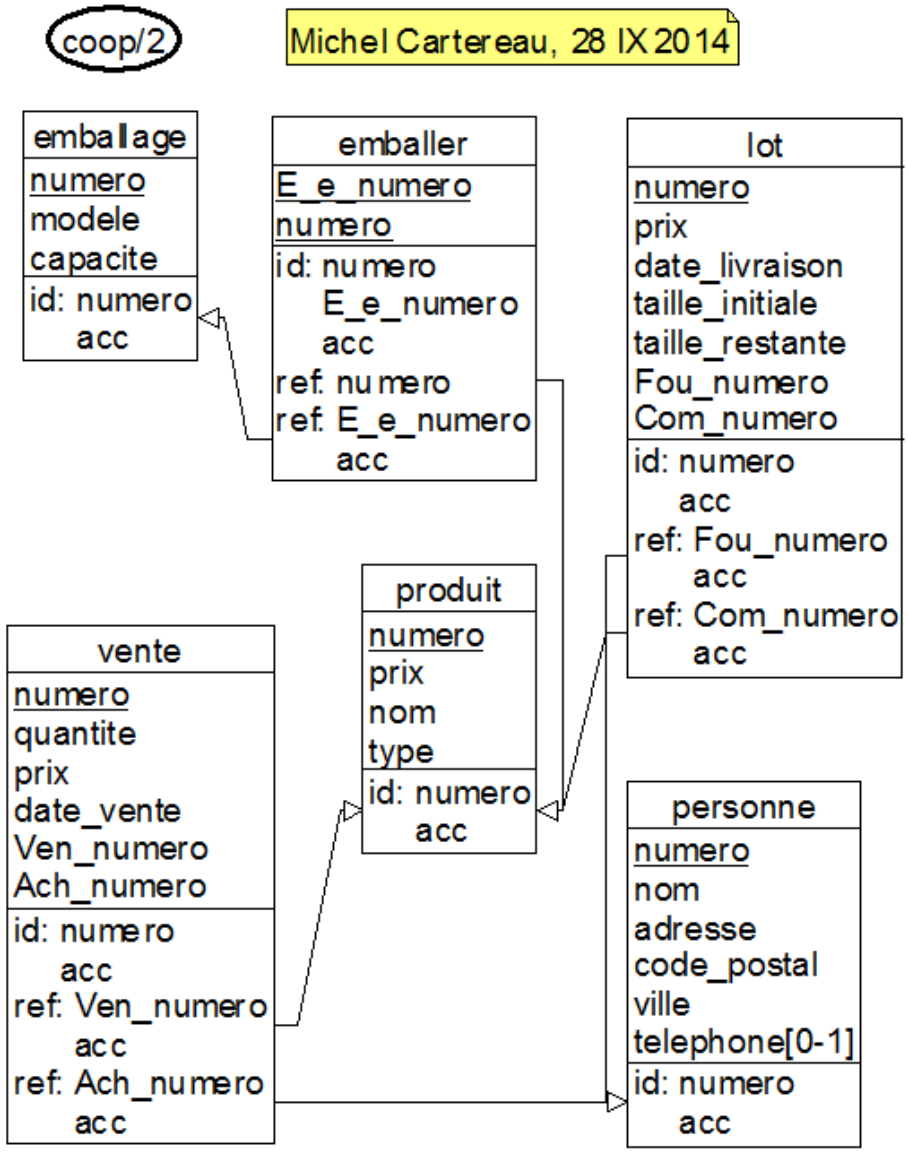
[1] Ouverture et-ou suppression d'un schéma par action sur sa représentation dans l'onglet du projet
[2] La documentation est automatiquement installée avec le logiciel (sous-dossier `doc`)

DB-MAIN : SCHÉMA RELATIONNEL

OBTENTION À PARTIR D'UN SCHÉMA CONCEPTUEL (UML OU ERA)

a) Réalisation d'une copie[1] du schéma conceptuel par PRODUCT COPY PRODUCT

b) Traduction du schéma en relationnel par TRANSFORM RELATIONAL MODEL



DB-MAIN ajoute les indications `ref:` (*reference*) sur les clefs étrangères, et `acc` (*access key*) pour indiquer un index (voir en SQL)

Possibilité de renommer les clefs étrangères, qui ont été créées automatiquement par DB-MAIN à l'aide du nom des associations

[1] Réalisation d'une copie car après la transformation du schéma en relationnel, il n'est plus possible de revenir au schéma conceptuel original.

SQL

Généralités

Définition des données

Manipulation des données

Transactions

Index de table

Programmation

LE LANGAGE SQL

LANGAGE DE REQUÊTE STRUCTURÉE (*STRUCTURED QUERY LANGUAGE*)

- HISTORIQUE
  - Recherche initiale d'Edgar CODD sur un langage du modèle relationnel
    langage ALPHA proposé en 1971
  - Premiers prototypes
    SGBD Ingres (*interactive graphics retrieval system*) et langage Quel (*query English language*), Eugene WONG et Michael STONEBRAKER, vers 1974, université de Berkeley (Californie)
    SGBD System R et langage Sequel [1] (*structured English query language*), Donald CHAMBERLIN et Raymond BOYCE, vers 1974, IBM
  - Premiers produits commercialisés
    SGBD Oracle V2, 1979, Relational software (devenu Oracle corporation)
    SGBD SQL/DS (*structured query language/data system*) en 1981 puis DB2 (*database 2*) en 1983, IBM

- NORMALISATION
  - Processus permanent
    travaux conjoints de la commission électrotechnique internationale (IEC) et de l'organisation internationale de normalisation (ISO)
    respect de la norme très variable selon les différents éditeurs de SGBD
  - Principales évolutions
    SQL 1 (1986 et 1989) : 1ère normalisation
    SQL 2 (1992) : nouveaux types (texte à taille variable, date, heure, durée), alphabet, opérations ensemblistes (intersection, différence), jointure explicite, assertion, contrainte, transaction, etc.
    SQL 3 (1999, 2003, 2006, 2008, 2011) : nouveaux types (bloc de bits/ caractères, logique, tableau, collection), type orienté-objet (héritage), comparaison avec expression rationnelle, déclencheur, requête récursive, procédure interne, analyse multidimensionnelle (*OLAP*), XML avec XQuery (2003, 2006, 2008), données temporelles (2011)

EXISTENCE D'AUTRES LANGAGES DONT QBE (*QUERY BY EXAMPLE*)

[1] Sequel a été renommé en SQL car c'était alors une marque déposée par une autre société ; cependant, SQL se prononce en anglais comme Sequel

CARACTÉRISTIQUES GÉNÉRALES DE SQL

UN LANGAGE COMPLET ET PUISSANT

- PARTICULARITÉS

  - Langage accessible aux utilisateurs
    objectif d'un langage simple, rigoureux, pour des non-informaticiens
  - Langage déclaratif à la base
    indication du résultat à obtenir mais pas de la procédure à suivre
  - Manipulation de la structure des données
    structure stockée dans la base avec les données
  - Prise en compte de l'absence de valeur
    logique particulière à trois états : vrai, faux ou inconnu
  - Possibilités de programmation
    ouvertures vers des langages et procédures internes

- SECTIONS PRINCIPALES DU LANGAGE

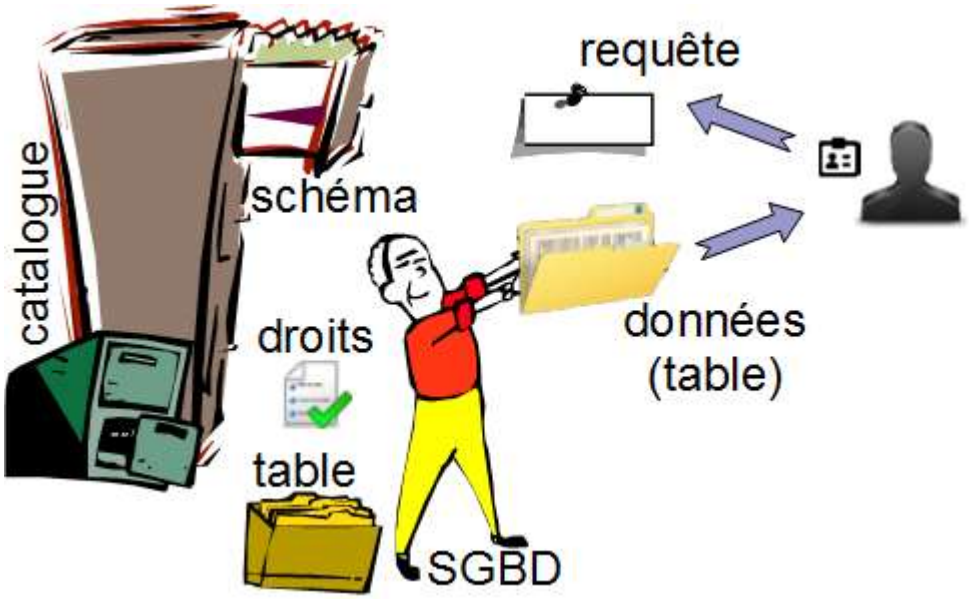
  - Définition des données (*data definition language*, *DDL*)
    création et modification de la structure des données
  - Manipulation des données (*data manipulation language*, *DML*)
    interrogation, ajout, mise à jour et suppression de données
  - Transactions (*transactional control language*, *TCL*)
    gestion des exécutions simultanées (concurrence d'accès)
  - Gestion du contrôle d'accès (*data control language*, *DCL*)
    définition de comptes d'accès et de droits associés
  - Programmation
    en interne à la base ou dans une application
  - Divers outils d'administration
    optimisations, sauvegardes, répartition, etc.

PRÉSENTATION ICI DE L'ESSENTIEL DU LANGAGE
DANS LE RESPECT DE LA NORME (SQL 2)

BASE DANS SQL

APPLICATION DU MODÈLE RELATIONNEL PARTICULIÈRE DANS SQL

- BASE
  - Organisation fondamentale en tables
    une relation représentée par une table, un attribut par une colonne
  - Regroupement de tables au sein d'un « schéma »
    structuration modulaire de la base avec des utilisateurs associés
  - Représentation d'une base par un « catalogue »
    regroupe un ou plusieurs schémas (dépend en fait du SGBD)
  - Définition des objets de la base au sein du schéma
    tables, vues, domaines, droits d'accès, etc. [1]



- REQUÊTE
  - Elément fondamental du langage SQL
    utilisée pour toute opération sur la base
  - Instruction exécutée par le SGBD
    renvoi de résultats sous forme générique d'une table

GÉNÉRALEMENT, CATALOGUE À SCHÉMA UNIQUE ET ASSIMILÉ À UNE BASE

[1] Cette présentation n'abordera pas des objets mineurs : définition d'alphabets, d'interclassements, d'assertions et de tables de conversion

SQL : SYNTAXE GÉNÉRALE

- Mots-clefs de SQL
  insensibles à la casse (minuscule et majuscules équivalentes)
  généralement réservés pour l'écriture d'une requête
  (ne pas s'en servir pour désigner autre chose, voir page 123)
  exemples : `select  KEY  Module`

- Identificateur (nom de base, table, colonne, etc.)
  composé de lettres, chiffres ou trait de souligné (« _ »)
  débute par une lettre et contient au maximum 128 symboles [1]
  a priori se limiter à l'alphabet ASCII (pas de lettres accentuées etc.)
  exemples : `coop  detail  QStock  code_postal  adresse1`
  contre-exemples : `_coop  2N  net&euro  H-T  type mobilier  réf`

- Commentaire
  texte débutant par un double trait d'union (« `--` ») jusqu'en fin de ligne
  texte placé entre les symboles « `/*` » et « `*/` » [SQL 1999]
  exemples :

```
-- vu par Tarzan
/* fait par Michel Cartereau le 14 X 2013 */
```

- Séparateur de mots (« blancs »)
  un ou plusieurs espaces, tabulations, ou retours à la ligne

- Instruction
  notation sur une ou plusieurs lignes,
  insertion possible de commentaire
  dans le cas d'une suite de requêtes ou d'une exécution interactive,
  point-virgule (« ; ») à la fin
  exemples :

```
insert into produit (numero, nom, type, prix)
values (1, 'CAROTTE', 'LEGUME', 1.2)

create domain type_entier as integer ;
create domain type_monetaire as decimal(7,2) ;
create domain code_pays as char(2) ;
```

[1] En pratique, on se limite à 60 caractères pour la compatibilité entre différents SGBD

SQL : VALEURS ÉLÉMENTAIRES

- Texte (« chaîne de caractères »)
  noté entre apostrophes (« ' »)
  une véritable apostrophe dans le texte est dédoublée
  possibilité de découper un texte long en plusieurs morceaux notés entre apostrophes et séparés par un blanc
  exemples :

```
'base de données'
'Gaia d''AgroParisTech'
'et voilà un texte vraiment très long '
'qui s''étale sur deux lignes !'
```

- Valeur numérique
  nombres entiers ou réels, notation à puissance de 10 avec « `e` » ou « `E` »
  exemples : `1235  -69  3.14159  -1.8E3`

- Valeur temporelle
  date : `date '`*année*`-`*mois*`-`*jour*`'`
  heure : `time '`*heure*`:`*minute*`:`*seconde*`'`
  horodate : `timestamp '`*année*`-`*mois*`-`*jour heure*`:`*minute*`:`*seconde*`'`
  durée : `interval '`*année*`', interval '`*année*`-`*mois*`', interval '`*mois*`', interval '`*jour heure*`:`*minute*`:`*seconde*`', interval '`*jour*`'` [etc.]
  exemples :

```
date '2013-9-30'
time '11:59:00'
timestamp '2013-9-30 11:59:00'
interval '10:30'
```

- Absence de valeur
  notée par le mot-clef `null`
  cela correspond généralement à une valeur inconnue (ou pas encore connue) ou à l'impossibilité d'attribuer une valeur à une colonne
  attention ! ce n'est pas une valeur nulle (0) ou un texte vide (`''`)

- Valeur logique [SQL 1999]
  `true` (vrai) ou `false` (faux) ou `unknown` (inconnu)

SQL : CRÉATION D'UNE BASE

- CRÉATION D'UNE BASE

  - Catalogue

    commande non définie par la norme et dépendant du SGBD

    syntaxe générale de l'instruction : `create database` *nom*

    option générales : définition de l'aphabet de référence, du propriétaire

    exemple : `create database coop`

  - Schéma

    syntaxe de l'instruction : `create schema` *nom options*

    option de l'aphabet de référence : `default character set` *nom*

    option du propriétaire : `authorization` *utilisateur* (voir page 81)

    exemple : `create schema coop default character set 'UTF8'`

- ALPHABET ET INTERCLASSEMENT

  - Jeu de caractèresj (*charset, encoding*)

    dépend du SGBD et du système d'exploitation

    généralement au-moins `'ascii'`, `'latin1'` (ISO 8859-1), `'UTF8'` (unicode)

    indication possible pour une chaîne de caractères en la préfixant par un trait de souligné « _ » suivi du nom de l'aphabet, ou par l'abréviation « `N` » (*national*) pour l'alphabet de référence du SGBD

    exemples : `_UTF8'Švejk'` `N'Švejk'`

  - Interclassement (*collation*)

    méthode de comparaison et de tri des symboles :

    - casse différenciée ou non

    - lettre accentuée identique ou non à la lettre sans accent

    - notations équivalentes ou non de lettres (exemple : « œ » et « oe »)

    dépend du SGBD et du système d'exploitation

    exemple : `create schema coop default collate utf8_general_ci`
    (MySQL, interclassement sans différence de casse, lettres avec ou sans accent identiques)

SQL : DÉFINITION DE TABLE ET DE COLONNE

- TABLE

syntaxe générale [1] de l'instruction de définition d'une table

`create table` *nom* `(`*liste des définitions de colonne et de contraintes de table*`)`

au sein de la liste, les définitions sont séparées par une virgule (« , »)

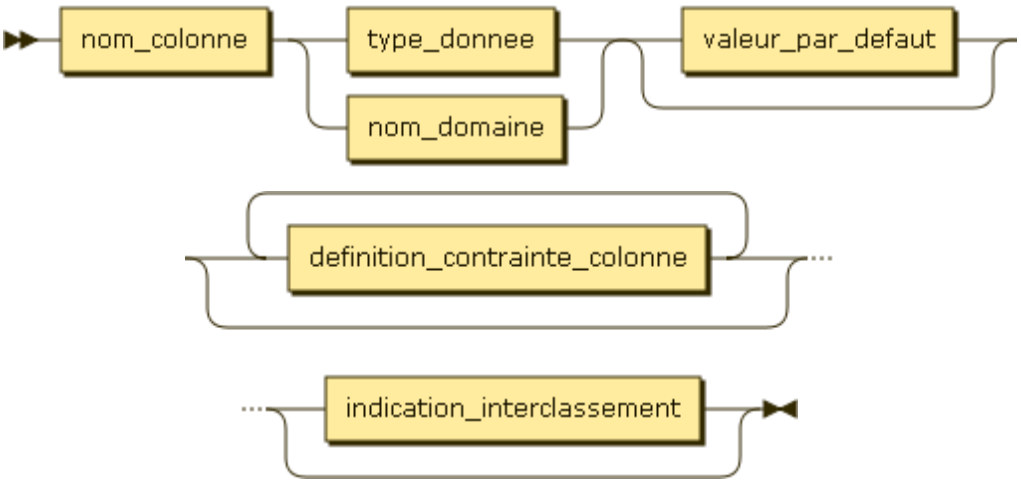
une ou plusieurs contraintes optionnelles sur la table (voir page 78)

notée tel que : *contrainte* ou nommée [2] via : `constraint` *nom contrainte*

exemple

```
create table personne (
 numero integer not null,
 nom varchar(40) not null,
 adresse varchar(60) not null,
 code_postal varchar(5) not null,
 ville varchar(40) not null,
 telephone varchar(14),
 primary key (numero)
)
```

- COLONNE



colonne définie au minimum par un nom et un type

types de donnée existants : textuels, numériques, temporels et logiques

possibilité d'indiquer une valeur par défaut automatiquement choisie en cas d'absence lors de la création de la ligne : `default` *valeur* [3]

une ou plusieurs contraintes optionnelles sur la colonne (voir page 78)

notées avec : `constraint` *nom contrainte* ou sans nom [1] via : *contrainte*

indication optionnelle d'un interclassement : `collate` *interclassement*

---

[1] Il existe aussi la possibilité de définir une table temporaire, non présentée ici.

[2] L'affectation d'un nom à une contrainte permet la suppression de la contrainte (page 83) et elle facilite aussi la compréhension d'un message du SGBD en cas d'anomalie

[3] La valeur par défaut doit être une constante ou une identification de l'usager (page 97) ou une fonction temporelle (page 94)

SQL : TYPES NUMÉRIQUES ET LOGIQUES

- TYPES NUMÉRIQUES
  - nombre entier : `integer`
    synonyme : `int`
    possibilité de taille réduite [1] : `smallint` ou augmentée : `bigint` [SQL 2003]
    possibilité de fixation automatique du numéro dépendant du SGBD [2] :
    via un type ou une propriété spécifique ou un générateur de numéros ;
    propriété introduite dans [SQL 2003] : `generated always as identity`
  - nombre réel exact : `decimal(`*précision*`, `*décimales*`)` ou `decimal`
    avec indication du nombre maximal de chiffres (*précision*) dont ceux situés dans la partie décimale (*décimales*)
    cas de calculs financiers avec une précision au chiffre près
    synonymes : `dec` et `numeric`
    précision et décimales fixés automatiquement si absents [1]
    exemple : `decimal (7,2)` pour une valeur entre `-99999,99` et `99999,99`
  - nombre réel approximatif : `float(`*précision*`)` ou `float` ou `real`
    avec indication de la précision maximale (*précision*) exprimée en nombre de chiffres (binaires selon la norme SQL)
    précision fixée automatiquement si absente et pour `real` [1]
    cas de calculs scientifiques sans nécessité d'exactitude [3]
    possibilité d'une meilleure représentation [1] : `double precision`

- TYPES BINAIRES ET LOGIQUES
  - valeur logique : `boolean`
    avec les valeurs possibles : `true` ou `false` ou `unknown` (inconnu)
  - suite de bits à taille fixe ou variable [SQL 1999] : `bit(`*taille*`)` ou `bit varying(`*taille*`)` ; taille maximale selon SGBD, fixée à 1 si absente
    valeur notée soit en binaire entre apostrophes et précédée de « `B` », soit en hexadécimal entre apostrophes et précédée de « `H` »
    exemples : `B'110011'  X'FA'`
    types binaires `bit` et `bit varying` abandonnés dans [SQL 2003]
  - suite de bits de très grande taille [SQL 1999] : `binary large object`
    taille largement au-delà de la taille maximale de `bit varying`, avec un stockage particulier
    synonyme : `blob`

[1] L'intervalle des valeurs représentées dépend du SGBD

[2] Numéro automatique dans PostgreSQL : `serial` et `sequence`, MySQL : `auto_increment`, Oracle : `sequence`, DB2 : `sequence` et `gen_id`, SQL server: `identity`, SQLite : implicite pour un entier ; généralement il suffit d'utiliser le mot-clef `default` quand il est nécessaire d'indiquer la valeur à l'insertion

[3] Par rapport au type `decimal`, le type `float` ne garantit pas l'exactitude des calculs au chiffre près mais ceux-ci sont effectués plus rapidement (représentation avec mantisse et exposant).

SQL : TYPES TEXTUELS ET TEMPORELS

● TYPES TEXTUELS

- texte à taille fixe : `char(`*taille*`)` avec exactement *taille* caractères [1]
  taille fixée à 1 si absente ; maximum dépendant du SGBD
  remplissage partiel possible mais stockage selon la taille fixée
  synonyme : `character`
- texte à taille variable : `varchar(`*taille*`)` avec au plus *taille* caractères [3]
  taille fixée à 1 si absente ; maximum dépendant du SGBD
  synonymes : `character varying` et `char varying`
- texte de très grande taille [SQL 1999] : `character large object`
  taille largement au-delà des tailles maximale de `char` et `varchar`, avec un stockage particulier mais avec un jeu de manipulations réduites (notamment pas de tri, de contrainte d'unicité et de jointure possibles)
  synonymes : `char large object` et `clob`

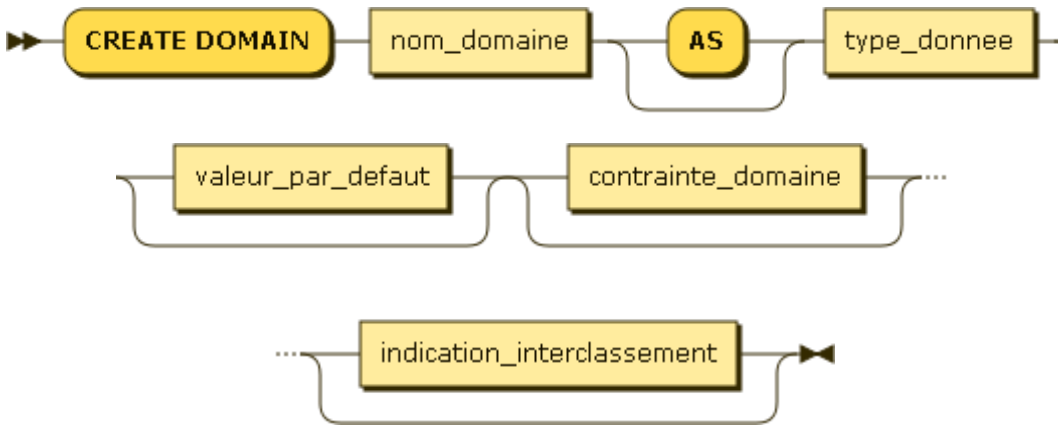
● TYPES TEMPORELS

- date (du calendrier grégorien) : `date`
- heure : `time` ou `time(`*précision*`)`
- horodate : `timestamp` ou `timestamp(`*précision*`)`
  précision en nombre de chiffres décimaux pour les fractions de seconde
  possibilité d'ajout du fuseau horaire : `with time zone` indiqué alors dans une valeur par : `local` ou +*heure*`:`*minute* ou −*heure*`:`*minute* ou `timezone` *durée*
  exemples : `time(3) with time zone`
  `timestamp with time zone`
- durée : `interval` *unité* `ou interval` *unité1* `to` *unité2*
  unités : `year`, `month`, `day`, `hour`, `minute` ou `seconde`
  précision optionnelle pour les fractions de seconde
  exemples : `interval minute`
  `interval hour to minute`

[1] Le nombre d'octets pour la représentation dépend de l'aphabet utilisé ; exemple : de 1 à 6 octets en UTF8

SQL : DOMAINES ET TYPES COMPLEXES

● DOMAINE PERSONNALISÉ



– définition d'un domaine personnalisé, avec attribution d'un nom :
`create` *nom* `domain as` *type options*

options (dans l'ordre) : *valeur par défaut, contrainte de domaine, interclassement*

contrainte de domaine : contrainte essentiellement de validation (page 78)

exemple : `create domain type_monetaire as decimal(7,2)`

● TYPES COMPLEXES [SQL 1999, présentation simplifiée]

– regroupement : `row(`*liste de définition de sous-élément* [1]`)`

définition de sous-élément (attribut avec un type) : *nom type* avec en option l'indication d'un interclassement

utilisation d'un sous-élément via l'élément global : *élément*`.`*sous-élément*

exemple : `velo row( marque varchar(40), modele varchar(30) )`
avec : `velo.marque` et `velo.modele`

– tableau : *type* `array(`*taille*`)` avec *taille* éléments

la taille maximale d'un tableau dépend du SGBD

élément accessible via son rang compté à partir de 1 : *tableau*`[`*rang*`]`

deux tableaux sont identiques si même taille et éléments identiques

opération de concaténation de deux tableaux disponible : t_1 `||` t_2

exemple : `telephones varchar[15] array(3)`
avec : `telephones[1]`, `telephones[2]` et `telephones[3]`

[1] La liste peut contenir un seul nom, ou plusieurs séparés alors par une virgule (« , »)

SQL : CONTRAINTE DE COLONNE ET DE TABLE

- CONTRAINTE AU NIVEAU D'UNE COLONNE

  - Contrainte de valeur obligatoire

    valeur obligatoire pour la colonne, notée : `not null`

  - Contrainte de valeur unique

    soit une colonne formant la clef primaire, notée : `primary key`
    soit une colonne en clef secondaire, notée : `unique`

  - Contrainte d'intégrité référentielle

    indication de la clef primaire associée à la clef étrangère (voir page 79)
    exemple : `references produit (numero)`

  - Contrainte de validation

    indication d'une condition devant être vérifiée : `check (`*condition*`)`
    condition validée après chaque modification du contenu de la colonne
    exemple : `check (prix >= 0)`

- CONTRAINTE AU NIVEAU D'UNE TABLE

  - Contrainte de valeur unique

    indication de la clef primaire : `primary key (`*liste de colonne* [1]`)`
    obligatoire si cela n'a pas déjà été indiqué au niveau de sa colonne ou si la clef primaire est constituée de plusieurs colonnes
    indication de(s) clef secondaire(s) : `unique (`*liste de colonne* [1]`)`
    si pas déjà indiqué au niveau de la (des) colonne(s)
    exemple : `primary key (matricule, enseignant)`

  - Contrainte d'intégrité référentielle

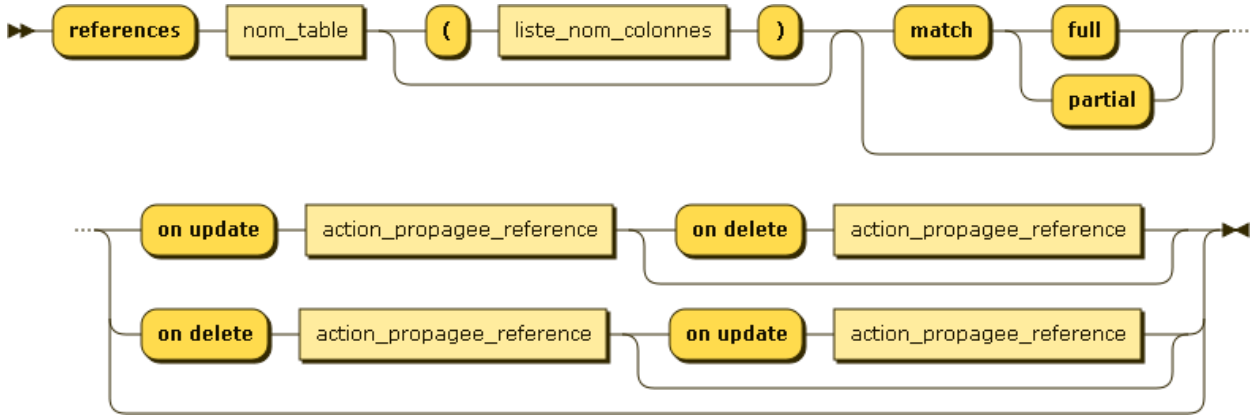
    indication de la clef primaire associée à la clef étrangère :
    `foreign key(`*clef_étrangère*`) references` *table*`(`*clef_primaire*`)` *options*
    si pas déjà indiqué au niveau de la (des) colonne(s) (voir page 79)
    exemple : `foreign key(ref_produit) references produit (numero)`

  - Contrainte de validation

    indication d'une condition devant être vérifiée : `check (`*condition*`)`
    condition validée après chaque modification du contenu de la table
    exemple : `check (taille_restante <= taille_initiale)`

[1] La liste peut contenir un seul nom, ou plusieurs séparés alors par une virgule (« , »)

SQL : CONTRAINTE D'INTÉGRITÉ RÉFÉRENTIELLE



- Définition de la contrainte

  Indication de la table et de(s) colonne(s) de la clef primaire associée : `references` *table* `(`*liste de colonne* [1]`)`

  ou si clef primaire identique à la clef étrangère : `references` *table*

  exemple : `ref_produit references produit(numero)`

- Mode de correspondance (`match`) si clef primaire sur plusieurs colonnes

  lors d'une nouvelle valeur ou d'une modification de la clef, option précisant les conditions du déclenchement du contrôle lorsque il y a une ou plusieurs valeurs manquantes parmi les colonnes de la clef étrangère :

mode	**condition de déclenchement**	**colonnes vérifiées**
absent	toutes les colonnes ont une valeur	toutes
`match full`	au-moins une colonne avec une valeur	toutes
`match partial`	au-moins une colonne avec une valeur	celles à valeur définie

- Conséquences en cas de mise à jour ou suppression

  lors d'une modification et-ou suppression de la ligne identifiée par la clef primaire, action à appliquer : `on update` *action* et-ou `on delete` *action*

action	**conséquences**
`no action`	opération refusée (contrôle en fin de transaction, voir page 104)
`restrict`	opération refusée immédiatement
`set default`	valeur par défaut mise dans chaque colonne de la clef étrangère des lignes liées à la clef primaire
`set null`	valeur devenant indéfinie pour chaque colonne de la clef étrangère des lignes liées à la clef primaire
`cascade`	opération répercutée sur les lignes liées à la clef primaire, sous la forme de leur mise à jour ou suppression

  exemple :
```
ref_producteur references personne(numero)
  on delete no action on update cascade
```

[1] La liste peut contenir un seul nom, ou plusieurs séparés alors par une virgule (« `,` »)

SQL : VUE

MÉCANISME RELATIF AU NIVEAU EXTERNE DU MODÈLE ANSI SPARC [1]

● PRINCIPE DE LA VUE

présentation des données pour un point de vue particulier sur la base, sous la forme d'une table virtuelle aux manipulations limitées

exemple : vue des fournisseurs de la coopérative

table virtuelle définie sous la forme d'une interrogation générale et réutilisable dans une interrogation comme une véritable table

modifications (ajout, mise à jour et suppression) via une vue possibles mais sous certaines conditions intrinsèques (faisabilité de l'opération) ou spécifiquement définies (validation optionnelle de cohérence)

● CRÉATION D'UNE VUE

syntaxe : `create view` *nom* `(`*liste de colonne*`) as` *interrogation option*
ou `create view` *nom* `as` *interrogation option*

si absence de la liste des colonnes, tous les résultats de l'interrogation

option associée à la validation optionnelle d'un ajout ou d'une mise à jour, qui doivent produire une ligne pouvant être obtenue via l'interrogation :

- `with local check option` : vérification limitée à la vue
- `with cascaded check option` ou `with check option` : vérification élargie à toutes les réutilisations de cette vue

exemple :
```
create view fournisseurs as
select distinct personne.*
from personne
join lot on lot.ref_producteur = personne.numero
```

● SUPPRESSION D'UNE VUE

syntaxe : `drop view` *nom action*

action définissant les conséquences sur les références externes :

- `cascade` : suppressions de toutes les réutilisations de la vue (par d'autres vues ou des contraintes)
- `restrict` : opération refusée si au-moins une réutilisation de la vue

exemple : `drop view fournisseurs restrict`

[1] Voir la présentation page 6

SQL : GESTION DES DROITS D'ACCÈS

● PRINCIPE

toute manipulation de la base est contrôlée par le SGBD sur la base d'une identification des utilisateurs

identification basée sur un nom d'utilisateur avec mot de passe et droits [1] associés, et un mécanisme de session de travail géré par le SGBD

droits élémentaires existants sur une table ou une vue :

- `select` : interrogation,
- `delete` : suppression de lignes,
- `insert` : ajout de lignes (toutes les colonnes ou seulement certaines),
- `update` : mise à jour de colonnes (toutes ou seulement certaines)
- `references` : utilisation d'une clef pour créer une clef étrangère dans une table liée (pour toutes les colonnes ou seulement certaines),

droit d'utilisation d'un domaine : `grant`

droits d'administration [2] : création/suppression d'utilisateur, de bases, ...

● ATTRIBUTION D'UN DROIT ÉLÉMENTAIRE

syntaxe : `grant` *liste de droits* `on` *objet* `to` *liste d'utilisateurs* *option*

objet : soit une table par la notation `table` *nom* ou simplement *nom*, soit un domaine par `domain` *nom* [autres cas non présentés ici]

liste de droits : un ou plusieurs droits séparés par une virgule (« , ») ou tous les droits par la notation `all privileges`

liste d'utilisateurs : un ou plusieurs utilisateurs séparés par une virgule (« , ») ou tous par la notation `public`

option : possibilité de permettre à l'utilisateur de retransmettre ses droits, notée par `with grant option`

exemple : `grant select, update(nom) on table produit to tarzan`

● SUPPRESSION D'UN DROIT ÉLÉMENTAIRE

syntaxe : `revoke` *option* *liste de droits* `on` *objet* `to` *liste d'utilisateurs* *mode*

option : suppression de la simple possibilité donnée à l'utilisateur de retransmettre les droits, notée par `grant option for`

mode : si des droits ont été transmis à d'autres utilisateurs, soit mode à `restrict` et dans ce cas rejet de l'opération, soit mode à `cascade` et dans ce cas annulation de tous les droits retransmis

exemple : `revoke select, update(nom) on table produit to tarzan`

---

[1] En anglais : *privilege*

[2] Les droits d'administration dépendent du SGBD

SQL : OPÉRATIONS SUR LA STRUCTURE DE LA BASE

- RETOUCHE DE LA STRUCTURE D'UNE TABLE
  - Forme générale : `alter table` *nom retouche*
    retouche portant sur une colonne ou sur une contrainte de la table,
    si suppression conséquence sur les objets liés (clefs étrangères, etc.)
    d'où *action* indiquant alors soit la suppression des objets liés (`cascade`),
    soit le rejet de l'opération en cas d'existence d'objets liés (`restrict`)
  - Retouche par ajout d'une colonne
    syntaxe : `add column` *nom définition* ou `add` *nom définition*
    exemple : `alter table personne  add column courriel varchar(50)`
  - Retouche sur la valeur par défaut d'une colonne
    modification : `alter column` *nom* `set default` *valeur* ou
    `alter` *nom* `set default` *valeur*
    suppression : `alter column` *nom* `drop default` ou
    `alter` *nom* `drop default`
    exemple : `alter table personne  alter courriel set default '?'`
  - Retouche par suppression d'une colonne
    supression en cascade ou non des objets liés à cette colonne
    syntaxe : `drop column` *nom colonne action* ou `drop` *nom colonne action*
    exemple : `alter table produit  drop prix restrict`
  - Retouche par ajout d'une contrainte sur la table
    syntaxe : `add` *contrainte*
    exemple : `alter table produit`
    `add constraint c_prix check (prix >= 0)`
  - Retouche par suppression d'une contrainte sur la table
    syntaxe : `drop constraint` *nom_contrainte action*
    exemple : `alter table produit  drop constraint c_prix cascade`

- AUTRES OPÉRATIONS
  - Vidage d'une table par suppression de toutes ses lignes [SQL 2008]
    syntaxe : `truncate table` *nom action*
    exemple : `truncate table emballage restrict`
  - Suppression d'une table
    syntaxe : `drop table` *nom action*
    exemple : `drop table emballage restrict`
  - Suppression d'un schéma de la base
    syntaxe : `drop schema` *nom action*
    exemple : `drop schema cultures restrict`

SQL : EXERCICES SUR LA DÉFINITION DE DONNÉES

1. Dans le cadre du système d'information de la coopérative (page 19), écrire les instructions en SQL pour la définition des tables de sa base

2. Compléter la définition précédente de la table `personne` pour prendre en compte :

   a) le parrainage (page 24)

   b) la fiche descriptive de produit, unique où avec des versions (page 26)

3. Définir pour la coopérative les vues suivantes, en indiquant si la mise à jour est possible via la vue :

   a) liste des acheteurs classés par département

   b) liste des produits avec leur stock actuel et leur prix moyen d'achat

SQL : MANIPULATIONS DES DONNÉES

● AJOUT DE LIGNES

syntaxe générale de l'instruction `insert`

`insert into` *table* (*liste de colonnes*) `values` (*liste de valeurs*)

valeur par défaut d'une colonne possible parmi les valeurs : `default`

plusieurs listes de valeurs possibles, séparées par une virgule (« , »)

forme alternative avec les valeurs issues d'une recherche

`insert into` *table* (*liste de colonnes*) *interrogation élémentaire*

formes alternatives sans liste des colonnes au cas où toutes utilisées

exemples

```
insert into produit (numero, nom, type, prix)
 values (1, 'CAROTTE', 'LEGUME', 1.2)
insert into produit values
 (1, 'CAROTTE', 'LEGUME', default), (2, 'KIWI', 'FRUIT', 3)
insert into produit
 select numero_affecte, nom, type, prix
 from nouveaux_produits where validation = 'fait'
```

● MISE À JOUR DE LIGNES

syntaxe générale de l'instruction `update`

`update` *table* `set` *liste des mises à jour* `where` *condition*

mises à jour séparées par une virgule et notées : *colonne* `=` *valeur*

forme alternative sans condition (toutes les lignes modifiées)

`update` *table* `set` *liste des mises à jour*

possibilité de valeur issue d'une sous-recherche avec prise en compte d'une colonne de la table mise à jour, et ce à chacune de ses lignes

exemples

```
update produit set type = 'FRUIT', prix = '4'
 where nom = 'KIWI'
update client set compte = compte + 10
update table1 set col1 =
 (select col2a from table2 where col1 = col2b)
```

● SUPPRESSION DE LIGNES

syntaxe générale de l'instruction `delete`

`delete from` *table* `where` *condition*

forme alternative sans condition (toutes les lignes supprimées)

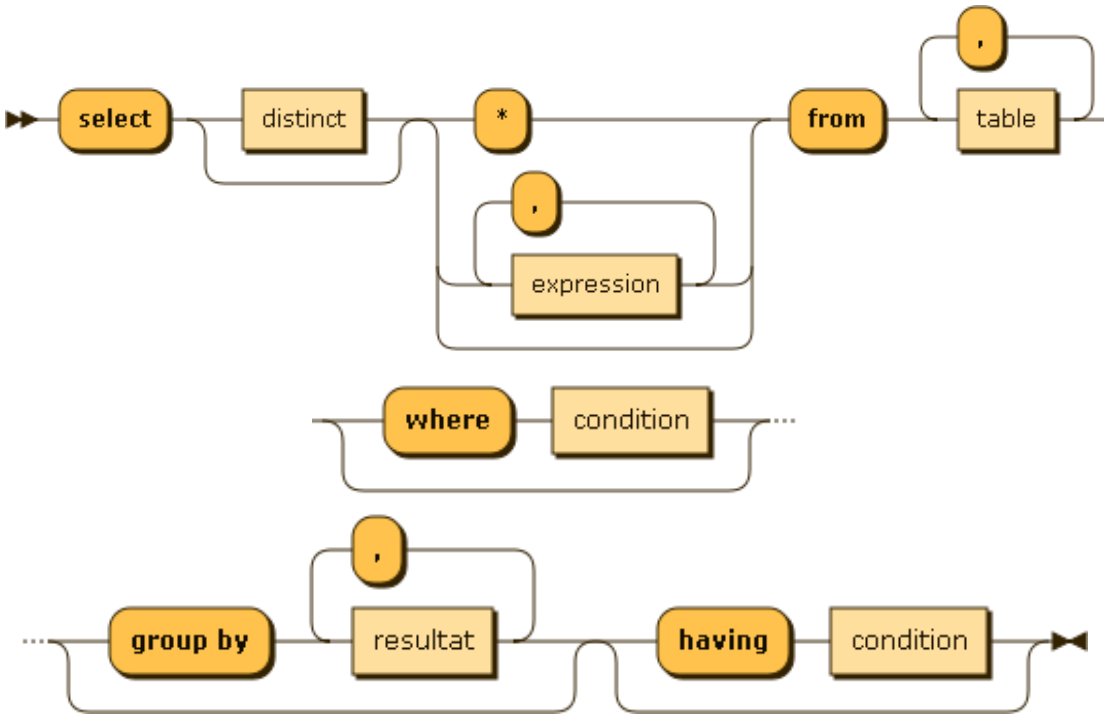
`delete from` *table*

exemples

```
delete from produit where nom = 'KIWI'
delete from nouveaux_produits
```

SQL : INTERROGATION ÉLÉMENTAIRE

- FORME GÉNÉRALE



syntaxe générale de l'instruction `select` (forme dite « SFW ») :

`select` *liste de résultats*
`from` *liste de tables*
`where` *restriction*
compléments

ou sans la restriction :

`select` *liste de résultats*
`from` *liste de tables*
compléments

compléments optionnels, dans l'ordre :

- regroupement d'agrégation (voir page 87)
- filtrage après regroupement (voir page 87)

exemples

```
select *
 from produit
 where type = 'FRUIT' and prix >= 5
select nom, (prix * 1.196) as ttc
 from produit
select ref_producteur, count(numero) as nb
 from lot
 where taille_restante = 0
 group by ref_producteur
 having nb >= 10
```

SQL : INTERROGATION ÉLÉMENTAIRE (SUITE)

● NOTATION DES RÉSULTATS

un résultat est soit une colonne, soit une expression
notations séparées dans la liste par une virgule (« , »)
notation particulière pour toutes les colonnes issues de(s) table(s) : *
élimination des doublons par le mot-clef `distinct` juste après `select`
nom de colonne éventuellement préfixé par nom de table suivi de « . »
possibilité d'attribuer un surnom à une expression ou à une colonne (réutilisable en regroupement et filtrage mais pas dans la restriction)
exemples :

```
type, nom
distinct prix
produit.nom, (vente.prix * 1.196) as ttc
```

● NOTATION DES TABLES

une table est soit :

- une véritable table désignée par son nom, éventuellement préfixée par le nom de son schéma
- l'expression d'une jointure de tables (voir page 88) ,
- une table dite « dérivée » résultat d'une sous-interrogation élémentaire (dont le résultat est une table), identifiée explicitement par un surnom : `(`*sous-interrogation*`) as` *surnom* ou `(`*sous-interrogation*`)` *surnom*

exemples :

```
produit
personne, lot
(select * from produit where type = 'FRUIT') as t
lot join produit on lot.ref_produit = produit.numero
```

● SURNOM (« ALIAS »)

possibilité d'attribuer un nom à une expression, ou définir un surnom à une colonne ou à une table : *élément* `as` *nom* ou *élément nom*
la définition du surnom annule localement la validité du nom original
pour une table, possibilité complémentaire de renommer ses colonnes : *table* `as` *surnom* `(`*liste des surnoms des colonnes*`)`
exemples :

```
produit.nom as nom_produit
avg(prix) moyenne
personne as p (num, nom, ad, cp, v, tel)
```

SQL : AGRÉGATION

CALCULS SUR TOUTES LES LIGNES OU PAR REGROUPEMENT

Résultats correspondant à l'appplication d'une fonction de calcul sur les lignes renvoyées par l'interrogation

`count(*)` (comptage de toutes les lignes)

`count(`*colonne*`)` (comptage des lignes où la valeur de la colonne est définie)

`avg(`*colonne*`)` (moyenne, *average*)

`sum(`*colonne*`)` (somme)

`min(`*colonne*`)` ou `max(`*colonne*`)` (minimum ou maximum)

attention ! la parenthèse ouvrante doit être collée au nom de fonction

à placer seulement dans la liste après `select` mais pas après `where`

si aucune ligne, résultat indéfini (`null`) pour `avg, sum, min et max`

Elimination des doublons avant les calculs

ajout du mot `distinct` avant le nom de la colonne entre les parenthèses

Regroupement des résultats avant les calculs

`group by` *colonne*$_1$, *colonne*$_2$, ... (à placer juste après la restriction `where`)

Filtrage des résultats après les calculs

`having` *condition* (à placer juste après le regroupement `group by`)

Exemples

```
select count(*) from produit
```
le nombre total de produits enregistrés

```
select count(distinct ville) from personne
```
le nombre total de villes de résidence des personnes

```
select type, count(nom) from produit
 group by type
```
les types et le nombre total de produit (avec un nom défini) pour chaque type

```
select avg(prix) from produit where type = 'LEGUME'
```
le prix moyen des légumes

```
select type, max(prix) from produit
 group by type
```
le prix maximal pour chaque type de produit

```
select type, min(prix) as min, max(prix) as max from produit
 group by type
```
minimum et maximum des prix pour chaque type de produit

```
select ref_produit, sum(quantite) as nb_total from vente
 group by ref_produit
 having nb_total > 100
```
les références des produits vendus en quantité totale de plus de 100

SQL : JOINTURE

COLONNES SUPPLÉMENTAIRES ISSUES D'UNE AUTRE TABLE

● JOINTURE INTERNE

syntaxe : *table₁* `join` *table₂* `on` *condition* ou
table₁ `inner join` *table₂* `on` *condition*

chaque ligne de la 1ère table est complétée par toute ligne de la 2nde table vérifiant la condition [1]

équivalent à la notation en SQL 1 avec la condition en restriction :
`select` *...* `from` *table₁*, *table₂* `where` *condition*

exemple :
```
select lot.numero, produit.nom, taille_restante
 from lot join produit on ref_produit = produit.numero
```

équivalent à :
```
select lot.numero, produit.nom, taille_restante
 from lot, produit
 where ref_produit = produit.numero
```

auto-jointure [2] d'une table sur elle-même possible à l'aide de surnom
exemple d'auto-jointure :
```
select etu.nom as stagiaire, ens.nom as encadrant
 from personne as ens
 join personne as etu on etu.tuteur = ens.numero
```

● JOINTURE EXTERNE

syntaxe : *table₁* *côté* `outer join` *table₂* `on` *condition* ou
table₁ *côté* `join` *table₂* `on` *condition*

jointure interne complétée selon *côté* par :

- `left` (à gauche) : toute ligne de la 1ère table sans concordance avec la 2nde table, complétée par des manques (`null`)
- `right` (à droite) : toute ligne de la 2nde table sans concordance avec la 1ère table, complétée par des manques (`null`)
- `full` (bilatérale) : toute ligne de la 1ère table sans concordance avec la 2nde table et toute ligne de la 2nde table sans concordance avec la 1ère table, complétées par des manques (`null`)

exemple :
```
select lot.numero, produit.nom, taille_restante
 from lot left join produit on ref_produit = produit.numero
```

[1] Une jointure est appleée « équi-jointure » quand la condition est une égalité, « inéqui-jointure » sinon
[2] L'auto-jointure correspond généralement à une association cyclique (voir page 24)

SQL : JOINTURE (SUITE)

CAS PARTICULIERS PEU UTILISÉS

- JOINTURE NATURELLE

  syntaxe : *table₁* `natural join` *table₂*

  jointure interne où la condition correspond à l'égalité des colonnes homonymes et de type identique de chaque table, avec reprise dans la table jointe d'une seule de ces colonnes homonymes

  forme déconseillée car les noms de colonnes ne sont pas explicités et l'instruction est donc de fait insensible à un nouveau schéma de table, ce qui est source d'erreurs difficiles à diagnostiquer

  exemple :

  ```
  select nom, solde from client natural join compte
  ```

- JOINTURE CROISÉE

  syntaxe : *table₁* `cross join` *table₂*

  chaque ligne de la 1ère table est complétée par toute ligne de la 2nde table (c'est le « produit cartésien » du modèle relationnel)

  exemple :

  ```
  select * from r cross join s
  ```

  équivalent à :

  ```
  select * from r, s
  ```

- JOINTURE D'UNION

  syntaxe : *table₁* `union join` *table₂*

  reprise sans correspondance de chaque ligne de la 1ère table et de la 2nde table, complétées par des manques (`null`)

  exemple :

  ```
  select * from r union join s
  ```

SQL : INTERROGATION GÉNÉRALISÉE

FORME PRINCIPALE NON RÉUTILISABLE DANS UNE SOUS-INTERROGATION

● CARACTÉRISTIQUES GÉNÉRALES

extraction de données à partir de tables et-ou calculs sous la forme d'une seule interrogation élémentaire ou d'une combinaison d'interrogations élémentaires (voir page 91)

résultat soit sous la forme d'une table, soit indéfini (`null`)

table en résultat éventuellement réduite à une unique valeur, dite « scalaire », sur une ligne à une seule colonne

possibilité d'indiquer un tri des lignes à la fin de l'interrogation

exemples

```
select personne.nom as producteur, count(lot.numero) as nb
from personne
join lot on personne.numero = lot.ref_producteur
group by lot.ref_producteur
order by personne.nom
```

```
(select nom, prix from produit
 where type = 'FRUIT' and prix >= 5)
union
(select nom, prix from produit
 where type = 'LEGUME' and prix >= 2)
order by nom
```

● TRI DES LIGNES

tri selon un ou plusieurs critères (tris successifs) : `order by` *liste de critères*
critères séparés dans la liste par une virgule (« , »)

un critère se note par un nom de résultat suivi du sens de tri indiqué explicitement par `asc` (croissant) ou `desc` (décroissant), ou simplement par un nom de résultat (sens croissant implicite)

attention ! le tri ne peut pas s'utiliser dans une sous-interrogation

exemples :

```
select nom from produit order by type, nom
select * from produit order by prix desc
```

SQL : COMBINAISONS D'INTERROGATIONS

● CARACTÉRISTIQUES GÉNÉRALES

syntaxe générale : *i_1 combinaison options i_2*

combinaison : `union` ou `intersect` ou `except` (différence)

i_1 et *i_2* constitués d'une interrogation élémentaire ou d'une table dérivée d'une sous-interrogation

i_1 et *i_2* doivent produire des résultats sur des colonnes en même nombre et de même type respectif

nom des résultats correspondant aux noms des colonnes de *i_1*

conservation optionnelle des doublons via l'option `all`

possibilité de restreindre le résultat à certaines colonnes via une seconde option (placée après `all` si présent) :

- soit les colonnes homonymes : `all corresponding`
- soit certaines colonnes : `all corresponding` *col_1, col_2* ...

● UNION

résultats de la seconde interrogation ajoutés à ceux de la première

exemple :

```
  select p.nom
  from personne as p join lot on lot.producteur = p.numero
union
  select p.nom
  from personne as p  join vente on vente.acheteur = p.numero
```

● INTERSECTION

résultats communs aux deux interrogations

exemple :

```
  select p.nom
  from personne as p join lot on lot.producteur = p.numero
intersect
  select p.nom
  from personne as p  join vente on vente.acheteur = p.numero
```

● DIFFÉRENCE

résultats de la première interrogation absents dans la seconde

exemple :

```
  select p.nom
  from personne as p join lot on lot.producteur = p.numero
except
  select p.nom
  from personne as p  join vente on vente.acheteur = p.numero
```

OPÉRATIONS ENSEMBLISTES PAS TOUJOURS DISPONIBLES

SQL : EXPRESSION

● FORME GÉNÉRALE

différents types d'expression selon le type du résultat :
logique, numérique, textuelle ou temporelle

conversion entre deux valeurs de types différents (« transtypage » [1]) :

- automatique : lors de l'évaluation d'une expression avec les valeurs, a priori seulement dans le cas de valeurs numériques mais existence de nombreux cas particuliers selon les SGBD
- explicite : via la fonction `cast(`*expression* `as` *type*`)` (voir page 96)

une valeur indéfinie (`null`) dans une expression entraîne un résultat indéfini (sauf pour le prédicat `is` ... ou `case`, `coalesce` et `nullif`)

● EXPRESSION NUMÉRIQUE

expression à valeur numérique

composants par ordre décroissant de priorité :

– un élément de base à valeur numérique (voir page 96)
– une fonction à résultat numérique (voir page 98)
– la négation : `-`
– la multiplication ou la division : `* /`
– l'addition ou la soustraction : `+ -`

exemples

```
prix * 1.196
extract(year from current_date)  - annee_naissance
```

● EXPRESSION TEXTUELLE

expression correspondant à un texte (une chaîne de caractères)

composants par ordre décroissant de priorité :

– un élément de base à valeur textuelle (voir page 96)
– une fonction à résultat textuel (voir page 98)
– l'application d'un interclassement : `collate` *interclassement*
– la concaténation de deux textes : *texte*$_1$ `||` *texte*$_2$

exemples

```
nom || ' ' || prenom
substring( trim(code_postal) from 1 for 2 )
```

[1] En anglais : *coertion*, *type conversion*, *typecasting*

SQL : EXPRESSION (SUITE)

● EXPRESSION LOGIQUE

expression à valeur logique, correspondant à une « condition »
logique à trois états : vrai (`true`), faux (`false`) ou inconnu (`unknown`)
composants par ordre décroissant de priorité :

– un élément de base à valeur logique (voir page 96)
– un prédicat logique (voir page 94)
– une sous-condition : `(` *condition* `)`
– un test élémentaire : `is` *constante logique* ou `is not` *constante logique*
où *constante logique* correspond à : `true`, `false` ou `unknown`
avec comme résultat `true` ou `false` (mais pas `unknown`)
– une négation : `not` *sous-expression*
`not unknown` vaut `unknown`
– une conjonction (et) : *sous-expression*$_1$ `and` *sous-expression*$_2$
– une disjonction (ou) : *sous-expression*$_1$ `or` *sous-expression*$_2$

exemples

```
(type = 'LEGUME') and (prix > 5)
(produit.prix > client.compte) is not unknown
```

● EXPRESSION TEMPORELLE

expression correspondant à une date, heure, horodate ou durée
composants par ordre décroissant de priorité :

– un élément de base à valeur temporelle (voir page 96)
– une fonction temporelle : `current_date` (date courante), `current_time` (heure courante) ou `current_timestamp` (horodate courante)
– un calcul sur une valeur temporelle :
augmentation de date/heure/horodate : *durée* + *valeur* ou *valeur* + *durée*
diminution de date/heure/horodate : *valeur* − *durée*
durée écoulée : *date*$_1$ − *date*$_2$ ou *heure*$_1$ − *heure*$_2$ ou *horodate*$_1$ − *horodate*$_2$
calcul de durée : *durée*$_1$ + *durée*$_2$ ou *durée*$_1$ − *durée*$_2$ ou *durée* * *nombre*
ou *nombre* * *durée* ou *durée* / *nombre*

exemples

```
current_date + interval '3' month
horodate_debut - horodate_fin
```

SQL : PRÉDICAT LOGIQUE

- Comparaison entre deux valeurs par : `=, <>` (différence), `>, <, >=, <=`
  s'applique aussi aux dates, heures et textes (ordre lexicographique)
  si une valeur est indéfinie (`null`), le résultat est inconnu (`unknown`)
  exemples : `type = 'FRUIT'  compte <> 0  an <= date '2001-01-01'`

- Valeur dans un intervalle : *valeur* `between` *valeur*$_1$ `and` *valeur*$_2$
  ou non : *valeur* `not between` *valeur1* `and` *valeur2*
  exemple : `compte between 500 and 1000`

- Dans une liste de valeurs : *valeur* `in` liste ou non : *valeur* `not in` *liste*
  exemple : `cat in ('A01, 'A02', 'A03')`

- Valeur définie : *valeur* `is not null` ou indéfinie : *valeur* `is null`
  exemple : `cat is not null`

- Existence des résultats d'une sous-recherche : `exists` *sous-recherche*
  exemple : `exists (select prix from produit where nom = 'radis')`

- Unicité du résultat d'une sous-recherche : `unique` *sous-recherche*
  exemple : `unique (select prix from produit where nom = 'radis')`

- Recouvrement de périodes temporelles : *période*$_1$ `overlaps` *période*$_2$
  période : `(`*debut*`,` *fin*`)` ou `(`*début*`,` *durée*`)` où début est une date ou une heure
  exemple : `(time '08:00:00', time '10:00:01') overlaps (h1, h2)`

- Comparaison d'une valeur avec chaque élément d'un ensemble donné :
  valeur comparateur mode liste
  comparateur : `=, <>, >, <, >=, <=`
  mode : `all` (chaque comparaison élémentaire est vérifiée),
  `any` ou `some` (au-moins une comparaison élémentaire est vérifiée)
  liste : liste de valeurs, ou la colonne en résultat d'une sous-recherche
  exemple : `produit.prix < all`
  `(select lot.prix from lot where lot.produit = produit.numero)`

SQL : PRÉDICAT LOGIQUE (SUITE)

- Correspondance entre une liste de valeurs et ligne(s) de la table en résultat d'une sous-recherche : `(` *liste de valeurs* `) match` *options sous-recherche*
  options dans l'ordre : `unique` (1 seule correspondance) puis *mode*
  mode : `full` (valeurs toutes définies dans la liste et correspondance) ou `partial` (correspondance avec les valeurs définies dans la liste) , ou si absent (au-moins 1 valeur indéfinie dans la liste ou sinon correspondance)
  exemple :
  ```
  (numero, prix) match unique
   (select ref_produit, prix from lot where taille_restante > 0)
  ```

- Concordance avec un motif : *valeur* `like` *motif* ou non : *valeur* `not like` *motif*
  motif peut contenir les méta-symboles suivants :
  « `%` » pour désigner toute suite de symboles (éventuellement vide),
  « `_` » pour désigner un et un seul symbole quelconque
  si un de ces méta-symboles figure dans le texte du motif comme véritable symbole, annulation possible de son effet par préfixage avec un symbole déclaré à la fin de l'expression par : `escape` *symbole*
  exemples : `code like '75___'` `mel like '%!_%@%' escape '!'`

- Concordance avancée à l'aide d'une expression rationnelle [SQL 1999] :
  valeur `similar` *motif* ou non : *valeur* `not similar` *motif*
  motif peut contenir comme `like` les méta-symboles suivants :
  `%` : toute suite de symboles (éventuellement vide),
  `_` : un et un seul symbole quelconque
  et aussi des notations empruntées aux expressions rationnelles :
  `*` : aucune, une ou plusieurs répétitions du symbole précédent
  `+` : une ou plusieurs répétitions du symbole précédent
  `[`*symbole*$_1$`-`*symbole*$_2$`]` : un symbole parmi ceux de l'intervalle
  `[^`*symbole*$_1$`-`*symbole*$_2$`]` : un symbole autre que ceux de l'intervalle
  `[:`*nom*`:]` : un symbole du jeu de nom `digit` (chiffres), `alpha` (lettres), `upper` (majuscules), `lower` (majuscules) ou `alnum` (lettres et chiffres)
  `|` : alternative
  `(`*expression*`)` : sous-expression
  l'expression rationnelle doit correspondre complètement à la valeur donnée
  si un de ces méta-symboles figure dans le texte du motif comme véritable symbole, annulation possible de son effet par préfixage avec un symbole déclaré à la fin de l'expression par : `escape` *symbole*
  exemple : `libelle similar '(num|NUM)!_[:digit:]+' escape '!'`

SQL : ÉLÉMENT DE BASE D'EXPRESSION

- Une notation de valeur numérique, textuelle, temporelle, logique, binaire
  exemples : `1.77 'Carotte' date '2011-12-28' true B'111' X'20'`

- Le nom d'une colonne, éventuellement préfixé par le nom de table
  exemples : `produit.prix nom`

- La velur définie par défaut pour la colonne considérée : `default`
  souvent utilisé pour un numéro fixé automatiquement (voir page 75)
  exemple :

```
insert into produit(numero, nom, type, prix)
 values(default, 'NOIX', 'FRUIT', default)
```

- L'identification de l'utilisateur du SGDB : `user` ou `current_user` ou `session_user` (sémantiques exactes dépendant du SGBD et du contexte)

- Le nom d'une variable définie dans l'environnement d'exécution

- Un calcul d'agrégation (voir page 87) :
  `sum()` : somme ; `avg()` : moyenne (*average*)
  `min()` : minimum ; `max()` : maximum
  `count()` : comptage (prise en compte des lignes à valeur définie, pas `null`)
  `count(*)` : nombre total de lignes
  attention ! pas d'espace entre nom de la fonction et parenthèse ouvrante
  exemples :

```
select type, avg(prix)
 from produit group by type
select count(*) as nb_client, count(distinct cat) as nb_cat
 from client
```

- Une sous-interrogation dont le résultat est une valeur simple (« scalaire ») ou `null` en cas d'absence de résultat
  exemple : `( select max(prix) from produit )`

- Une sous-expression notée entre parenthèses : `(` expression `)`
  exemple : `(prix * 1.77)`

SQL : ÉLÉMENT DE BASE D'EXPRESSION (SUITE)

- Une conversion de type ou « transtypage » : `cast(`*expression* `as` *type*`)`
  nom de domaine utilisable à la place du type
  exemple : `cast('12:20' as time)`

- La première valeur définie dans une liste : `coalesce(`*liste d'expressions*`)`
  exemple : `coalesce(nom_jeune_fille, nom_famille, '?')`

- La valeur d'une expression, remplacée par `null` dans le cas où elle est égale à une valeur particulière : `nullif(`*expression*, *valeur*`)`
  exemple : `nullif(nom, '?')`

- Une valeur définie au cas par cas selon les valeurs possibles d'une expression donnée : `case` *expression liste de cas alternative* `end`
  liste est un ou plusieurs cas à la suite, chacun noté : `when` *valeur* `then` *résultat*
  alternative : optionnelle, s'applique si aucun cas n'est vérifié : `else` *valeur*
  si aucun cas vérifié et pas d'alternative, c'est alors la valeur `null`
  exemple :
```
case sexe
 when 'F' then 'femme'
 when 'H' then 'homme'
 else 'inconnu'
end
```

- Une valeur définie au cas par cas selon des conditions au sein d'une interrogation : `case` *liste de cas alternative* `end`
  liste : plusieurs cas à la suite, chacun noté : `when` *condition* `then` *résultat*
  alternative : optionnelle, s'applique si aucun cas n'est vérifié : `else` *valeur*
  si aucun cas vérifié et pas d'alternative, c'est alors la valeur `null`
  exemple :
```
select code_postal,
  case
   when pays = 'FR' then 'France'
   when pays is null then 'Inconnu'
   else 'Etranger'
  end
 from personne
```

SQL : FONCTION

PRÉSENTATION MINIMALE

- Position d'une sous-chaîne : `position(`*sous-chaîne* `in` *chaîne*`)`
  renvoie le rang de la sous-chaîne si présente dans chaîne, 0 sinon
  renvoi 0 si la sous-chaîne est vide
  rang compté à partir de 1
  exemple : `position('@' in courriel)`

- Nombre de caractères dans une chaîne : `char_length(`*chaîne*`)` ou `character_length(`*chaîne*`)`
  renvoi le nombre de caractères selon l'alphabet défini pour la chaîne
  exemple : `char_length(nom)`

- Nombre d'octets dans une chaîne : `octet_length(`*chaîne*`)`
  renvoie le nombre d'octets occupés par les caractères de la chaîne
  exemple : `octet_length(nom)`

- Nombre de bits dans une chaîne : `bit_length(`*chaîne*`)`
  exemple : `bit_length(code)`

- Changement de casse par passage en majuscules : `upper(`*chaîne*`)`
  ou en minuscules : `lower(`*chaîne*`)`
  exemple : `upper(produit.type)`

- Extraction dans une valeur temporelle : `extract(`*champ* `from` *valeur temporelle*`)`
  champ est au choix :
  - `year` (l'année)
  - `month` (le mois)
  - `day` (le jour)
  - `hour` (l'heure)
  - `minute` (les minutes)
  - `second` (les secondes)
  - `timezone_hour` (décalage en heures du fuseau horaire)
  - `timezone_minute` (décalage en minutes du fuseau horaire)

  le résultat est une valeur numérique
  exemple : `extract(year from date_arrivee)`

SQL : FONCTION (SUITE)

- Extraction d'une partie d'une chaîne débutant à un rang donné
  soit jusqu'à la fin de la chaîne : `substring(`*chaîne* `from` *début*`)`,
  soit sur une certaine longueur : `substring(`*chaîne* `from` *début* `for` *longueur*`)`
  rang compté à partir de 1
  chaîne vide en résultat si le rang *début* est inférieur ou égal à 1
  exemple : `substring(code from 1 for 2)`

- Elimination des occurences d'un caractère situées à l'une ou aux deux extrémités d'une chaîne : `trim(`*mode caractère* `from` *chaîne*`)`
  mode est au choix :
  - `both` ou absent (les deux extrémités),
  - `leading` (au début seul)
  - `trailing` (à la fin seule)

  par défaut d'indication le caractère à éliminer est l'espace
  en absence d'indication de mode et caractère, mot-clef `from` optionnel
  exemple : `trim(both from courriel)` ou `trim(courriel)`

- Conversion du texte d'une chaîne dans un alphabet donné :
  `convert(`*chaîne* `using` *alphabet*`)`
  fonction utilisable uniquement dans la spécification d'un résultat d'une requête d'interrogation
  exemple : `select convert(nom using utf8) from personne`

- Remplacement ciblé de caractères selon un modèle de conversion :
  `translate(`*chaîne* `using` *modèle de conversion*`)`
  le modèle de conversion indique les caractères à remplacer et pour chacun son caractère de remplacement ; sa création n'est pas présentée ici.

- Valeur absolue d'une valeur numérique [SQL 1999] : `abs(`*valeur*`)`
  exemple : `abs(x - moyenne)`

- Modulo ou reste de la division entière [SQL 1999] : `mod(`*valeur, diviseur*`)`
  exemple : `mod(duree_minutes, 60)`

NOMBREUSES AUTRES FONCTIONS DISPONIBLES SELON LE SGBD UTILISÉ

SQL : EXERCICES SUR LES MANIPULATIONS

1. Dans le cadre du système d'information de la coopérative :
   a) Peut-on supprimer tous les choux en une seule requête ?

   b) Quelle est la requête pour doubler le prix des fruits ?

   c) Comment obtenir le nom en majuscule des acheteurs de panais ?

   d) Calculer le nombre et le poids total des livraisons pour les produits en stock, puis pour tous les produits

   e) Déterminer les fournisseurs qui ne sont pas aussi acheteurs, à l'aide d'une sous-interrogation ou sans

2. Est-ce que les formulations suivantes sont correctes :
   a) `select nom from personne where telephone not null`

   b) `select current_date`

   c)
```
select m.titre, m.pays, m.annee, etudiant.numero
from memoire m join etudiant as etu on m.auteur = etu.numero
where promotion between 2000 and 2012
order by m.pays, m.annee desc
```

   d)
```
select upper(nom), ville, cp, m from (
 select p.nom, p.code_postal cp, sum(taille_initiale * prix) m
 from personne p join lot on ref_producteur = p.numero
 where substring(code_postal, 1, 2)
  in (75, 77, 78, 91, 92, 93, 94, 95)
 order by cp
) having m > 1000
```

3. Quelle est la valeur de :
   a) `null is null`

   b) `null and null`

   c) `true is unknown`

4. Est-ce que `not predicat` est équivalent à `predicat is not true` ?

TRANSACTIONS : PRINCIPES

GESTION DE LA CONCURRENCE D'ACCÈS AUX DONNÉES

●PROBLÉMATIQUE

- Exécution simultanée de deux mises à jours d'une même colonne

  exemple théorique de requête correcte donnant un résultat faux :

```
update lot
set taille_restante =  taille_restante - 5
where numero = 72
```

  qui se décomposerait en 3 étapes :

  a) lecture de `taille_restante`
  b) calcul de `(taille_restante - 5)`
  c) mise à jour de `taille_restante`

  et qui serait exécutée simultanément par deux utilisateurs :

instant	taille_restante	utilisateur A	utilisateur B
t1	510	a) → 510	
t2	510		a) → 510
t3	510	b) → 505	
t4	505	c)	
t5	505		b) → 505
t6	***505***		c)

  entrelacement néfaste des étapes d'exécution des deux requêtes

● PRINCIPES

- Concept de transaction

  opération s'exécutant complètement, ou annulée en cas d'anomalie sans alors aucune modification effectuée prise en compte finalement

- Critères ACID

  atomicité : transaction exécutée globalement ou pas du tout, pas de modification des données en cas d'anomalie lors de son exécution

  cohérence : garantie de conservation de la cohérence des données à l'issue de l'exécution de la transaction

  isolation : transactions exécutées indépendamment les unes des autres, avec un résultat identique à leur exécution l'une après l'autre (en série)

  durabilité : données garanties mises à jour durablement dès la fin de l'exécution de la transaction, même en cas de défaillance du système

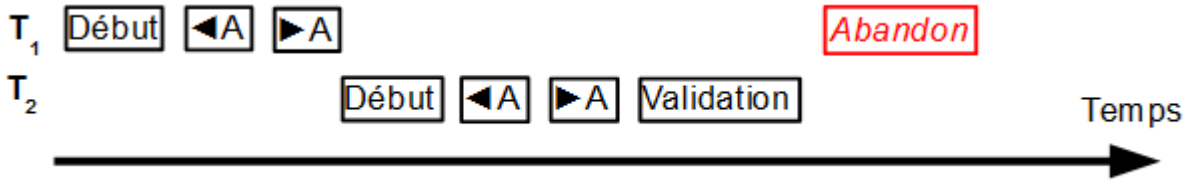
TRANSACTIONS GÉRÉES PAR LE SGBD AVEC DES TECHNIQUES SPÉCIALISÉES

TRANSACTIONS : CAS PROBLÉMATIQUES

SITUATIONS ANORMALES DÉFINIES PAR SQL

- LECTURE IMPROPRE (*DIRTY READ*)

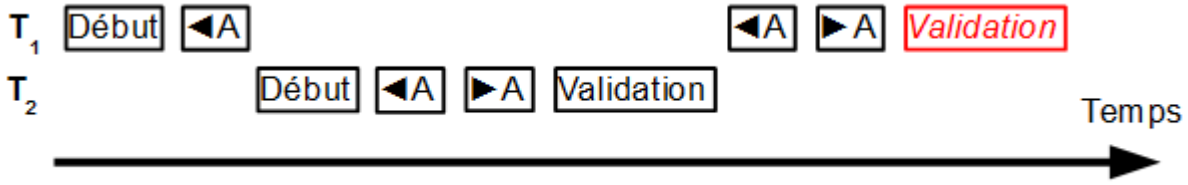
  lecture d'une donnée en cours de mise à jour par une autre transaction pas encore validée



  exemple : la transaction T_2 lit une valeur de A qui n'est pas valide car juste modifiée par la transaction T_1 en fait annulée plus tard

- LECTURE NON REPRODUCTIBLE (*NONREPEATABLE READ*)

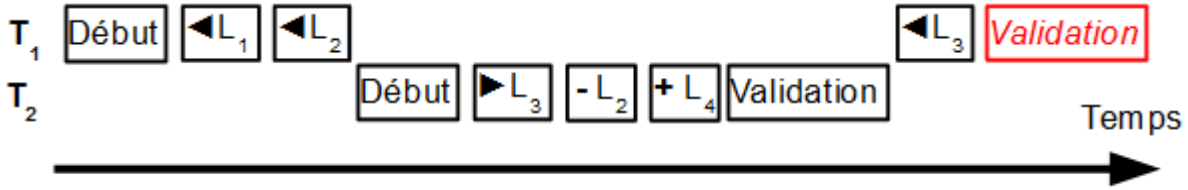
  lectures successives d'une donnée modifiée entre deux lectures par une autre transaction



  exemple : la transaction T_1 lit successivement deux valeurs différentes de A car sa valeur est modifiée entre-temps par la transaction T_2

- LECTURE FANTÔMATIQUE (*PHANTOM*)

  interrogations successives d'une table dont le nombre de lignes a changé entre temps du fait de mises à jour (ajout ou supression de ligne) effectuées par une autre transaction



  exemple : la transaction T_1 lit un ensemble de lignes dont la sélection initiale est entre-temps rendue incohérente par les mises à jour de T_2

TRANSACTIONS : TECHNIQUES DE GESTION

CONTRÔLE DES EXÉCUTIONS SIMULTANÉES (« CONCURRENCE »)

● CARACTÉRISTIQUES GÉNÉRALES

- Sérialisation des transactions

  exécution simultanée d'un ensemble de transactions rendue équivalente à leur exécution séquentielle, produisant les mêmes résultats

  prise en compte des situations conflictuelles où deux opérations de deux transactions manipulent le même objet, dont au-moins une en écriture

- Deux grandes familles de stratégies

  stratégie pessimiste (conflits fréquents) : prévention par mise en attente de l'opération si détection d'un conflit potentiel

  stratégie optimiste (conflits rares) : exécution avec écriture des modifications dans un fichier temporaire, validation en fin d'exécution avec intégration des modifications si pas de conflit ou sinon réexécution

● PRINCIPALES TECHNIQUES

- Verrous (*locks*)

  garantie d'un accès exclusif à une donnée attribué à une transaction, soit en mode partagé (lecture) ou en mode exclusif (lecture et écriture), demande de verrou mise en attente tant que la garantie est impossible

  protocole de verrouillage à deux phases (*two phase locking*, *2PL*) où la transaction doit prendre tous les verrous nécessaires avant d'en libérer

  problème d'étreinte fatale ou verrou mortel (*deadlock*), où 2 transactions demandent chacune un verrou exclusif qui a déjà été attribué à l'autre

- Estampilles (*timestamps*)

  identification des transactions par une marque (horodate de création ou numéro d'ordre, « estampille ») définissant une relation d'ordre temporel

  donnée marquée de 2 estampilles définies par l'estampille maximale de transactions y ayant accédé en lecture, et celle maximale en écriture

  en cas de conflit entre deux transactions, la plus jeune est prioritaire tandis que la plus vieille est réinitialisée avec une nouvelle estampille

  problème de famine (*livelock*) où 2 transactions redemandent sans cesse chacune un verrou exclusif qui a déjà été attribué à l'autre

TECHNIQUE DU VERROUILLAGE MAJORITAIREMENT UTILISÉE DANS LES SGBD

TRANSACTIONS EN SQL

MISE EN OEUVRE POUR UNE SÉQUENCE D'INSTRUCTIONS

● DÉFINITION D'UNE TRANSACTION

- Mode implicite au niveau de la requête par défaut d'activation
  a priori une requête s'exécute sous la forme d'une transaction
  (`create`, `alter`, `drop`, `select`, `insert`, `update`, `delete`, `grant`, `revoke`)
- Mode explicitement activé en cas d'une séquence de requêtes
  indication des début et fin de transaction par des instructions spécifiques

● DÉBUT DE TRANSACTION

- Démarrage de la transaction [SQL 1999] : `start transaction` *paramètres*
  cas où la transaction se compose d'une séquences d'instructions
  paramètres liées aux caractéristiques de son exécution (voir page 102)
  exemple : `start transaction serializable read only`
- Fixation des paramètres de la transaction : `set transaction` *paramètres*
  réglages s'appliquant à la prochaine transaction ou celle en cours (mais pas celles d'après), ou entraînant le démarrage de la transaction
  sémantique variable selon les SGBD et le mode de programmation
  exemple : `set transaction serializable read only`

● FIN DE TRANSACTION

- Validation finale de la transaction : `commit` ou `commit work`
  possibilité d'enchaîner immédiatement sur une autre transaction démarrée alors avec les mêmes caractéristiques [SQL 1999] :
  `commit and chain` ou `commit work and chain`
- Abandon de la transaction : `rollback` ou `rollback work`
  possibilité d'enchaîner immédiatement sur une autre transaction démarrée alors avec les mêmes caractéristiques [SQL 1999] :
  `rollback and chain` ou `rollback work and chain`

TRAITEMENT TRANSACTIONNEL EN LIGNE
(*ONLINE TRANSACTION PROCESSING*, OLTP)

TRANSACTIONS EN SQL (SUITE)

● PARAMÈTRES D'UNE TRANSACTION

- Déclarations sur son mode de fonctionnement au démarrage (page 104)
  destinée à faciliter la gestion des transactions par le SGBD
- Paramètre du niveau d'isolation vis-à-vis des autres transactions
  soit `read uncommitted` : aucune isolation
  soit `read committed` : si lecture d'une donnée modifiée par une autre transaction alors attente de sa validation
  soit `repeatable read` : si lecture d'une donnée accédée par une autre transaction alors attente de sa validation
  soit `serializable` : exécution de la transaction comme si les transactions s'exécuteraient l'une après l'autre

Situation possible selon le niveau	*lecture impropre*	*lecture non reproductible*	*lecture fantômatique*
`read uncommitted`	**oui**	**oui**	**oui**
`read committed`	non	**oui**	**oui**
`repeatable read`	non	non	**oui**
`serializable`	non	non	non

- Paramètre du mode d'accès aux données
  soit `read only` : aucune modification de données par la transaction,
  soit `read write` : modification de données par la transaction possible
- Paramètre de la profondeur de diagnostic (accès par programmation)
  correspond à un nombre maximal d'anomalies (*exception*) mémorisées
  indication par `diagnostic size` *entier*

● IMPACT SUR LES CONTRAINTES

- Validation des contraintes au niveau des transactions
  soit à chacune de leurs requêtes, soit reportée à la fin de transaction :
  `set contraints` *contraintes mode*
  contraintes : `all` (toutes) ou la liste des contraintes notées par leur nom
  mode : `immediate` (en fin de requête) or `deferred` (en fin de transaction)
  exemple : `set constraints c_stock, c_vente deferred`
- Déclaration facultative à la fin de la définition d'une contrainte
  soit non reportable en fin de transaction (par défaut) : `not deferrable`
  soit sinon : `deferrable initially` *mode* avec mode de validation
  a priori, en fin de transaction : `deferred` ou de requête : `immediate`

INDEX DE TABLE

ELÉMENT ESSENTIEL DE LA PERFORMANCE D'UNE BASE DE DONNÉES

● CARACTÉRISTIQUES GÉNÉRALES

- Mécanisme d'optimisation dans l'accès aux données
  localisation à l'aide d'une information caractéristique (« clef »), beaucoup plus rapide qu'une recherche par parcours séquentiel des données
  accélération des opérations de recherche, tris et jointure mais au détriment d'un ralentissement des opérations de mises à jour
- Définition automatique ou manuelle
  index systématiquement attribué à une clef primaire par le SGBD, à créer manuellement pour les autres cas (clef étrangère, etc.)
- Réalisation par un algorithme et une structure de données spécifiques
  séquentiel indexé (*ISAM* [1]), arbres B/B+ (*B/B+ tree*), hachage (*hashing*)
  possibilités et appplication très dépendantes du SGBD utilisé

● MISE EN ŒUVRE EN SQL

- Instructions non normalisées
  variantes spécifiques à chaque éditeur de SGBD
- Formes génériques quasi universelles
  création : `create index` *nom d'index* `on` *table* `(` *liste de colonnes* `)`
  ou `create unique index` *nom d'index* `on` *table* `(` *liste de colonnes* `)`
  (cas particulier ou toutes les valeurs de la clef sont distinctes)
  suppression : `drop index` *nom d'index*
- Exemples

```
create index i_vente_produit on vente(ref_produit)
create index i_perso_id on personnage(nom, prenom)
drop index i_vente_produit
```

[1] ISAM : *indexed sequential acess method*, inventé par IBM dans les années 1960.

INDEX AVEC ARBRE B+

PRINCIPALE TECHNIQUE UTILISÉE POUR LA RÉALISATION D'UN INDEX [1]

● PRINCIPE

- Arbre trié et équilibré pour représenter les valeurs de clefs [2]

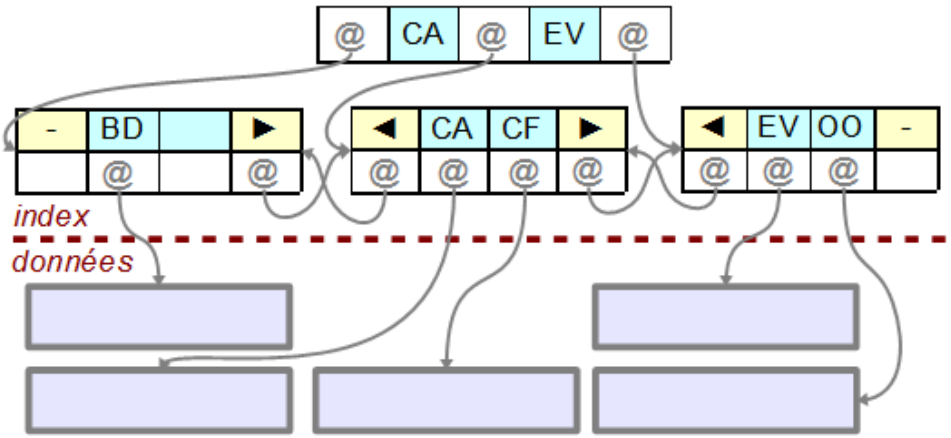
  noeuds et feuilles contenant un nombre fixe maximal L de valeurs de clef triées et avec un taux de remplissage Tx minimal (sauf la racine)

  noeuds partitionnant les valeurs de clefs selon la relation d'ordre, avec pour chaque clef les sous-arborescences des clefs plus petites/grandes

  feuilles avec les adresses des données associées aux valeurs de clef, situées toutes à la même profondeur (équilibrage de l'arbre), avec un chaînage symétrique selon la relation d'ordre

- Exemple

  2 clefs au maximum par noeud et feuille, remplissage minimal à 50 %



● MANIPULATIONS

- Recherche de donnée selon une clef

  recherche dichotomique en temps d'exécution d'ordre logarithmique : Log_{L} (nombre de feuilles)

- Tri des données

  parcours ordonné des feuilles dans le sens du tri

- Ajout ou suppression de donnée

  mise à jour consécutive de l'arbre d'index

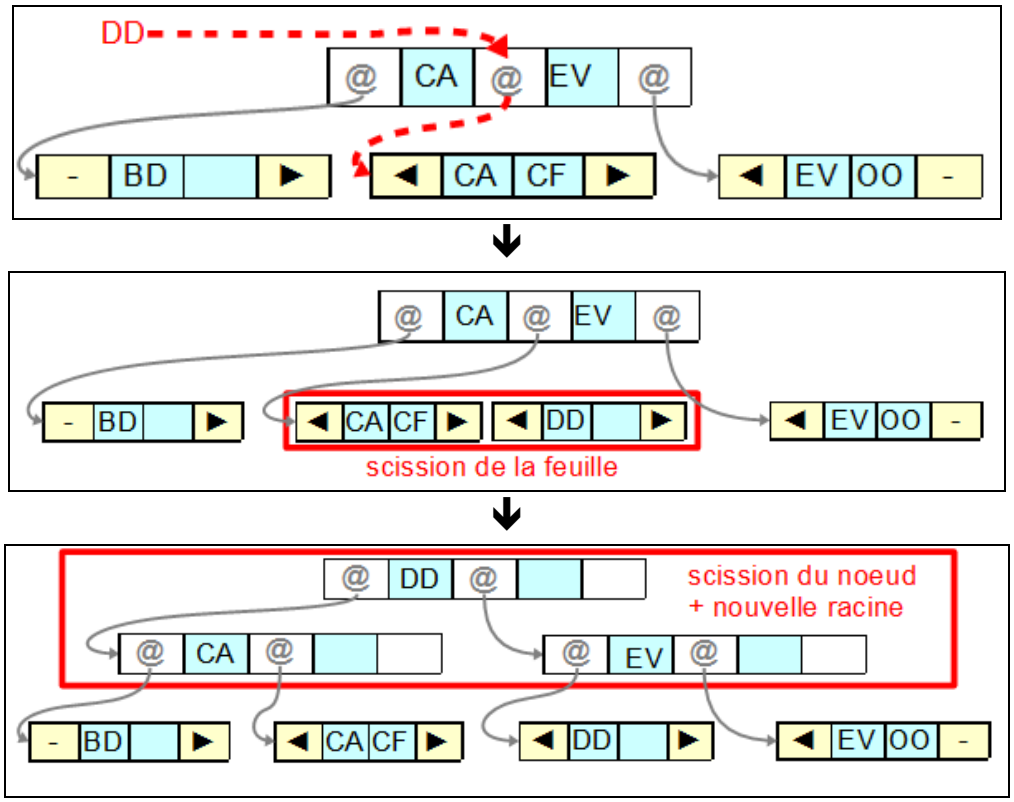
[1] Invention en 1971 par R. Bayer et E.M. McCreight chez Boeing.
[2] On considère ici le cas d'une clef primaire où chaque n-uplet de données est identifié par une clef unique.

ALGORITHME D'INSERTION DANS UN ARBRE B+

Procédure d'insertion dans l'arbre d'une clef C
recherche de la feuille d'accueil F pour la clef C
si encore une place libre alors
    insertion de la clef C dans la feuille F
sinon
    création d'une nouvelle feuille F'
    répartition des clefs en y incluant C, entre la feuille d'accueil F et la nouvelle feuille F'
    insertion de la clef médiane et de l'adresse de la feuille F' dans le noeud parent
fin si

Procédure d'insertion dans un noeud N d'une clef M et de l'adresse associé(e)
si encore de la place libre alors
    insertion de la clef M et de l'adresse dans le noeud N
sinon
    création d'un nouveau noeud N'
    répartition des clefs/adresses entre le noeud N et le nouveau noeud N'
        en y incluant M mais sans la clef médiane
    si au niveau de la racine alors
        création d'une nouvelle racine avec l'adresse des noeuds N, N' et la clef médiane
    sinon
        insertion de la clef médiane dans le noeud parent avec l'adresse du noeud N'
    fin si
fin si

Exemple



CROISSANCE DE L'ARBRE EN LARGEUR, ET EN HAUTEUR PAR LA RACINE

OPTIMISATION DES REQUÊTES

COMMENT MINIMISER LE TEMPS D'EXÉCUTION DES REQUÊTES ?

- PRINCIPES
  - Eléments pénalisants dans l'exécution d'une requête
    nombre important d'accès aux données sur disque (« entrées-sorties »), exemples : jointure brassant un grand nombre de lignes, sous-requête
    temps de calcul pour des comparaisons complexes, à opérandes autres qu'une colonne et une valeur fixe, ou utilisant un motif à début variable
    exemples : `year(arrivee) = 2013` `code like '%D3%'`
  - Optimisations potentielles
    ajustement de tables pour limiter volume et nombre d'accès au disque
    exemples : ajout d'index, réduction de la taille des valeurs de colonne
    réécritures de comparaisons complexes sous une forme plus simple
    exemple : `arrivee >= '2013-01-01' and arrivee < '2014-01-01'`
    réorganisation des opérations constitutives d'une requête
    exemple : sélection effectuée avant une jointure

- MÉCANISMES
  - Modes d'interventions
    soit manuel : réécriture de requêtes, définition d'index etc.
    soit automatique : à l'exécution par l'« optimiseur de requête » du SGBD
  - Optimisation automatique
    analyse de la requête et décomposition en un graphe d'opérations selon l'algèbre relationnelle (sélection, projection, jointure, etc.)
    réorganisation des opérations selon des règles d'optimisation afin d'obtenir une requête équivalente mais d'exécution plus rapide
    a) optimisation « logique » du graphe selon des propriétés algébriques : sélections et projections en 1er, ordonnancement de jointures, etc.
    b) optimisation « physique » à partir des caractéristiques des tables : utilisation d'index et de statistiques sur les distribution de valeurs
    construction d'un plan d'exécution composé d'opérations élémentaires : lecture séquentielle, accès via index, projection, jointure, etc.

OPTIMISEUR SPÉCIFIQUE À CHAQUE SGBD

EXEMPLES D'OPTIMISATION DE REQUÊTE

● OPTIMISATION LOGIQUE DE CONDITIONS

Exemple n° 1

```
select produit.nom
from produit
join lot on produit.numero = lot.ref_produit
where produit.type = 'LEGUME' and produit.type = 'FRUIT'
```

→ détection par l'optimiseur de la condition jamais satisfaite

● RÉORDONNANCEMENT ALGÉBRIQUE

Exemple n° 2

```
select produit.nom, personne.nom
from produit
join lot on lot.ref_produit = produit.numero
join personne on lot.ref_fournisseur = personne.numero
where produit.type = 'FRUIT'
```

→ sélection sur produit
puis jointure sur lot
puis jointure sur personne

Exemple n° 3

```
select distinct personne.nom
from personne
join lot on personne.numero = lot.ref_fournisseur
where personne.numero not in (
 select distinct ref_acheteur from vente
)
```

→ sélection sans doublons sur vente
puis sélection sur personne
puis jointure sur lot
puis élimination des doublons

COMMANDE « EXPLAIN » SPÉCIFIQUE AUX SGBD
INDIQUANT LE PLAN D'EXÉCUTION POUR UNE REQUÊTE

EXERCICES SUR LES COMPLÉMENTS DE SQL

1. Donner un exemple d'étreinte fatale dans le cas de la gestion de transactions à l'aide de verrous.

2. Soit le cas d'une opération de mise à jour s'appliquant sur plusieurs n-uplets comme par exemple l'instruction suivante :

```
update client set compte = compte - (
 select sum(valeur) from achat
 where achat.acheteur = client.id_client
)
```

   quelles sont les conséquences d'une anomalie sur la mise à jour d'un n-uplet (cas par exemple d'un client sans achat) ?

3. On considère un arbre B+ avec une taille L de noeud/feuille fixée à 200 et un taux de remplissage moyen de 67 %.

   a) Calculer les nombres minimal et moyen de clefs stockées dans 2 et 3 niveaux de cet arbre.

   b) Déterminer l'ordre de grandeur du coût de la recherche d'une clef.

4. Proposer une réécriture optimisée de la requête suivante :

```
select sum(taille_restante) from lot
where (prix * 5 < 100) and not (numero < 5 or numero >= 59)
```

PROGRAMMATION EN SQL

PRÉSENTATION DES CARACTÉRISTIQUES GÉNÉRALES

- PROGRAMMATION INTERNE À LA BASE [SQL 1999]
  - Programme stocké dans la base (*persistant stored module*, PSM)
    langage minimal [1] : variables, requêtes, branchements, itérations, tests
    instructions terminées par un point-virgule (« ; »)
    méthode de création spécifique au SGBD
  - Déclencheur (*trigger*)
    action lors d'une modification de table (ajout, mise à jour ou suppression)
  - Procédure stockée (*stored procedure*)
    procédure dans la base appelée via une instruction en SQL
  - Fonction (*function*)
    cas de procédure renvoyant une valeur et utilisable dans une expression

- PROGRAMMATION DANS UNE APPLICATION
  - Deux manières d'exécuter une instruction en SQL
    SQL statique : instruction figée lors de l'écriture du programme
    SQL dynamique : instruction construite à l'exécution du programme
  - SQL intégré à un langage de programmation (*embedded SQL*)
    instructions de SQL compilées avec le programme (C, Cobol, Ada, etc.)
    code statique aux possibilités limitées mais plus efficace à exécuter
  - Interface de programmation avec un SGBD (*call level interface*, CLI)
    instructions de SQL préparées au sein du programme, transmises pour exécution par le SGBD, puis résultats récupérés par le programme
    code dynamique d'écriture plus puissante avec possibilité d'introduction de variable dans une requête, mais moins efficace à exécuter
    interface générique ; exemple : *open database connectivity* (ODBC, Microsoft), *java database connectivity* (JDBC, Sun/Oracle), etc.
    interface dédiée à un SGBD et à un langage de programmation ; exemple : MySQL, Oracle, PostgreSQL, Sybase, Sqlite, etc. en PHP

UNE TRÈS GRANDE BIODIVERSITÉ !

[1] Langage souvent spécifique au SGBD : PL/SQL (Oracle), PL/pgSQL (PostgreSQL), SQL/PL(DB2), etc.

PROGRAMMATION EN SQL : VARIABLE ET BLOC

- VARIABLE
  - Déclaration obligatoire
    syntaxe : `declare` *liste d'identificateur*[1] *type option* `;`
    option : valeur initiale de chaque variable de la liste, *default valeur*
    exemple : `declare nb, rang int default 0 ;`
  - Instruction d'affectation
    syntaxe : `set` *identificateur* `=` *expression* `;`
    exemple : `set nb = nb + 1 ;`
  - Récupération de résultat(s) d'une interrogation
    syntaxe : `select` *résultats* `into` *liste de variables* `from` *liste de tables options* `;`
    options : *sélection regroupement filtrage* (voir page 85)
    exemple : `select prix into valeur from lot where numero = 155 ;`
  - Réutilisation dans une requête
    utilisation du nom de la variable comme une valeur
    exemple : `update produit set prix = tarif2 where numero = 125 ;`

- BLOC D'INSTRUCTIONS
  - Regroupement d'une séquence d'instructions
    syntaxe : `begin` *option déclarations liste d'instructions* `end`
    déclarations : essentiellement les variables locales au bloc
    option : mode d'exécution ordinaire (`not atomic`, par défaut d'indication) ou comme une transaction (`atomic`)
    exemple :

```
begin
  declare nb int default 0 ;
  select sum(taille_restante) into nb from lot
    where taille_restante > 0 ;
  update suivi set stock = nb where nature = 'lot' ;
end
```

[1] La liste peut contenir un seul élément, ou plusieurs séparés alors par une virgule (« , »)

PROGRAMMATION EN SQL : TEST ET ITÉRATION

- INSTRUCTIONS CONDITIONNELLES
  - Plusieurs syntaxes

    `if` *condition* `then` *instruction* `;` `end if`

    `if` *condition* `then` *liste₁ d'instructions* `;` `else` *liste₂ d'instructions* `;` `end if`

    `if` *condition₁* `then` *instruction₁* `;` *liste de sous-tests* `;` `end if`

    liste de sous-tests composée d'un ou plusieurs sous-tests notés chacun :

    `elseif` *condition* `then` *liste₁ d'instruction* `;`

    `if` *condition* `then` *liste₁ d'instructions* `;` *liste de sous-tests* `;`
    `else` *liste₂ d'instructions* `;` `end if`
  - Exemples

```
if nb <= 1 then pluriel = false ; end if
```

```
if stock = 0 then libelle = 'vide' ;
elseif stock < 100 then libelle = 'minimal' ;
elseif stock < 500 then libelle = 'normal' ;
else libelle = 'important' ;
end if
```

- ITÉRATIONS
  - Condition initiale (tant que)

    `while` *condition* `do` *liste d'instructions* `;` `end while`
  - Condition finale (répéter jusque)

    `repeat` *liste d'instructions* `until` *condition* `;` `end repeat`
  - Forme générale (itérer sortirsi recommencer)

    étiquette `:` `loop`
    liste d'instructions₁
    `if` *condition* `then leave` *étiquette* `;` `end if`
    liste d'instructions₂
    `end loop` *étiquette* `;`
  - Exemples

```
set nb = 1 ;
while nb <= 100 do nb = nb * nb ; end while
```

```
set nb = 1 ;
repeat nb = nb * nb until nb > 100 ; end repeat
```

```
set rang = 1 ;
set id = null ;
parcours: loop
  select nom into id from personne where numero = rang ;
if (rang = 10) or (id is not null) then leave parcours ; end if
  set rang = rang + 1 ;
end loop parcours ;
```

PROGRAMMATION EN SQL : CHOIX MULTIPLE

- CHOIX MULTIPLES
  - Choix selon la valeur d'une expression

    `case` *expression* *liste de cas* `;` *alternative* `;` `end case`

    liste de cas : un ou plusieurs cas à la suite, chacun noté :
    `when` *valeur* `then` *liste d'instructions* `;`

    alternative : optionnelle, s'applique si aucun cas n'est vérifié :
    `else` *liste d'instructions* `;`

    le 1er cas trouvé est exécuté puis l'instruction s'achève

    exemple :

```
case genre
 when 'F' then set libelle = 'femme' ;
 when 'H' then set libelle = 'homme' ;
 else set libelle = 'inconnu' ;
end case
```

  - Choix selon une des conditions d'une liste

    `case` *liste de cas* `;` *alternative* `;` `end case`

    liste de cas : un ou plusieurs cas à la suite, chacun noté :
    `when` *condition* `then` *liste d'instructions* `;`

    alternative : optionnelle, s'applique si aucun cas n'est vérifié :
    `else` *liste d'instructions* `;`

    le 1er cas trouvé est exécuté puis l'instruction s'achève

    exemple :

```
case
 when pays = 'FR' then set libelle = 'France' ;
 when pays is null then set libelle = 'Inconnu' ;
 else set libelle = 'Etranger' ;
end case
```

PROGRAMMATION EN SQL : FONCTION, PROCÉDURE

- FONCTION
  - Sous-programme renvoyant un résultat
    renvoi d'une valeur avec éventuellement des paramètres d'entrée
    réutilisation dans une expression
  - Syntaxe
    `create function` *nom* `(` *paramètres* `) returns` *type du résultat*
    bloc d'instructions
    paramètres : zéro, une ou plusieurs définitions de paramètre par *nom type*
    renvoi de la valeur par : `return` *valeur* `;`
    exemple :

```
create function en_pourcent (ratio float) returns decimal(5,2)
-- renvoi de la notation decimale en pourcentage de ratio
begin
 return cast( (ratio * 100) as decimal(5,2) ) ;
end
```

- PROCÉDURE
  - Sous-programme
    ensemble d'actions avec éventuellement des paramètres d'entrée-sortie
  - Syntaxe
    `create procedure` *nom* `(` *paramètres* `)` *bloc d'instructions* `;`
    paramètres : zéro, une ou plusieurs définitions de paramètre d'entrée et-ou de sortie par *option nom type*
    option indique si le paramètre est en entrée (`in`, par défaut), en résultat (`out`) ou les deux à la fois (`inout`)
    appel de la procédure par : `call` *nom* `(` *paramètres d'appel* `) ;`
    exemple :

```
create procedure trace ( texte varchar(60) )
-- enregistrement horodate de texte dans le journal
begin
  insert into journal(numero, horodate, libelle)
    values(default, current_date, texte) ;
end
```

    avec l'appel suivant :

```
call trace ('arret du serveur') ;
```

PROGRAMMATION EN SQL : ANOMALIES

- PRINCIPES

  - Variable d'état

    variable `sqlstate` indiquant le résultat de la dernière requête

    texte de 5 chiffres dont les 2 premiers indiquent le niveau d'erreur :
    `'00'` (pas d'anomalie), `'01'` (avertissement), `'02'` (pas de données), etc.

  - Exceptions

    anomalies provoquant le déroutement de l'exécution du programme

    possibilité de définir un gestionnaire pour une anomalie particulière

- MISE EN OEUVRE EN SQL

  - Consultation de `sqlstate`

    déclaration de la variable obligatoire

    exemple :

```
declare sqlstate varchar(5) ;
declare fin boolean default false ;
if substring(sqlstate from 1 for 2) = '02' then
 set fin = true ;
end if
```

  - Définition d'un gestionnaire

    `declare` *type* `handler for` *liste d'anomalies* *instruction* `;`

    type spécifie le devenir du programme à l'origine de l'anomalie :
    poursuite (`continue`), abandon (`exit`), annulation de transaction (`undo`)

    liste d'anomalies contient une ou plusieurs [1] anomalies (« conditions ») :

    - `sqlwarning` : avertissement
    - `not found` : plus ou pas de donnée
    - `sqlexception` : autres cas d'anomalie
    - un nom associé à une anomalie particulière

    nommage d'une anomalie particulière selon sa valeur de `sqlstate` :

```
declare nom condition for valeur ;
```

    exemple :

```
declare continue handler for not found set fin = true ;
```

[1] Séparation dans la liste par une virgule (« , »)

PROGRAMMATION EN SQL : DÉCLENCHEUR

- DÉCLENCHEUR (*TRIGGER*) [SQL 1999]
  - Action associée à une modification d'une table

    cas d'un ajout, d'une mise à jour (seulement sur certaines colonnes en option) ou d'une suppression de ligne(s) dans la table

    action déclenchée juste avant ou juste après la requête de modification, et s'appliquant soit globalement à l'ensemble des lignes modifiées, soit individuellement à chacune de ces lignes

    accès aux anciennes et-ou aux nouvelles valeurs de colonnes par : `new.`*colonne* (insertion, mise à jour) ou `old.`*colonne* (suppression) (possibilités de renommage non présentées ici)

  - Syntaxe

    `create trigger` *nom moment opération* `on` *table* $option_{portée}$ $option_{condition}$ *action ;*

    moment par rapport à l'opération : `before` (avant) ou `after` (après)

    opération : `insert`, `delete`, `update` ou `update of` *liste de colonnes*

    $option_{portée}$ pour les lignes concernées : `for each row` (individuellement) ou `for statement` (globalement, par défaut d'indication)

    $option_{condition}$ pour restreindre le déclenchement : `when` *condition*

    action définie par une requête ou un bloc d'instructions transactionnel : `begin atomic` *liste d'instructions* ; `end`

    exemple :

```
create trigger suivi_suppr before delete on produit
for each row
-- trace de la suppression d'un produit
begin atomic
  declare texte varchar(60) default '?' ;
  declare nb int default 0 ;
  select count(*) into nb from lot where numero = old.numero ;
  set texte = 'suppr produit ' || old.numero ||
    ' "' || old.nom || '" ' || nb || ' lots ' ;
  insert into journal(numero, horodate, libelle)
    values(default, current_date, texte) ;
end
```

PROGRAMMATION EN SQL : CURSEUR

PRINCIPES GÉNÉRAUX

- Rôle

  manipulation de l'ensemble des lignes issues d'une requête, à l'aide d'un repère parcourant les lignes une à une (dispositif de pointage)
- Utilisation en 4 phases

  1) déclaration : définition avec un nom et une requête associée

  2) ouverture : activation du mécanisme

  3) accès aux lignes : lecture de la ligne suivante ou selon une position

  4) fermeture : désactivation du mécanisme
- Déclaration

  `declare` *nom* *option*$_{maj}$ *option*$_{accès}$ `cursor for` *interrogation* *option*$_{nature}$ *;*

  option$_{maj}$, lignes non modifiables par d'autres transactions : `insensible`

  option$_{accès}$, accès aux lignes l'une après l'autre : `not scroll` (par défaut), ou accès non séquentiel par position : `scroll`

  option$_{nature}$, accès en vue de la modification des lignes : `for update`, ou de certaines colonnes : `for update of` *liste de colonnes*, ou sans modification : `for read only` (par défaut si `insensible`, `scroll`, tri)

  exemple :
  ```
  declare curseur_fruit cursor for
  select nom, prix from produit where type_produit = 'FRUIT'
  for read only
  ```
- Ouverture

  `open` *nom ;*
- Accès

  cas d'accès séquentiel : `fetch` *nom* `into` *liste de colonnes ;*

  cas d'accès par position : `fetch` *position* `from` *nom* `into` *liste de colonnes ;*

  ou sans `from` : `fetch` *position nom* `into` *liste de colonnes ;*

  position : `next` (suivante), `prior` (précédente), `first` (1ère), `last` (dernière), `absolute` *n* (au rang *n*, à partir de 1), `relative` *n* (décalage de *n*)

  fin de lecture ou lecture impossible : exception de niveau `'02'`

  désignation de la ligne courante dans une requête de mise à jour via la condition de restriction : `current of` *nom du curseur*

  exemples :
  ```
  fetch curseur_fruit into nom_fruit, prix_fruit ;
  update produit set prix = 0 where current of curseur_fruit
  ```
- Fermeture

  `close` *nom ;*

PROGRAMMATION EN SQL : EXEMPLE

CAS D'UNE FONCTION UTILISANT UN CURSEUR

- DÉFINITION DE LA FONCTION

```
create function liste_stock() returns varchar(200)
-- renvoi du texte de la liste des produits en stock
--  sous la forme individuelle "nom:taille/"
begin

declare resultat  varchar(200);
declare libelle varchar(40);
declare taille  integer;
declare fait boolean default false;
-- curseur de lecture des stocks
declare un_stock cursor for
    select produit.nom, sum(taille_reste ) as stock
    from produit join lot on lot.ref_produit = produit.numero
    where type = 'LEGUME' and taille_reste > 0
    group by lot.ref_produit
    order  by stock desc ;
-- gestionnaire pour la fin de lecture via le curseur
declare continue handler for not found set fait = true;

-- iteration de lecture des stocks
set resultat = '';
open un_stock;
iteration: loop
  fetch un_stock into libelle, taille;
if fait then leave iteration; end if;
  set resultat = resultat || libelle || ':' || taille || '/';
end loop iteration;
close un_stock;
-- renvoi du resultat
return resultat;

end
```

- EXEMPLE D'UTILISATION

```
select liste_stock()
```

→ BETTERAVE:900/CAROTTE:534/BROCOLIS:398/PANAIS:177/AIL:8/

EXERCICES SUR LA PROGRAMMATION EN SQL

1. Ecrire une fonction qui calcule l'état du stock d'un produit donné.

2. Ecrire une procédure qui majore tous les prix par un coefficient donné.

3. Créer un déclencheur qui lors de la fin du stock d'un produit ajoute une ligne dans un journal à l'aide de la procédure `trace()` définie page 116.

4. En utilisant un curseur, créer une fonction qui retourne la liste de tous les fournisseurs situés dans un département donné.

ANNEXES

MOTS RÉSERVÉS DE SQL

LISTE DES MOT-CLEFS RÉSERVÉS DANS SQL 2

absolute
action
add
all
allocate
alter
and
any
are
as
asc
assertion
at
authorization
avg
begin
between
bit
bit_length
both
by
cascade
cascaded
case
cast
catalog
char
char_length
character
character_length
check
close
coalesce
collate
collation
column
commit
connect
connection
constraint
constraints
continue
convert
corresponding
create
cross
current
current_date
current_time
current_timestamp
current_user
cursor
date
day
deallocate
dec
decimal
declare
default
deferrable
deferred
delete
desc
describe
descriptor
diagnostics
disconnect
distinct
domain
double
drop
else
end
end-exec
escape
except
exception
exec
execute
exists
external
extract
false
fetch
first
float
for
foreign
found
from
full
get
global
go
goto
grant
group
having
hour
identity
immediate
in
indicator
initially
inner
input
insensitive
insert
int
integer
intersect
interval
into
is
isolation
join
key

language
last
leading
left
level
like
local
lower
match
max
min
minute
module
month
names
national
natural
nchar
next
no
not
null
nullif
numeric
octet_length
of
on
only
open
option
or
order
outer
output
overlaps
pad
partial
position
precision
prepare
preserve
primary
prior
privileges
procedure
public
read
real
references
relative
restrict
revoke
right
rollback
rows
schema
scroll
second
section
select
session
session_user
set
size
smallint
some
space
sql
sqlcode
sqlerror
sqlstate
substring
sum
system_user
table
temporary
then
time
timestamp
timezone_hour
timezone_minute
to
trailing
transaction
translate
translation
trim
true
union
unique
unknown
update
upper
usage
user
using
value
values
varchar
varying
view
when
whenever
where
with
work
write
year
zone

LISTE DES MOT-CLEFS RÉSERVÉS ADDITIONNELS DANS SQL 3

abs
action
after
aggregate
alias
array
before
binary
blob
boolean
breadth
call
cardinality
clob
completion
cube
current_path
cycle
data
depth
deref
destroy
dictionary
each
equals
every
factor
free
general
grouping
hold
host
ignore
initialize
iterate
less
large
limit
locator
map
mod
modifies
modify
nclob
new
no
none
object
off
old
operation
operator
ordinality
overlay
parameter
parameters
path
preorder
reads
recursive
ref
referencing
relative
replace
result
return
returns
rollup
role
routine
row
savepoint
search
sensitive
sequence
session
sets
similar
space
specific
specifictype
sqlexception
sqlwarning
start
state
static
structure
sublist
symbol
term
terminate
the
treat
trigger
type
under
variable
without

BIBLIOGRAPHIE

OUVRAGES UTILISÉS POUR LA PRÉPARATION DE CE DOCUMENT

HAINAUT Jean-Luc. **Bases de données : concepts, utilisation et développement**. Dunod, 2012, 2ème édition, 700 pages, 35,90 €.

Ouvrage complet et très pédagogique, traitant non seulement les concepts fondamentaux, mais aussi les méthodologies de conception.

`http://info.fundp.ac.be/~dbm/mediawiki/index.php/DUNOD2009`

BROUARD Frédéric, BRUCHEZ Rudi, SOUTOU Christian. **SQL**. Pearson, 2010, 3ème édition, 300 pages, 25 €.

Ouvrage pédagogique rédigé par des praticiens de ce langage.

`http://www.pearson.fr/livre/?GCOI=27440100939490`

AUTRES OUVRAGES

RAMAKRISHNAN Raghu, GEHRKE Johannes. **Database management systems**. Mcgraw-Hill , 2002, 3ème édition, 1 100 pages, 56 € (poche).

Ouvrage de référence très complet sur les bases de données.

`http://shop.oreilly.com/product/0636920022879.do`

DATE Chris. **SQL and relational theory: how to write accurate SQL code**. O'Reilly, 2011, 2ème édition, 450 pages, 40 $

Ouvrage de référence sur le modèle relationnel et SQL.

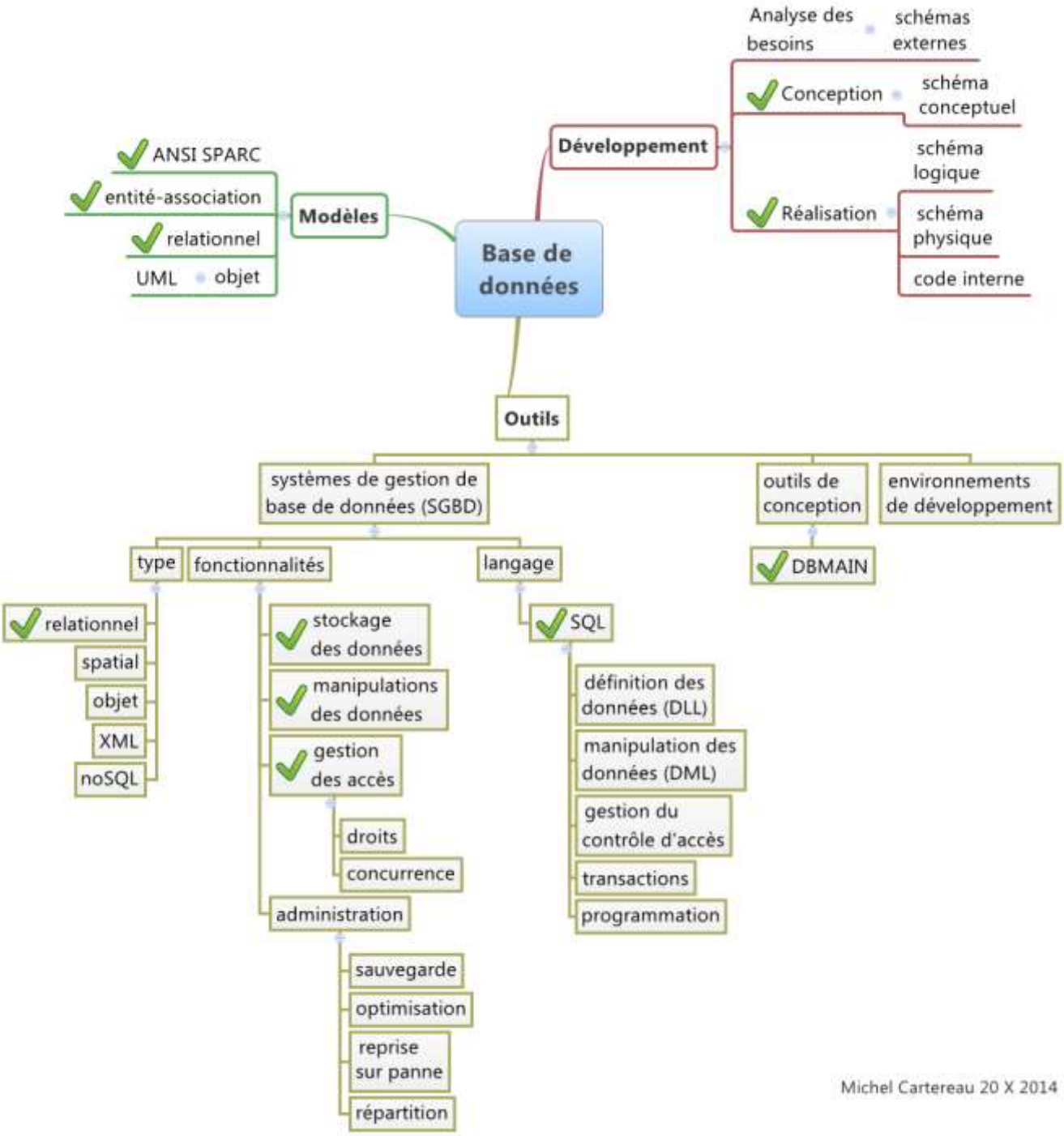
`http://shop.oreilly.com/product/0636920022879.do`

GARDARIN Georges. **Bases de données**. Eyrolles, 2003, 790 pages, gratuit en ligne.

Ouvrage de référence un peu ancien par l'un des universitaires français spécialistes du domaine.

`http://izibook.eyrolles.com/produit/2385/9782212175035/Bases%20de%20donnees`

VUE SYNTHÉTIQUE DU DOMAINE



INDEX

SOMMAIRE