Command SQL

Algèbre Relationnel

- Projection (éliminer les doublants)

  R = Projection (R1, a1, a2, ...) / R = π (R1, a1, a2, ...) / R = $\pi_{a1, a2 ...}$(R1)
- Restriction

  R = Restriction (R1, condition) / R = σ (R1, condition) / R = $\sigma_{condition}$(R1)
- Produit

  R = Produit (Homme, Voiture) / R = X (R1, R2) / R = R1 X R2
- Jointure = Produit + Restriction

  R = Jointure (R1, R2, condition) / R = ⋈ (R1, R2, condition) / R = R1 ⋈(condition) R2

  Jointure (R1, R2, condition) = Restriction (Produit (R1, R2), condition)
- Séléction = Produit + Restriction + Projection

R1

#A	B	C=>R2
1	Alpha	10
2	Bravo	10
3	Charlie	20
4	Delta	

R2

#X	Y
10	Echo
20	Fox
30	Golf

R

A	B	C	X	Y
1	Alpha	10	10	Echo
2	Bravo	10	10	Echo
3	Charlie	20	20	Fox

- Jointure naturelle

  R = JointureNaturelle (R1, R2) / R = ⋈ (R1, R2) / R = R1 ⋈ R2

  Jointure(R1,R2,R1.A=R2.A) = JointureNaturelle(R1,R2) avec A l'attribut commun

  SELECT * FROM R1, R2, Ri WHERE <condition> = SELECT * FROM (R1 INNER JOIN R2 ON <condition>) INNER JOIN Ri ON <condition>
- Jointure externe

  R = JointureExterne (R1, R2, condition) / R = +⋈+ (R1, R2, condition) / R = R1 +⋈+(condition) R2

  R = JointureExterneGauche (R1, R2, condition) / R = +⋈ (R1, R2, condition) / R = R1 +⋈(condition) R2

  SELECT * FROM R1 LEFT JOIN R2 ON R1.C = R2.X

  (Pour sélectionner tous les tuple du côté N d'une relation 1:N, même si il ne sont pas référencés)

  R = JointureExterneDroite (R1, R2, condition) / R = ⋈+ (R1, R2, condition) / R = R1 ⋈+(condition) R2

  SELECT * FROM R1 RIGHT JOIN R2 ON R1.C = R2.X

  (Pour sélectionner tous les tuples d'une relation 0:N, y compris ceux qui ne font pas de référence)

  Trouver les enregistrements non joints

  SELECT * FROM R1 RIGHT JOIN R2 ON R1.C = R2.X WHERE R1.A IS NULL

- Union

  R = Union (R1, R2) / R = R1 ∪ R2

  SELECT * FROM R1 UNION SELECT * FROM R2
- Différence

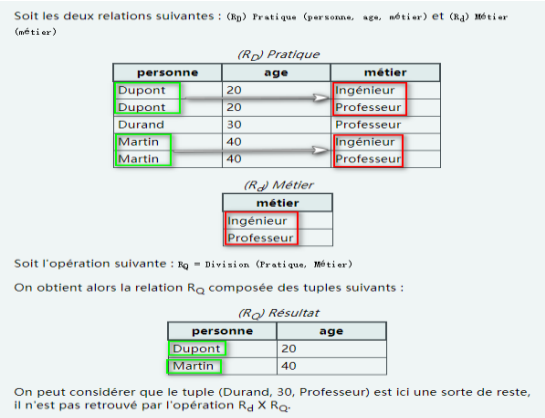
  R = Différence (R1, R2) <> Différence (R2, R1) / R = R1 − R2

  SELECT * FROM R1 EXCEPT SELECT * FROM R2
- Intersection

R = Intersection (R1, R2) / R = R1 ∩ R1
Intersection (R1, R2) = Difference (R1, Difference (R1, R2))
SELECT * FROM R1 INTERSECT SELECT * FROM R2

- Division = Produit + Restriction + Différence

R_Q = Division (R_D, R_d) / R_Q = ÷ (R_D, R_d) / R_Q = R_D ÷ R_d


Soit les deux relations suivantes : (R_D) Pratique (personne, age, métier) et (R_d) Métier (métier)

(R_D) Pratique

personne	age	métier
Dupont	20	Ingénieur
Dupont	20	Professeur
Durand	30	Professeur
Martin	40	Ingénieur
Martin	40	Professeur

(R_d) Métier

métier
Ingénieur
Professeur

Soit l'opération suivante : R_Q = Division (Pratique, Métier)

On obtient alors la relation R_Q composée des tuples suivants :

(R_Q) Résultat

personne	age
Dupont	20
Martin	40

On peut considérer que le tuple (Durand, 30, Professeur) est ici une sorte de reste, il n'est pas retrouvé par l'opération R_d X R_Q.

Database SQL

- CREATE DATABASE *databasename*;
- DROP DATABASE *databasename*;
- SHOW DATABASES;

Table SQL

- CREATE TABLE *table_name (column datatype constraint);*
  - UNIQUE (<liste d'attibuts>)
  - PRIMARY KEY (<liste d'attibuts>)
  - FOREIGN KEY (<liste d'attibuts>) REFERENCES <nom table> (<nom colonnes>)
  - CHECK (<condition>)
- DROP TABLE *table_name*;
- TRUNCATE TABLE *table_name*; - *delete the data inside table*
- ALTER TABLE
  - add column
    ALTER TABLE *table_name* ADD *column_name datatype*;
  - drop column
    ALTER TABLE *table_name* DROP *column_name*;
  - change data type
    ALTER TABLE *table_name* MODIFY *column_name datatype*;
  - NOT NULL constraint
    ALTER TABLE *table_name* MODIFY *column_name datatype* NOT NULL;
  - UNIQUE constraint
    ALTER TABLE *table_name* ADD UNIQUE *(column_name);*
  - PRIMARY KEY constraint
    ALTER TABLE *table_name* ADD PRIMARY KEY *(column_name);*
    ALTER TABLE *table_name* DROP PRIMARY KEY;
  - FOREIGN KEY constraint
    ALTER TABLE *table_name1* ADD FOREIGN KEY *table_name2* (*column_name*);

- CHECK constraint
  ALTER TABLE *table_name* ADD CHECK *(column_name condition);*
  ALTER TABLE *table_name* DROP CHECK;
- DEFAULT constraint
  ALTER TABLE *table_name* ALTER *column_name* SET DEFAULT *default_value*;
  ALTER TABLE *table_name* ALTER *column_name* DROP DEFAULT;
- AUTO_INCREMENT constraint
  ALTER TABLE *table_name* AUTO_INCREMENT=start_value;
  ALTER TABLE *table_name* CHANGE *column_name column_name datatype* UNSIGNED *contraint*;

- INSERT
  - INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>) VALUES (<Liste ordonnée des valeurs à affecter aux propriétés spécifiées ci-dessus>)
    ```
    INSERT INTO livre (pubdate) VALUES (TO_DATE('20170330','YYYYMMDD'));
    ```
  - INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>) SELECT ...
    ```
    INSERT INTO Credit (Date, Montant, Objet) SELECT Date, Montant, 'Annulation de débit' FROM Debit WHERE Debit.Date = 25-12-2001;
    ```
- UPDATE <Nom de la relation> SET <Liste d'affectations Propriété=Valeur, Propriété=Valeur> WHERE <Condition pour filtrer les tuples à mettre à jour>
- DELETE FROM <Nom de la relation> WHERE <Condition pour filtrer les tuples à supprimer>
- SELECT DISTINCT <liste d'attributs projetés> FROM <liste de relations du produit cartésien> WHERE <condition de la restriction> ORDER BY <liste ordonnée d'attributs> (DESC)
  - Renommage
    R = Renommage (R1, a1, a2, ...) / R = ρ (R1, a1, a2...) / R = $\rho_{a1, a2, ...}$(R1)

Opérateurs de comparaisons et logiques

P=C, P<>C, P<C, P>C, P<=C, P>=C, P BETWEEN C1 AND C2, P IN (C1,C2,..), P LIKE '%_', P IS NULL, OR, AND, NOT

Alias

SELECT t1.attribut1 AS a1, t2.attribut2 AS a2 FROM table1 t1, table2 t2 ;

GROUPE BY

SELECT V.NR, **SUM/AVG/COUNT/MIN/MAX** (**DISTINCT** V.QT) **FROM** VENTES V, PRODUITS P **WHERE** V.NP=P.NP **AND** P.COUL='Rouge' **GROUP BY** V.NR **HAVING** COUNT(*)>5 **ORDER BY** V.NR **DESC**;

Héritage

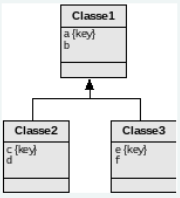
Tout ce qui est vrai pour la classe mère est vrai pour ses classes filles

Classe abstraite : sans association ni clé au niveau de la classe mère

Héritage	Héritage (presque) complet	Classe mère abstraite	Associations liées
Par référence			
Par classes filles	NON	OUI	
Par classe mère	OUI	NON	N : N - mère

- Héritage par références(de filles vers la mère)

  - Chaque classe, mère ou fille, est représentée par une relation.
  - La clé primaire de la classe mère est utilisée pour identifier chacune de ses classes filles : cette clé étant pour chaque classe fille à la fois la clé primaire et une clé étrangère vers la classe mère.

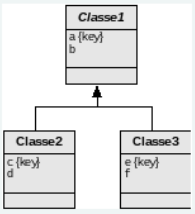


Héritage

```
Classe1(#a, b)

Classe2(#a=>Classe1, c, d) avec c KEY

Classe3(#a=>Classe1, e, f) avec e KEY
```

- Héritage par les classes filles

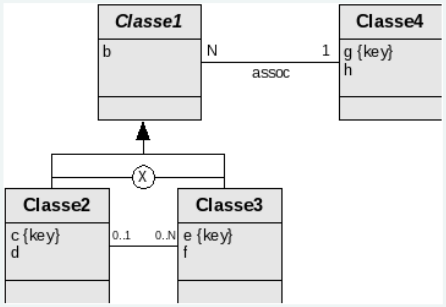
  - Chaque classe fille est représentée par une relation, la classe mère n'est pas représentée (si elle est abstraite).
  - Tous les attributs de la classe mère sont répétés au niveau de chaque classe fille.
  - La clé primaire de la classe mère est utilisée pour identifier chacune de ses classes filles.



Héritage (classe mère abstraite)

```
Classe2(#a, b, c, d) avec c KEY

Classe3(#a, b, e, f) avec e KEY
```

Si la classe mère possède une association sortante (c'est elle qui référence, mais elle n'est jamais référencée), alors l'**héritage pas les classes filles** fonctionne toujours très bien.

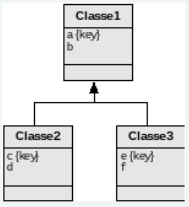


Héritage exclusif non complet (classe mère abstraite, association sortante)

```
Classe2(#c, b, d,  fkg=>Classe4)

Classe3(#e, b, f, fka=>Classe2,  fkg=>Classe4)

Classe4(#g, h)
```

- Héritage par la classe mère

  - Seule la classe mère est représentée par une relation (ses classes filles ne sont pas représentées par des relations).
  - Tous les attributs de chaque classe fille sont réintégrés au niveau de la classe mère.
  - La clé primaire de la classe mère est utilisée pour identifier la relation.
  - Un attribut supplémentaire de discrimination t (pour "type"), est ajouté à la classe mère, afin de distinguer les tuples. Cet attribut est de type énumération et a pour valeurs possibles les noms de la classe mère ou des différents classes filles.

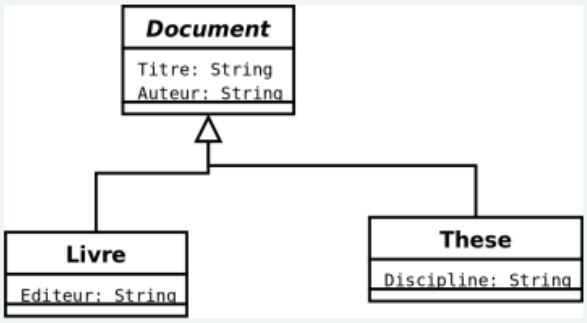


Héritage

`Classe1(#a, b, c, d, e, f, t:{1,2,3}) avec c UNIQUE, e UNIQUE, t NOT NULL`

Contraintes : on devra vérifier que les nullités et non nullités de c, d, e et f en fonction du type t (cela peut se faire grâce à un CHECK en SQL)

- Example1



Représentation de documents

Héritage absorbé par les classes filles

These(#id:int, titre:text, discipline:text, auteur:text)
Livre(#id:int, titre:text, editeur:text, auteur:text)
Contrainte complexe :
- Intersection (PROJECTION(These, id), Projection(Livre, id)) = {}

Héritage représenté par une référence

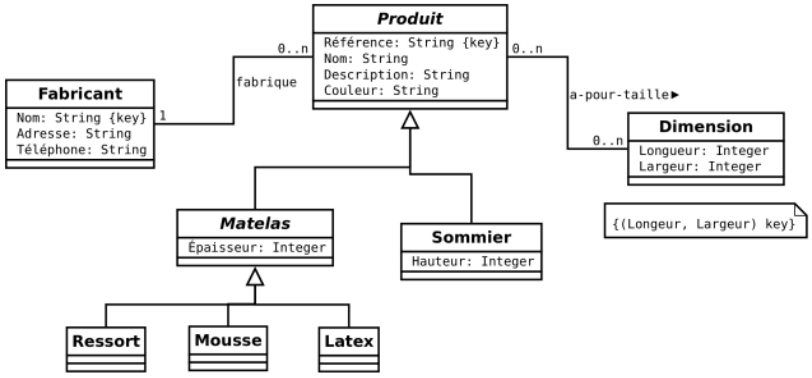
Document(#id:int, titre:text, auteur:text) avec titre et auteur NOT NULL
These(#id=>Document, discipline:text)
Livre(#id=>Document, editeur:text)
Contraites compliquées :
- Intersection (PROJECTION(These, id), Projection(Livre, id)) = {}
- Projection(Document,id) = Union (Projection(These,id), Projection(Livre,id))
Vues :
vThese = JointureNaturelle (Document, These)

vLivre = JointureNaturelle (Document, Livre)

Héritage absorbé par la classe mère

Document(#id:int, titre:text, discipline:text, editeur:text, auteur:text, type:{These|Livre})
Contraintes simples :
- (discipline NOT NULL AND type=These) OR (editeur NOT NULL AND type=Livre)
- NOT (discipline NOT NULL AND type=Livre)
- NOT (editeur NOT NULL AND type=These)
Vues :
vThese = Projection (Restriction (Document, type=These), id, titre, discipline, auteur)
vLivre = Projection (Restriction (Document, type=Livre), id, titre, editeur, auteur)

- Example2



Transformation par classe mère

Fabricant (#nom:string, adresse:string, telephone:string) WITH adresse, telephone NOT NULL
Produit (#reference:string, nom:string, description:string, couleur:string, fabricant=>Fabricant, epaisseur:integer, hauteur:integer, type:{ML, MM, MR, S}) WITH nom, couleur, fabriquant, type NOT NULL
Dimension (#longueur:integer, #largeur:integer)
Taille (#reference=>Produit, #longueur=>Dimension, #largeur=>Dimension)
Contraites simples :
- NOT (epaisseur AND type=S)
- NOT (hauteur AND type=ML OR type=MM OR type=MR)
- (epaisseur NOT NULL AND type=ML OR type=MM OR type=MR) OR (hauteur NOT NULL AND type=S)

Transformation par référence

Fabricant (#nom:string, adresse:string, telephone:string) WITH adresse, telephone NOT NULL
Produit (#reference:string, nom:string, description:string, couleur:string, fabricant => Fabricant) WITH nom, couleur, fabriquant NOT NULL
Matelas (#reference => Produit, epaisseur:integer, type:{'L','M','R'}) WITH epaisseur, type NOT NULL
Sommier (#reference => Produit, hauteur:integer) WITH hauteur NOT NULL
Dimension (#longueur:integer, #largeur:integer)
Taille (#reference=>Produit, #longueur=>Dimension, #largeur=>Dimension)

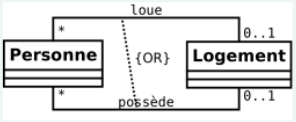
Contrainte complexe : INTERSECTION (PROJECTION (MATELAS, reference), PROJECTION (SOMMIER, reference)) = {}

Contraintes avancées

Contraintes exprimées sur le MCD

Les contraintes exprimées au niveau conceptuel doivent être traduites au niveau relationnel.

Si les contraintes ne sont pas trop nombreuses, on peut commenter chaque relation directement lors de la formalisation du MLD.



Exemple de contrainte OR entre deux associations

```
1 Personne (... loue=>Logement, possède=>Logement) avec (loue OR possède)
2 Logement (...)
```

- Contrainte de cardinalité minimale 1 dans les associations 1:N

Association 1:N

  - Si la cardinalité est exactement 1 (1..1) côté 1, alors on ajoutera une contrainte de non nullité sur la clé étrangère,
  - si la cardinalité est au moins 1 (1..N) côté N, on ajoutera une contrainte d'existence de tuples référençant pour chaque tuple de la relation référencée.

  R1(#a,b)

  R2(#c,d,a=>R1)

  Contraintes : R2.a NOT NULL et PROJECTION(R1,a) = PROJECTION(R2,a)

  On veut que tous les éléments de R1 soient référencés au moins une fois dans R2, donc il n'existe pas de tuple de R1 qui ne soit pas référencé par R2.a, donc PROJECTION(R1,a) = PROJECTION(R2,a)

  Si la cardinalité côté 1 est 0..1 alors il peut exister la valeur NULL dans R2.a et donc la contrainte devient : PROJECTION(Classe1,a) $\subseteq$ PROJECTION(Classe2,a).

- Contrainte de cardinalité minimale 1 dans les associations N:M

Association N:M

  Si la cardinalité est au moins 1 (1..N) d'un côté et/ou de l'autre, alors des contraintes d'existence simultanée de tuples devront être ajoutées.

R1(#a,b)

R2(#c,d)

A(#a=>R1,#c=>R2)

Contraintes : PROJECTION(R1,a) = PROJECTION(A,a) AND
PROJECTION(R2,c) = PROJECTION(A,c)

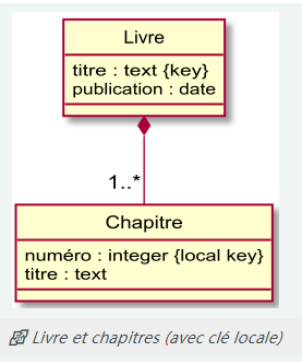
Si la cardinalité est 0..N:1..N seule une des deux contraintes est exprimée.

Composition et agrégation

On appelle composition une association particulière qui possède les propriétés suivantes : ***1.*** La composition associe une classe composite et des classes parties, tel que tout objet partie appartient à un et un seul objet composite. C'est donc une association 1:N (voire 1:1). ***2.*** La composition n'est pas partageable, donc un objet partie ne peut appartenir qu'à un seul objet composite à la fois. ***3.*** Le cycle de vie des objets parties est lié à celui de l'objet composite, donc un objet partie disparaît quand l'objet composite auquel il est associé disparaît. ***4.*** La composition est une association particulière (binaire de cardinalité contrainte). ***5.*** La composition n'est pas symétrique, une classe joue le rôle de conteneur pour les classes liées, elle prend donc un rôle particulier a priori. ***6.*** La composition est une agrégation avec des contraintes supplémentaires (non partageabilité et cycle de vie lié). ***7.*** La cardinalité côté composite est toujours de exactement 1. Côté partie la cardinalité est libre, elle peut être 0..1, 1, * ou bien 1..*.

L'agrégation (noté avec un losange blanc) est différent de la composition, puisqu'elle ne pose aucune dépendance au cycle de vie (c'est une association classique)

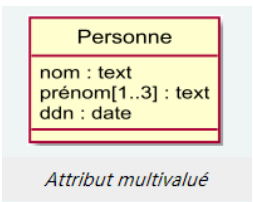
- Clé local



Livre et chapitres (avec clé locale)

Il existe deux plusieurs livres avec un chapitre numéro 1, mais il n'existe pas deux chapitre numéro 1 au sein d'un même livre.

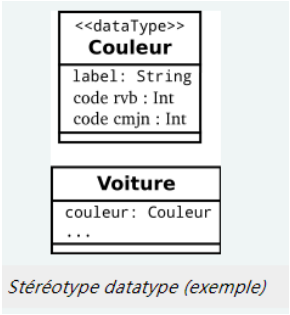
- Attribut mulvalué : un attribut multivalué est un attribut qui peut prendre plusieurs valeurs distinctes dans son domaine.
  attribut_mv[nbMinValeurs..nbMaxValeurs]:type



Attribut multivalué

- Attribut composé et Data type

  Un attribut peut être composé (ou composite) joue le rôle d'un groupe d'attributs (par exemple une adresse peut être un attribut composé des attributs numéro, type de voie, nom de la voie). Dans UML on préfère l'usage de compositions (ou de dataType) aux attributs composés.



Stéréotype datatype (exemple)

- Transformation de composition

Pour identifier une classe partie dans une composition, on utilise une clé locale concaténée à la clé étrangère vers la classe composite, afin d'exprimer la dépendance entre les deux classes.

Une composition

- est transformée comme une association 1:N,
- puis on combine une clé locale de la classe partie et la clé étrangère vers la classe composite pour construire une clé primaire composée.

Composition

```
Classe1(#a, b)

Classe2(#c, #a=>Classe1, d)
```

- Transformation des attributs multivalués

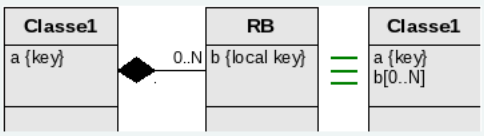
Pour chaque attribut multivalué b d'une classe C,

- on crée une nouvelle relation RB,
- qui comprend un attribut monovalué correspondant à b,
- plus la clé de la relation représentant C ;
- la clé de RB est la concaténation des deux attributs.

Attribut multivalué

```
R1(#a)

RB(#b, #a=>Classe1)
```

La transformation d'une composition avec un seul attribut pour la classe composante donne un résultat équivalent à la transformation d'un attribut multivalué.



Composition et attribut multivalué

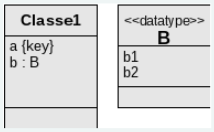
```
R1(#a)     RB(#b,#a=>Classe1)
```

- Transformation des datatypes

Datatype

Pour chaque datatype comprenant N sous-attributs,

- on crée N attributs correspondants,
- dont les noms sont la concaténation du nom de l'attribut composite avec celui du sous-attribut.

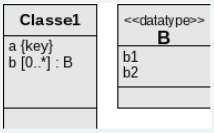


Attribut composé

```
R1(#a, b_b1, b_b2)
```

Datatype multivalué

On combine les règles énoncées pour les attributs composés et pour les attributs multivalués.



Attribut composé multivalué

```
R1(#a)

RB(#b_b1, #b_b2, #a=>Classe1)
```

La transformation des datatype multivalué est similaire à la transformation des compositions.

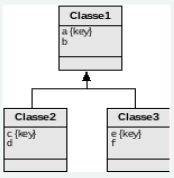
Vue

- Création

CREATE VIEW <nom de vue> <nom des colonnes> AS <spécification de question>

- Héritage par une référence et vues

Une vue est créée pour chaque classe fille en réalisant une jointure avec la classe mère.



Héritage

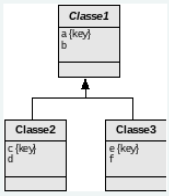
```
Classe1(#a, b)

Classe2(#a=>Classe1, c, d) avec c KEY

Classe3(#a=>Classe1, e, f) avec e KEY

vClasse2=jointure(Classe1, Classe2, a=a)

vClasse3=jointure(Classe1, Classe3, a=a)
```

Soit la classe A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A, comprenant la clé K' et les attributs B1 et B2. Le modèle relationnel correspondant selon cette transformation est :

A (#K, A1, A2)

B (#K=>A, K', B1, B2)
vB = Jointure (A, B, A.K=B.K)

- Héritage par les classes filles
  Cas - classe mère abstraite



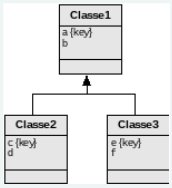
Héritage (classe mère abstraite)

```
Classe2(#a, b, c, d) avec c KEY

Classe3(#a, b, e, f) avec e KEY

vClasse1=Union(Projection(Classe2, a, b), Projection(Classe3, a, b))
```

Soit la classe abstraite A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C2. Le modèle relationnel correspondant selon cette transformation est :

B (#K, A1, A2, B1, B2)
C (#K, A1, A2, C1, C2)
vA = Union (Projection (B, K, A1, A2), Projection (C, K, A1, A2))

Cas - classe mère non abstraite

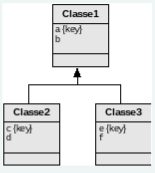


Héritage

```
Classe1(#a, b)

Classe2(#a, b, c, d) avec c KEY

Classe3(#a, b, e, f) avec e KEY

vClasse1=Union(Union(Classe1, Projection(Classe2, a, b)), Projection(Classe3, a, b))
```

- Héritage par la classe mère

Chaque classe est représentée par une vue qui restreint aux tuples de la relation correspondants et les projette sur les attributs correspondants.



Héritage

```
Classe1(#a, b, c, d, e, f, t: {1, 2, 3}) avec c UNIQUE et e UNIQUE

vClasse1=projection(restriction(Classe1, t=1), a, b)

vClasse2=projection(restriction(Classe1, t=2), a, b, c, d)

vClasse3=projection(restriction(Classe1, t=3), a, b, e, f)
```

Soit la classe A abstraite avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C. Le modèle relationnel correspondant selon cette transformation est :

A (#K, A1, A2, B1, B2, C1, C2, T:{'B','C'})
vB = Projection (Restriction (A, T='B'), K, A1, A2, B1, B2)
vC = Projection (Restriction (A, T='C'), K, A1, A2, C1, C2)

- Transformation des méthodes par des vues
  Lorsqu'une méthode est spécifiée sur un diagramme UML pour une classe C, si cette méthode est une fonction relationnelle (elle renvoie une unique valeur et elle peut être résolue par une requête SQL), alors on crée une vue qui reprend les attributs de la classe C et ajoute des colonnes calculées pour les méthodes.

  Les *attributs dérivés* étant apparentés à des méthodes, ils peuvent également être gérés par des vues.

Gestion du droit

- Attribution de droits
  GRANT <SELECT / INSERT / DELETE / UPDATE / ALTER/ ALL PRIVILEGES > ON <nom table/nom vue> TO <utilisateur / PUBLIC> [WITH GRANT OPTION]
  - || est possible de spécifier le droit ALL PRIVILEGES qui donne tous les droits à l'utilisateur (sauf celui de transmettre ses droits).
  - La clause WITH GRANT OPTION est optionnelle, elle permet de préciser que l'utilisateur a le droit de transférer ses propres droits sur la table à d'autres utilisateur. Une telle clause permet une gestion décentralisée de l'attribution des droits et non reposant uniquement dans les mains d'un administrateur unique.
  - La spécification PUBLIC à la place d'un nom d'utilisateur permet de donner les droits spécifiés à tous les utilisateurs de la BD
  - Il est possible de spécifier des droits sur des vues plutôt que sur des tables, avec une syntaxe identique (et un nom de vue à la place d'un nom de table).
  - Les droits sont stockés dans le catalogue des données, il est généralement possible de modifier directement ce catalogue à la place d'utiliser la commande GRANT. Cela reste néanmoins déconseillé.
- Révocation de droits
  REVOKE <SELECT / INSERT / DELETE / UPDATE / ALTER/ ALL PRIVILEGES > ON <nom table/nom vue> FROM <utilisateur / PUBLIC> [GRANT OPTION]
  - Pour retirer les droits de donner les droits à un utilisateur (qui l'a donc obtenu par la clause WITH GRANT OPTION), il faut utiliser la valeur GRANT OPTION dans la liste des droits révoqués.
  - Lorsque qu'un droit est supprimé pour un utilisateur, il l'est également pour tous les utilisateurs qui avait obtenu ce même droit par l'utilisateur en question.

Normalisation

- Problèmes soulevés par redondances
  Redondances conduiront à des problèmes de contrôle de la cohérence de l'information, de mise à jour, de perte d'information lors de la suppression de données et de difficulté à représenter certaines informations.

- Dépendance fonctionnelle

Soient R(A1, A2, ... , An) un schéma de relation, X et Y des sous-ensembles de A1, A2, ... , An. On dit que X détermine Y, ou que Y dépend fonctionnellement de X, si et seulement s'il existe une fonction qui à partir de toute valeur de X détermine une valeur unique de Y.

A	B	C	D	E
a1	b2	c2	d3	e2
a1	b2	c2	d1	e4
a2	b3	c2	d1	e5
a2	b4	c5	d1	e5

E→D / E→A / B→C / B→A – Clé :{B, E}

Dépendance Fonctionnelle Elémentaire(DFE)

Dépendance fonctionnelle élémentaire

Soit G un groupe d'attributs et A un attribut, une DF ☆ G→A est élémentaire si A n'est pas incluse dans G et s'il n'existe pas d'attribut A' de G qui détermine A.

DF élémentaires

- AB→C est élémentaire si ni A, ni B pris individuellement ne déterminent C.
- Nom, DateNaissance, LieuNaissance→Prénom est élémentaire.

DF non élémentaires

- AB→A n'est pas élémentaire car A est incluse dans AB.
- AB→CB n'est pas élémentaire car CB n'est pas un attribut, mais un groupe d'attributs.
- N°SS→Nom, Prénom n'est pas élémentaire.

Fermeture transitive des DFE

Fermeture transitive

On appelle fermeture transitive F+ d'un ensemble F de DFE ☆, l'ensemble de toutes les DFE qui peuvent être composées par transitivité à partir des DFE de F.

Soit l'ensemble F = {A→B, B→C, B→D, A→E}.

La fermeture transitive de F est F+ = { A→B, B→C, B→D, A→E, A→C, A→D }

Couverture minimale des DFE

Couverture minimale

La couverture minimale d' un ensemble de DFE ☆ est un sous-ensemble minimum des DFE permettant de générer toutes les autres DFE.

Synonymes : Famille génératrice

Tout ensemble de DFE (et donc tout ensemble de DF) admet au moins une couverture minimale (et en pratique souvent plusieurs).

L'ensemble F = {A→B, A→C, B→C, C→B} admet les deux couvertures minimales :

CM1 = {A→C, B→C, C→B} et CM2 = {A→B, B→C, C→B}

Définition formelle d'une clé

Clé

Soient une relation R(A1,A2,...,An) et K un sous-ensemble de A1,A2,... ,An.

K est une clé de R si et seulement si :

1. K→A1,A2,...,An
2. et il n'existe pas X inclus dans K tel que X→A1,A2,...,An.

Une clé est donc un ensemble minimum d'attributs d'une relation qui détermine tous les autres.

Clés candidates et clé primaire

Si une relation comporte plusieurs clés, chacune est dite clé candidate et l'on en choisit une en particulier pour être la clé primaire.

Les clés candidates sont des clés !

Toutes les clés candidates sont des clés, pas seulement la clé primaire.

Les clés candidates se déterminent mutuellement

Toute clé candidate détermine les autres clés candidates, puisque qu'une clé détermine tous les attributs de la relation.

- Forme normale
  1NF : Une relation est en 1NF si elle possède au moins une clé et si *tous ses attributs sont atomiques*.
  2NF : Une relation est en 2NF si elle est en 1NF et il n'existe pas aucune *partie de clé* détermine un attribut qui n'appartient pas à une clé
  3NF : Une relation est en 3NF si elle est en 2NF et il n'existe pas aucun attribut qui n'appartient pas à une clé détermine d'attribut qui n'appartient pas à une clé
  BCNF : Une relation est en BCNF si elle est en 3NF et et il n'existe pas aucun attribut qui n'appartient pas à une clé détermine une partie de clé

Example

On considère une relation R construite sur les attributs Propriétaire, Occupant, Adresse, Noapt, Nbpièces, Nbpersonnes, un nuplet (p, o, a, n, nb1, nb2) ayant la signification suivante : La personne o habite avec nb2 personnes l'appartement de numéro n ayant nb1 pièces dont le propriétaire est p. Une analyse de cette relation nous fournit un ensemble initial E de dépendances fonctionnelles :

occupant → adresse, occupant → noapt, occupant → nbpersonnes, adresse, noapt → propriétaire, adresse, noapt → occupant, adresse, noapt → nbpièces

✧ Donner l'ensemble des DFE engendrées par E (fermeture transitive F+).

- occupant→adresse et occupant→noapt donc occupant→(adresse,noapt)
- occupant → propriétaire (Transitivité)
- occupant → nbpièces (Transitivité)

On a donc F+ :

- occupant → adresse, noapt, nbpersonnes, propriétaire, nbpièces
- adresse, noapt → propriétaire, occupant, nbpièces, nbpersonnes

- ✧ Quelles sont les clés candidates de R?
  Une clé est un ensemble d'attributs qui détermine tous les autres. Si on regarde la fermeture transitive de E, on voit que (occupant) ainsi que (adresse, noapt) sont dans ce cas. Il y a donc deux clés candidates.
- ✧ Montrer que R est en 3NF de deux façons différentes (sans passer et en passant par la BCNF).

  <u>Démonstration exhaustive</u>

  Pour déterminer la forme normale de R, il faut d'abord distinguer :

  - les attributs clés : (occupant) et (adresse, noapt)
  - et les attributs n'appartenant pas à une clé : propiétaire, nbpièces et nbpersonnes

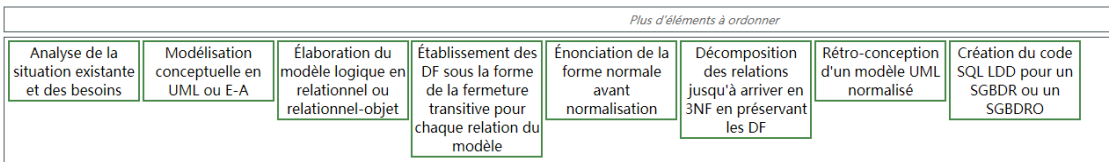
  Étudions chaque NF :

  1. Considérons que la relation est en 1ère forme normale.
  2. Elle est en 2ème forme normale si, pour chaque clé, tous les attributs n'appartenant pas une clé dépendent pleinement des clés et non de parties de clés. Ici c'est le cas. (occupant) est une clé à un seul attribut, la réponse est donc triviale. On vérifie pour (adresse, noapt) :
     - adresse, noapt → propriétaire, nbpièces, nbpersonnes,
     - et il n'existe pas de DF depuis adresse ou noapt.
  3. Une relation est en 3ème forme normale s'il n'existe pas de dépendance fonctionnelle entre des attributs n'appartenant pas à une clé. C'est le cas ici, les seuls DF sont issues de clés. R est donc en 3ème forme normale

  <u>Démonstration courte</u>

  On peut directement montrer que R est en BCNF (donc en 3NF) : **Toutes les DFE sont de la forme K→A avec K clé**.

Étape de conception de base



Python

- Lecture

```
#!/usr/bin/python3
```

```
import psycopg2

HOST = "localhost"
USER = "me"
PASSWORD = "secret"
DATABASE = "mydb"

# Open connection
conn = psycopg2.connect("host=%s dbname=%s user=%s
password=%s" % (HOST, DATABASE, USER, PASSWORD))

# Open a cursor to send SQL commands
cur = conn.cursor()

try:
# Execute a SQL SELECT command
sql = "SELECT name, century FROM v_philosopher"
cur.execute(sql)

# Fetch data line by line
raw = cur.fetchone()
while raw:
  print (raw[0], raw[1])
  raw = cur.fetchone()

except psycopg2.IntegrityError as e:
  print("Message système :", e)


# Fetch all the data
res = cur.fetchall()
// [('Aristote', '-4e'), ('Epicure', '-4e'), ('Platon', '-5e')]

for raw in res:
  print (raw[0], raw[1])

# Close connection
conn.close()
```

- Ecriture

```
#!/usr/bin/python3

import psycopg2

HOST = "localhost"
USER = "me"
PASSWORD = "secret"
DATABASE = "mydb"

# Open connection
conn = psycopg2.connect("host=%s dbname=%s user=%s
password=%s" % (HOST, DATABASE, USER, PASSWORD))

# Open a cursor to send SQL commands
cur = conn.cursor()

try:
# Execute a SQL INSERT command
 sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
```

```
 cur.execute(sql)
 conn.commit()
 res = cur.fetchone()[0] # Retourner un résultat de requête
 print(res)
except psycopg2.IntegrityError as e:
  conn.rollback()
  print("Message système :", e)

# Close connection
conn.close()
```

- Création d'un menu

```
choice = '1'
while choice == '1' or choice == '2':
  print ("Pour ajouter un philosophe à la base, entrez 1")
  print ("Pour voir la liste des philosophes, entrez 2")
  print ("Pour sortir, entrez autre chose")
  choice = input()
  if choice == '1':
    forms.addPhilo(conn)
  if choice == '2':
    reports.printPhilo(conn)
  print(choice)
```