Chapitre 1 : Concepts Fondamentaux

1) Introduction

On fait appel à l'approche base de données lorsque les données à gérer sont de natures diverses (exemple : étudiants, cours, enseignants, salles, ...) et possèdent de nombreux liens entre elles (exemple : un étudiant suit un cours, un cours est assuré par un enseignant, ...).
Cours(nom du cours, section, prérequis, nom enseignant),
Enseignant (nomenseignant, grade, date de recrutement),
Etudiant (NomEtudiant, section, groupe…)

Définitions : Bases de données et SGBD:

- Une **base de données** (BD) représente l'ensemble (cohérent, intégré, partagé) des informations nécessaires au fonctionnement d'une entreprise, ensemble dont la gestion est assurée par un logiciel appelé **système de gestion de bases de données** (SGBD).

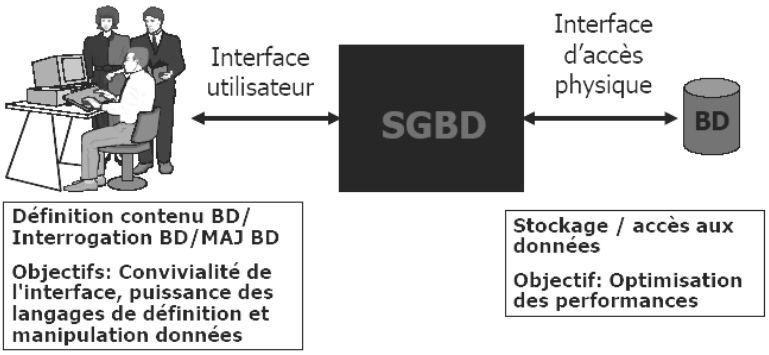
Cohérent : L'ensemble des données a une signification (pas n'importe quelles données)
Intégré : elles sont regroupées au sein d'un même ensemble
Partagé : utilisées par plusieurs utilisateurs et/ou types d'utilisateurs

- Un **SGBD** est un ensemble de programmes i.e. software permettant de :

Définir la BD : Spécifier les types de données, la structure, contraintes…ex: structure de Billet, de ses champs,…
Construire la BD: Stocker les données sur disque
Manipuler la BD : Récupérer des données stockées (requêtes sur la BD), ex: liste des billets de train
Mettre à jour les données : ex: changer l'heure de départ d'un train
Maintenir la BD : Gestion des données (concurrence, fiabilité,…) et des utilisateurs (droits)

2) Architecture d'un SGBD

Au niveau d'abstraction le plus élevé, un SGBD peut être vu comme une boite noire, assurant la gestion de la BD conformément aux requêtes de ses utilisateurs:



- L'**interface utilisateur** permet aux utilisateurs d'exprimer des requêtes: soit pour définir le contenu de la BD (avec le LDD), soit pour interroger la BD (en extraire des informations), soit enfin pour apporter des modifications à ce qui a été enregistré.
- L'**interface d'accès physique** permet au SGBD d'accéder aux données sur les supports (disques, ...).

Un SGBD gère des problèmes très différents, avec des objectifs particuliers:

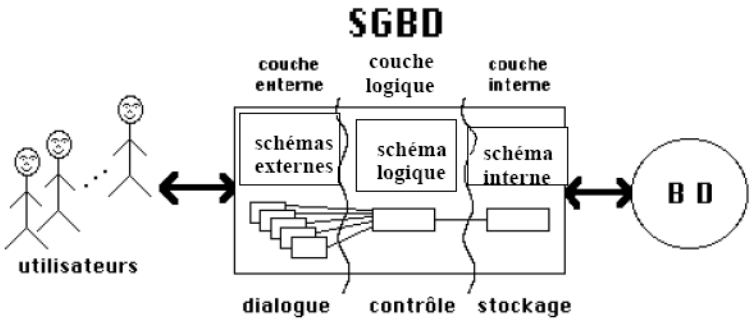
- **interface utilisateur:** compréhension, analyse et vérification des requêtes; objectifs: convivialité de l'interface, puissance des langages de description et de manipulation;
- **interface d'accès physique**: optimisation du stockage des données (en termes d'espace occupé sur les supports) et de l'accès aux données (en temps); objectif: avoir les meilleures performances.

2) Les couches du SGBD

Un SGBD étant utilisé simultanément par plusieurs utilisateurs, il a à résoudre plusieurs problèmes internes de coordination de ses actions, de cohérence et de contrôle du bon déroulement de ses activités.
Pour ce faire on distingue trois couches :

- **niveau externe** prend en charge le problème du dialogue avec les utilisateurs, c'est-à-dire l'analyse des demandes de l'utilisateur, le contrôle des droits d'accès de l'utilisateur, la présentation des résultats
- **niveau interne** s'occupe du stockage des données dans les supports physiques et de la gestion des structures de mémorisation (fichiers) et d'accès (gestion des index, des clés, ...)
- **niveau intermédiaire**: assure les fonctions de contrôle global:
  - optimisation globale des requêtes
  - gestion des conflits d'accès simultanés de la part de plusieurs utilisateurs
  - contrôle général de la cohérence de l'ensemble
  - garantie du bon déroulement des actions entreprises même en cas de panne
  - ...

La couche intermédiaire de contrôle est appelée niveau logique.



4) Modèle de données et schéma

Pour chaque couche d'un SGBD, une description est élaborée. Chacune des descriptions fait appel à un langage formel, basé sur un certain nombre de concepts bien établis. Exemple de concepts **objets, lien propriété**.
Ainsi pour chaque couche un modèle de donnée est défini auquel est associé un schéma.

On appelle **modèle de données** l'ensemble des concepts qui permettent la description de données d'une base et les règles d'utilisation de ces concepts.
On appelle **schéma** d'une base de données l'expression de la description de la base de données d'une entreprise obtenue en employant un modèle de données.

4.1 Niveau logique

On définit un Schéma logique qui représente une description des données présentes dans la base de données dans les concepts du **modèle** utilisé par le SGBD choisi.
On appelle **Modèle logique** le modèle sur lequel est construit un SGBD. Il existe plusieurs modèles de SGBD :

Relationnel, objet,…

Exemple
Schéma Logique (SL) Relationnel (application institut de formation)

Étudiant (nom, prénom, date naissance, n°étudiant)
Enseignant (nom, prénom, statut, no_compte)
Cours (nomC, cycle, nom_enseignant)
Inscription (n°étudiant, nomC, note1, note2)

4.2 Niveau externe

On définit pour chaque groupe d'utilisateur un **schéma externe. Le Schéma externe** définit la vue de base pour ces utilisateurs. Le schéma externe représente le sous ensemble de la base données auquel à accès le groupe d'utilisateurs. Les avantages de cette approche sont la simplicité ainsi que la protection (confidentialité). Dans les SGBD actuels, le modèle de données employé pour décrire les schémas externes est le même que celui du schéma logique.

Exemples

Schéma externe du professeur de base de données :

Etudiant _BD : nom, prénom, note1, note2, note_finale
tel que Etudiant _BD résulte de la combinaison de Etudiant et Inscription du SL, tels qu'il existe une Inscription de cet étudiant pour le cours BD (n°étudiant dans Etudiant = n°étudiant dans Inscription et nom_cours dans Inscription = BD), et
tel que note_finale = (note1 + note2)/2
Schémas externe : service personnel enseignant

- Professeur : nom, prénom, n°compte_bancaire, nombre_de_cours, liste(nom_cours)
  tel que Professeur résulte de la combinaison de Enseignant et Cours du SL, tels que liste(nom_cours) est la liste de nomC qui se trouvent dans Cours tel que nom_enseignant dans Cours = nom dans Enseignant, et
  tel que nombre_de_cours = Cardinalité (liste(nom_cours))

4.3 Schéma interne (SI)

L'implantation des données elles-mêmes, c'est-à-dire le chargement de la base de données avec la version initiale, nécessite que soient fixés les choix en matière de structuration de données sur la mémoire secondaire (quels types de fichiers? quels index? ...). Ces choix sont fait par les administrateurs système qui, en fonction de leur analyse des traitements qui vont être effectués sur la future base de données, détermineront les paramètres effectifs pour l'implantation de la base sous forme d'un ensemble de fichiers. L'ensemble de ces choix sera consigné dans ce que l'on appelle le **schéma interne** de la base de données: description de comment les données de la base sont enregistrés dans les fichiers. Cette description fait donc appel à un nouveau modèle, appelé **modèle interne**, où les concepts seront ceux de fichier, organisation, index, chemin d'accès, clé, ...

Exemple
- Etudiant : fichier **FEtud**,
  contenu : nom, prénom, date de naissance, n°étudiant
  indexé sur n°étudiant,
  index secondaire sur nom+prénom

- Enseignant + Cours: fichier **FEnsCours,**

contenu : nom, prénom, statut, n°compte_bancaire, liste(nomC, cycle)
tel que nom_enseignant dans Cours = nom dans Enseignant
indexé sur nom,
deux index secondaires, l'un sur nomC, l'autre sur cycle

- Inscription : fichier **FInscrits**,
  contenu : n°étudiant, nom_cours, note1, note2
  indexé sur n°étudiant,
  index secondaire sur nom_cours

5) Cycle de vie d'une base de donnée :

L'établissement d'une bases de données passe par 4 phases principales :
Phase 1 : Conception de la base : dans cette phase un schéma conceptuel est élaboré à partir des données de l'application.
Phase 2 : Implémentation de la base. Dans cette phase le schéma logique, le schéma interne ainsi que la population de la base sont déterminés.
Phase 3 : Utilisation de la base. Correspondant à l'interrogation, le développement des programme et les mises à jour.
Phase 4 : Maintenance de la base.

5.1 Conception :

Cette phase est une phase de réflexion et d'analyse sur la manière de structurer les données en fonction des besoins de l'application. Son objectif principal est de déterminer le futur contenu de la base de données. Il faut ainsi que l'ensemble des utilisateurs actuels et futurs de cette base de données se mettent d'accord sur la nature et les caractéristiques des informations qu'il faut garder pour assurer la gestion de l'entreprise. Le résultat de cette phase est le **Schéma conceptuel.**

5.2 Implantation

La phase d'implantation correspond à la transmission de la description des données au SGBD choisi. Le schéma conceptuel sera traduit en schéma logique et le schéma logique sera traduit en schéma interne. Ceci est fait moyennant un langage de description de données (LDD) spécifique au SGBD choisi. Une fois que le SGBD aura pris connaissance de cette description, il sera possible aux utilisateurs d'entrer les données, c'est-à-dire de constituer la première version, initiale, de la base de données. On appelle **implantation de la base de données** cette phase qui consiste à décrire la base de données dans le langage du SGBD et construire cette première version.

5.3 Utilisation de la base

L'**utilisation de la base de données** se fait au moyen d'un langage, dit **langage de manipulation de données** (LMD), qui permet d'exprimer aussi bien les requêtes d'interrogation (pour obtenir des informations contenues dans la base) que des requêtes de mise à jour (pour ajouter de nouvelles informations, supprimer des informations périmées, modifier le contenu des informations).

5.4 Maintenance de la base

On distingue deux types de maintenances : la maintenance corrective et la maintenance évolutive. La maintenance corrective consiste à supprimer les données erronées, périmées ou redondantes. La maintenance évolutive consiste à faire évoluer la structure des données, exemple ajouter une colonne adresse à Etudiant.

Chapitre 2

Modèle Conceptuel Entité-Association

1 Modélisation Conceptuelle

La modélisation conceptuelle vise à la définition du schéma conceptuel de la base de données. Le Schéma conceptuel est une description de la BD obtenue en utilisant un modèle de données.

Il existe plusieurs modèles conceptuels : Entité-Association – EA (ER: Entity-Relationship), UML, Autres (OO, OR).

2 Modèle Entité Association (Concepts généraux)

Le modèle EA étant un modèle conceptuel il se base dans la description du monde réel sur les concepts générique **objets, liens et propriétés** renommées. En effet, la correspondance entre les trois concepts génériques et la terminologie du modèle EA est la suivante:

objet → entité, lien→ association, propriété→attribut

Entité : est une représentation d'un objet du monde réel (concret ou abstrait) ayant une existence propre, indépendamment des autres objets. Exemple : Bouguerra mohamed, service comptabilité, atelier de fabrication A22….

Type Entité (TE) c'est la représentation d'un ensemble d'entités perçues comme similaires et ayant les mêmes caractéristiques. Exemple employé (ensemble des employé d'une entreprise), atelier, service

Association : c'est un lien entre plusieurs entités. Exemple : le lien travail entre bouguerra mohamed et le service comptabilité

Type association (TA) : représentation d'un ensemble d'association ayant la même sémantique et décrites par les mêmes caractéristiques. Exemple : le TA « fabrique » entre TE atelier et TE pièce.

Rôle d'une association : dans une association, chaque entité joue un rôle déterminé

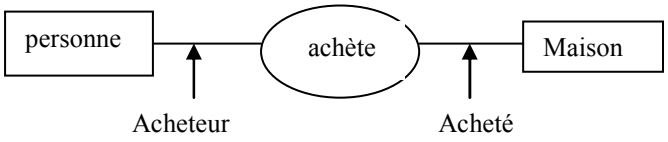


fig 2.1 Rôles acheteur et acheté dans l'association acheté

Une Association peut être **binaire** (2 rôles), **Cyclique** (2 rôles liant le même TE) dans ce cas les rôles doivent être nommés, ou **ternaire** (3 rôles obligatoire).

Attribut : représentation d'une propriété associée à un TE, ou à un TA. L'ensemble des attributs d'un TE (TA) représente l'ensemble des informations inhérentes que l'on souhaite conserver sur les entités (associations) du TE (TA). Exemples: nom, prénoms, salaire sont des attributs du TE Employé; quantité-en-fabrication est un attribut du TA fabrique; date-de-mariage est un attribut du TA est-marié-avec; jour, mois, année sont des attributs composant un attribut date, ..

On appelle **occurrence** d'un TE (TA) toute entité (association) appartenant à l'ensemble décrit par le TE (TA).

Une **occurrence de TE** est vue par l'utilisateur comme un ensemble de valeurs: une valeur pour chaque attribut du TE (NB: il est possible que la valeur d'un attribut soit en fait une absence de valeur ou un ensemble de valeurs).

Une **occurrence de TA** est vue par l'utilisateur comme un ensemble de valeurs (éventuellement vide) et un ensemble d'occurrences de TE : une valeur pour chaque attribut du TA (s'il en existe), et, pour chaque rôle associé au TA, une occurrence du TE qui joue ce rôle.

On appelle **population** d'un TE (TA) l'ensemble des occurrences du TE (TA).

La **base de données** décrite par un schéma EA est donc l'ensemble des populations des TE et TA apparaissant dans le schéma.

Représentation graphique du modèle EA : Le modèle EA permet une représentation graphique du schéma d'une base de données. Dans cette représentation, appelée **diagramme** EA, les types d'entités sont représentés par des rectangles; les types d'associations sont représentés par des hexagones ou autre symbole similaire (ovale, losange...). Les attributs sont listés à l'intérieur du rectangle TE/TA.

Exemple :

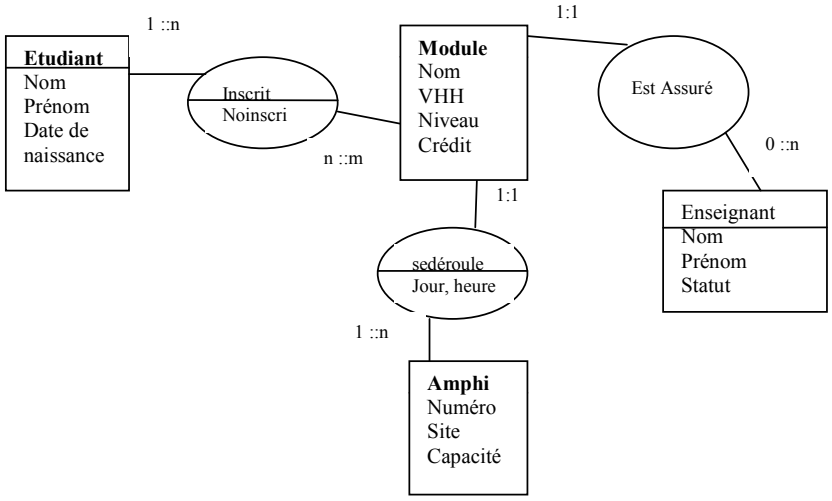


Fig 2.2 Exemple d'un diagramme EA

3 Représentation multiples : La généralisation/spécialisation

Un type d'entité représente une classe d'objets du monde réel perçus comme similaires et ayant les mêmes caractéristiques. Or, il arrive parfois qu'un même ensemble d'objets soit perçu d'un certain point de vue comme une seule classe, mais en même temps perçu d'un autre point de vue comme plusieurs classes, différentes malgré l'existence de caractéristiques communes.

Exemple: dans une application de gestion d'un Hypermarché, le TE « Article » regroupe tous les articles vendus, le TE « article » est dit **générique** si l'on définit des TE plus **spécialisés**, tels que « Article

alimentaire », « Article d'habillement », « Article Hi-Fi »…ces types entités représentent les sous-classes du TE « Article ».

Pour décrire une telle situation, les modèles de données récents incluent le concept de **généralisation/spécialisation**: un lien, orienté, d'un TE spécialisé (ou spécifique) vers un TE générique.

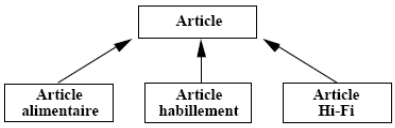
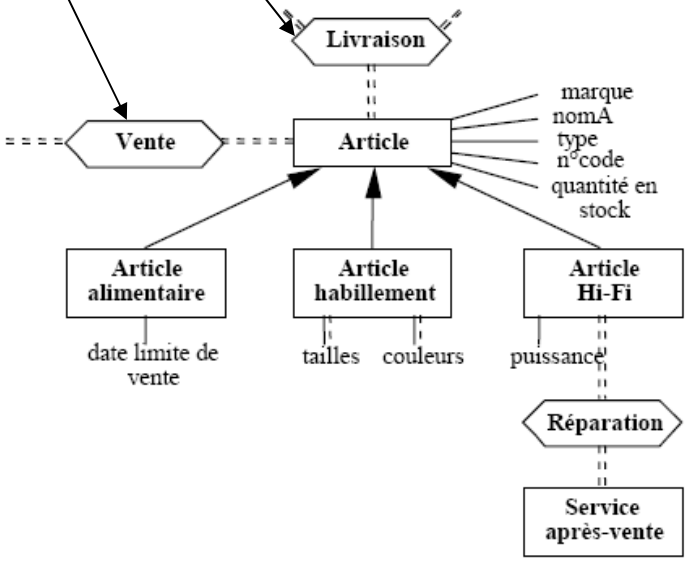


Fig 2.3 : Diagramme EA avec Spécialisation

Les liens de généralisation/spécialisation sont souvent appelés liens "est-un" (IS A); on dit que "Article alimentaire **est-un** Article".
On dit également que le TE spécifique est un **sous-type** du TE générique qui, lui, est **sur-type** du TE spécifique.
Par convention, les attributs communs au TE générique et aux TE spécifiques ne sont décrits, dans le schéma, que comme attributs du TE générique. Néanmoins, ils sont implicitement inclus dans les attributs des TE spécifiques: on dit que ces derniers "**héritent**" des attributs du TE générique. En plus des attributs hérités, les TE spécifiques peuvent avoir des attributs propres. Exemple : date de péremption

Ce qui a été dit pour la description et **l'héritage des attributs** s'applique également à **l'héritage des rôles d'association**. Par exemple, le TE "Article Hi-Fi", comme les autres TE spécifiques, est implicitement lié par les TA Vente et Livraison, hérités du TE Article. Il pourrait, en plus, être lié par un TA Réparation à un TE Service après-vente (tel type d'articles de Hi-Fi est réparé par tel Service après-vente). Ce dernier TA n'est défini alors que pour les articles du TE Article Hi-Fi.
Un diagramme plus précis pour l'exemple hypermarché est donc:



4 Description d'un Schéma EA

4.1 Identifiant

L'identifiant d'une TE (TA) est un ensemble **minimum** d'attributs tel qu'il n'existe pas deux occurrences de TE (TA) qui ont la même valeur pour ces attributs.

Exemple : le TE Etudiant c'est NI, pour le TE MODULE c'est l'intitulé et le niveau.

4.2 Description TE et TA

Un TE est décrit par:

- le nom du type d'entité;
- le nom du (ou des) type(s) d'entité sur-type de ce type d'entité, s'il en existe;
- une définition libre (commentaire) précisant la population exacte du type d'entité;
- la description des attributs du TE;
- les identifiants du TE.

Exemple : Description du TE EMPLOYE :

Nom : Employé

sur-types: / ;

Définition : « toute personne qui perçoit un salaire dans une entreprise »

Attributs : nom, prénom, date de naissance, NSS, adresse, salaire

Identifiants : NSS

Un TA est décrit par :

- le nom du TA;
- une définition libre (commentaire) précisant la population exacte du type d'association;
- les noms des TE participant au TA, avec le nom du rôle les associant au TA. En pratique, le nom de rôle n'est obligatoire que pour les associations cycliques;
- pour chaque rôle, ses **cardinalités**: c'est une information supplémentaire exprimant la règle de participation des entités dans les associations (au niveau des occurrences). Les cardinalités consistent en deux nombres, min et max, spécifiant le nombre minimal et le nombre maximal d'occurrences du TA qui peuvent, à un instant donné, lier par ce rôle une occurrence quelconque du TE en question; min et max sont deux entiers tels que max≥min, min≥0, max≥1.
- les attributs du TA , s'il en existe;
- les identifiants du TA, s'il en existe.

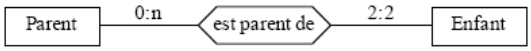


Fig 2.4 Diagramme EA avec cardinalité

4.3 Attributs

On distingue différents types d'attributs :

attribut simple: un attribut qui n'est pas décomposé en d'autres attributs: ses valeurs sont atomiques. Un domaine lui est associé. Exemple: salaire, téléphones

attribut complexe: un attribut qui est décomposé en d'autres attributs: ses valeurs sont des valeurs composées. Exemple: adresse, composé de: rue, ville, code postal

attribut monovalué: un attribut qui ne peut prendre qu'une seule valeur par occurrence. Exemple: nom, date de naissance

attribut multivalué: un attribut qui peut prendre plusieurs valeurs par occurrence. Exemple: prénoms, téléphones

attribut obligatoire: un attribut qui doit prendre une valeur au moins par occurrence. Exemple: nom, prénoms

attribut facultatif: un attribut qui peut ne pas prendre de valeur dans une occurrence. Exemple: salaire, téléphones

5. Contraintes d'intégrité

Les concepts d'entité, d'association, d'attribut et de sous-type ne suffisent pas à décrire tout ce qui caractérise les données d'un schéma EA. Des règles, définissant les états (ou transitions d'état) possibles de la base de données et qui ne peuvent pas être décrites avec les concepts du modèle sont ajoutées. On parle alors de **contraintes d'intégrité**. Si les valeurs de la base de données ne satisfont pas ces contraintes, il y a une "erreur" dans la base de données; on dit que la base de données est **incohérente**.

Exemple :

- si mois ∈ {4, 6, 9, 11} alors jour∈ [1:30] , sinon si mois=2 alors jour∈ [1:29] sinon jour ∈ [1:31] ;
- dans la relation parent-enfant, l'attribut nombre d'enfants du TE parent doit être égal au nombre d'occurrences du TA « est parent de » qui lient ce parent.

Contraintes d'intégrité sur les généralisations/spécialisations

Dans une hiérarchie de généralisation/spécialisation, il est fréquent de trouver des contraintes d'intégrité décrivant le partage de population entre les sous-types d'un même sur-type:

- contrainte de couverture, pour spécifier que l'union des populations de certains TE spécifiques d'un même TE générique est égale à la population du TE générique;
- contrainte de disjonction, pour spécifier que les populations de certains TE spécifiques d'un même TE générique n'ont aucune occurrence en commun;
- contrainte de partition, pour spécifier que la population d'un TE générique se distribue complètement et sans intersection entre certains de ses TE spécifiques (partition = couverture + disjonction sur les mêmes TE spécifiques).

Par exemple, dans la base de l'hypermarché, les trois sous-types de Articles sont disjoints: un article alimentaire n'est jamais un article Hi-Fi et vice-versa.
Un objet du monde réel peut se transformer et changer de représentation. Par exemple un étudiant de première année devient étudiant de seconde année, un étudiant devient assistant…

En conclusion:

Un schéma conceptuel entité association est un ensemble de descriptions de types d'entité et de types d'association (avec leurs attributs et les liens de généralisation entre types d'entité), et des contraintes d'intégrité (CI) associées:

schéma conceptuel EA = ({TE}, {TA}, {CI})

Concepts du modèle relationnel

1. Introduction

Le modèle relationnel a été défini par E.F Codd dans les années 70 et de nombreux chercheurs ont contribué à son développement. Les premiers SGBD bâtis sur ce modèle ont été SQL/DS et DB2 de IBM, d'où est né le langage de manipulation de bases relationnelles, SQL (Structured Query Language).

Simplicité de la structure des données :

Une base relationnelle est composée de tables et est perçue par l'utilisateur comme un ensemble de tables et rien d'autre. Dans une table, une ligne correspond à un enregistrement et une colonne à un champ de cet enregistrement.

2. Définitions de base

- Le modèle relationnel est fondé sur la *théorie* ***mathématique des relations*** qui se construit à partir de la ***théorie des ensembles.***
- Il représente une base de données comme une collection de ***relations***
- Informellement, une relation ressemble à une table de valeurs, ou, à un fichier plat d'articles (enregistrements).

Etudiant	N°Etud	Nom	Prénom	Age
	345	Bouali	Lotfi	22
	123	Kebir	Mohamed	21
	101	Chaafi	Imene	19
	456	Mehdaoui	Sara	23

2.1. Domaine

- Un domaine est un ensemble de valeurs caractérisé par un nom et qui corresponde normalement, dans le MR, à un type élémentaire. C'est l'ensemble de valeurs que peut prendre un attribut

Exemples:

- ENTIER
- REEL
- CHAINES DE CARACTERES (String)
- EUROS
- SALAIRE = {4 000..100 000}
- COULEUR= {BLEU, BLANC, ROUGE}
- POINT = {(X:REEL,Y:REEL)}
- TRIANGLE = {(P1:POINT,P2:POINT,P3:POINT)}

- Une information placée dans une colonne d'une table possède forcément un domaine construit sur un type de base.
- La valeur spéciale NULL appartient à chaque domaine

2.2. Attribut et schéma dôune relation

- Un attribut est une *colonne* dôune relation caractérisée par un nom. Elle identifie une catégorie dôinformations dont le type est fixée.
- Un attribut sert à nommer une colonne et à la référencer pour effectuer des opérations (interrogation, mise à jour).
- Une relation est définie par :
  - Son nom
  - Une liste de couples (nom dôattribut : domaine)
  - Son (ses) identifiant(s)
  - Sa définition (phrase en français)
  - Les trois premières informations constituent le ***schéma de la relation.***

- La forme générale **d'un schéma de relation** est *R*(*A1:D1*, é , *An:Dn*) avec *R* le nom de la relation, *Ai* le nom des attributs et *Di* le nom des domaines respectifs, *n* est le **degré** ou lô***arité*** de la relation.

Exemple : : schéma de la relation Etudiant,
Etudiant(N°Etud : Dnum, Nom : Dnom, Prénom : Dnom, Age : Dâge)

2.3. Tuple

- Tuple est une Ligne dôune relation correspondant à un enregistrement. Un tuple est une liste de *n* valeur *(v1j, é , vnj) où chaque valeur vij est la valeur dôun attribut de domaine Di pour la ligne j.*

Exemple : (456, " Mehdaoui", " Sara",23)

2.4 Extension d'un relation

Une ***extension*** (***état*** ou ***instance*** ou ***population***) *r* dôun schéma de relation *R*(*A1:D1*, é , *An:Dn*) , notée r(R), est un ensemble de tuples *r={t1, t2,é ,tm}*

Lôextension dôune relation est variable au cours de la vie dôune base de données

Exemple :
Soit le schéma personne **:** Personne (Nom:Chaîne, Age:Entier, Marié:Booléen)
Et dôextension : {{Nom=ôôHakimôô, Age=23, Marié = Faux},{Nom=ôôSaidaôô, Age=28, Marié =Vrai}}

Propriétes dôune relation

- Ordre des attributs peut importe.
- Chaque tuple distinct ; il nôy a pas des tuples dupliqués.
- Théoriquement, lôordre des tuples nôa pas dôimportance.

Graphe dûne relation

- Relation binaire R(A1,A2)
- Une relation n-aire est une généralisation à n dimensions

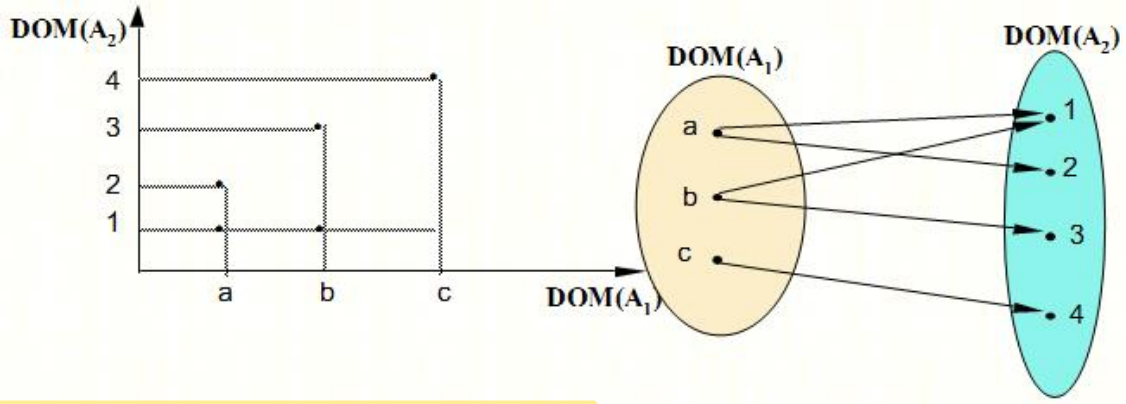


Schéma dûne base de données relationnelles

- un ensemble S de schémas de relations appartenant à la même base de données. S est le nom de la base de données.
- S = {R1, R2, ..., Rn}

3.Contraintes dûintégrité

- Les contraintes dûintégrité sont les assertions (affirmations) qui doivent être vérifiées par les données contenues dans une base (instances des relations)
- Il y a deux type de contraintes
- Les contraintes structurelles qui sont inhérentes au modèle de données (nécessaire à sa mise en î uvre)
- Les contraintes de comportement propres au schéma particulier dûne application (contraintes sémantiques)
- Les contraintes structurelles sont :
- Unicité de clé
- Contraintes de référence
- Contraintes dûentité
- Contraintes de domaine

3.1. Unicité de clé

- Super-clé de R: un ensemble dûattributs K de R tel que , il nûexiste pas deux tuples t1, t2 dans nûimporte quelle instance valide r(R) ayant la même valeur de K. Cûest-à-dire, $\forall$ t1 et t2 dans r(R), t1[k] $\neq$ t2[k].
- Clé de R: une super-clé minimale; cûest-à-dire, une super-clé K telle que la suppression de nûimporte quel attribut de K abouti à un ensemble dûattributs qui nûest pas une super-clé.

Exemple:

- Le schéma de relation Voiture:
  Voiture(N°Immatriculation, N°Serie, Marque, Modèle, Année)
- possède deux clés Clé1 = {N°Immatriculation}, Clé2 = {N°Série}, qui sont des super-clés, alors que {N°Serie, Marque} est une super-clé mais n'est pas une clé
- Si une relation possède plusieurs clés candidates (candidate keys) , on en choisit en générale arbitrairement qui est appelée clé primaire (primary key)

Exemple :

- N°Immatriculation peut constituer une clé primaire pour la relation Voiture
- Les attributs formant la clés primaire sont souligné dans le schéma d'une relation

3.2. Contraintes de Références (CR) ou Referential Constraints

- Une CR est une contrainte impliquant deux relation (ne sont pas forcément distinctes), R1 et R2, qui utilisée pour spécifier une association (relationship) qui porte sur les tuples des deux relations: la relation référençante R1 et la relation référencée R2

- Les tuples dans la relation référençante R1 possèdent des attributs FK (appelées clé étrangère ou foreign key) qui référencent les attributs clé primaire PK de la relation référencée R2
- FK dans R1, et PK dans R2 doivent avoir le/les même(s) domaine(s)
- Un tuple t1 dans R1 référence un tuple t2 dans R2 $\Rightarrow$ t1[FK] = t2[PK].
- Une contrainte d'intégrité référentielle peut être illustrée dans un schéma de base de données relationnelle comme un arc partant de R1.FK à R2.PK
- La valeur dans la/les colonne(s) clé étrangère FK de la relation référençante R1peut être :
- Une valeur qui apparaissent comme une valeur clé primaire PK dans la relation référencée R2
- Une valeur NULL, FK dans R1 ne doit pas faire partie de sa clé primaire

3.3. Contraintes d'entité

- Si lors de l'insertion de tuples dans une relation, il y a un attribut qui est inconnu (ex., email d'un étudiant) ou non applicable (ex., situation vis-à-vis le service national pour les femmes), alors, conventionnellement, une valeur spéciale, appelée ***nulle*** (***NULL***), est introduite pour représenter la valeur de cet attribut.
- Les attributs clé primaire PK (identifiant)de chaque schéma de relation R dans S ne peuvent pas avoir des valeurs nulles dans un tuple de r(R), c'est-à-dire $\forall t \in r(R) \Rightarrow t(PK) \neq NULL$
- Les autres attributs de R peuvent être contraints à prendre des valeurs non nulles même s'ils ne font partie de la clé primaire de cette relation.
- La ***contrainte d'entité*** (***Entity constraint***) impose que toute relation possède une clé primaire et que tout attribut participant à cette clé primaire soit non nul.

3.4. Contraintes de domaine (Domain constraint)

- Une contrainte de domaine est une contrainte d'intégrité imposant qu'un attribut d'une relation doit comporter des valeurs vérifiant une assertion logique (appartenance à une plage de valeurs ou à une liste de valeurs)

Exemple : Colueur $\in$ {Rouge, Blanc, Noir} , $15000 \leq$ Salaire ≤ 1000000

- La non-nullité d'un attribut peut être considérée comme une contrainte de domaine

3.5. Contraintes d'intégrité sémantique

- Ces contraintes sont basées les sémantiques de l'application par le modèle relationnel.

Exemple : le salaire d'un employé doit être inférieur à tous les salaires de ses supérieurs

- Un langage de spécification de contraintes peut être utilisé pour exprimer ces contraintes
- SQL-99 permet la définition des déclencheurs (triggers) et assertions (assertions) pour spécifier certains types de contraintes sémantiques.

Chapitre IV : LôAlgèbre relationnelle

- ÅUne algèbre est un ensemble dôopérateurs de base, formellement définis, qui peuvent être combinés pour construire des expressions algébriques
- ÅUne algèbre est dite fermée si le résultat de tout opérateur est du même type que les opérandes (ce qui est indispensable pour construire des expressions)
- ÅComplétude: toute manipulation pouvant être souhaitée par les utilisateurs devrait pouvoir être exprimable par une expression algébrique

Algèbre relationnelle

- Å Lô***algèbre relationnelle*** est une collection dôopérations formelles qui agissent sur des relations et produisent les relations en résultats
- Å Cêst la force du modèle relationnel
- Å Il y a deux types dôopérations de base :
  - ï Les opérations ensemblistes traditionnelles (une relation doit être traité comme un ensemble). Ces opérations sont des opérations binaires (à partir de deux relations elles en construisent une troisième) (lôunion, l'intersection,la différence et le produit cartésien)
  - ï Les opérations spécifiques sont les opérations unaires de projection et restriction qui, à partir dôune relation , en construisent une autre, et lôopération binaire de jointure

1. Les opérations ensemblistes

1.1. Union : opération portant sur deux relations ***de même schéma R et S*** consistant à construire une relation de même schéma T ayant pour tuples ceux appartenant à R ou S ou aux deux relations.

Notations :

- ï $R \cup T$
- ï UNION (R, S)

Algèbre relationnelle

1.1. Union

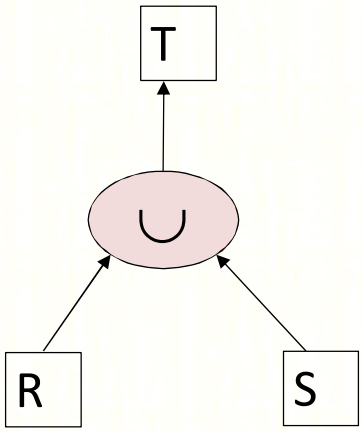
A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
α	2
β	1
β	3



T = R ∪ S

Algèbre relationnelle

1.2. Différence : opération portant sur deux relations de même schéma R et S consistant à construire une relation de même schéma T ayant pour tuples ceux appartenant à R et nôappartenant pas à S.

Notations :

- R - S
- DIFFERENCE (R, S)
- MINUS (R, S)

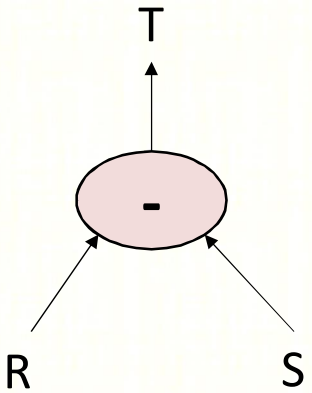
A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
β	1



T = R - S

Algèbre relationnelle

1.3. Produit cartésien : opération portant sur deux relations R et S, consistant à construire une relation T ayant pour schéma la concaténation de ceux des relations opérandes et pour tuples les combinaisons des tuples des relations opérandes.

Notations :

- R × S
- PRODUCT (R, S)

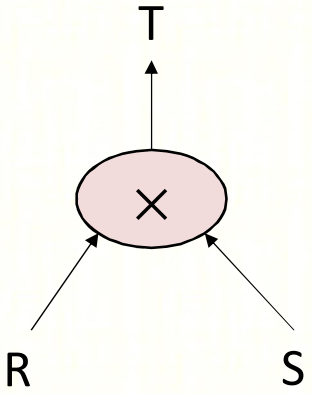
A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b



T = R ×S

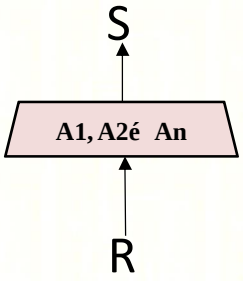
Algèbre relationnelle

2. Les opérations spécifiques

2.1. Projection: opération sur une relation R, consistant à composer une relation S en enlevant à la relation initiale tous les attributs non mentionnés en opérandes (aussi bien au niveau du schéma que des tuples) et en éliminant les tuples en double qui sont conservés une seule fois.

Notations : *Attribut*$_i$, *Attribut*$_j$***é*** *Attribut*$_m$ ***:*** sont les attributs de projection

- $\Pi_{Attribut_i, Attribut_j ... Attribut_m}(R)$



A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

R

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

$S=\Pi_{A,C}(R)$

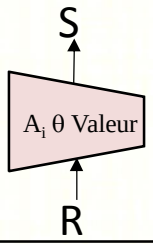
Algèbre relationnelle

2.2. Restriction ou sélection : opération sur une relation R produisant une relation S de même schéma, mais comportant les seuls tuples qui vérifient la condition précisée en argument.

- Les conditions possibles sont des formules dans le calcul propositionnel constituées de termes connectés par : $\wedge$ (**et**), $\vee$ (**ou**), $\neg$ (**non**). Chaque terme est du type :
  - <Attribut> <Opérateur> <Attribut> ou <Constante>. Tel que <Opérateur> $\in \{=, >, \geq, <, \leq, \neq\}$,

Notations :

- $\sigma_{condition}(R)$



$\theta \in \{=, >, \geq, <, \leq, \neq\}$

R

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

A	B	C	D
α	α	1	7
β	β	23	10

$S = \sigma_{A=B \text{ and } D > 5}(R)$

Algèbre relationnelle

2.3. Jointure : Opération consistant à rapprocher selon une condition les tuples de deux relations R et S afin de former une troisième relation T qui contient tous les tuples obtenus en concaténant un tuple de R et un tuple de S vérifiant la condition de rapprochement

- La condition de jointure est du type : ***<Attribut1> <opérateur> <Attribut2>***, tel que Attribut1 $\in$ R et Attribut2 $\in$ S
- Si <Opérateur> est = alors on parle de l'***équi-jointure*** qui est une véritable composition de relation au sens mathématique
- Si <Operateur> $\in \{>, \geq, <, \leq, \neq\}$ alors on parle de l'***inéqui-jointure***
- Dans l'équi-jointure les deux attributs égaux apparaissent chacun dans le résultat : il y a donc duplication d'une même valeur dans chaque tuple.

Jointure Naturelle

Jointure naturelle : Opération consistant à rapprocher selon une condition les tuples de deux relations R et S afin de former une troisième relation T dont les attributs sont lûunion des attributs de R et S, et dont les tuples sont obtenus en composant un tuple de R et un tuple de S **ayant mêmes valeurs pour les attributs de même nom**.

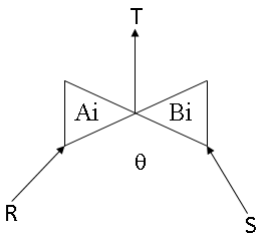
Notations :

ï JOIN(R, S, Condition)

Å **PRECONDITION: Au moins un attributs de même nom en commun entre les deux relations**

Algèbre relationnelle

2.4. Jointure



$\theta \in \{=, >, \geq, <, \leq, \neq\}$

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

T=R ⋈ S

Théta Jointure

Theta-jointure ⋈[p]

- but: créer toutes les combinaisons significatives entre tuples de deux relations
  - significatif = critère de combinaison explicitement défini en paramètre de l'opération
- précondition: les deux relations n'ont pas d'attribut de même nom
- exemple : R ⋈ [B ≠ C] S

R

A	B
a	b
b	c
c	b

S

C	D	E
b	c	d
b	a	b
c	a	c

R⋈ [B ≠ C] S

A	B	C	D	E
a	b	c	a	c
b	c	b	c	d
b	c	b	a	b
c	b	c	a	c

Algèbre relationnelle

3. Les opérations dérivées

3.1. Intersection : opération portant sur deux relations de même schéma R et S consistant à construire une relation de même schéma T ayant pour tuples ceux appartenant à la fois à R et S

Notations :

- R ∩ S
- INTERSECT(R, S)
- AND (R, S)

Lôopération dôintersection sôobtient à partir de lôopération de différence :

- R ∩ S = R − (R-S)
- R ∩ S=S-(S-R)

PRECONDITION:

- R et S doivent être de même schéma

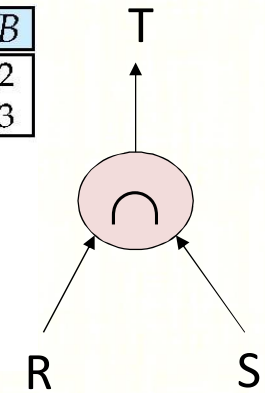
A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	2



T = R ∩ S

Algèbre relationnelle

3.2. Division : Opération consistant à construire le quotient de la relation

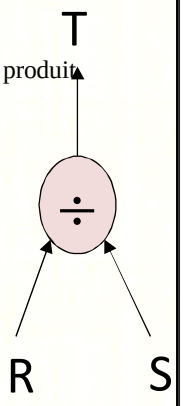
R (A_1, A_2é A_p, A_{p+1}é A_n) par la relation S (A_{p+1}é A_n) comme la relation Q (A_1, A_2é A_p) dont les tuples sont ceux concaténés à tout tuple de S donnent un tuple de R

Notations :

- ï R ÷ S
- ï DIVISION (R, S)

Å Lôopération de division sôobtient à partir de lôopération de différence, du produit cartésien et de la projection:

- ï R ÷ S =R1-R2
- ï R1=$\Pi_{A1, A2é\ Ap}$(R)
- ï R2= $\Pi_{A1, A2é\ Ap}$((R1×S)-R)



Algèbre relationnelle

A	B	C
a1	b1	c1
a1	b2	c2
a2	b1	c1
a2	b2	c1
a3	b3	c1

R

B	C
b1	c1
b2	c1

S

A
a2

Q=R÷S

Algèbre relationnelle

3.3. Complément : Ensemble des tuples du produit cartésien des domaines des attributs dôune relation R nôappartenant pas à cette relation. Le complément suppose a priori que les domaines sont finis (sinon on obtient des relations infinies).

Notations :

- ï $\neg R$
- ï -R

Å Opération peu utilisée, car elle permet de générer des tuples qui ne sont pas dans la base, en général très nombreux.

Å Si lôon note par $\times D_i$ le produit cartésien des domaines, le complément dune relation R est obtenu à partir de la différence comme suit :

✖ $\neg R = \times D_i - R$

Algèbre relationnelle

3.3. Complément

Exemple :

Domaines:

Å A={a1, a2, a3}

Å B= {b1, b2, b3}

R	A	B
	a1	b1
	a1	b2
	a2	b1
	a3	b2
	a3	b3

¬R	A	B
	a1	b3
	a2	b2
	a2	b3
	a3	b1

Algèbre relationnelle

3.4. Nommage : opération permettant de nommer, et par conséquent de sôy référer à, les résultats des expressions en algèbre relationnelle.

- Permet de référencer une relation par plus dôun nom.
- On renomme les attributs dôune relation

Exemple:

- Soit R (A1,A2,A3)
- R1=α (A1:B1, A2,:B2,A3:B3) R : crée une nouvelle relation R1 en renommant tous les attributs de R
- On peut renommer quôun sous ensemble des attributs dôune relation:
- Exemple:
  - R2 = α (A1:B1,A3:C1)R

Algèbre relationnelle

3.5. Jointure externe (External Join) : opération générant une relation T à partir de deux relations R et S, par jointure de ces deux relations et ajout de tuples de relations R et S ne participant pas à la jointure, avec des valeurs nulles pour les attributs de lôautre relation

Notations :

- T=R⟕S, ou T=R⟖S
- T=EXT-JOIN(R, S)

- La jointure externe permet de conserver les tuples sans correspondant avec des valeurs nulles associées quand nécessaire.
- Opération très utile en pratique, en particulier pour composer des vues sans perte dôinformations (ex., on peut joindre des tables CLIENTS et COMMANDES sur un numéro de client commun, en gardant les clients sans commande et les commandes sans client associé
- On en distingue deux types :
  - La ***jointure externe droite*** qui garde seulement les tuples sans correspondant de la relation droite. Celle-ci est notée ⟕, ⟖, ou REXT-JOIN
  - ***La jointure externe gauche*** qui garde seulement les tuples sans correspondant de la relation gauche. Celle-ci est notée ⟕, ⟖, ou LEXT-JOIN

Algèbre relationnelle

3.5. Jointure externe (External Join) :

R	A	B
	a1	b1
	a1	b2
	a2	b1
	a1	b4

S	B	C
	b1	c1
	b3	c1
	b5	c2

R ⟕ S

T	A	B	C
	a1	b1	c1
	a1	b2	-
	a2	b1	c1
	a1	b4	-
	-	b3	c1
	-	b5	c2

R ⟖ S

T	A	B	C
	a1	b1	c1
	a1	b2	-
	a2	b1	c1
	a1	b4	-

R ⟗ S

T	A	B	C
	a1	b1	c1
	a2	b1	c1
	-	b3	c1
	-	b5	c2

Algèbre relationnelle

4. Les expressions de lôalgèbre relationnelle

- A partir des opérations de lôalgèbre relationnelle, il est possible de composer un langage dôinterrogation de bases de données.
- Chaque requête (question) peut être représentée par arbre dôopérateurs relationnels
- Le paraphrasage en anglais de telles expressions est à la base du langage SQL.

4.1. Langage algébrique

- La réponse à la plupart des questions que lôon peut poser à une base de données relationnelle peut sôélaborer en utilisant des expression dôopérations de lôalgèbre relationnelle.

Algèbre relationnelle

4.1. Langage algébrique

Å Soit le schéma relationnel dôune base de données touristiques (les clés sont soulignées)

```
STATION (NUMSTA, NOMSTAT, ALTITUDE, REGION)
HÔTEL (NUMHOT, NOMHOT, CATEGORIE, NUMSTA)
CLIENT (NUMCLI, NOMCLI, ADRCLI, TELCLI)
CHAMBRE (NUMCH, NUMHOT, NBLITS)
RESERVER (NUMCLI, NUMCH, NUMHOT, DATEDEB, DATEFIN,
NBPERS)
```

Exemple de requêtes

Å Ces requêtes peuvent être exprimées comme des expressions dôopérations, ou comme des opérations successive appliquées sur des relations intermédiaires ou de base, générant des relations intermédiaires.

Å ***(Q1) Donner les noms de station de la région Haute-Normandie et dôaltitude > 1000 M :***

- $R1 = \sigma_{REGION=" Haute\text{-}Normandie"}(STATION)$
- $R2 = \sigma_{ALTITUDE>1000}(STATION)$
- $R3 = R1 \cap R2$
- $RESULTAT = \Pi_{NOMSTA}(R3)$

Algèbre relationnelle

Å La requête (Q1) peut simplement sôexprimer comme suit

$$\prod_{NOMSTA}(\sigma_{\substack{REGION="HAUTE-NORMANDIE" \\ AND\ ALTITUDE>1000}}(STATION))$$

Å ***(Q2) Donner les numéros et noms des hôtels de la station de Paris ou dôAlsace :***

- $R1 = \sigma_{REGION=" Paris"}(STATION)$
- $R2 = \sigma_{REGION=" Alsace"}(STATION)$
- $R3 = R1 \cup R2$
- $R4 = R3 \bowtie HÔTEL$
- $RESULTAT = \Pi_{NUMHOT,NOMHOT}(R4)$

Ou tout simplement

$$\prod_{NUMHOT,NOMHOT}(\sigma_{\substack{REGION="PARIS" \\ OR \\ REGION="ALSACE"}}(STATION) \bowtie HÔTEL)$$

Å ***(Q3) Donner les noms et adresses des clients ayant réservé plus de 5 places dans un hôtel de 3 étoiles avec le nom de cet hôtel :***

Algèbre relationnelle

- $R1 = \sigma_{NBPERS>5}(RESERVER)$
- $R2 = \sigma_{CATEGORIE = "****"}(HÔTEL)$
- $R3 = \Pi_{NUMCLI,NUMHOT}(R1)$
- $R4 = CLIENT \bowtie R3$
- $R5 = R2 \bowtie R3$
- $R6 = R4 \bowtie R5$
- RESULTAT = $\Pi_{NOMCLI,ADRCLI,NOMHOT}$(R6)

ou tout simplement

$$\prod_{NOMCLI,\ ADRCLI,\ NOMHOT} (((\prod_{NUMCLI,\ NUMHOT} (\sigma_{NBPERS\ >5}(RESERVER\ ))) \bowtie (\sigma_{CATEGORIE\ ="****"}(HÔTEL\ ))) \bowtie CLIENT\ )$$

- ***(Q4) Donner le nom des client nâayant réservé que dans des hôtels 4 étoiles :***
  - $R1 = CLIENT \bowtie RESERVER \bowtie HÔTEL$
  - $R2 = \sigma_{CATEGORIE\ =" ****"}$ (R1)
  - $R3 = \Pi_{NOMCLI}(R2)$
  - $R4 = \sigma_{CATEGORIE\ \neq" ****"}$ (R1)
  - $R5 = \Pi_{NOMCLI}(R4)$, $RESULTAT = R3 - R5$

Algèbre relationnelle

Ou tout simplement

$$\prod_{NOMCLI} (\sigma_{CATEGORIE\ ="****"}(HÔTEL\ ) \bowtie RSERVER \bowtie CLIENT\ ) -$$
$$\prod_{NOMCLI} (\sigma_{CATEGORIE\ \neq"****"}(HÔTEL\ ) \bowtie RSERVER \bowtie CLIENT\ )$$

- Si lôon accepte les relations constantes, lôalgèbre relationnelle permet aussi dôeffectuer les mise à jour . Ces opérations sont exprimées en utilisant lôopérateur dôaffectation (←)

Algèbre relationnelle

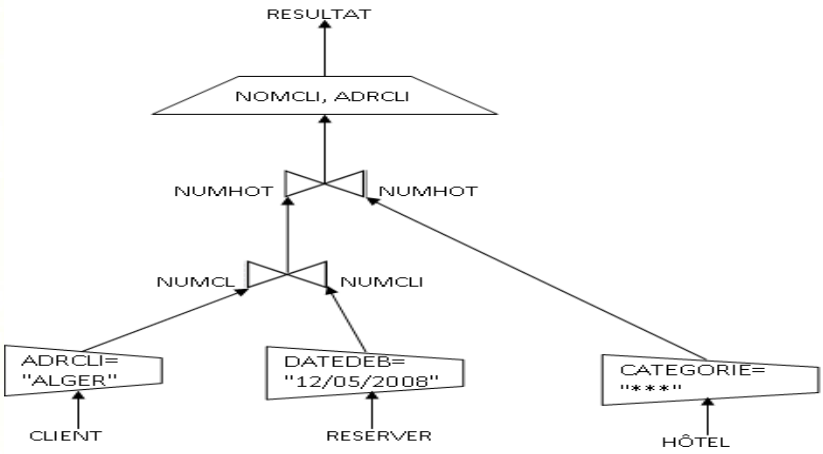
4.2. Arbre algébrique

- Une question exprimée sous forme d'un programme d'opérations de l'algèbre relationnelle peut être représentée par un ***arbre relationnel***.
- L'arbre relationnel est un arbre dont les nî uds correspondent aux représentations graphiques des opérations de l'algèbre relationnelle et les arcs à des relations temporaires ou de base représentant des flots de données entre opérations.
- Plusieurs arbre équivalents peuvent être déduits d'un arbre donné à l'aide de règles de transformation simples, telles que la permutation des jointures et restrictions, la permutation des projections et des jointures, le regroupement des intersections sur une même relation, etc.
- Ces transformations sont à la base des techniques d'optimisation de question
- La composition d'opérations de l'algèbre relationnelle ne nécessite pas toujours d'attendre le résultat de l'opération précédente pour exécuter l'opération suivante.
- Les opérations de restrictions, jointures et projections peuvent ainsi être exécutées par des algorithmes à flots de données.
- Les opérateurs qui se succèdent dans un arbre peuvent être exécutés en parallèle par des algorithmes "pipe-line"
- Les opérateurs figurants sur des branches distinctes d'un arbre peuvent aussi être exécutés en parallèle de manière indépendante.
- La représentation par arbre algébrique met ainsi en évidence les possibilités de parallélisme et les enchaînements nécessaires. Ces propriétés sont importantes pour optimiser les questions dans des contextes parallèles.

Algèbre relationnelle

Exemple :

La question "Noms et Adresses des clients habitant Alger ayant effectué une réservation le 12/05/2008 dans un hôtel de trois étoiles" peut être exprimée à l'aide de l'arbre suivant :



Le langage SQL

1

SQL: Structured Query Language

- ***SQL*** (Structured Query Language) créé dans les années 70 est dérivé de l'algèbre relationnelle et de ***SEQUEL*** (Structured English QUEry Language) et ***SQUARE*** (Specifying Queries As Relational Expressions)
- SEQUEL et *SQUARE* ont été conçus et implémentés par IBM à San Jose Research Laboratory comme l'interface pour un SGBD relationnel expérimental appelé R, précurseur de plusieurs SGBD relationnels, notamment du système SGBD DB2 .
- C'est un langage non procédural dans sa formulation
- C'est un paraphrasage en anglais des expressions de l'algèbre relationnelle.
- Il a été intégré à SQL/DS, DB2, puis ORACLE, INGRES, é
- Actuellement, c'est le langage standard des SGBDs relationnels commerciaux
- SQL existe en trois versions normalisées, du simple au complexe. Elles sont les résultats d'un effort joint de l'ANSI (American National Standards Institute) et de l'ISO (International Standards Organization).
- ***SQL1 ou SQL-86 version minimale*** : permet l'expression des requêtes composées d'opérations de l'algèbre relationnelle et d'agrégats, et comporte les fonctionnalités de définition, de recherche et de mise à jour..
- **SQL1 ou SQL-89 addendum (intégrité)** : extension de SQL-86 avec des fonctions de contrôle (gestion d'intégrité) qui sont nécessaires pour programmer des applications transactionnelles
- ***SQL2 ou SQL-92 langage complet à 3 niveaux*** : ***entrée*** (amélioration de SQL1), ***intermédiaire*** et ***complet***. Ces deux dernier niveaux supportent totalement le modèle relationnel avec des domaines variés, tels date et temps.

2

SQL: Structured Query Language

- ***SQL3*** –***SQL: 99***, ***SQL: 2003***, ***SQL: 2006*** ***et*** ***SQL: 2008***– apportent des extension pour la gestion de l'orientée objet, des règles, XML, datamining, données spatiales, données temporelles, data warehousing, traitement analytique enligne, données multimédias, etc.
- La plupart des systèmes SGBDs relationnels supportent SQL1 et la majorité des spécifications de SQL2, et des ensembles variables de caractéristiques définies par les derniers standards SQL, et offrent leurs propres caractéristiques spéciales.
- Le langage SQL comporte trois facettes fortement intégrées peu importe le SGBD relationnel :
  1. ***DDL*** (***Data Definition Language***), pour la spécification des information concernant les relations, telles que :
     - Le schéma de chaque relation;
     - Le domaine des values associées à chaque attribut;
     - Les contraintes d'intégrité;
     - D'autre informations comme l'ensemble d'indxes à maintenir pour chaque relation, les information d'autorisation et de sécurité pour chaque relation.
  2. ***DML*** (***Data Manipulation Language***), pour la manipulation des tables et plus précisément les manipulations des tuples de relations.
  3. ***DCL*** (***Data Control Language***), pour gérer la définition physique des accès (index), la spécification des fichiers physiques et la validation des opérations exécutées dans un contexte multiposte.

3

SQL: Structured Query Language

10.1. Notations utilisées pour la présentation de la syntaxe de SQL

- Basées sur ***BNF*** **(*Backus-Naur Form*)** avec les extensions suivantes:
  - Les crochet ([]) indiquent des éléments optionnels;
  - Les pointillés (é) suivent un élément qui peut être répété plusieurs fois;
  - Les accolades ({}) groupent comme un seul élément une séquence d'élément;
  - Un exposant plus ($^{+}$) indique que l'élément qui précède peut être répété n fois (n >0), chaque occurrence est séparée de la précédente par une virgule;
  - Un exposant (*) indique que l'élément qui précède peut être répété n fois (n ≥0), chaque occurrence est séparée de la précédente par une virgule.

4

SQL: Structured Query Language

Manipulation de données

Recherche de données

- ***SELECT*** est la requête de recherche de données qui est à la base de SQL
- L'objectif de la commande **SELECT** est de rechercher et afficher les données à partir d'une ou plusieurs tables ou vues d'une base de données
- C'est une commande extrêmement puissante qui est capable d'effectuer, en une seule instruction, l'équivalent des opérations de restriction, projection et jointure de l'algèbre relationnelle.
- C'est la commande de SQL la plus fréquemment utilisée
- Sa syntaxe est la suivante :

SELECT [DISTINCT | ALL] *{* | [<expression de colonne>* ***[AS*** *<nouveau nom>]] [, . . .]}*
FROM *T<nom de table>* ***[***<alias>***]*** *[, . . .]*
[WHERE *<condition de recherche>]*
[GROUP BY *<liste de colonnes>****[HAVING*** *<condition de recherche>]***
[ORDER BY *<liste de colonnes>]*

5

SQL: Structured Query Language

Recherche de données

- *<expression de colonne>* représente une *spécification de colonne* ou une expression de valeurs (composée avec les opérateurs +, -, * et /), éventuellement parenthésée, de spécification de constantes (constantes, une variable de programme ou le nom de l'usager (mot clé ***USER***))
- **<nom de table>** est le nom dûne table ou d'une vue de la base de données que l'on ait le droit dôy accéder
- **<alias>** est une abréviation optionnelle pour *<nom de table>*
- La séquence de traitement dans une commande ***SELECT*** est comme suit :
- ***FROM*** spécifie la table ou les tables à utiliser
- ***WHERE*** filtre les lignes selon une certaine condition (critères) de recherche
- ***GROUP BY*** forme des groupes de lignes avec la même valeur de colonne
- ***HAVING*** filtre les groupes conformément à une certaine condition de recherche
- **SELECT** spécifie les colonnes à apparaître dans le résultat
- ***ORDER BY*** spécifie lôordre du résultat
- Lôordre des différentes clauses dans une commande ***SELECT*** ne peut être changé.
- Les deux clauses obligatoires sont les deux premières : ***SELECT*** et ***FROM*** ; les autres sont optionnelles.
- Lôopération ***SELECT*** est ***fermée*** (i.e., le résultat dûne requête sur une table est une autre table)
- Il y a plusieurs variantes de la commande ***SELECT***

6

SQL: Structured Query Language

Recherche de données

- Une spécification de colonne désigne le nom d'une colonne précédé d'un désignateur de relation éventuel (nom de table ou variable), comme suit :

  <spécification de colonne> ::= [<désignation>.] <nom de colonne>

  <désignation> ::= <nom de table> | <nom de variable>
- Une condition de recherche définit un critère, qui est appliqué à une ligne, est vrai, faux ou inconnu, selon le résultat de l'application d'opérateurs booléens (ET, OU, NOT) à des conditions élémentaires (appelées ***prédicats*** en SQL). L'expression booléenne de conditions élémentaires peut être parenthésée.
- Il y a plusieurs type de prédicats :
  - de comparaison d'un ***terme*** (spécification de colonne)à une constante ou à un autre ***terme*** l'aide des opérateurs {=, <, >, ≠, ≥, ≤}
  - de comparaison de texte (***LIKE***) permettant de tester si un terme de type chaîne de caractères contient une ou plusieurs sous-chaînes
  - d'intervalle (***BETWEEN***) permettant de tester si la valeur d'un terme est comprise entre la valeur de deux constante;
  - d'appartenance (***IN***) qui permet de tester si la valeur d'un terme appartient à une liste de constantes ou au résultat d'une sous-requête
  - de test de nullité (***IS NULL, IS NOT NULL***) qui permet de tester si un terme a une valeur convenue ***NULL*** ou non, signifiant que sa valeur est inconnue

7

SQL: Structured Query Language

Recherche de données

- Possibilité de blocs imbriqués (sous-question) par :
  - ***IN*** : permet de tester si la valeur d'un terme appartient au résultat (liste de valeurs) d'une sous-question
  - ***EXISTS*** : permet de tester si le résultat d'une sous-question n'est pas vide
  - ***NOT EXISTS*** : permet de tester si le résultat d'une sous-question est vide
  - ***ALL*** ("quel que soit") : permet de tester si la valeur d'un terme satisfait un opérateur de comparaison avec tous les éléments du résultat d'une sous-question
  - ***SOME*** ou ***ANY*** ("il existe") : permet de tester si la valeur d'un terme satisfait un opérateur de comparaison avec au moins un élément du résultat d'une sous-question

8

SQL: Structured Query Language

Recherche de données

Forme générale d'une condition de recherche

*<condition de recherche> ::= [**NOT**]*
<nom_colonne> θ constante | <nom_colonne>
*<nom_colonne> **LIKE** <modèle_de_chaîne>*
*<nom_colonne> **IN** <liste_de_valeurs>*
*<nom_colonne> θ (**ALL** | **ANY** | **SOME**) <liste_de_valeurs>*
***EXISTS** <liste_de_valeurs>*
*<nom_colonne> **BETWEEN** constante **AND** constante*
*<condition de recherche > **AND** | **OR** <condition de recherche >*

avec :

*θ **::=** > | < | = | <> | >= |<=*

- ***Remarque***: *<liste_de_valeurs>* peut être dynamiquement déterminée par une requête
- *<modèle_de_chaîne>* est une constante de chaîne de caractères pouvant contenir le caractères génériques "%" (signifie une quelconque sous-chaîne de caractères), ou "_" (soulignement) (signifie un quelconque caractère). Si l'un de ces deux caractères devrait être utilisé en tant que caractère constant dans la chaîne de caractères constante, alors il doit être précédé par un ***caractère d'échappement*** (**Escape Character**). Ce dernier est spécifié après la chaîne de caractères en utilisant le mot clé ***ESCAPE***

9

SQL: Structured Query Language

Recherche de données

- Traitement de chaines de caractères en SQL

<expression_caractère >::=
'chaîne_caractère'
<nom_colonne>
USER
***UPPER** (<expression_caractère>)*
***LOWER** (<expression_caractère)>*
***CHARACTER_LENGTH** (<expression_caractère>)*
***SUBSTRING** (<expression_caractère > **FROM** <début >**FOR** <longueur>)*
***POSITION** (expression_caractère **IN** <expression_caractère>)*
***CAST** (expression **AS** <type> | <domaine>)*
<expression_caractère > | | <expression_caractère >

- || est l'opérateur de concaténation de deux chaines

10

SQL: Structured Query Language

Squelette de relations

Soit le schéma de la base de données suivante:

Produit(NPRO,NOMP,QTE,Couleur)
Client(NOMC,Adresse,Tel,Fax)
Fournisseur(NOMF,Adresse,Tel,Fax)
Vente(NVEN,NOMC,NPRV,QTEV,Date)
Achat(NACH,Date,NPRA,QTEA,NOMF)

11

SQL: Structured Query Language

Recherche de données

Exemples de requêtes

- (Q1) : Lister les noms et les couleurs des produits

```
SELECT NOMP, COULEUR
FROM PRODUIT;
```

```
SELECT ALL NOMP, COULEUR
FROM PRODUIT;
```

- (Q2) : Lister toutes les informations des produits

```
SELECT NPRO, NOMP, QTES, COULEUR
FROM PRODUIT;
```

```
SELECT *
FROM PRODUIT;
```

- (Q3) : Lister les noms et les quantités en stock de tous les produits de couleur "Rouge"

```
SELECT NOMP, QTES
FROM PRODUI
WHERE COULEUR='Rouge';
```

- (Q4) : Lister toutes les couleurs (éventuellement en doubles) des différents produits dont la quantité > 10

```
SELECT COULEUR
FROM PRODUIT
WHERE QTES > 10;
```

12

SQL: Structured Query Language

Recherche de données

Exemples de requêtes

- (Q5) : Lister toutes les couleurs (sans doubles) des différents produits dont la quantité > 10

```
SELECT DISTINCT COULEUR
FROM PRODUIT
WHERE QTES > 10;
```

- (Q6) : Lister les désignations et les adresses de tous les clients n'ayant pas de numéro de fax

```
SELECT DESIGNATION, ADRESSE
FROM CLIENT
WHERE FAX IS NULL;
```

- (Q7) : Lister les désignations et les adresses de tous les clients ayant un numéro de fax

```
SELECT DESIGNATION, ADRESSE
FROM CLIENT
WHERE FAX IS NOT NULL;
```

13

SQL: Structured Query Language

Recherche de données

Exemples de requêtes

- (Q8) : Lister les noms des produits dont la couleur soit Rouge, Bleu, Vert ou Marron

```
SELECT NOMP
FROM PRODUIT
WHERE COULEUR IN ('Rouge', 'Bleu', 'Vert', 'Marron');
```

- (Q9) : Lister les noms des produits dont la couleur est différente de Rouge et Bleu

```
SELECT NOMP
FROM PRODUIT
WHERE COULEUR NOT IN ('Rouge', 'Bleu');
```

- (Q10) : Lister les noms des produits dont le numéro est >= 1000 et <=5000

```
SELECT NOMP
FROM PRODUIT
WHERE NPRO BETWEEN 1000 AND 5000;
```

14

SQL: Structured Query Language

Recherche de données

Exemples de requêtes

- (Q11) : Lister les noms, quantités en stock des produits dont le nom commence par 'B'.

```
SELECT NOMP, QTES
FROM PRODUIT
 WHERE NOMP LIKE 'B%';
```

- (Q12) : Lister les noms des produits contenant la chaine "SAPIN"

```
SELECT NOMP
FROM PRODUIT
WHERE NOMP LIKE '%SAPIN%';
```

- (Q13) : Lister les noms des produits dont le 1ère et le 3ème caractères sont respectivement 'C' et 'd'

```
SELECT NOMP
FROM PRODUIT
WHERE NOMP LIKE "C_d%;
```

15

QUALIFICATION DES NOMS D'ATTRIBUTS

- Un attribut qualifié est un attribut accompagné du nom de la table auquel il appartient
- Notation attribut qualifié:
  - <nom_table>.<nom_attribut>
- La qualification est nécessaire lorsque le nom_attribut existe dans des tables utilisées dans la même requête.
- Un nom d'attribut non qualifié, référence la relation la plus interne qui a l'attribut de ce nom là.

16

SQL: Structured Query Language

Recherche de données

Exemples de requêtes

- (Q14) : Lister les désignations et catégories des clients qui ont acheté un produit de couleur 'Rouge' fourni par un fournisseur 'Algérois'

```
SELECT DISTINCT C.DESIGNATION, CAT AS CATEGORIE
FROM CLIENT C, VENTE V, PRODUIT P, ACHAT AS A, FOURNISSEUR AS F
WHERE C.NUMC=V.NUMC AND P.NPRO=V.NPRO AND A.NPRO=P.NPRO AND
A.NUMF=F.NUMF AND COULEUR='Rouge' AND F.ADRESSE LIKE '%Alger%';
```

- (Q16) : Lister les désignations et les adresses des clients avec les noms et les couleurs des produits qui les ont achetés, en triant les résultats suivant l'ordre décroissant des désignations des clients et l'ordre croissant des noms et des couleurs des produits.

```
SELECT DESIGNATION, ADRESSE, NOMP as NOM_PRODUIT, COULEUR
FROM CLIENT , ACHAT
WHERE CLIENT.NUMC=ACHAT.NUMC
ORDER BY DESIGNATION DESC, NOMP, COULEUR ASC;
```

- L'ordre de tri par défaut (si on ne le spécifie pas explicitement en utilisant les mots clés ***ASC*** ou ***DESC***) est l'ordre ascendant des valeurs (***ASC***).

17

SQL: Structured Query Language

Recherche de données

Exemples de requêtes

Exemples de requêtes imbriquées

- (Q17) : Lister les désignations et les adresses des clients ayant acheté un produit de couleur rouge ou un produit fourni par un fournisseur de Annaba

```
SELECT DESIGNATION, ADRESSE
FROM CLIENT
WHERE NUMC IN
                (SELECT NUMC
                 FROM ACHAT, PRODUIT
                 WHERE COULEUR='Rouge' AND ACHAT.NPRO=PRODUIT.NPRO)
        OR
        NUMC IN
                (SELECT NUMC
                 FROM ACHAT A, VENTE AS V, FOURNISSEUR F
                 WHERE F.ADRESSE LIKE '%Annaba%' AND A.NPRO=V.NPRO AND
                        A.NUMF=F.NUMF);
```

18

SQL: Structured Query Language

Recherche de données

Exemples de requêtes imbriquées

- (Q18) : Lister les noms des produits de couleur vert ayant une quantité en stock et un prix de vente identiques à la quantité en stock et le prix de vente des produits de couleur rouge.

```
SELECT NOMP
FROM PRODUIT
WHERE COULEUR='Vert'
      AND (QTES, PRIX_VENTE) IN (SELECT QTES, PRIX_VENTE
                                 FROM PRODUIT
                                 WHERE COULEUR='Rouge');
```

- (Q19) : Lister les noms des produits de couleur vert ayant une quantité en stock supérieure à la quantité en stock de tous les produits de couleur jaune

```
SELECT NOMP
FROM PRODUIT
WHERE COULEUR='Vert'
      AND QTES > ALL (SELECT QTES
                      FROM PRODUIT
                      WHERE COULEUR='Jaune');
```

19

SQL: Structured Query Language

Recherche de données

Exemples de requêtes imbriquées

(Q20) : Lister les noms des produits de couleur vert ayant une quantité en stock égale à la quantité en stock de certains produits de couleur jaune

```
SELECT NOMP
FROM PRODUIT
WHERE COULEUR='Vert'
      AND QTES =SOME (SELECT QTES
                      FROM PRODUIT
                      WHERE COULEUR='Jaune');
```

Ou bien

```
SELECT NOMP
FROM PRODUIT
WHERE COULEUR='Vert'
      AND QTES =ANY (SELECT QTES
                     FROM PRODUIT
                     WHERE COULEUR='Jaune');
```

20

SQL: Structured Query Language

Recherche de données

Exemples de requêtes imbriquées

- (Q22) : Lister les désignations et les adresses des fournisseurs qui ont fourni un produit de couleur bleu.

```
SELECT DESIGNATION, ADRESSE
FROM FOURNISSEUR AS f
WHERE EXISTS (SELECT *
              FROM ACHAT, PRODUIT p
              WHERE COULEUR='Blue' AND
              ACHAT.NUMF=f.NUMF AND ACHAT.NPRO=p.NPRO);
```

  - En SQL, la fonction ***EXISTS*** est utilisée pour vérifier si le résultat d'une sous-requête imbriquée est vide (ne contient pas de tuples) ou non. Le résultat de la fonction ***EXISTS*** est une valeur booléenne: ***TRUE*** ou ***FALSE***.

- (Q23) : Lister les désignations et les adresses des fournisseurs qui n'ont pas fourni un produit de couleur bleu.

```
SELECT DESIGNATION, ADRESSE
FROM FOURNISSEUR AS f
WHERE NOT EXISTS (SELECT *
                  FROM ACHAT a, PRODUIT p
                  WHERE COULEUR='Blue' AND a.NUMF=f.NUMF
                  AND a.NPRO=p.NPRO);
```

21

SQL: Structured Query Language

Recherche de données

Exemples de requêtes imbriquées

- (Q24) : Lister les noms des produits qui ont été achetés et vendus

```
SELECT NOMP
FROM PRODUIT p
WHERE EXISTS (SELECT *
              FROM ACHAT a
              WHERE a.NPRO=p.NPRO)
    AND
    EXISTS (SELECT *
            FROM VENTE v
            WHERE v.NPRO=p.NPRO)
```

- (Q25) : Lister les désignations des clients qui ont acheté au plus un seul produit de couleur rouge

```
SELECT DESIGNATION
FROM CLIENT c
WHERE UNIQUE(SELECT COULEUR
             FROM ACHAT a, PRODUIT p
             WHERE COULEUR='Rouge' AND a.NUMC=c.NUMC
             AND a.NPRO=P.NPRO);
```

  - La fonction ***UNIQUE(Q)*** renvoie ***TRUE*** s'il n'y a pas de tuples doubles dans le résultat de la sous-requête ***Q***, autrement elle renvoie ***FALSE***.

22

SQL: Structured Query Language

Recherche de données

Expression de l'union, de l'intersection et de la différence

- Les opérations d'union, d'intersection et de différence de l'algèbre relationnelle s'expriment en SQL par :
  *<Clause Select> <Opération> [**ALL**] <Clause Select>*
  *<Opération> ::= **'UNION'** | **'INTERSECT'** | **'MINUS'** | **'EXCEPT'***

- Si le mots clé ***ALL*** est spécifié, le résultat de l'opération *<Opération>* peut inclure les tuples en doubles.
- Certains dialectes de SQL ne supportent pas l'opération d'intersection et de différence; d'autres utilisent **'MINUS'** à la place de ***'EXCEPT'***.

23

SQL: Structured Query Language

Recherche de données

Expression de l'union, de l'intersection et de la différence

- (Q26) : Lister les adresses des clients et des fournisseurs en éliminant les doublons

```
SELECT ADRESSE
FROM CLIENT
UNION
SELECT ADRESSE
FROM FOURNISSEUR ;
```

(Q27) : Lister les adresses des clients et des fournisseurs en retenant les doublons

```
SELECT ADRESSE
FROM CLIENT
UNION ALL
SELECT ADRESSE
FROM FOURNISSEUR ;
```

- (Q28) : Lister les adresses où il y a des clients et des fournisseurs

```
SELECT ADRESSE
FROM CLIENT
INTERSECT
SELECT ADRESSE
FROM FOURNISSEUR ;
```

24

SQL: Structured Query Language

Recherche de données

(Q28) : Lister les adresses où il y a des clients et des fournisseurs

Avec blocs imbriqués

```
SELECT ADRESSE
    FROM CLIENT
    WHERE ADRESSE IN (SELECT ADRESSE FROM
                          FOURNISSEUR);
```

Expression de l'union, de l'intersection et de la différence

- (Q29) : Lister les adresses où il y a des clients et il n'y a pas de fournisseurs

```
SELECT ADRESSE
FROM CLIENT
EXCEPT
SELECT ADRESSE
FROM FOURNISSEUR
```

Avec blocs imbriqués

```
SELECT ADRESSE
    FROM CLIENT
    WHERE ADRESSE NOT IN  (SELECT ADRESSE
                               FROM FOURNISSEUR);
```

25

Fonctions de calculs et agrégats

En SQL, elles existent des possibilités de calcul de fonctions. Les fonctions implantés sont :

- ***COUNT*** qui permet de compter le nombre de valeurs d'un ensemble;
- ***SUM*** qui permet de sommer les valeurs d'un ensemble;
- ***AVG*** qui permet de calculer la valeur moyenne d'un ensemble;
- ***MAX*** qui permet de calculer la valeur maximum d'un ensemble;
- ***MIN*** qui permet de calculer la valeur minimum d'un ensemble.

- Ces fonctions peuvent être utilisées dans la clause SELECT. Elles sont aussi utilisables pour effectuer des calcules d'agrégats.
- Un agrégat est un partitionnement horizontale d'une table en sous-tables en fonction des valeurs d'un ou de plusieurs attributs de partitionnement, suivi de l'application d'une fonction de calcul à chaque attribut des sous-tables obtenus.

26

SQL: Structured Query Language

Recherche de données

Fonctions de calculs et agrégats

Cette fonction est choisie parmi celle indiquées ci-dessus.

- En SQL, le partitionnement s'exprime par la clause :
  GROUP BY *<Spécification de Colonne>⁺*
- La clause GROUP BY permet de préciser les attributs de partitionnement, alors que les fonctions de calculs appliquées aux ensembles générés sont directement indiquées dans les expressions de valeurs suivant le ***SELECT***.
- Une restriction peut être appliquée avant calcul de l'agrégat au niveau de clause WHERE, mais aussi après calcul de l'agrégat sur les résultats de ce dernier en utilisant une clause spéciale qui est ajoutée à la clause SELECT :
  HAVING *<Expression de Valeurs>⁺*

Exemples de calculs d'agrégats

- (Q30) : Calculer la somme, la moyenne, le maximum et le minimum des quantités en stock des produits de couleur rouge.

```
SELECT SUM(QTES), AVG(QTES) , MAX(QTES), MIN(QTES)
FROM PRODUIT
WHERE COULEUR='Rouge';
```

27

SQL: Structured Query Language

Recherche de données

Exemples de calculs d'agrégats

- (Q31) : Calculer le nombre de couleurs distinctes des produits

```
SELECT COUNT(DISTINCT COULEUR)
FROM PRODUIT
```

- (Q32) : Calculer le nombre de produits par couleur.

```
SELECT COULEUR, COUNT(*)
FROM PRODUIT
GROUP BY COULEUR;
```

- (Q33) : Calculer le nombre de produits distincts vendus par clients

```
SELECT NOMC, COUNT(DISTINCT V.NPRV)
FROM CLIENT C JOIN VENTE V ON C.NOMC=V.NOMC
GROUP BY NOMC;
```

- (Q34) : Calculer la moyenne des quantités en stock pour toutes les couleurs des produits dont la quantité minimale est supérieur à 10

```
SELECT COULEUR, AVG(QTES)
FROM PRODUIT
GROUP BY COULEUR
HAVING MIN(QTES) >10;
```

28

SQL: Structured Query Language

Recherche de données

Exemples de calculs d'agrégats

- (Q35) : Rechercher tous les numéros de produits vendus en quantité supérieure à 20 à plus de 100 clients

```
SELECT NPRV
FROM VENTE
WHERE QTEV >20
GROUP BY NPRV
HAVING COUNT(NUMC) > 100;
```

- (Q36) : Rechercher les clients ayant acheté depuis 2000 une quantité de produits de couleur rouge inférieur à 200.

```
SELECT NOMC
FROM CLIENT
WHERE NUMC IN (SELECT NUMC
               FROM  VENTE V, PRODUIT P
               WHERE V.NPRV = P.NPRO AND DATEV >= 01-01-2000
               AND COULEUR='Rouge'
               GROUP BY NUMC
               HAVING SUM(QTEV) <200);
```

29

SQL: Structured Query Language

Recherche de données

Expression de jointures

Jointure simple (equi-jointure)

- Lister les noms de tous les produits vendus

```
SELECT NOMP
FROM  PRODUIT
WHERE  NPRO IN (SELECT DISTINCT NPRV FROM VENTE);
```

Ou :

```
SELECT DISTINCT NOMP
FROM PRODUIT , VENTE
WHERE NPRO=NPRV;
```

Jointure Naturelle (equi-jointure)

- Lister les désignations des clients qui ont acheté au moins un produit

```
SELECT DISTINCT DESIGNATION
FROM CLIENT JOIN VENTE ON CLIENT.NOMC=VENTE.NOMC ;
```

30

SQL: Structured Query Language

Recherche de données

Expression de jointures

Jointure Naturelle (equi-jointure)

- Lister les désignations des clients qui ont acheté au moins un produit

```
SELECT DISTINCT NOMC
FROM CLIENT JOIN VENTE ON CLIENT.NOMC=VENTE.NOMC;
```

ou

```
SELECT DISTINCT DESIGNATION
FROM CLIENT NATURAL JOIN VENTE;
```

Jointure externe à gauche (LEFT JOIN ou LEFT OUTER JOIN)

- Lister en ordre croissant les désignations de tous les clients avec le nombre de produits achetés par chacun d'eux.

```
SELECT CLIENT.NOMC, COUNT(DISTINCT NPRV)
FROM CLIENT LEFT JOIN VENTE ON CLIENT.NOMC=VENTE.NOMC
GROUP BY CLIENT.NOMC
ORDER BY CLIENT.NOMC ASC;
```

31

SQL: Structured Query Language

Recherche de données

Jointure externe à droite (RIGHT JOIN ou RIGHT OUTER JOIN)

- Lister en ordre décroissant les noms de tous les produits avec le nombre de clients qui les ont achetés.

```
SELECT NOMP, COUNT(DISTINCT NOMC)
FROM VENTE RIGHT JOIN PRODUIT ON NPRO=NPRV
GROUP BY NOMP
ORDER BY NOMC DESC;
```

Jointure externe complète (FULL JOIN ou FULL OUTER JOIN)

- Lister en ordre croissant les noms de tous les produits et la désignation de tous les clients qui les ont achetés ou non .

```
SELECT NOMP, NOMC
FROM VENTE FULL JOIN PRODUIT ON NPRO=NPRV FULL JOIN CLIENT ON
    VENTE.NOMC=CLIENT.NOMC
ORDER BY NOMP, VENTE.NOMC ASC;
```

32

SQL: Structured Query Language

Les mises à jours

Insertion de tuples

- Elle peut s'effectuer :
  - ***Variante directe*** : par fourniture directe au terminal d'un tuple à insérer (ou d'une partie de tuple, les valeurs inconnues étant positionnées à NULL),
  - ***Variante indirecte***: soit par construction à partir d'une question des tuples à insérer.
- La syntaxe de la commande d'insertion est :

  INSERT INTO *<nom de table> [(<nom de colonne>⁺)]*

  *{**VALUES** (<constantes>⁺)|<commande de recherche>}*
- Dans le cas où la liste de colonnes n'est pas spécifiée, tous les attributs de la relation doivent être fournis dans l'ordre de déclaration. Si seulement certaines colonnes sont spécifiées, les autres sont insérées avec la valeur nulle.
- Une insertion à partir d'une commande de recherche permet de composer une relation à partir des tuples d'une relation existante, par recherche dans la base.

Exemple :

INSERT INTO *PRODUIT (NPRO, NOMP, QTES, COULEUR)* ***VALUES*** *(3444, 'Ecran', 5, 'Gris');*

33

SQL: Structured Query Language

Exemple :

- Insertion dans une table ***PRODUITS_ROUGES_VENDUS*** tous les produits de couleur rouge vendus

  INSERT INTO *PRODUITS_ROUGES_VENDUS*
  SELECT *DISTINCT NPRO, NOMP*
  FROM *PRODUIT, VENTE*
  WHERE *NPRO=NPRV AND COULEUR='Rouge';*

10.4. 2. Modification de tuples

- Elle permet de changer des valeurs d'attributs de tuples existant.
- Elle s'effectuer :
  - Fourniture directe des valeurs à modifier
  - Elaboration des valeur à modifier à partir d'une expression
- Les seules tuples modifiés sont ceux vérifiant une condition de recherche optionnelle fournit en argument d'une clause WHERE.
- La syntaxe de la commande de modification de tuples est :

  UPDATE *<nom de table>*
  SET *{<nom de colonne>= {<expression de valeur> | **NULL**}}⁺*
  WHERE *<condition de recherche>*

34

SQL: Structured Query Language

Exemple :

- Changement des produits de couleur rouge en couleur bleu

```
UPDATE PRODUIT
SET CLOULEUR='Bleu'
WHERE COULEUR='Rouge';
```

- Augmenter la quantité en stock de 100% pour tous les produit de couleur vert

```
UPDATE PRODUIT
SET QTES= QTES*2
WHERE COULEUR='Vert';
```

10.4.3. Suppression de tuples

- Elle permet d'enlever d'une relation des tuples existants.
- Les tuples à supprimer sont spécifiés par une condition de recherche. Dans le cas où on ne spécifie pas cette condition, tous les tuples de la relation seront supprimés.
- La syntaxe de la commande de suppression est :

```
DELETE FROM <nom de table>
WHERE <condition de recherche>
```

35

SQL: Structured Query Language

Exemple :

- Suppression de tous les produits non achetés

```
DELETE FROM PRODUIT
WHERE NRRO NOT IN
(SELECT NPRV
 FROM VENTE)
```

36

SQL: Définition des données

37

SQL: Structured Query Language

Définition de Schémas

- En SQL2, on trouve dans un ***environnement SQL*** (i.e., une installation d'un SGBD relationnel conforme à SQL sur une machine) des relations et d'autres objets base de données tels que les contraintes, les vues, les domaines, les triggers (déclencheurs) et d'autres constructions comme l'attribution des droits d'accès, etc.
- Chaque environnement SQL contient un ou plusieurs (***groupe*** ou ***cluster***) ***catalogues*** , et chaque catalogue est composé d'un ensemble de ***schémas***.
- Un ***schéma*** est une collection nommée d'objets base de données interdépendants.
- Les objets d'un schéma peuvent être des tables, vues, domaines, assertions, collations (correspondances), translations, et des jeux de caractères. Ils ont tous le même propriétaire.
- Un schéma de base de données SQL est identifié par ***nom***, et contient un ***identificateur d'autorisation*** indiquant l'utilisateur ou le compte détenant le schéma, et également des ***descripteurs*** pour chaque élément dans le schéma.
- Un schéma est créé par l'instruction ***CREATE SCHEMA*** selon la syntaxe suivante :

  CREATE SCHEMA <nom de schéma> ***AUTHORIZATION*** <identificateur du créateur>;

38

SQL: Structured Query Language

Exemple

CREATE SCHEMA *COMMERCE AUTHORIZATION 'Samir';*

- Cette construction permet de créer un schéma de base de données appelé commerce, et détenu par l'utilisateur dont l'identificateur d'autorisation est 'Samir'.

Remarques

- Chaque instruction en SQL se termine par ";"
- SQL, vis-à-vis ses mots clés, est insensible à la casse.
- En général, tous les utilisateurs ne sont pas autorisés à créer des schémas et les éléments d'un schéma. Le privilège de créer des schémas, des tables, et d'autres constructions doit être explicitement accordé aux comptes d'utilisateurs concernés par l'administrateur système ou le DBA.
- Chaque catalogue contient un schéma spécial appelé SCHEMA_INFORMATION qui fournit des informations sur tous les schémas de ce catalogue et tous les descripteurs d'éléments dans ces schémas. Les contraintes d'intégrité telles que l'intégrité référentielle peut être définie entre les relations uniquement si elles existent dans les schémas du même catalogue. Les schémas d'un même catalogue peuvent également partager certains éléments, tels que les définitions de domaines.

39

SQL: Structured Query Language

Création de tables

- SQL permet de créer des relations sous forme de tables et de définir lors de la création des contraintes d'intégrité variés sur les attributs.
- Une commande de création d'une relation permet de spécifier le nom de la table et de définir les éléments de la tables correspondant aux colonnes ou aux contraintes, selon la syntaxe suivante :

  CREATE TABLE *<nom de table> (<élément de table>⁺);*
- <nom de la table> peut être un nom simple (exemple PRODUIT) ou un nom composé d'un nom de schéma (identifiant du schéma dans lequel la table est déclarée) suivi par le nom simple de table en notation pointée (ex. COMMERCE.PRODUIT)
- Un élément de table est soit une définition de colonne, soit une définition de contrainte, comme suit :

  <élément de table> ::= <définition de colonne>|<contrainte de table>
- Chaque colonne est définie par un nom et un type de données. Une valeur par défaut peut être précisée. Une contrainte de colonne peut être aussi spécifiée à ce niveau.

  <définition de colonne > ::= <nom de colonne> <type de données>
  [<clause défaut>][<contrainte de colonne>]
  - La clause défaut permet de spécifier une valeur par défaut selon la syntaxe ***DEFAULT*** *<valeur>*, la valeur ***NULL*** étant permise. Les contraintes d'intégrité de table et de colonne seront examinées dans la section qui suit.

40

SQL: Structured Query Language

Types de données supportés

- ***CHAR(n)*** ou ***CHARACTER(n)***: les chaînes de caractères de longueurs fixes, la valeur par défaut de la longueur étant 1.
- ***VARCHAR(n)***, ***CHAR VARYING(n)*** ou ***CHARACTER VARYING(n)***: les chaînes de caractères de longueurs variables, avec au maximum n caractères. Les chaînes de caractères constantes sont placées entre simples quotes (guillemets) (ex., 'Bonjour tout le monde ')
- ***CLOB(n)***: ***Character Large Object***, les chaines de caractères de longueurs variables qui peuvent stocker d'importants volumes de données caractères (ex. documents). La longueur maximum d'un CLOB peut être spécifiée en kilo-octets (K), méga-octets (M) ou en giga-octets (G). Par exemple, *CLOB(2G)* spécifie une chaine de caractères de longueur maximum de 2 GO.
- ***INT*** ou ***INTEGER*****:** entier (un sous-ensemble fini d'entiers qui dépend de l'implémentation)
- ***SMALLINT*** : entier court (un sous-ensemble du type de domaine entier qui dépend de l'implémentation)
- ***DECIMAL(p, d)***, ***DEC(p, d)*** ou ***NUMERIC(p, d)***: nombre réel à point fixe , avec une précision (nombre total de chiffres décimaux) de *p* chiffres et une échelle (nombre de chiffres après la virgule) de *d* chiffres
- ***REAL***, ***DOUBLE PRECISION***: nombres réels à virgule flottante avec simple et double précision dépendant de l'implémentation.
- ***FLOAT(n)*** : nombre réel avec une précision optionnelle d'au moins n chiffres décimaux, la valeur par défaut de n étant 0, et si n=0 on ***FLOAT*** ⇔***REAL***
- ***BIT(n)***, ***ou BIT VARYING(n)***: les chaînes de bits (i.e., séquences de chiffres binaires 0 ou 1) de longueurs fixes ou variables ï n étant le nombre maximum de bits, par défaut n=1. Les chaînes de bits constantes sont placées entre simples quotes qui doivent être précédées de la lettre B (ou X c'est la longueur de la chaine est un multiple de 4) afin de les distinguer des chaines de caractères ordinaires (ex., B'01011111' ou X'5F').

41

SQL: Structured Query Language

Types de données supportés

- ***BLOB***: ***Binary Large Object***, chaines de bits de longueurs variables pouvant contenir de gros volumes de données binaires (ex. images, vidéos, sons, etc.). Comme pour le type CLOB, la longueur maximum d'un BLOB peut être spécifiée en kilo-octets (K), méga-octets (M) ou en giga-octets (G). Par exemple, ***BLOB(20G)*** spécifie une chaine de bits de longueur maximum de 20 gigabits.
- ***BOOLEAN*** : type de données logique binaire qui possède deux valeurs ordinaires possibles : ***TRUE*** ou ***FALSE***. En SQL, étant donnée la présence des valeurs nulles, on utilise une logique à trois valeurs, de telle sorte une troisième valeur "***UNKOWN***" est ajoutée pour le type de données booléen.
- ***DATE:*** utilisé pour représenter des données de date. Il possède 10 positions et est composé de 3 parties : année (***YEAR***), mois (***MONTH***), et jour (***DAY***) sous la forme de ***AAAA-MM-JJ*** (ou ***YYYY-MM-DD***). Les valeurs de dates constantes sont représentées par des chaines placées entre simples quotes précédées par le mot clé ***DATE*** (ex. DATE'2007-09-21'). Les environnements SQL autorisent uniquement les dates valides (i.e., 1≤ mois ≤ 12, 1≤ jours ≤ 31 et la valeur du jour doit être adéquate avec la valeur du mois).
- ***TIME*****:** utilisé pour représenter des données de temps. Il possède au moins 8 positions, avec les composantes de l'heure (***HOUR***), minute (***MINUTE***), et seconde (***SECOND***) sous la forme de HH:MM :SS. Les valeurs de temps constantes sont représentées par des chaines simples placées entre quotes précédées par le mot clé TIME (ex. *TIME'12 :23 :23'*). Les environnements SQL autorisent uniquement les temps valides (i.e., 0≤ Heure ≤ 23, 0≤ Minute ≤ 59 et 0≤ Second ≤ 59).

42

SQL: Structured Query Language

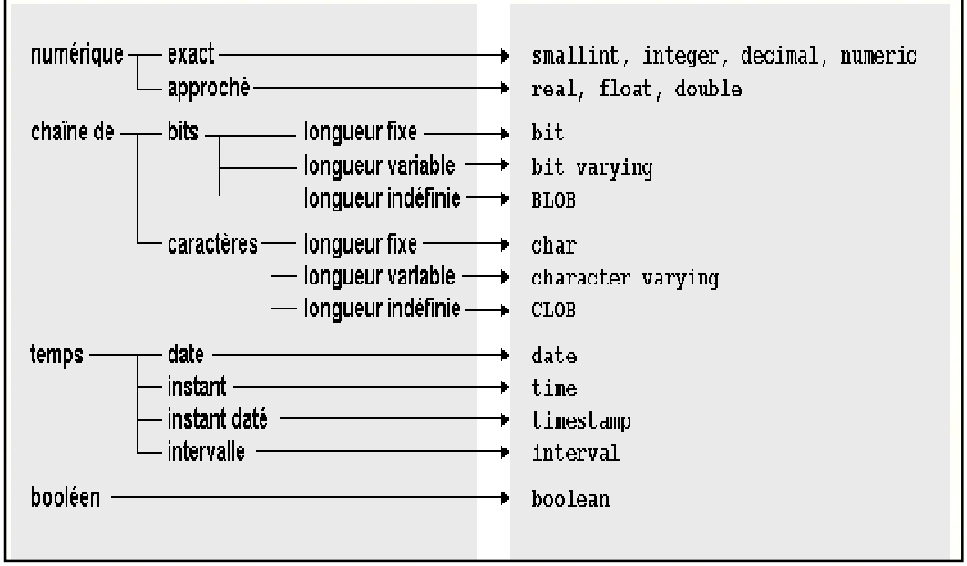
Types de données supportés

- ***TIME WITH TIME ZONE*** : idem que le type TIME , il comporte en plus 6 positions pour spécifier le déplacement par rapport à la zone du temps universel standard, qui est dans lôintervalle de +13 :00-12 :59 en unité dô*heures:minutes*. Si ***WITH TIME ZONE*** nôest pas incluse, la valeur par défaut est le fuseau horaire local de la session de lôenvironnement SQL.
- **TIMESTAMP** : type de données composé de champs date et heure (horodateur), en plus de 6 positions pour les fractions décimales de secondes et un qualificateur optionnel "WITH TIME ZONE". Les valeurs constantes de type *TIMESTAMP* sont placées entre simples quotes et précédées par le mot clé *TIMESTAMP* (ex., TIMESTAMP'2007-08-21 00:30:47.648302').
- **INTERVAL**: spécifie un intervalle ð une valeur relative qui peut être utilisée pour incrémenter ou décrémenter une valeur absolue d'une date, d'un temps ou d'un horodateur. Les intervalles sont qualifiés soit comme des intervalles *YEAR/MONTH* (i.e., avec précision en année-mois, ex., ***INTERVAL YEAR(2) TO MONTH***) , soit comme des intervalles *DAY/TIME* (i.e., avec précision en jour-heure-minute-seconde, ex., **INTERVAL HOUR TO SECOND(4)**).
- Les format de *DATE*, *TIME et TIMESTAMP* peuvent être considérés comme un type spécial de chaîne de caractères. Par conséquent, ils peuvent généralement être utilisés, après les avoir converties en chaînes de caractères équivalentes, dans les comparaisons de chaînes de caractères.

43

SQL: Structured Query Language

Types de données supportés



SQL: Structured Query Language

Types de données supportés

- Il est possible de spécifier les type de données par l'utilisateur (i.e., création de domaines par l'utilisateur) au sein du schéma d'une base de données avec un meilleur contrôle de l'intégrité de domaine. En SQL2, ces type restent purement des sous-ensembles des types prédéfinis; aucune opération spécifique ne leur être associée. La syntaxe pour la création d'un domaine est :

  CREATE DOMAIN <nom de domaine> ***[AS]*** *<type de données>*

  *[**DEFAULT** <option défaut>]*

  *[**CHECK**(<condition de recherche>)]*

  - <option défaut> : permet de spécifier une valeur par défaut
  - <condition de recherche> : permet de spécifier une contrainte générale (de contrôle) sur les valeurs autorisées
- La création de domaine permet
  - d'effectuer le changement de son type de données en un seul endroit, si ce domaine est utilisé par plusieurs attributs;
  - L'utilisation de ce domaine par plusieurs attributs du schéma de base de données
  - Améliorer la lisibilité du schéma de base de données

45

Contrôle des Contraintes d'intégrité

- Un SGBD doit garantir la cohérence des Données lors des MAJ
- Les dépendances entre les données sont déclarées par des règles appelées Contraintes d'intégrité (CI)
- Ces Contraintes sont déclarées explicitement et sont mémorisées dans le catalogue (dictionnaire de Données)
- Les MAJ portant sur des données dépendantes doivent faire passer la base d'un état cohérent à un autre état cohérent,
- Pour garder la base de données cohérente, le SGBD doit vérifier que les CI sont satisfaites à chaque fin d'opération de MAJ,
- NB. Il faut ordonner les MAJ en cas de contraintes d'intégrité référentielles: il faut insérer dans la relation Maitre (la relation référencée) avant d'insérer dans la relation qui la référence

Méthodes de vérification de cohérences

- Méthode de détection:
  - Elle consiste à évaluer la CI après exécution de la transaction. Ainsi après une opération de modification de données (INSERT, DELETE ou UPDATE) un test appelé post-test est lancé par le SGBD qui consiste à vérifier si une CI a été violée,
- Méthode de prévention
  - Elle consiste à empêcher les modifications de la base qui violeraient une quelconque CI. Pour cela un test avant MAJ appelé pré-test permet de garantir la non violation dûne CI. Ce test est lancé par le SGBD

Contrôle sur les contraintes simples

- Unicité de la clé
  - Tout SGBD gère en général un index (table dans laquelle il y a toutes les clés primaires) sur les clés primaires. Un pré-test simple consiste à vérifier que la nouvelle clé ne figure pas dans lûindex. Ce pré-test est effectué lors de lûinsertion dûun tuple ou de la modification dûune clé dans la table.
- Contrainte dûintégrité référentielle (clé étrangère)
  - Deux tables, la table référencée et la table dépendante dont mises en jeu par une contrainte référentielle. 4 types de modification nécessitent des vérifications:
- 1) Insertion dans la table dépendante: pré-test simple qui consiste à vérifier lûexistence de la valeur de la colonne référençante dans lûindex (table dans laquelle il y a toutes les clés)
- 2) MAJ de la colonne référençante dans la table dépendante. Le pré-test est identique au précédent
- 3) suppression dans la table maitre: le pré-test consiste à vérifier quûil nûexiste pas de tuple contenant la valeur de clé à supprimer dans la colonne référençante. Il y a plusieures possibilités:
  - soit rejeter la MAJ,
  - Soit supprimer les tuples de la table dépendantes

Contrôle sur les contraintes simples

- Contrainte de domaine
  - Pour les contraintes de type CHECK <condition>, il suffit de vérifier avant dôappliquer la MAJ que la valeur qui serait donnée à lôattribut après MAJ vérifie la condition

SQL: Structured Query Language

Expression des contraintes d'intégrité

- Les ***contraintes de colonnes*** permettent de spécifier différentes contraintes d'intégrité portant sur un seul attribut, y compris les contraintes référentielles. Les différentes variantes sont :
  - Valeur nulle impossible (syntaxe ***NOT NULL***),
  - Unicité de l'attribut (syntaxe ***UNIQUE*** ou ***PRIMARY KEY***),
  - Contrainte référentielle –syntaxe ***REFERENCES*** *<table référencée> [(<colonne référencée>)]* –, le nom de la colonne référencée est optionnel s'il est identique à celui de la colonne référençante,
  - Contrainte générale (de contrôle) (syntaxe ***CHECK*** *<condition>*); la condition est une condition pouvant spécifier des plages ou des liste de valeurs possibles (voir condition de recherche ci-après)

50

SQL: Structured Query Language

Expression des contraintes d'intégrité

- Les ***contraintes de relations*** peuvent porter sur plusieurs attributs. Cela peut être des contraintes d'unicité, référentielles ou générales. Elle sont exprimées à l'aide des syntaxes suivantes:
  - Contrainte d'unicité ***UNIQUE*** *<attribut>*$^+$,
  - Contrainte référentielle, permettant de spécifier quelles colonnes référencent celles d'une autre table, *[**FOREIGN KEY** (<colonne référençante>$^+$)] **REFERENCES** <table référencée> [(<colonne référencée>$^+$)]*,
  - Contrainte générale, ***CHECK*** *<condition>*.
- Il est possible de donner un nom à une contrainte (syntaxe *CONSTRAINT <nom de contrainte>*). Le nom d'une contrainte doit être unique l'intérieur d'un schéma de base de données.

51

SQL: Structured Query Language

```
CREATE SCHEMA COMMERCE AUTHORIZATION 'Samir';

CREATE TABLE PRODUIT (
NPRO INT NOT NULL,
NOMP VARCHAR(50) NOT NULL UNIQUE,
QTES INT DEFAULT 0 CHECK (QTES >=0),
COULEUR CHAR(10) NOT NULL DEFAULT 'Blue',
PRIX _VENTE DECIMAL(9, 2),
PRIMARY KEY (NPRO));

CREATE TABLE Client (
NUMC INT NOT NULL,
DESIGNATION VARCHAR(50) NOT NULL UNIQUE,
ADRESSE VARCHAR(100),
TEL CHAR(14), FAX CHAR(14), CAT CHAR(2),
CONSTRAINT CLICU
PRIMARY KEY (NUMC));
```

52

SQL: Structured Query Language

```
CREATE TABLE Fournisseur(
NUMF INT NOT NULL,
DESIGNATION VARCHAR(50) NOT NULL UNIQUE,
ADRESSE VARCHAR(100),
TEL CHAR(14), FAX CHAR(14),
CONSTRAINT  FOURCU
PRIMARY KEY  (NUMF));

CREATE TABLE Vente (
NUMV INT NOT NULL,
NPRO INT NOT NULL, NUMC INT NOT NULL,
QTEV SMALLINT NOT NULL, DATEV DATE NOT NULL,
PRIMARY KEY (NUMV),
FOREIGN KEY NPRO REFERENCES PRODUIT ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY NUMC REFERENCES  CLIENT(NUMC) ON DELETE
CASCADE ON UPDATE CASCADE);
```

53

SQL: Structured Query Language

```
CREATE TABLE  Achat (
NUMA INT NOT NULL,
NPRO INT NOT NULL,
NUMF INT,
QTEA SMALLINT NOT NULL,
DATEA DATE NOT NULL,
PRIMARY KEY (NUMA),
FOREIGN KEY NPRO REFERENCES PRODUIT ON
DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY NUMF REFERENCES
FOURNISSEUR(NUMF) ON DELETE  SET NULL ON
UPDATE CASCADE);
```

54

SQL: Structured Query Language

Variante : contraintes de colonne

```
CREATE TABLE CLIENT (
 NCLI CHAR(10) PRIMARY KEY, . . .);

CREATE TABLE COMMANDE (
NCOM  CHAR(12) NOT NULL,
NCLI  CHAR(10) NOT NULL REFERENCES CLIENT, . . . );
```

Variante : contraintes nommées

```
CREATE TABLE COMMANDE (
NCOM  CHAR(12) NOT NULL,
NCLI  CHAR(10) NOT NULL REFERENCES CLIENT, . . . );

CREATE TABLE COMMANDE (
NCOM  CHAR(12) CONSTRAINT COMPK NOT NULL,
NCLI  CHAR(10) NOT NULL CONSTRAINT COMFK
REFERENCES CLIENT, . . . );
```

55

SQL: Structured Query Language

Suppression des tables

- La suppression d'une table se fait par la commande suivante :

  DROP TABLE *<nom de table>* ***[RESTRICT|CASCADE]***
- L'option ***RESTRICT***, permet d'annuler la suppression de la table s'il y a des objets qui en dépendent .
- L'option **CASCADE** permet aussi la suppression de tous les objets (et aussi les objets qui en dépendent) dépendant de la table.

Exemple

```
DROP TABLE ACHAT
```

56

SQL: Structured Query Language

Modification des schémas de tables

- La modification d'un schéma de table se fait à l'aide de la commande :

  ALTER TABLE *<nom de table>*
  [ADD [COLUMN] *<nom de colonne> <type de données>* ***[NOT NULL] [UNIQUE]***
  [DEFAULT *<option défaut>]* ***[CHECK*** *(<condition de recherche>)]]*
  [DROP [COLUMN] *<nom de colonne>* ***[RESTRICT | CASCADE]]***
  [ADD [CONSTRAINT *[<nom de contrainte>]] <définition de contrainte de la table>]*
  [DROP CONSTRAINT *<nom de la contrainte>* ***[RESTRICT | CASCADE]]***
  [ALTER [COLUMN] SET DEFAULT *<option de défaut>]*
  [ALTER [COLUMN] DROP DEFAULT]

- Différents types d'altération sont possibles:
  - Ajout d'une colonne (***ADD COLUMN***)

    ALTER TABLE *CLIENT* ***ADD COLUMN*** *EMAIL* ***VARCHAR(100);***
  - Suppression d'une colonne (***DROP COLUMN***)

    ALTER TABLE *CLIENT* ***DROP COLUMN*** *FAX;*

57

SQL: Structured Query Language

Modification des schémas de tables

- Modification de la définition d'une colonne (***ALTER /MODIFY COLUMN***)

  //Modifier la taille de l'attribut EMAIL

  ALTER TABLE CLIENT ALTER COLUMN EMAIL VARCHAR(200);

  //Modifier la valeur par défaut de la couleur du produit
- ***ALTER TABLE PRODUIT MODIFY COLUMN COULEUR SET 'DEFAULT 'Rouge';***

  //Supprimer la valeur par défaut de la couleur du produit

  ALTER TABLE PRODUIT MODIFY COLUMN COULEUR DROP DEFAULT;
- Ajout d'une contrainte (***ADD CONSTRAINT***)

  Contraintes sans nom

  ALTER TABLE *PROSPECT* ***ADD CONSTRAINT PRIMARY KEY*** *(NCLI);*
  ALTER TABLE *CLIENT* ***ADD CONSTRAINT UNIQUE*** *(ADRESSE, TEL);*
  ALTER TABLE *CLIENT* ***MODIFY*** *CAT* ***NOT NULL;***
  ALTER TABLE CLIENT MODIFY *ADRESSE* ***NULL;***
  ALTER TABLE CLIENT ADD FOREIGN KEY (CAT) REFERENCES CATEGORIE ;

58

Normalisation d'une relation

La normalisation permet de garantir que le schéma relationnel est un bon schéma c-à-d possède de bonnes propriétés.
Exemple de mauvais schéma
soit la relation **livraison (Nofourn,adrF, Noprod, couleur, prixP, date, quantité)** ; cette relation présente plusieurs anomalies :
Anomalie de redondance: S'il existe N livraisons pour un fournisseur, son adresse adrF est répété N fois, il faut vérifier que c'est la même.
Incohérence en modification : La redondance de l'information entraîne également des risques en cas de modification d'une donnée répétée en différents endroits : on oublie fréquemment de modifier toutes ses occurrences (en général par simple ignorance des différentes places où figure l'information). Si un fournisseur change d'adresse, il faut changer toutes les adresses de toutes les occurrences de ce fournisseur.
Anomalie d'insertion
Pour insérer une nouvelle livraison, il faut enregistrer à nouveau cette adresse adrF. De même la couleur du produit est répétée autant de fois qu'il y a de livraison de ce produit.

On dit alors que la relation présente des **redondances.**
Les redondances impliquent des anomalies de mise à jour , exemple : si un fournisseur change d'adresse et qu'un seul tuple est mise à jour, nous aurons une incohérence dans la base.
Une relation qui présente des redondances est incorrecte. Il faut la normaliser.

La normalisation d'une relation est un processus de décomposition d'une relation présentant des mise à jour **complexe**s en plusieurs relations à mises à jour **simples.**
Exemple la relation livraison
livraison (Nofourn,adrF, Noprod, couleur, prixP, date, quantité) sera décomposée en :
Livraison (N°fourn, N°prod, date, qté)
Fournisseur (N°fourn, adrF)
Produit (N°prod, couleur)

Pour décomposer une relation à mise à jour complexe (présentant des redondances) nous faisons appel à la notion de dépendance fonctionnelle et au graphe minimum des dépendances

4.1 Dépendances fonctionnelles et graphe des dépendances fonctionnelles

Définition :
Les dépendances fonctionnelles permettent d'établir des liens sémantiques entre attributs ou groupe d'attributs

Etant donné une relation, R (X, Y, Z), il existe une **dépendance fonctionnelle**, ou "DF", de Y vers Z, notée Y→Z, si :
Etant donné deux tuples quelconques de R, s'ils ont même valeur pour Y, alors ils ont nécessairement même valeur pour Z.

On appelle Y source de la DF, et Z cible de la DF

Définition : une DF, X –> B, est une **dépendance fonctionnelle élémentaire** si B est un attribut unique, et si X est un ensemble minimum d'attributs (ou un attribut unique).
Définition Une DF non élémentaire est dite Augmentée :si X→Y alors quelque soit A A,X→ Y est une DF Augmentée et donc non élémentaire

Propriété des dépendances fonctionnelles : Si dans une relation, on a les deux dépendances fonctionnelles, X→Y et Y→Z, alors on a aussi la dépendance fonctionnelle X→Z qui est dite **déduite** des deux autres.

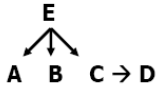
Graphe minimum des dépendances fonctionnelles
Définition : Etant donné une relation et un ensemble de DF portant sur les attributs de cette relation, on appelle **graphe minimum** des DF de la relation, tout ensemble de DF élémentaires non déduites, équivalent à en ce sens que toute DF ne peut être déduite des DF du graphe.
En d'autres termes les DF **augmentées et déduites** doivent être supprimer du graphe des DF pour obtenir un graphe minimum des DF.

Une méthode pour savoir si une DF, X→ Y, est déduite des autres est la suivante:
- établir un graphe (non minimum) des DF,
- supprimer la DF X→Y du graphe,
- parcourir tous les chemins possibles partant de X et suivant les DF. La DF, X→Y, est déduite si un (ou plusieurs) de ces chemins atteint Y. Le graphe minimum des DF sert essentiellement à définir des relations normalisées.

Exemple 1:

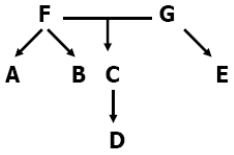
E→ D est une DF déduite des DFs E→C et C→ D. Il faut supprimer E→ D du graphe.

Identifiants : les identifiants peuvent être cherchés à partir du graphe minimum des DF,. Les identifiants correspondent à l'ensemble minimum d'attributs X qui nous permettent, en suivant, les DF d'atteindre tous les autres attributs.
Exemples:
Exemple 2 Soit R(A,B,C,D,E)
Avec le graphe minimum suivant :



E est l'identifiant de R

Exemple 3 Soit une relation R(A,B,C,D,E,F,G) ayant le graphe minimum suivant :



(F,G) est l'identifiant de R

Les formes Normales

3 Première et deuxième formes normales

Normaliser une relation qui présentent des redondances (relation non normalisée), consiste à décomposer cette relation en relations sans redondances (relations normalisées). La méthode à suivre est la suivante:

1/ vérifier que la relation est en première forme normale (voir définition ci-dessous);
2/ établir son graphe minimum des dépendances;
3/ déterminer, à l'aide du graphe, tous ses identifiants;
4/ déterminer, à l'aide du graphe, sa forme normale (voir définitions ci-dessous);
5/ si la relation n'est pas normalisée, décomposer, à l'aide du graphe, la relation en relations mieux normalisées.

Définition : Une relation est en **première forme** normale si chaque valeur de chaque attribut de chaque tuple est une valeur simple (tous les attributs sont simples et monovalués).

Définition 2ième Forme Normale :
Une relation est en 2ième forme normale (2FN) si :

- Elle est en 1FN, et
- Chaque attribut qui ne fait pas partie de l'identifiant dépend d'un identifiant entier (connu à l'étape 3) et non pas d'une partie de l'identifiant.

Dans l'exemple 2 la relation R(A,B,C,D,E) est en 2FN
Dans l'exemple 3, la relation R(A,D,C,D,E,F,G) n'est pas en 2FN car A et B dépendent de F qui est une partie de l'identifiant (F,G) de même E dépend de G qui est une partie de l'identifiant (F,G).

Définition 3ième forme Normale
Une relation est en 3ième forme Normale (3FN) si

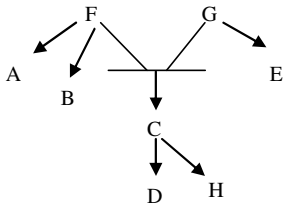
- Elle est en 1FN, et
- Chaque attribut qui ne fait pas partie d'aucun identifiant dépend **directement** d'une identifiant entier.

Exemples :
Dans l'exemple 2 la relation R(A,B,C,D,E) n'est pas en 3FN car D dépend de C (C→D) et C n'est pas identifiant.
Dans l'exemple 3 la relation R(A,B,C,D,E,F,G) n'est pas en 3FN car D dépend de C qui n'est pas identifiant.

Algorithme de décomposition / DF
Plusieurs algorithmes pour décomposer selon les DF existe nous avons retenu un algorithme de décomposition selon une méthode pragmatique.

Algorithme : méthode pragmatique
Algorithme 2 (à partir des sources de DF)
R (A1, A2, A3, … An)
Tant qu'il existe une source de DF élémentaire non déduite faire
    Choisir si possible une source dont toutes les cibles sont des extrémités terminales du graphe
    Soient Ai la source et Ai→Aj, Ak,… Al les DF
    Créer la relation Ri (Ai, Aj, Ak, … Al)
    Supprimer les attributs Aj, Ak, …Al de R
Si aucune des relations Ri ne contient un identifiant de R
alors ajouter la relation : R0 (un identifiant de R).

Exemple :
soit la relation R(A,B,C,D,E,F,G,H) avec le graphe minimum :

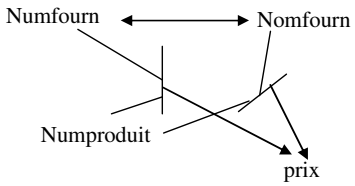


Après décomposition : (les identifiants des nouvelles relations normalisées sont soulignés)
R1 (F,A,B)
R2 (G,E)
R3 (C,D,H)
R4(F,G,C)

Relations à plusieurs identifiants. Forme normale de Boyce Codd

Toute relation peut toujours être décomposée en 3FN sans aucune perte d'information. A savoir sans perte de DF et sans perte d'information.
Ce qui n'est pas vrai pour les formes normales supérieures.

Cependant une relation en 3FN peut encore présenter des redondances. Exemple : soit la relation : catalogue(Numfourn,Nomfour,Numproduit,prix)
Avec le graphe minimum des dépendances suivant :



Cette relation présente deux identifiants : (numfourn,numproduit) et (nomfourn, numproduit)
La relation catalogue est en 3FN car tout attribut qui ne fait pas partie de la clé dépend directement de la clé entière. Cependant elle présente toujours des redondances (le nom de fournisseur est répété autant de fois qu'il y a de produit disponible chez ce même fournisseur.

La forme normale de Boyce Codd (FNBC) généralise le 3FN des relations à plusieurs identifiants.
Définition : une relation est en FNCB si elle est en 1FN et si toute source de DF est un identifiant entier.

Exemple : dans la relation catalogue, la DF numfourn→nomfourn présente une source qui n'est pas identifiant entier de même la DF nomfourn→numfourn. Donc n'est pas en FNBC

On décompose donc la relation catalogue en

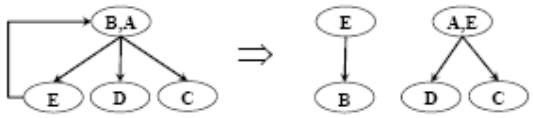
R1(numFour, NomFour) avec deux identifiants numfour et nomfour

R2(numfourn,numproduit,prix) avec identifiant (numfour + nnumproduit),

Remarque : une relation en FNBC est en 3ième forme normale, le contraire n'est pas vrai.

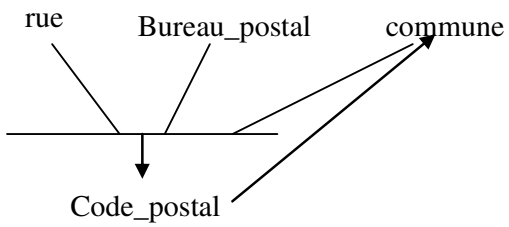
Normaliser en FNBC :

1. Isoler la DF problématique dans une nouvelle relation
2. Éliminer la cible de cette DF et la remplacer par sa source dans la relation initiale.



Exemple 2 :

Soit la relation adresse (rue, commune, bureau_postal,code_postal)
Ayant le graphe minimum suivant :



La relation adresse est en 3FN mais présente toujours des redondances car elle n'est en FNBC à cause de la DF code_postal → commune (source n'est pas identifiant).

Normalisation en FNBC : adresse1(rue,bureau_postal,code_postal)
R2(code_postal,commune)

La normalisation en FNBC peut conduire à des pertes d'information ou de DF.

Qualité d'une décomposition : notion de projection et de jointure

Soit une relation R qui contient des redondances et pose des problèmes lors des mises à jour : "Elle n'est pas normalisée"
Il faut la décomposer en plusieurs relations meilleures ("normalisées") :

- par **projection**
- en suivant les DF

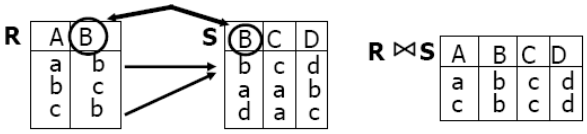
Après décomposition, il faut s'assurer de conserver le même contenu via l'opérateur de jointure : La jointure des nouvelles relations = R

Opérateur de projection :
Définition : **Projection** d'une relation:
Etant donné une relation R (A1, A2,...,An) et un sous-ensemble de ses attributs Ai1, Ai2,..., Aip, la projection de R sur Ai1, Ai2,..., Aip: **π**[Ai1, Ai2,..., Aip] R , crée une nouvelle relation d'attributs Ai1, Ai2,..., Aip, et de population égale à l'ensemble des tuples de R tronqués à Ai1, Ai2,..., Aip (sans doubles);

R (B , C, D) π (B , C) R

trois tuples deux tuples

Opérateur de jointure naturelle de deux relations:
La jointure de R1(X,Y) et R2 (Y,Z) s'indique de la façon suivante: R1 * R2.
C'est une nouvelle relation d'attributs (X,Y,Z), et dont la population est constituée de l'ensemble des tuples de R1 concaténés à ceux de R2 qui ont même valeur pour Y;



Définition: Une décomposition d'une relation R(X,Y,Z) en deux relations R1=π[X,Y]R et R2=π[X,Z]R (obtenues par projection de la relation R respectivement sur les attributs X,Y et X,Z) est dite "sans perte d'information" si R = R1*R2.

Exemple : soit R(NomEmp, adresse,poste,age) la décomposition de R en R1 et R2 tel que R1=π[NomEmp, adresse,poste] et R2= π[Nomp,age] est sans perte d'information mais elle est inutile.

Tandis que la décomposition de R en R1' = π[NomEmp,adresse,poste] et R2'= π[poste, age] est une décompostion avec perte d'information car R1'*R2'<> R.

Le théorème suivant permet de savoir quand une décomposition est sans perte d'information.

Théorème de Heath:
Toute relation R(X,Y,Z) est décomposable sans perte d'information en R1=π[X,Y]R et R2=π[X,Z]R s'il y a dans R une dépendance fonctionnelle de X vers Y (X→Y).

Il ne faut jamais faire une décomposition avec perte de données car les informations contenues dans la base sont alors fausses et il est impossible de rétablir les bonnes données

Pertes de dépendances fonctionnelles

Lorsque l'on décompose R en deux relations R1 et R2. la projection de l'ensemble des dépendances de R sur R1 s'obtient en ne gardant des dépendances de départ de R que celles dont les attributs (source et cible) sont les attributs de la relation R1.
Une décomposition en plusieurs relations est dite sans pertes de dépendances si l'on peur retrouver logiquement les dépendances de départ à partir de la projection de l'ensemble des dépendances sur les relations de la décomposition.

Si on décompose avec une perte de dépendances, on ne peut travailler indépendamment sur chacune des relations de la décomposition sans s'occuper des autres relations. Lors des MAJ on ne peut s'assurer que les dépendances (perdues) sont vérifiées qu'en consultant les autres relations, ce qui occasionne des traitements couteux .

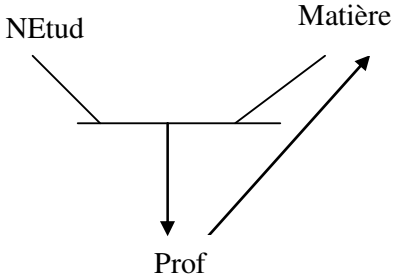
Il faut préférer les décompositions sans pertes de dépendances. Mais une perte de dépendances est mois grave qu'une perte de données.

On démontre que l'on peut toujour normaliser un schéma en 3FN sans perte de données ni perte de dépendances. Pour les FNBC et les formes normale supérieure il n'est pas toujours possible de conserver les dépendances.

Exemple

Soit la relation enseignant (NEtud,Matière,Prof)
Avec les règles de gestion :

1) chaque professeur enseigne une seule matière
2) pour chaque matière, chaque étudiant suit les cours d'un seul professeur



Cette relation présente deux identifiants : (NEtud+Matiére) et (NEtud+Prof). Elle est en troisième forme normale puisque tout attribut fait patie d'un identifiant mais elle n'est pas en FNBC puisque l'attribut Prof est source de DF et ne constitue pas un identifiant entier.
Cette relation présente des redondances dues au fait que la DF prof → matière n'est pas traduite par la relation : exemple si un enseignant change de spécialité, il faut penser à faire cette MAJ dans tous les tuples où apparaît ce professeur.
Cependant on ne peut pas décomposer sans perte de dépendances fonctionnelles.

La décomposition (Prof,Matière) (NEtud,Prof) pert la DF (NEtud, Matière) → Prof. Cette perte de DF va permettre d'insérer le fait qu'un même étudiant suit deux cours portant sur la même matière avec deux professeurs différents, ce qui est interdit dans la relation enseignant (règle de gestion 2)

Il n'y a pas de solution idéale. Pratiquement on conserve la relation initiale et on ajoute une contrainte d'intégrité spécifiant qu'un professeur ne peut enseigner qu'une seule matière .

Université Badji Mokhtar
Faculté des sciences de l'ingénieur
Département informatique

Module Bases de données
2ième année LMD
Série 1 : Entité Association (révision)

Proposer des diagrammes entité-association qui modélisent les cas ci-dessous. Précisez les contraintes d'intégrité.

Exercice 1 : Aéroport

Pour les besoins de la gestion d'un aéroport on souhaite mémoriser dans une base de données les informations nécessaires à la description des faits suivants:
- chaque avion géré est identifié par un numéro d'immatriculation. Il est la propriété soit d'une société, soit d'un particulier: dans les deux cas on doit connaître le nom, l'adresse et le numéro de téléphone du propriétaire, ainsi que la date d'achat de l'avion;
- chaque avion est d'un certain type, celui-ci étant caractérisé par son nom, le nom du constructeur, la puissance du moteur, le nombre de places;
- la maintenance des avions est assurée par les mécaniciens de l'aéroport. Par sécurité, les interventions sont toujours effectuées par deux mécaniciens (l'un répare, l'autre vérifie). Pour toute intervention effectuée, on conserve l'objet de l'intervention, la date et la durée;
- pour chaque mécanicien on connaît son nom, son adresse, son numéro de téléphone et les types d'avion sur lesquels il est habilité à intervenir;
- un certain nombre de pilotes sont enregistrés auprès de l'aéroport pour chaque pilote on connaît son nom, son adresse, son numéro de téléphone, son numéro de brevet de pilote et les types d'avion qu'il est habilité à piloter avec le nombre total de vols qu'il a effectué sur chacun de ces types.

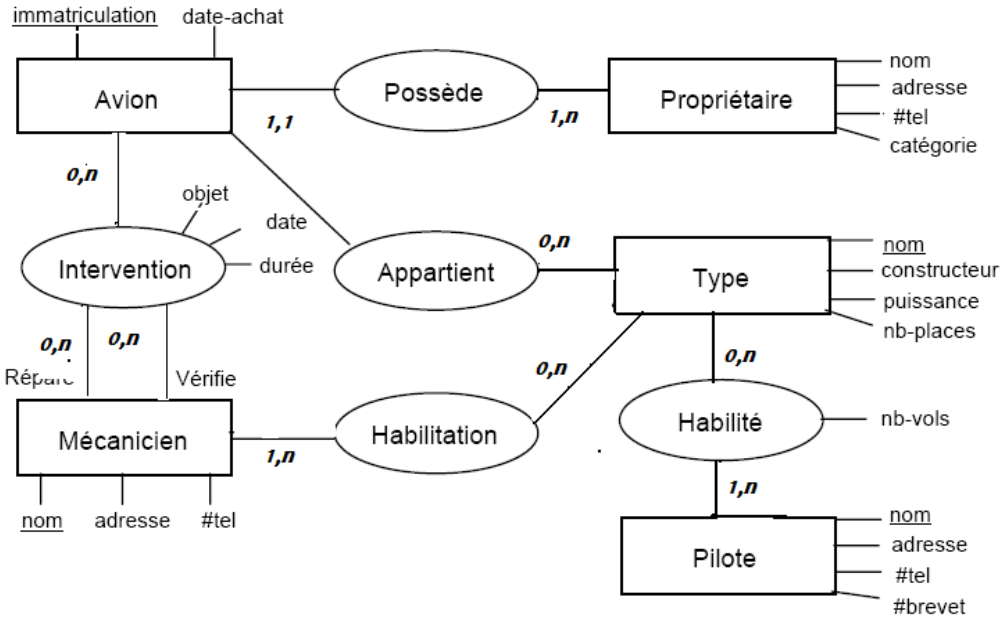
Des questions types auxquelles l'application doit pouvoir répondre sont les suivantes:
- liste des avions de la société "Voltige";
- liste des avions propriété de particuliers;
- durée totale des interventions faites par le mécanicien Durand au mois d'août;
- liste des avions de plus de 4 places, avec le nom du propriétaire;
- liste des interventions (objet, date) faites sur l'avion numéro 3242XZY78K3.

Exercice 2 : Personnel

On veut représenter le personnel d'une entreprise et son affectation. L'entreprise est organisée en services auxquels est affecté le personnel. Chaque service est décrit par son nom, son chef (qui est nécessairement un cadre de l'entreprise) et la liste de ses locaux. Le personnel est réparti en trois catégories, les administratifs, les techniciens et les cadres. Tous possèdent un numéro d'employé, un nom, un prénom, une adresse, une identification bancaire (nom banque, nom agence, numéro de compte), un salaire et sont rattachés à un service. Chaque catégorie possède en outre des renseignements qui lui sont propres:
- pour un administratif ou un technicien, le prix de l'heure supplémentaire;
- pour un technicien, les machines dont il est responsable;
- pour un administratif, le(s) cadre(s) pour le(s)quel(s) il travaille;
- pour un cadre, son bureau, son numéro de poste téléphonique et l'(les) administratif(s) (s'il en existe) qui lui est (sont) attaché(s).

Solution série 1
Entité Association

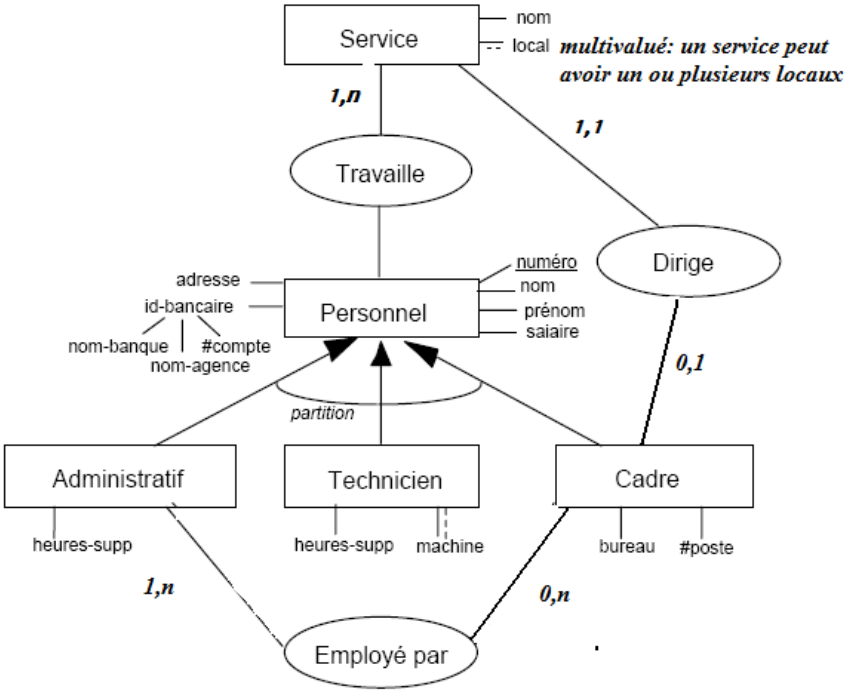
Exercice 1: Aéroport



Contrainte d'intégrité: Un mécanicien ne peut faire des interventions (en tant que vérificateur ou réparateur) que sur les types d'avions pour lesquels il est habilité : S'il existe un chemin Mécanicien-Vérifie-Intervention-Avion ou Mécanicien-Répare-Intervention-Avion liant une occurrence de Mécanicien et une occurrence d'Avion alors ces deux occurrences doivent être liées par un chemin Avion-Appartient-Type-Habilitation-Mécanicien.

Exercice 2 : Personnel

Cet exercice a pour but de faire travailler les étudiants sur lœextension du modèle EA pour supporter la notion dœhéritage. De plus un exemple dœattributs multi valués est présenté à savoir lœattribut « local » dans lœentité service et lœattribut « machine » dans lœentité technicien**.**



Autre solution :

Ce schéma peut être représenté sans les trois sous-types, Administratif, Technicien et Cadre. Dans ce cas, les attributs des trois sous-types sont rattachés à Personnel et deviennent facultatifs ; le fait que la valeur d'un de ces attributs ne soit pas nulle signifierait alors que l'employé appartient à cette catégorie de personnel.

Université Badji Mokhtar
Département d'informatique

Série 2 de TD : Du modèle Conceptuel au Modèle relationnel
Règles de passage

Exercice 1 : Institut de formation

Soit un institut de formation permanente qui veut gérer avec une base de données ses cours, ses enseignants et ses étudiants, avec leurs inscriptions et leurs résultats.
Les cours, identifiés par leur nom, sont répartis sur trois cycles (1, 2 et 3). Chaque cours peut avoir zéro, un ou plusieurs autres cours du même cycle ou des cycles précédents en prérequis.
Un enseignant, identifié par son nom, peut assurer un ou plusieurs cours; mais un cours est assuré par un seul enseignant. L'institut mémorise, pour chaque enseignant, ses nom, prénom, adresse, numéro de téléphone, statut (universitaire, professionnel...), et renseignements bancaires.
Les étudiants s'inscrivent à un ou plusieurs cours et paient un droit d'inscription pour chaque cours (qui est le même pour tous les cours).
Lors de sa première inscription à l'institut, l'étudiant reçoit un numéro qu'il conserve tout au long de sa formation. Chaque étudiant est décrit par son nom, prénoms, numéro, adresse, études antérieures (diplômes et année) et date de naissance. L'institut conserve, pour chaque étudiant, la liste des cours qu'il a obtenus, avec la note et l'année. On suppose que l'offre de cours est la même d'une année à l'autre.

Questions :

1) Elaborer un diagramme E/A pour la base de données de l'institut de formation. Préciser les contraintes d'intégrité.
2) Transformer le diagramme E/A en schéma relationnel

Exercice 2 : Bibliothèque

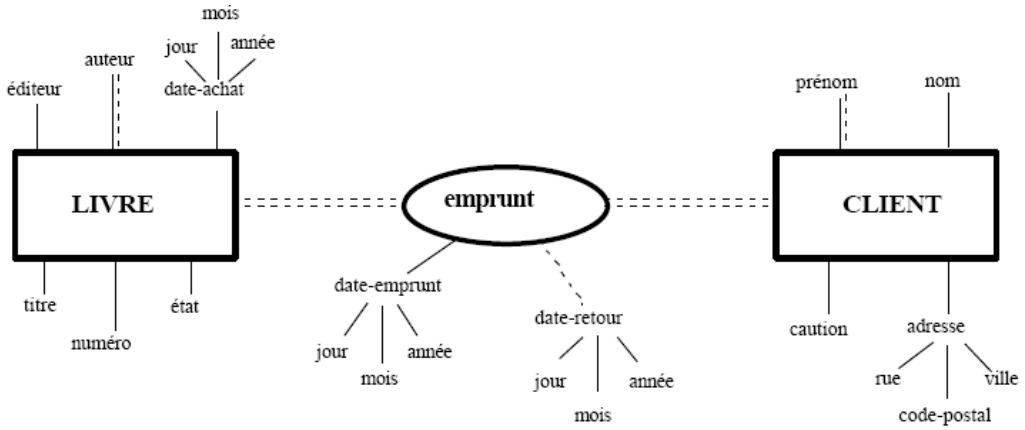
La Bibliothèque d'un syndicat intercommunal consiste en 5 points de prêt. Ces centres disposent d'ordinateurs personnels interconnectés qui doivent permettre de gérer les emprunts.
L'interview des bibliothécaires permet de déterminer les faits suivants:
- un client qui s'inscrit à la bibliothèque verse une caution. Suivant le montant de cette caution il aura le droit d'effectuer en même temps de 1 à 10 emprunts;
- les emprunts durent au maximum 8 jours;
- un livre est caractérisé par son numéro dans la bibliothèque (identifiant), son éditeur et son (ses) auteur(s);
- on veut pouvoir obtenir, pour chaque client les emprunts qu'il a effectué (nombre, numéro et titre du
livre, date de l'emprunt) au cours des trois derniers mois;
- toutes les semaines, on édite la liste des emprunteurs en retard : nom et adresse du client, date de l'emprunt, numéro(s) et titre du (des) livre(s) concerné(s);
- on veut enfin pouvoir connaître pour chaque livre sa date d'achat et son état.

1) Elaborez un diagramme entité-association pour la base de données de la Bibliothèque. Préciser les contraintes d'intégrité. Par exemple: pour chaque livre la date d'achat doit être antérieure aux dates d'emprunt.
2) Transformer le diagramme E/A en schéma relationnel.

Éléments de réponse Série II

Exo 1 : Bibliothèque

Schéma E/A



Contraintes d'intégrité:
- Un client a le droit d'emprunter en même temps de 1 à 10 livres selon le montant de la caution qu'il a versée.
- Si la date-retour existe, alors elle doit être supérieure à la date d'emprunt.
- Pour chaque livre la date-achat doit être inférieure aux dates d'emprunt.

Pour la transformation en schéma relationnel on applique les règles ci-dessus

Pour résoudre le problème des attributs multi-valués il faut mettre les relations en 1^{ière} forme normale.

Pour chaque relation R d'identifiant X, contenant un attribut multi-valué Y, on crée une relation R1 d'attribut X et Y ayant comme identifiant la concaténation de X et Y, et on supprime l'attribut multivalué de R (la relation initiale).

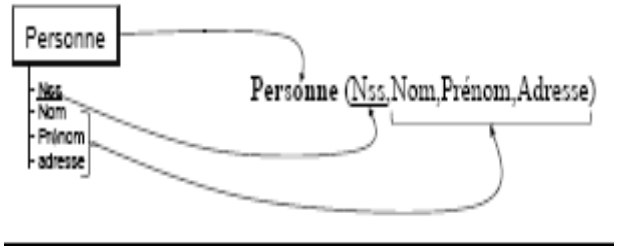
R(X, A,B, Y,…..) avec Y multivalué

On crée R1(X,Y) avec comme identifiant (X,Y) et on supprime Y de R on obtient :
R(X, A,B, ✗,…..)

Règles de passage du schéma conceptuel E/A vers le relationnel

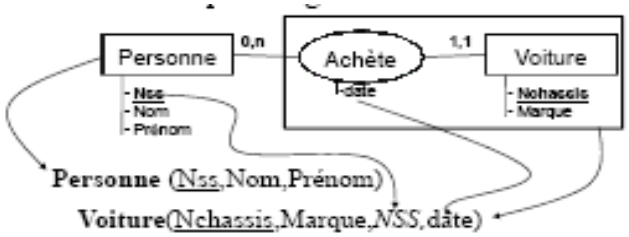
Étant donné un bon shéma E/A on peut passer au relationnel en appliquant des règles de passage :

Règle 1 :
Une entité **E** est représentée par une relation **T** dont les attributs simples sont les attributs de l'entité **E**. De plus, la clé de **T** est l'identifiant de **E**.



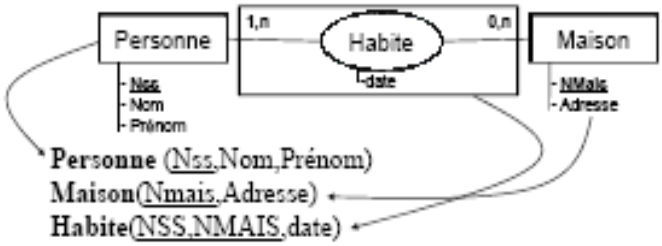
Règle 2 :

Dans le cas d'une association (Père/fils), les attributs de l'association migrent vers la relation représentant le fils et la clé du père migre vers le fils comme clé etrangère.

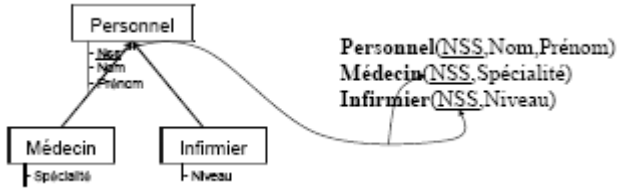


Règle 3 :

Dans le cas d'une association n:m l'association **A** est représentée par une relation **T** dont les attributs sont les attributs de **A** et la clé est la concaténation des clés des entités participant à l'association **A**.



Règle 4 : Généralisation spécialisation :



Module Base de Données
2LMD
Série 3 : Algèbre relationnelle

Soit la base de données relationnelle , PUF, de schéma :

U (NU, NomU, Ville)

P (NP, NomP, Couleur, Poids)

F (NF, NomF, Statut, Ville)

PUF (NP, NU, NF, Quantité)
NP référence P.NP
NU référence U.NU
NF référence F.NF

décrivant le fait que :

U: une usine est décrite par son numéro NU, son nom NomU, la ville Ville dans laquelle elle est située;

P: un produit est décrit par son numéro NP, son nom NomP, sa Couleur, son Poids;

F: un fournisseur est décrit par son numéro NF, son nom NomF, son Statut (fournisseur sous-traitant, fournisseur-client, ...), la Ville où il est domicilié;

PUF: le produit de numéro NP a été livré à l'usine de numéro NU par le fournisseur de numéro NF dans une Quantité donnée.

Exprimer en algèbre relationnelle les requêtes suivantes:

1) Donner le numéro, le nom et la ville de toutes les usines.

2) Donner le numéro, le nom et la ville de toutes les usines de Londres.

3) Donner les numéros des fournisseurs qui approvisionnent l'usine n° 1 en produit n° 1.

4) Donner le nom et la couleur des produits livrés par le fournisseur n° 1.

5) Donner les numéros des fournisseurs qui approvisionnent l'usine n° 1 en un produit rouge.

6) Donner les noms des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge.

7) Donner les numéros des produits livrés à une usine par un fournisseur de la même ville.

8) Donner les numéros des produits livrés à une usine de Londres par un fournisseur de Londres.

9) Donner les numéros des usines qui ont au moins un fournisseur qui n'est pas de la même ville.

10) Donner les numéros des fournisseurs qui approvisionnent à la fois les usines n° 1 et n° 2.

11) Donner les numéros des usines qui utilisent au moins un produit disponible chez le fournisseur n° 3 (c'est-à-dire un produit qu'il livre mais pas nécessairement à cette usine).

12) Donner les numéros des produits qui sont livrés **à toutes** les usines dôAnnaba .

13) Donner les numéros des usines qui sôapprovisionnent uniquement chez le fournisseur n°3.

Université Badji Mokhtar Annaba
Faculté des Sciences de lôingénieur
Département Informatique

Solution TD3 : Algèbre relationnelle

Soit la base de données relationnelle PUF, de schéma :

U (NU, NomU, Ville)
P (NP, NomP, Couleur, Poids)
F (NF, NomF, Statut, Ville)
PUF (NP, NU, NF, Quantité)
NP référence P.NP
NU référence U.NU
NF référence F.NF

Exprimer en algèbre relationnelle les requêtes suivantes :

1) Donner le numéro, le nom et la ville de toutes les usines :

   Π [NU, NomU, Ville]U

2) Donner le numéro, le nom et la ville de toutes les usines dôAnnaba :

   σ [Ville=''londres'']U

3) Donner les numéros des fournisseurs qui approvisionnent les usines numéro1 en produit numéro1 :

   Π [NF] σ [NU=1 et NP=1]PUF

4) Donner le nom et la couleur des produits livrés par le fournisseur n°1 :

   Π[NomP, Couleur] P $\bowtie$ σ [NF=1] PUF

5) Donner les numéros des fournisseurs qui approvisionnent lôusine n°1 en un produit rouge :

   Π[NF] σ [NU=1 et Couleur= ''rouge''] P $\bowtie$ PUF (c'est une jointure naturelle)

   Ou bien avec la thêta jointure (il faut renommer)

   P1=α [NP :NP1] P (renommage pour préparer la théta jointure)

   Π[NF] [PUF $\underset{[NP=NP1 \text{ et } NU=1]}{\bowtie}$ (Π[NP1] σ [Couleur= ''rouge'']) P1]

6) Donner les noms des fournisseurs qui approvisionnent une usine dôAnnaba ou dôAlger en un produit rouge :

Π[NomF] σ [(Ville=ôôAnnabaôô ou Ville=ôôAlgerôô) et Couleur= ''rouge''] PUF $\bowtie$ U $\bowtie$ P $\bowtie$ Π[NF,NomF] F

7) Donner les numéros des produits livrés à une usine par un fournisseur de la même ville :

   Π[NP] PUF $\bowtie$ Π[NF,Ville] F $\bowtie$ Π[NU, Ville] U

8) Donner les numéros des produits livrés à une usine dôAlger par un fournisseur dôAlger :

   Π[NP] (PUF $\bowtie$ σ [Ville= ''Alger''] (F $\bowtie$ U))

9) Donner les numéros des usines qui ont au moins un fournisseur qui nôest pas de la même ville :

On doit renommer car il y a une condition dôinégalité entre les villes des fournisseurs et des usines

U1=α [Ville : Ville1], PUF1=α [NF :NF1, NU : NU1] PUF

$\Pi[NU1]\ (F \underset{[NF=NF1 \text{ et } Ville \neq Ville1]}{\bowtie} (PUF1 \underset{[NU=NU1]}{\bowtie} U1))$

10) Donner les numéros des fournisseurs qui approvisionnent à la fois les usines n°1 et n°2 :

Jointure sur une même table PUF, renommage obligatoire

PUF1= α (NP :NP1, NU : NU1, NF : NF1, Quantité : Quantité1)PUF

$\Pi[NF]\ (PUF \underset{[NF=NF1 \text{ et } NU=1 \text{ et } NU1=1]}{\bowtie} PUF1)$

11) Donner les numéros des usines qui utilisent au moins un produit disponible chez le fournisseur n°3 (cêst-à-dire un produit quôil livre mais pas nécessairement à cette usine) :

$\Pi[NU]\ (\ PUF \bowtie (\Pi\ [NP]\ \sigma\ [NF=3]\ PUF)\ )$

12) Donner les numéros des produits qui sont livrés **à toutes** les usines dôAnnaba :

Il sôagit dôune division : (produit, usine)/usine (restriction)=produit donc

$\Pi[NP, NU]\ PUF\ /(\Pi[NU]\ (\sigma\ [Ville=\text{ôôAlgerôô}]\ U))$

13) Donner les numéros des usines qui sôapprovisionnent uniquement chez le fournisseur n°3 :

Il sôagit de la soustraction

$(\Pi[NU]\ \sigma\ [NF=3]\ PUF) - (\Pi[NU]\ \sigma\ [NF \neq 3]\ PUF)$

Remarque 1 : pour toutes les requêtes on peut répondre par étapes. Exemple avec la requête 13

R1=σ [NF=3] PUF
R2= Π[NU] R1
R3= σ [NFÍ 3] PUF
R4= Π[NU] R3
R5= R2 ï R4

Remarque 2 : Ce TD sera résolu en TP où vous trouverez une base de données et chaque réponse est accompagnée dôun exemple dôexécution de la requête (Voir TP 2 TD 3 SQL).

Remarque 3 : les opérateurs de division et de soustraction nôexistent pas en SQL.

Exemple de Modèle relationnel :

Enoncé :

Soit un institut de formation permanente qui veut gérer avec une base de données ses cours, ses enseignants et ses étudiants, avec leurs inscriptions et leurs résultats.

Les cours, identifiés par leur nom, sont répartis sur trois cycles (1, 2 et 3). Chaque cours peut avoir zéro, un ou plusieurs autres cours du même cycle ou des cycles précédents en prérequis.

Un enseignant, identifié par son nom, peut assurer un ou plusieurs cours; mais un cours est assuré par un seul enseignant. L'institut mémorise, pour chaque enseignant, ses nom, prénom, adresse, numéro de téléphone, statut (universitaire, professionnel...), et renseignements bancaires.

Les étudiants s'inscrivent à un ou plusieurs cours et paient un droit d'inscription pour chaque cours (qui est le même pour tous les cours).

Lors de sa première inscription à l'institut, l'étudiant reçoit un numéro qu'il conserve tout au long de sa formation. Chaque étudiant est décrit par ses nom, prénoms, numéro, adresse, études antérieures (diplômes et année) et date de naissance. L'institut conserve, pour chaque étudiant, la liste des cours qu'il a obtenus, avec la note et l'année. On suppose que l'offre de cours est la même d'une année à l'autre.

Schéma base de données relationnelles

Domaines:

Dnom : chaînes de caractères de longueur inférieure à *30*
Dch100 : chaînes de caractères de longueur inférieure à *100*
Dannée : [*1970* : *1990*]
Dnote : [*0.0* : *10.0*]
Ddate : [*1* :*31*] / [*1* :*12*] / [*1920* :*2100*]

Relation : *Etudiant*
Attributs: *n°E* : entier sans nul
Nom :Dnom sans nul
dateN : *Ddate* sans nul
Identifiants: (*n°E*)
Définition: *tout individu qui est actuellement inscrit à l'institut, ou qui a déjà passé avec succès un des cours de l'institut*

Relation : *PrénomEtudiant*
Attributs : *n°E : entier sans nul*
Prénom : Dnom san nul
Identifiants : *(n°E+Prénom)*
Identifiant externe *n°E référence un Etudiant*

Relation : *PrénomEnseignant*
Attributs : *Nom : Dnom sans nul*
Prénom : Dnom san nul
Identifiants : *(Nom+Prénom)*
Identifiant externe *Nom référence un Enseignant*

Relation : *EtudiantEtudes*
Attributs: *n°E* : entier sans nul
année : *Dannée* sans nul
diplôme : *Dnom* sans nul
Identifiant: (*n°E + diplôme*)
Identifiant externe : *n°E* référence un *Etudiant.n°E*
Définition: études antérieures des étudiants

Relation : Enseignant
Attributs: *Nom* :Dnom sans nul
tel: : entier sans nul
*statu*t : Dnom sans nul
banque : Dnom sans nul
agence : Dnom sans nul
compte : entier sans nul
Identifiant: (Nom)
Définition: tout individu assurant actuellement un ou plusieurs cours à l'institut

Relation : Cours
Attributs: nomC : Dnom sans nul
cycle : entier sans nul
n°Ens : entier sans nul
Identifiant: (nomC)
Identifiant externe : n°Ens référence un Enseignant
Définition: tout cours offert par l'institut

Relation : Obtenu
Attributs: n°E : entier sans nul
nomC : Dnom sans nul
note : Dnote sans nul
année : Dannée sans nul
Identifiant: (n°E + nomC)
Identifiants externes : n°E référence un Etudiant.n°E
nomC référence un Cours
Définition: l'étudiant n°E a réussi le cours nomC telle année et a obtenu telle note

Relation : Inscrit
Attributs: n°E : entier sans nul
nomC : Dnom sans nul
Identifiant: (n°E + nomC)
Identifiants externes : n°E référence un Etudiant.n°E
nomC référence un Cours
Définition: actuellement, l'étudiant n°E est inscrit au cours nomC

Relation: Prérequis

Attributs: nomC : Dnom sans nul
nomCprérequis : Dnom sans nul
Identifiant: (nomC + nomCprérequis)
Identifiants externes : nomC référence un Cours
nomCprérequis référence un Cours
Définition: le cours nomCprérequis est un prérequis pour le cours nomC
Contrainte d'intégrité: dans tout tuple, nomCprérequis doit être différent de nomC

Contraintes d'intégrité:

Pour tout tuple de Prérequis <nomC, nomCprérequis>, le cycle de nomCprérequis dans Cours doit être inférieur ou égal à celui de nomC.
Pour tout tuple de Etudiant: année_en_cours - dateN ≥ 18 53
Pour tout tuple de EtudiantEtudes <n°E , année, diplome>, soit dateN la date de naissance de l'étudiant dans la relation Etudiant, alors: dateN < année

Université Badji Mokhtar — Faculté des sciences
de l'ingénieur
Département d'informatique

Révision 2018-2019

Exercice 1:

Soit R1 (A,B,C,D,E,F) une relation avec l'ensemble X des dépendances Fonctionnelle suivantes :
X={AB → C, D,E ; B →C; D → E ; F → B ; F → C}

1) Donner le **graphe minimum** des dépendances. Justifier les étapes de transformation du graphe de dépendance d'origine. Quelle est la clé de R1?

2) Proposer une décomposition en 3ième forme normale de R1. Y-a-t-il perte d'information ou dépendance? justifier.

3) On ajoute la dépendance E → F à l'ensemble X.

a) Quelles sont les clés candidates de la nouvelle relation R?

b) La nouvelle relation est-elle en FNBC? justifier.

c) Décomposer R en FNBC.

Problème

Soit le schéma de la base de données CINEMA :

FILM (Num-F **CHAR(5),** Titre **VARCHAR(30),** année **NUMBER(4),** Durée **NUMBER(3),** Budget **NUMBER(6),**Réalisateur***CHAR(3)**, Salaire-R **NUMBER(4)),**
GENERIQUE (Film **CHAR(5),** Acteur **CHAR(3),** Rôle **VARCHAR(20),** Salaire **NUMBER(4)),**
PERSONNE (Num-P **CHAR(3),** Nom **VARCHAR(20),** Prénom **VARCHAR(20),** Date-Nais **DATE,** Sexe **CHAR(1),** Adresse **VARCHAR(30),** Tél **NUMBER(8)),**
ACTEUR (Num-A***CHAR(3),**Spécialité **VARCHAR(10))**
CINEMA (Num-C **CHAR(3),** Nom **VARCHAR(20),** Adresse **VARCHAR(20),** Téléphone **NUMBER(8),** Compagnie **VARCHAR(10)),**
PROJECTION (Film **CHAR(5),** Cinéma **CHAR(3),** Salle **NUMBER(2),** Date-Deb **DATE,** Date-Fin **DATE,** Horaire **NUMBER(4),** Prix **NUMBER(3)),**
SALLE (Cinéma***CHAR(3),** Num-S **NUMBER(2),** Taille-Ecran **NUMBER(6),** Places **NUMBER(3)),**
RECOMPENSE (Num-R **NUMBER(2),** Nom **VARCHAR(20) ,** Festival **VARCHAR(10)),**
RECOMP-FILM (Film **CHAR(5),** Récompense **NUMBER(2),** Année **NUMBER(4)**)
RECOMP-ACTEUR (Acteur **CHAR(3),** Récompense **NUMBER(2),** Année **NUMBER(4)**)

Une personne peut être soit un acteur, soit un réalisateur. L'attribut Sexe prend ses valeurs dans {F,M}. Un film est réalisé par un seul réalisateur et fait jouer plusieurs acteurs.
la relation GENERIQUE représente les acteurs identifiés par l'attribut Acteur qui ont joué dans le même film identifié par l'attribut Film avec le Rôle de chaque acteur et son Salaire dans le film.

1) En utilisant un tableau,

relation	Clé primaire	Intégrité référentielle

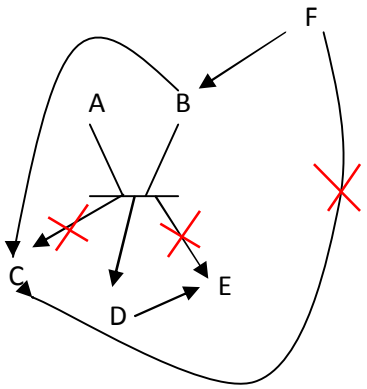
indiquer les clés primaires (les clés concaténées doivent être séparées par "+") et les contraintes d'intégrité référentielles (la table référencée avec sa clé doit être indiquée) des relations suivantes:
- GENERIQUE, PROJECTION, SALLE, RECOMP-FILM, RECOMP-ACTEUR répondant aux règles suivantes:
Un film fait jouer plusieurs acteurs. Un film peut être projeté plusieurs fois (à des heures différentes) dans le même cinéma, le même jour et dans la même salle. Un cinéma peut avoir plusieurs salles. Les récompenses sont identifiées par un code unique. Un film peut avoir plusieurs récompenses. Un acteur peut aussi avoir plusieurs récompenses .

2) Ecrire en SQL LDD les commandes pour la création des tables suivantes:
FILM, GENERIQUE, ACTEUR et PERSONNE. Dans quel ordre ces tables doivent être créées? justifier votre réponse.

CORRIGE

Exercice 1: 5 points

1) Graphe minimum des DF 0,75 pt (0,25 pt pour chaque DF supprimée)



Justification

DF B-->C, AB-->C donc AB--> C est augmentée (non élémentaire) à enlever

AB-->D, D-->E ⇒ AB-->E est déduite Donc à enlever

F-->B et B-->C ⇒ F-->C est déduite Donc à enlever

Clé de R1 (F,A)

2) Décomposition en 3FN

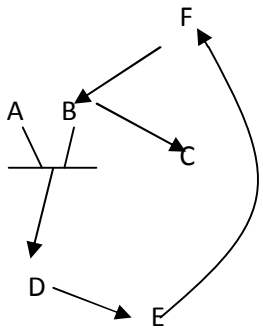
R11(**D**,E)

R12(**A,B**,D)

R13(**F**,B)

Il n'ya pas perte d'information car toute décomposition qui tient compte des DF aboutissant à 3FN c-à-d que tous les attributs non clé dépendent de la clé entière (source de DF) est sans perte d'information. ..... 0,25 pt

3)



a) les clés candidates sont (A,F) et (A,E).

b) la nouvelle relation R(A,B,C,D,E,F) n'est pas en FNBC car toute source de DF n'est pas un identifiant entier. C'est le cas des DF A,B-->D , F-->B et E-->F

c) Décomposition en FNBC :

- R'1 (E,F)
- R'2(A,F,D)
- R'3(B,C)
- R'4(D,E).

Problème: 13 points

1)

Relation	Clé primaire	Intégrité référentielle	
GENERIQUE	Film+Acteur.....	.....	0,25
		Film référence FILM(Num-F).....	0,25
		Acteur référence ACTEUR(Num-A).....	0,25
PROJECTION	Film+Cinéma+Salle+Date-Deb+Horaire	.....	0,25
		Film référence FILM(Num-F).....	0,25
		Cinéma référence CINEMA (Num-C).....	0,25
SALLE	Cinéma+Num-S	.....	0,25
		Cinéma Référence CINEMA(Num-C).....	0,25
RECOMP-FILM	Film+Récompense	.....	0,25
		Film référence film(Num-F).....	0,25
		Récompense référence RECOMPENSE(Num-R).....	0,25
RECOMP-ACTEUR	Acteur+Récompense	.....	0,25
		Acteur référence ACTEUR(Num-A).....	0,25
		Récompense référence RECOMPENSE(Num-R).....	0,25

2)

CREATE TABLE **FILM** (Num-F **CHAR(5)NOT NULL**, Titre **VARCHAR(30)NOT NULL**, année **NUMBER(4)NOT NULL**, Durée **NUMBER(3)**, Budget **NUMBER(6)**,Réalisateur **CHAR(3)**, Salaire-R **NUMBER(4)**,

PRIMARY KEY Num-F, ...0,25

UNIQUE(Titre,Année),

FOREIGN KEY Réalisateur REFERENCES PERSONNE(Num-P)ON DELETE CASCADE ON UPDATE CASCADE), ...0,25

ou bien (contrainte nommée)

Constraint un_film unique (Titre, Année)

Constraint pk_film Primary key (Num-F),

Constraint fk_pers_film Foreign Key realisateur references personne (Num-P)...

N.B toutefois lorsque la contrainte porte sur un seul attribut elle peut être déclarée en colonne(en ligne) exemple Num-F char(5) Primary Key. Dans ce cas

il est inutile de mettre NOT NULL car elle est implicite lorsque l'attribut est Clé Primaire.

CREATE TABLE GENERIQUE (Film **CHAR(5)NOT NULL REFERENCES FILM(Num-F),**0,25

Acteur **CHAR(3)NOT NULL REFRENCE ACTEUR(NUM-A),** 0,25

Rôle **VARCHAR(20) NOT NULL,**

Salaire **NUMBER(4),**

PRIMARY KEY (Film,Acteur)), ...0,25

oubien

Constraint pk_gen primary key film,

constraint fk_film_gen foreign key film references film(num-F) ....)

CREATE TABLE ACTEUR (Num-A **CHAR(3)PRIMARY KEY,**.....0,25

Spécialité **VARCHAR(10),**

FOREIGN KEY NUM-A REFERENCES PERSONNE(Num-P)) 0,25

CREATE TABLE PERSONNE (Num-P **CHAR(3)PRIMARY KEY,** 0,25

Nom **VARCHAR(20) NOT NULL,**

Prénom **VARCHAR(20),** Date-Nais **DATE,**

Sexe **CHAR(1)CHECK IN {'M','F'},** 0,25

Adresse **VARCHAR(30),** Tél **NUMBER(8)**)

Université Badji Mokhtar
Département d'informatique

Série de TD
Modèle relationnel : Normalisation d'une relation

Exercice1
Pour chaque relation ci-dessous:
- identifier les redondances éventuelles dans sa population,
- établir le (un) graphe minimum de ses dépendances,
- définir son (ses) identifiant(s),
- définir sa forme normale et la justifier,
- si nécessaire, proposer une décomposition optimale.

a. Pièce: description des pièces employées dans un atelier de montage.
Pièce (N°pièce, prix-unit, TVA, libellé, catégorie)
avec les dépendances fonctionnelles suivantes:
N°pièce → prix-unit, TVA, libellé, catégorie
catégorie → TVA

b. Prime: liste des primes attribuées au personnel technique en fonction du type de machine sur lequel il travaille
Prime (N°type-machine, nom-machine, N°techn, montant-prime, nom-techn)
avec les dépendances fonctionnelles suivantes:
N°type-machine → nom-machine
N°techn → nom-techn
(N°type-machine, N°techn) → montant-prime

c. Employé: description d'un employé travaillant sur un projet d'un laboratoire.
Employé (NoEmp, NoLab, NoProj, NomEmp, NomProj, adresse)
avec les dépendances fonctionnelles suivantes:
(NoEmp, NoLab) → NoProj
NoEmp → NomEmp
NoEmp → adresse
NoProj → NomProj

d. Adresse
Adresse (rue, ville, code-postal)
avec les dépendances fonctionnelles suivantes:
code-postal → ville
(rue, ville) → code-postal

Exercice 2
Soit R la relation suivante, avec les dépendances:
R (A, B, C, D, E, F, G)
AB → C, AB → D, AB → E, AB → F, B → C, D → E, D → F, G → A
a. Etablir le (un) graphe minimum de dépendances. Quel(s) est (sont) l'identifiant(s) de R ?
b. Quelle est la forme normale de R? Justifier votre réponse.
c. Proposer une décomposition optimale de R.

Exercice 3 :
Reprendre les exercices de la série 1 et proposer une décomposition optimale.

Corrigé SEFIE2

Exo 1 :

a) Pièce (N° pièce, prix unitaire, TVA, libellée, catégorie)

DF: N° pièce → prix_unitaire, TVA, libellé, catégorie
Catégorie → TVA

- Redondances éventuelles

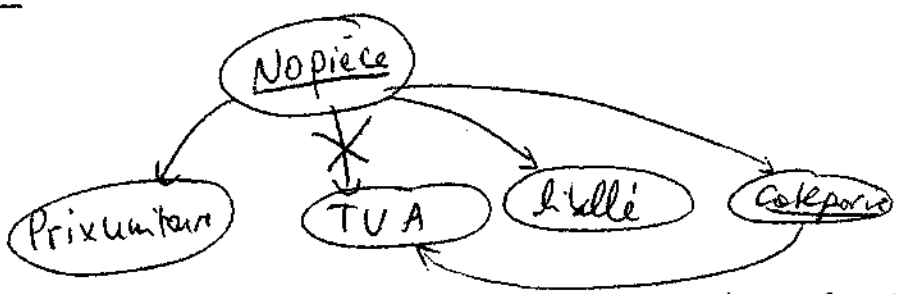
Pièce

N° pièce	prix unitaire	TVA	libellé	Catégorie
123	1230	10	qwerty	2
129	1650	10	axby	2
230	1300	17	xxx	3
530	2600	10	cccc	2
⋮				

s'il y a 1000 pièces de catégorie 2 la TVA=10 sera répété 1000 fois

- Graphe minimum des dépendances : Sans DF Augmentée et sans DF déduite

N° pièce → TVA (DF déduite) à enlever



- identifiants : l'ensemble minimum d'attributs qui nous mène vers tous les autres attributs (dans le graphe minimum)
identifiant N° pièce, Catégorie
- 1FN OK (Tous les attributs sont atomiques)
2FN tous les attributs ≠ identifiant doivent dépendre de l'identifiant entier
TVA dépend seulement de Catégorie. donc la Relation pièce n'est pas en 2ème FN.
Décomposition

pièce (Nopiece, prixunitaire, libellé, Catégorie)
Categ (Catégorie, TVA)

3FN OK : tous les Attributs ≠ identifiant ne dépendent que des Identifiants (il n'y a pas d'attribut ≠ identifiant qui dépend d'un Attribut autre que l'identifiant

b) Prime (Notypemachine, nom_machine, Notech, Nom_tech, montantprime)

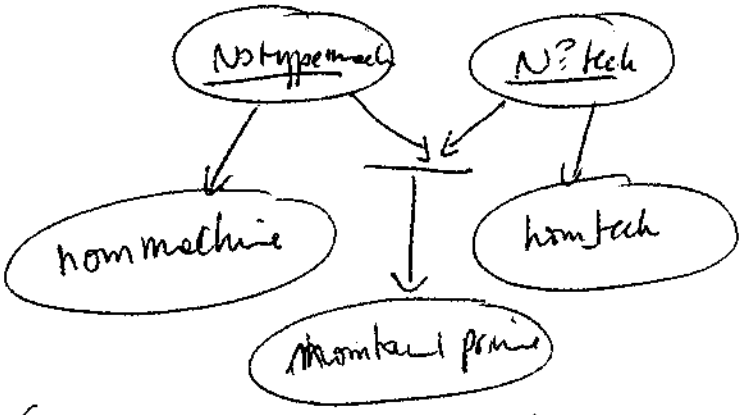
DFs: N° type machine → nom_machine

~~N° type~~ No tech → nom_tech.

(Notypemach, No tech) → montant prime

- Redondances Eventuelles : s'il y a plusieurs tech qui travaillent sur la m machine identifié par Notypemachine, son nom est répété plusieurs fois

-



Graphe minimum OK

- Identifiant (Notypemachine, N° tech)
- 1FN OK
- 2FN non Décomposition R1 (Notypemach, No tech, montantprime)
  R2 (N° type machine, nom machine)
  R3 (No tech, Nom_tech)
- 3FN OK

---

c) Employé (NoEmp, NoLABS, NoProj, NomEmp, NomProj, @)

DF: (NoEmp, NoLABS) → NoPROJ

NoEmp → NomEmp ; NoEmp → adresse ; NoPROJ → NomPROJ

les DFs montrent qu'un employé peut être dans plusieurs projets à conditions que les projets sont dans le labo ≠ts

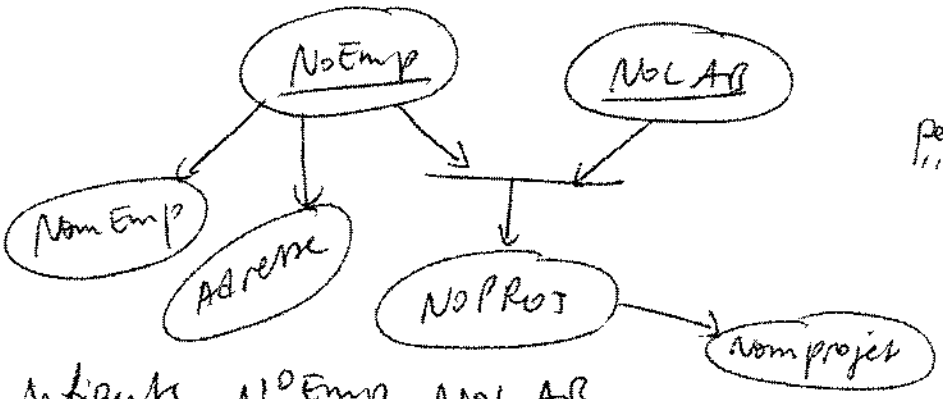
Redondances Employé et le nom Employé

NoEmp	NoLABS	NoPROJ	NomEmp	NomPROJ	@
196	1	2	aaaa	XXXX	4
128	1	2	bbbb	XXXX	2
196	2	3	aaaa	bbbXX	1

l'adresse de l'employé se répètent autant de fois qu'il est employé dans des projet ≠ts.

s'il y a 100 Employés dans un Projet le nom est répété autant de fois du projet

- Graphe minimum des dépendances



Graphe minimum de dép
pas de DF déduite
" " " Armstrong

- Identifiants N°Emp, NoLAB
- 1FN OK
* 2FN non OK Décomposition ~~N°~~ Employé (NoEmp, NomEmp, Adresse,
R1 (NoEmpl, NoLAB, NoPROJ, Nom

3FN non OK pour R1 (1 Attribut NomProj dépend NoPROJ ≠ ~~Attribut~~ Identifiant

Décomposition 1 ⇒ R'1 (NoEmpl, NoLAB, NoPROJ)
R''1 (NoPROJ, NomPROJ)

Exo2

$R(A, B, C, D, E, F, G)$

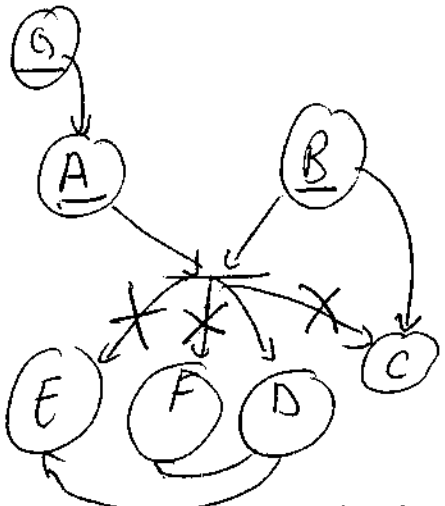
$AB \rightarrow C, D, E, F$

$B \rightarrow C$

$D \rightarrow E$

$D \rightarrow F$

$G \rightarrow A$



Graphe minimum de dépendance

$AB \rightarrow F$ est déduite on peut arriver à F en passant par le chemin $AB \rightarrow D \rightarrow F$

même chose $AB \rightarrow E$

$B \rightarrow C$ } donc $AB \rightarrow C$ est Augmenté à enlever (non élémentaire)
$AB \rightarrow C$ }

Identifiants $\underline{G}, \underline{A}, \underline{B}$

Forme normale 1FN OK
2FN non Décomposition

$R1(\underline{G}, A)$

$R2(\underline{A}, \underline{B}, D, E, F)$

$R3(\underline{B}, C)$

E et F dépendent car $D \rightarrow E$ de D qui n'est pas $D \rightarrow F$ Identifiant

3FN non OK pour la Relation R2 / Décomposition

$R2'(\underline{A}, \underline{B}, D)$

$R2''(\underline{D}, E, F)$

Corrigé micro BDD 2018-2019

Exercice 1 : (7 points)

Personne(nss,adresse,département*).....0,5

Nss clé primaire, ...0,5

Département référence département(dnom)..... 0,5

Personne_tel(nss*,téléphone).....0,5

Clé primaire (nss+téléphone) 0,5

nss référence personne(nss)..... 0,5

département (dnom, adresse, Maire*)..... 0,5

dnom clé primaire..... 0,5

Maire référence Maire(nss)..... 0,5

Maire(nss*, nom)..... 0,5

nss clé primaire référence personne (spécialisation)..... 0,5

budget (bnum, montant, annee,dnom*).... 0,5

bnum clé primaire..... 0 ,5

dnom référence département(dnom)..... 0,5

Exercice 2 Algèbre relationnelle 8

1. Lister les noms et prénoms des employés d'Annaba actuellement en congé dans cette liste on désir avoir les libellés des congés. Total 2,5 pts

$$\pi_{(nom,\ prénom,\ LibelléCongé)}\ \sigma_{(DateD<=\ '07/05/2016'\ et\ DateF>=\ '07/05/2016'\ et\ ville='Annaba')}(Employé) \bowtie_{Matricule} EnCongé \bowtie_{CodeC} Congé$$

2,5 points répartis comme suit :

- jointure congé et EnCongé avec attribut de jointure (CodeC) 0,5 pt (sans attribut de jointure ... 0,25)
- jointure Encongé et Employé avec attribut de jointure (Matricule) 0,5 pt (sans attribut de jointure ... 0,25)
- selection 0,25 pt pour chaque condition
- projection 0,25 pour chaque attribut

2. Lister les noms et prénoms des employés du service 'Comptabilité' qui sont actuellement en congé. Total 2,75 pts

$\pi_{(nom, prénom)}\ \sigma_{(DateD<= '07/05/2016' \text{ et } DateF>= '07/05/2016' \text{ et } libelléService='Comptabilité')}$ Employé $\bowtie_{Matricule}$ EnCongé $\bowtie_{Matricule}$ Affecté $\bowtie_{NumSce}$ Service

- jointure Service et Affecté avec attribut de jointure (NumSce) 0,5 (sans attribut de jointure ... 0,25)
- Jointure Affecté et EnCongé avec attribut de jointure (Matricule) 0,5 (sans attribut de jointure ... 0,25)
- jointure EnCongé avec Employé avec attribut de jointure (Matricule) 0,5 ((sans attribut de jointure ... 0,25)
- selection 0,25 pour chaque condition
-projection 0,25 pour chaque attribut

3. Lister les matricules, noms et prénoms des employés qui ne sont jamais sortis en congé..... 2 pt

$\pi_{(Matricule, nom, prénom)}$ (Employé) - [$\pi_{(Matricule, nom, prénom)}$ Employé $\bowtie_{Matricule}$ EnCongé]

- Jointure employé et EnCongé en précisant l'attribut de jointure ...0,5 (0,25 sans attribut de jointure)
- projection sur les trois attributs sur la table résultant de la jointure ....(0,5) 0,25 sans attribut de jointure)
- projection sur les trois attributs de la table employé 0,5
- opérateur de soustraction 0,5

4. Lister les matricules des employés qui ont travaillé dans tous les services. ..0,75 pts (0,25 projection sur la relation Affecter, 0,25 projection sur la relation service et 0,25 sur l'opérateur de division lorsqu'il est conforme)

$\pi_{(Matricule, NumSce)}$ (Affecter)/ $\pi_{(NumSce)}$ (Service)

Exercice 3 5 points

UNION: R5=R1 U R2 .....1 pt

A	B	C
a1	b2	c1
a2	b3	c2
a3	b4	c2
a1	b1	c1
a2	b2	c3
a4	b4	c4

INTERSECTION: R6=R1 ž R2 = Φ .....1 pt

PRODUIT CARTESIEN : R7=R1 x R3 ……1 pt

A	B	C	A	C
a1	b2	c1	a1	c1
a1	b2	c1	a2	c2
a2	b3	c2	a1	c1
a2	b3	c2	a2	c2
a3	b4	c2	a1	c1
a3	b4	c2	a2	c2

JOINTURE : R8=R2 Join R3 ……..1 pt

A	B	C
a1	b1	c1

DIVISION : R9= R4 ÷ R3 ……1pt

B
b2

CORRIGE EMD BDD
2018-2019

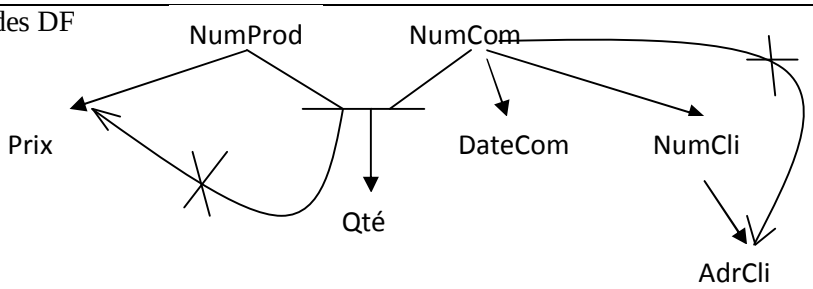
Exercice 1: (4 points)
Soit la relation suivante:

Commandes (NumCom, DateCom, NumCli, AdrCli, NumProd, Prix, Qte)
a. NumCli →AdrCli;
b. NumCom → NumCli, DateCom, AdrCli ;
c. NumProd → Prix;
d.NumCom,NumPro→Prix, Qté

1) Classer les dépendances Fonctionnelles en:

DF Elémentaire	DF Non élémentaire	FD déduite
√ a 0,25 □ b √ c 0,25 □ d	□ a √ b 0,25 □ c √ d 0,25	□ a □ b NumCom→AdrCli 0,25 □ c □ d

2) Etablir le Graphe minimum des DF



0,75 (0,25 .... la DF augmentée
+0,25.....DF déduite
+0,25..... pour tout le GM correct)

3) Identifier la ou les clé(s) Candidate (s)	(NumProd+NumCom) 0,25

4) Décomposer en 2FN *	5) Décomposer en 3FN*
R1 (<u>NumCom</u>, DateCom, NumCli, AdrCli) 0,25 R2 (<u>NumProd</u>, Prix) 0,25 R3(<u>NumProd,NumCom</u>,Qté) 0,25 Les clés doivent être soulignées sinon -0,5 sur les questions 5 et 6	R11(<u>NumCom</u>, DateCom, NumCli) 0,25 R12(<u>NumCli</u>, AdrCli) 0,25 R2 (<u>NumProd</u>, Prix) 0,25 R3(<u>NumProd,NumCom</u>,Qté) 0,25

** indiquer les clés primaires des relations obtenues après décomposition*

Exercice 2: (4,5 points)

Une base de données de schéma R(A,B,D) et Q(A,B) contient les n-uplet suivants:

R

A	B	D
a1	b1	d1
a1	b1	d2
a1	b2	d1
a2	b2	d1
a3	b1	d2
a2	b2	d2
a1	b2	d3
a1	b1	d4

Q

A	B
a1	b1
a2	b2

Question: Déterminer l'extension des tables résultant de ces opérations algébriques (*on rappelle que la projection en algèbre relationnelle élimine les doublons).*

1.

R1= α(A:A1,B:B1,D:D1) R

J = Π[A1,B1,A,B](Π [A1,B1] R1 ⋈ R)

[A1Í A et B1=B]

A1	B1	A	B
a1	b1	a3	b1
a1	b2	a2	b2
a2	b2	a1	b2
a3	b1	a1	b1

0,5 schéma
1 pour toute l'extension correcte sinon 0

2.

u = Π[A,B]R÷ Π[A]Q

B
b2

0,5 pour le schéma et
0,5 pour l'extension

3.

V= R ⋈ (Π[A]R-Π[A]Q)

A	B	D
a3	b1	d2

0,5 schéma
0,5 pour l'extension

4.

W= Π [A,B]R - Q

A	B
a1	b2
a3	b1

0,5 schéma
0,5 extension

Exercice 3: (11,5 points)

Soit le schéma de la base de données SPECTACLE :

```
1) SPECTACLE (Num-SPECTACLE NUMBER (3), NOM VARCHAR(30), Durée NUMBER(3), TYPE VARCHAR(7)) ;/*TYPE ={"théatre"|"concert"|"Danse"}*/

2) SALLE (NUM-S NUMBER(2), Nbr-Places NUMBER(3)),

3) REPRESENTATION (Date TIMESTAMP, Num-Spec NUMBER (3), NUM-SALLE NUMBER(2), PRIX DECIMAL).
```

I/ Après avoir identifier les clés primaires et étrangères, écrire les commandes SQL pour la création des tables. On vous demande de **nommer** les contraintes de clés primaires, d'intégrité référentielle et de valeur et de les écrire en **contrainte table**. De plus, toute **modification sur les tables mères sera propager sur les tables filles**.

N.B : TIMESTAMP = (YYYY-MM-DD hh:mm:ss)= (année-mois-jour heure:minute:seconde).
Les fonctions SQL MONTH(DATE) et YEAR(DATE) retournent respectivement le mois et l'année de DATE de type TIMESTAMP

Relation	Création
SPECTACLE	CREATE TABLE SPECTACLE (Num-SPECTACLE NUMBER(3) NOT NULL, NOM VARCHAR(30) NOT NULL DUREE NUMBER(3) NOT NULL, TYPE VARCHAR(7) NOT NULL, } 0,5 CONSTRAINT pk-SPEC PRIMARY KEY (Num-SPECTACLE), ...0,5 CONSTRAINT ck-type CHECK TYPE IN {"théatre,"concert","Danse"}) 0,5
SALLE	CREATE TABLE SALLE (NUM-S NUMBER(2) NOT NULL,.....0,25 Nbr-Places NUMBER(3) NOT NULL,.....0,25 CONSTRAINT pk-salle PRIMARY KEY (NUM-S);.....0,5
REPRESENTATION	CREATE TABLE REPRESENTATION (DATE TIMESTAMP NOT NULL, NUM-Spectacle NUMBER (3) NOT NULL, NUM-SALLE NUMBER(2) NOT NULL, PRIX DECIMAL NOT NULL, } 0,5 CONSTRAINT fk-Repre-Spec Foreign Key NUM-Spectacle REFERENCES SPECTACLE(Num-SPECTACLE).....0,5 ON DELETE CASCADE ON UPDATE CASCADE.....0,5 CONSTRAINT fk-Repre-Salle foreign key NUM-SALLE REFERENCES SALLE (NUM-S) ....0,5 ON DELETE CASCADE ON UPDATE CASCADE,.....0,5 CONSTRAINT pk-Repre PRIMARY KEY(Num-SPECTACLE,Num-SALLE,DATE)); 0,5

II/ Ecrire en SQL les requêtes suivantes:

a) donner le nombre et les durées des spectacles de type " théatre". (1 point)

```
SELECT  COUNT(*)  0,25, SUM(durée) 0,25
FROM SPECTACLE 0,25
WHERE TYPE="Théatre"; 0,25
```

b) Donner le numéro, le nom et le type de spectacle qui n'ont jamais été présentés dans les salles où se sont déroulés les spectacles de type " théatre". 2 points

```
SELECT Num-SPECTACLE, NOM, TYPE    0,25
FROM SPECTACLE S                                                    } 0,5
WHERE NOT EXISTS (SELECT Num-SPECTACLE
                  FROM   REPRESENTATION R    0,25
                  WHERE S.Num-SPECTCLE=R.NUM-Spectacle   0,25
                   and R.NUM-SALLE IN (SELECT R1. NUM-SALLE    0,5
                                       FROM REPRESENTATION R1, SPECTACLE S1
                                      WHERE R.NUM-SPECTACLE=S1.Num-SPECTACLE   } 0,5
                                       ANS S1.TYPE="Théatre"));
```

c) Donner le nombre de spectacles, dépassant 10 spectacles, par type. 1 point

```
SELECT TYPE, COUNT(*)   0,25
FROM SPECTACLE   0,25
GROUP BY TYPE   0,25
HAVING COUNT(*) >10;   0,25
```

d) Donner le nombre de spectacles de même type par Mois de l'année 2018 ayant un nombre de places entre 200 et 350 places. 2 points

```
SELECT TYPE 0,25, COUNT(*)   0,25
FROM SPECTACLE S, SALLE, REPRESENATION R    0,25
WHERE S.Num-Spectacle=R.NUM-Spectacle 0,25 AND R.NUM-SALLE= SALLE.NUM-S 0,25 AND
Nbr-Places between 200 and 350 AND YEAR(DATE)="2018" 0,25
GROUP BY TYPE,MONTH(DATE);  0,5
```