U.B.S. - I.U.T. de Vannes
Département Informatique

R1.05
Introduction aux Bases de Données

Période 1
TD1
Découverte du modèle relationnel et normalisation

BUT Informatique S1-1A

R. Fleurquin, A. Ridard

Septembre 2022

Contenu du TD

- Introduction au modèle relationnel
- Importance de la qualité des schémas relationnels

Partie 1 : découverte du paradigme relationnel

Le ***paradigme relationnel*** est le plus utilisé dans les SGBD. Il se fonde sur une théorie rigoureuse (comprendre mathématiquement bien fondée !) établie il y a plus de 50 ans. Vous allez dans le module de mathématiques discrètes (R1.06) découvrir les concepts mathématiques qui sont mis en œuvre pour fonder ce paradigme : les opérateurs de la théorie des ensembles et la logique des prédicats. Nous nous contentons ici d'une introduction informelle et donc non « mathématisée » de ce paradigme.

Le modèle relationnel propose de stocker l'information sous la forme de tableaux à 2 dimensions appelés « ***relations*** » ou encore « tables ». Une base de données consiste en une ou plusieurs relations. Chaque relation à un nom. Une ligne d'une relation est appelée un « ***n-uplet*** » ou « tuple » ou « enregistrement ». **Par hypothèse du paradigme relationnel il est formellement interdit pour deux lignes d'une même relation d'être identiques**.

Une ligne rassemble un ensemble d'informations sémantiquement « cohérent » associé à une entité de l'univers d'information que l'on souhaite stocker : les informations sur un étudiant, sur une entreprise, sur une commande, sur un produit, etc. L'information contenu par un n-uplet est structuré en compartiments appelé colonnes : une colonne pour le nom de l'étudiant, une pour son prénom, une pour sa série de baccalauréat, etc. Les colonnes d'une relation sont identifiées par un nom unique que l'on appelle un « ***attribut*** ». Chaque attribut se voit associé un « ***domaine de valeurs*** ». On parle plus souvent de « ***type*** » en informatique par exemple le type `int`, `String`, `float`, etc. Toutes les valeurs présentes dans la colonne désignée par l'attribut doivent appartenir au domaine (au type) de l'attribut.

Prenons en petit exemple de base de données relationnelle comportant deux relations.

La première relation (table) s'appelle `Étudiants`.

numEtudiant (int)	nomEtudiant (String)	bacEtudiant (String)	formationEtudiant (String)
11	Le moal	STI2D	BUT INFO
8	Le moal	Général	BUT STID
114	Veran	STI2D	BUT STID

Cette relation comporte quatre attributs (colonnes) : `numEtudiant` de type `int`, `nomEtudiant` de type `String`, `bacEtudiant` de type `String` et `formationEtudiant` de type `String`.

Elle comporte en l'état 3 n-uplets (on parle aussi de lignes ou d'enregistrements). Le premier n-uplet recueille les informations associées à l'étudiant de numéro 11, qui s'appelle Le Moal, de bac d'origine STI2D et inscrit en But informatique. Chaque ligne de cette relation est un ensemble cohérent d'informations associé à un étudiant.

On constate, comme l'impose la théorie, que toutes les lignes de cette relation sont deux à deux distinctes. Cette propriété est assurée ici par le premier attribut `numEtudiant` qui diffère pour chacune des 3 lignes (11, 8 et 114). Cet attribut, à lui seul, joue dans le domaine modélisé le rôle de « clé candidate ».

Un sous-ensemble d'attributs d'une relation est dit « ***clé candidate*** » lorsque :

- la valeur sur cet ensemble d'attribut est et sera toujours différente pour chaque n-uplet d'une relation (propriété d'*unicité*)
- il n'est pas possible de retirer l'un de ses attributs sans perdre la propriété précédente (propriété de *minimalité*).

Une clé candidate est donc un ensemble d'attributs minimal qui garantit dans l'univers modélisé pour une valeur fixée de chaque attribut membre de la clé l'unicité de la ligne correspondante. Ainsi l'ensemble (`numEtudiant, nomEtudiant`) n'est pas une clé candidate car non minimal. L'attribut `bacEtudiant` n'est pas non plus une clé candidate car il ne garantit pas l'unicité de la ligne pour une valeur fixée : par exemple deux lignes correspondent à la valeur STI2D.

Attention de prendre garde aux clés candidates de « circonstance » qui ne sont pas un invariant temporel dans l'univers modélisé. Ainsi dans la relation précédente le couple (`bacEtudiant, formationEtudiant`) semble (avec les données actuelles) jouer le rôle de clé candidate. Mais si dans le domaine rien n'interdit que deux étudiants différents soient titulaires du même bac et inscrits dans la même formation, ce couple peut à tout moment suite à l'insertion dans la base d'un nouvel étudiant perdre la propriété d'unicité. Cette propriété est temporaire et absolument non garantie à l'avenir. En l'état des données stockées on peut penser que ce couple joue le rôle de clé candidate mais la connaissance du domaine nous permet de dire que ce n'est pas le cas.

Notez qu'il y a toujours une clé candidate dans une relation. L'hypothèse d'unicité de chaque ligne du paradigme relationnel entraîne l'existence d'une telle clé : car tous les tuples d'une relation étant nécessairement différents, au pire la seule clé candidate est obtenue en prenant tous les attributs de la relation !

Il peut y avoir également plusieurs clés candidates. Dans ce cas, on choisit toujours l'une d'entre elles comme clé privilégiée : c'est la « ***clé primaire »***. Les clés peuvent émerger du domaine mais il est fréquent de créer des clés artificielles avec un seul attribut et de type entier pour simplifier et optimiser les accès. Cette volonté en provenance du monde informatique impacte notre quotidien avec l'apparition d'identifiant artificiel dans nos vies : votre numéro de sécurité sociale, votre numéro d'étudiant, votre numéro de client EDF, etc.

La seconde relation s'appelle `Formations`.

formation (String) (1)	nomChef (String)	nomSecrétaire (String)
BUT INFO	Kamp	Volin
BUT STID	Frambourg	Sorin
BUT TC	Bourdoncle	NULL

Sa clé primaire est l'attribut `formation`. On constate que certaines des valeurs du domaine de cet attribut se retrouvent dans la colonne `formationEtudiant` de la relation `Etudiants`. L'importation d'une clé primaire d'une table dans une autre table s'appelle une « ***clé étrangère*** ». Le mécanisme de clé étrangère est fondamental. Ce concept permet de stocker de l'information en la structurant de manière à réduire les problèmes de redondance. Il sera ensuite possible en passant par ces clés étrangères en usant d'un mécanisme de « jointure » d'agglomérer et de déduire de nouvelles informations en mettant en correspondance de manière cohérente les données provenant de plusieurs relations.

L'attribut `formationEtudiant` de la relation `Etudiants` étant une clé étrangère, son domaine de valeurs doit nécessairement être inclus dans celui des valeurs actuellement présentes dans la relation `Formations`. On parle de « ***contrainte d'intégrité référentielle*** ». Ainsi, il est impossible de citer BUT GEA

dans une valeur de cet attribut de la table `Etudiants` car elle n'apparait pas dans la table `Formations`. Par contre on pourrait observer la valeur BUT TC.

La structure d'une relation et toutes les contraintes atemporelles qui relient les données stockées s'appelle le « ***schéma*** » de la relation. Ce schéma décrit *l'intension* de la relation : ce sont des informations sur les données qui sont à stocker (la liste des attributs avec leur type, les contraintes d'intégrité, etc.), de l'information sur de l'information ! On parle en conséquence de *métadonnées*. La description en *extension* de la relation est la table avec les données (tuples) qu'elle contient.

En généralisant, on parle du « ***schéma d'une base de données*** » pour désigner tous les schémas de toutes les relations de cette base. Il est fréquent de confondre (à tort !) dans le discours la *relation* (la table qui contient les données) avec son *schéma relationnel* (la structure de la relation).

Le schéma associé à la base précédente est donné ci-dessous. Étudiez bien sa syntaxe avec votre enseignant. Est-il compatible avec l'extension des relations ci-dessus ? Notez que le langage utilisé pour décrire ce schéma n'est pas normalisé. Il y a autant de syntaxes différentes que d'auteurs d'ouvrage en base de données dans la littérature informatique ! Nous utiliserons donc un langage « à nous » IUT de Vannes !

```
Etudiants(  numEtudiant : int (1),
            nomEtudiant : String (NN)
            bacEtudiant : String
            @formationEtudiant : String REF Formations(formation) (NN)
)

Formations(
            formation : String (1),
            nomChef : String (2),
            nomSecrétaire : String (UQ)
)
```

Fin du cours, passons aux exercices !

Exercice 1. Notion de clé candidate

On considère la relation `Exo1` donnée ci-dessous.

a) Identifiez dans cette relation : les attributs, les tuples, les domaines.
b) Cette relation vérifie-t-elle le critère d'unicité des lignes ?
c) Donnez des exemples d'ensembles d'attributs qui ne ne sont pas des clés candidates parce qu'elles ne respectent pas soit le critère d'unicité soit le critère de minimalité. Identifiez trois clés candidates *potentielles*. Est-on certains que ces clés potentielles sont réellement candidates ?
d) Quelle clé primaire aurait-on intérêt à choisir ici ?
e) Proposez un schéma relationnel pour décrire cette relation si l'on suppose que i) toutes les clés trouvées en c) sont candidates et que ii) vous avez retenu la clé primaire choisie à la question précédente.

Exo1

A	B	C	D	E
1	a	50	x	5
1	a	60	y	6
2	b	50	y	4
2	c	60	x	8
5	d	60	z	7

Exercice 2. Notion de clé étrangère

On considère les 3 relations R, S et T données ci-dessous.

a) Quels attributs pourraient jouer le rôle de clé étrangère potentielle ? Est-on certains que ce sont des clés étrangères ?
b) On suppose que les clés étrangères potentielles le sont effectivement. Comment les déclarer dans un schéma relationnel ?
c) On supprime la troisième ligne de la table S. Quel est l'impact potentiel de cette action sur les autres tables ? Même question si l'on supprime ensuite la première ligne de S.

R

A	B	C	D
1	a	50	x
2	a	50	y
3	b	50	t
4	d	60	x
5	b	50	x

S

E	F	G
a	ab	50
b	aa	60
c	bb	50
d	c	60

T

H	I	J
w	a	50
x	a	60
y	b	50
z	b	60

Exercice 3. Notion de schéma relationnel

On considère le schéma relationnel d'une base de données ci-dessous. Proposez 3 relations avec des données respectueuses de ce schéma et comportant chacune au moins 3 tuples.

```
Articles(   numArticle : int (1),
            titreArticle : String (NN)
            journal : String (NN)
)

Auteurs(
            numAuteur : int (1),
            nomAuteur : String (2),
            prenomAuteur : String (2)
            statutAuteur : String
)

AEcrit(
            @auteur : int REF Auteurs(numAuteur) (1)
            @article : int REF Articles(numArticle) (1)
            date : DATE (UQ)
)
```

Exercice 4 : usage d'une base de données, requêtage à la main

On considère la base de données ci-dessous. A l'aide cette base on cherche à répondre aux questions qui suivent. Essayez pour chaque réponse de lister les « opérations » que vous faites à la main. Vous verrez en période 2 comment le langage SQL vous permet de réaliser simplement toutes ces opérations via un SGBDR !

Q1 (projection) : quels sont les noms de tous les enfants ?
Q2 (sélection) : Quelles sont les informations sur l'enfant Jouve ?
Q3 (sélection puis union) : Quelles sont les informations sur les enfants de 11 ans ou 12 ans
Q4 (sélection puis intersection) : Quelles sont les informations sur les filles de 11 ans
Q5 (sélection puis projection) : Quel est l'âge de Tania ?
Q6 (jointure et projection) : Quels sont les intitulés des clubs qui ont des enfants inscrits ?
Q7 (double jointure et sélection et projection) : Quels sont les noms des enfants dont s'occupe Fleurquin ?

Inscriptions

@enfant (1)	@club (1)	annéeInscription
1	1	2020
2	1	2021
1	2	2019
4	2	2020
5	4	2022

Enfants

numéroEnfant (1)	nomEnfant	prénomEnfant	âgeEnfant	sexeEnfant
1	Jouve	Paul	8	M
2	Le Brix	Titouan	12	M
3	Goff	Tania	11	F
4	Rivoal	Alexia	9	F
5	Rivoal	Tonio	11	M

Clubs

numéroClub (1)	intituléClub (2)	responsableClub	salleClub	montantInscription
1	Basket	André	S1	100
2	Tennis	Le Fur	S1	150
3	Judo	Fleurquin	S2	200
4	Karaté	Fleurquin	S3	250

Partie 2 : de l'importance de la qualité d'un schéma relationnel

Le modèle relationnel offre une organisation de stockage très simple sous forme de tables à deux dimensions. On pourrait penser (naïvement) qu'il suffit d'user d'une seule table pour stocker l'intégralité des informations nécessaires au fonctionnement d'une application informatique. On parle alors de « ***table universelle*** ». On va voir que cette approche, que l'on retrouve trop souvent dans certaines applications, est, le plus souvent, très mauvaise. Il est toujours profitable de bien réfléchir à la structure de ces tables. Si l'on ne prend pas ce soin, on s'expose à de nombreuses redondances (perte de place mémoire), des problèmes potentiels d'intégrité des données (apparition d'incohérences entre les informations), une moindre maintenabilité et des pertes de performance.

Il existe toute une théorie « mathématiques » appelée « ***théorie de la normalisation*** » qui permet de limiter au maximum l'apparition de ces problèmes et donc de quantifier le niveau de qualité d'un schéma relationnel. La littérature identifie pour cela des « ***formes normales*** ». Chaque forme normale impose des contraintes atemporelles à respecter par les données stockées dans une relation. Les formes normales s'emboîtent les unes dans les autres et fournissent un moyen d'étalonner la qualité d'un schéma. Le respect d'une forme normale de niveau supérieur implique nécessairement le respect des formes normales des niveaux inférieurs (par exemple le respect de la 3NF implique le respect de la 2NF et donc par transitivité de la 1NF).

Dans le modèle relationnel **il existe huit formes normales manifestant par leur numérotation une qualité croissante**. Les quatre premières (1NF, 2NF, 3NF et BCNF) sont les plus connues et utilisées en pratique. La première (1NF) et la deuxième forme normale (2NF) sont nécessaires pour avoir un modèle relationnel « juste ». Leur respect est non négociable sauf exigence de performance particulière ! Mais **l'usage est de viser au moins la 3ème forme normale (3NF)**. Il est toujours possible de le faire. Cette forme normale offre en effet un bon compromis entre l'impératif de gestion de la redondance, la non perte d'information, la conservation des dépendances fonctionnelles entre les données et l'efficacité des accès.

Les formes normales de niveau supérieur à la 3NF ont leurs avantages et leurs inconvénients. L'inconvénient essentiel tient à la multiplication des relations et donc aux temps d'accès potentiellement plus longs (nombreuses jointures) et à la perte lors de la normalisation de certaines contraintes structurelles entre données. L'augmentation potentielle des temps d'accès du fait de nombreuses jointures est heureusement souvent en partie compensée par la qualité des optimiseurs modernes et par la mise en place d'index pertinents.

La normalisation systématique d'une base de données passe en théorie par une étude « mathématique » des **dépendances** sémantiques entres les données. Elle entraine souvent le découpage de relations en sous-relations. Ce découpage impose malheureusement des requêtes plus compliquées et plus longues avec l'usage de « jointures » pour reconstruire l'information d'origine en « recollant » les sous-relations. En pratique, on évite souvent cette étude mathématique fastidieuse en dérivant les relations depuis des modèles de conception (type diagramme de classes UML par exemple) « propres ». Le développement de cette compétence est l'objet du début de cette ressource !

Remarque : pour des bases de données très importantes (tout particulièrement celles relevant du big data), il est parfois nécessaire après normalisation et étalonnage, de relâcher le niveau de normalisation et d'effectuer une ***dénormalisation*** intelligente dont un benchmark prouve le gain en performance. Ce faisant on accepte de la redondance au bénéfice de temps d'accès beaucoup plus courts.

Exercice 5. Découverte de la notion de qualité d'un schéma

Soit la relation ci-dessous produite par un informaticien débutant. Il s'agit de conserver des informations sur les ressources (cours) enseignées dans un BUT informatique et sur les enseignants qui interviennent sur ces ressources. La clé primaire de cette relation est le couple (`numEns, codeRess`). Un enseignant est identifié de manière unique par son `numEns`. Un enseignant peut enseigner (ou non) dans une ou plusieurs ressources et avec des volumes horaires différents. Une ressource est identifiée de manière unique par son `codeRess`.

numEns (int) (1)	codeRess (String) (1)	nomRess (String)	coeffRess (int)	semestre (String)	nomEns (String)	gradeEns (String)	volHoraire (int)
1	R1.05	Intro BDR	2	S1	Fleurquin	Mcf	15
4	R1.05	Intro BDR	2	S1	Ridard	Prag	30
3	R1.05	Intro BDR	2	S1	Naert	Prag	30
2	R1.05	Intro BDR	2	S1	Pham	Mcf	15
1	R2.01	Dev OO	3	S2	Fleurquin	Mcf	15
5	R2.01	Dev OO	3	S2	Borne	Pr	30
3	R2.01	Dev OO	3	S2	Naert	Prag	30
20	R2.01	Dev OO	3	S2	Paul	Vac	15
1	R3.14	PPP Pro	1	S3	Fleurquin	Mcf	15
5	R3.14	PPP Pro	1	S3	Borne	Pr	15
6	R4.12	Automates	4	S4	Gaudin	Prag	45

a) L'enseignant *Fleurquin* obtient une promotion et obtient le grade de *Pr*. Que faut-il faire ?
b) La ressource *R4.12* disparait de la formation. Que faut-il faire ?
c) L'enseignant *Gaudin* obtient une mutation et quitte l'IUT. Que faut-il faire ?
d) On observe qu'il y a beaucoup de redondance dans cette relation. Lesquelles ? Pourquoi ?

Un enseignant de base de données propose de scinder cette relation en 3 relations de schéma :

```
Affectations(    @ens : int REF Enseignants(numEns) (1)
                 @ress : String REF Ressources(codeRess) (1)
                 volHoraire : int (NN)
)

Enseignants(     numEns : int (1)
                 nomEns : String (NN)
                 gradeEns : String(NN)
)
```

```
Ressources(     codeRess : String (1)
                nomRess : String (NN)
                coeffRess : int (NN)
                semestre : String (NN)
)
```

e) Donnez le contenu de chacune des 3 relations pour retranscrire une information « cohérente » avec celle de la relation d'origine.

f) Cette réorganisation conduit-elle à un gain ou à une perte de place en mémoire ?

g) Montrez qu'il est possible sans perte d'information de reconstruire par « jointure naturelle » la relation originale depuis ces 3 nouvelles relations.

h) Que deviennent les difficultés identifiées en question a), b), c) ?

Un étudiant de BUT 1 voulant impressionner son enseignant de base de données propose une autre manière de stocker toutes ces informations. Il propose de scinder la relation d'origine en 5 relations de schéma :

```
Affectations(   @ens : int REF Enseignants(numEns) (1)
                @ress : int REF Ressources(codeRess) (1)
)

Services(       gradeEns : String (1)
                volHoraire : int (1)
)

Enseignants(    numEns : int (1)
                nomEns : String (NN)
                @grade : String REF Services(gradeEns) (NN)
)

Ressources(     codeRess : String (1)
                nomRess : String (NN)
                @coeff : int REF Maquettes(coeffRess) (NN)
)

Maquettes(      coeffRess : int(1)
                semestre : String (NN)
)
```

i) Donnez le contenu de chacune des 5 relations pour retranscrire une information « cohérente » avec celle de la relation d'origine.

j) Montrez qu'il n'est pas possible de retrouver par des jointures avec ce schéma l'information d'origine concernant les volumes horaires (il y a donc perte en partie de cette information) ? Expliquez le pourquoi de ce phénomène.

k) Montrez qu'il est possible de retrouver ici par jointure pour une ressource son coefficient et son semestre (il n'y a donc pas ici de perte de cette information). Est-il possible d'avoir deux ressources de même coefficient mais dans des semestres différents dans le schéma d'origine et dans ce nouveau schéma ? Montrez que ce phénomène est lié à des modifications dans les dépendances entre les données.

Cet exercice illustre le fait que tous les schémas relationnels ne se valent pas pour stocker la même information ! Les pires schémas font perdre de l'information. Certains schémas ne font pas perdre d'information mais altèrent les dépendances entre les données et donc entrainent des risques de corruption des données par mise à jour dans le futur. L'art de la conception d'une base de données consiste à proposer des schémas qui réalisent un compromis entre cinq besoins parfois non réconciliables : i) une faible redondance des données, ii) la non perte de données lors des jointures iii) la conservation des dépendances (sémantiques) entre les données, iv) la rapidité des accès et v) la maintenabilité de la base face aux évolutions. L'étude du **niveau de normalisation** d'un schéma est un outil clé dans la recherche du juste compromis.

Exercice 6. Inconvénients du non-respect de la 1NF

Soit la relation ci-dessous.

numÉtudiant (int)	codeModule (String)	Notes (String)
11	BCO	10,11, 15
8	PROG	9, 12, 8, 4
11	PROG	11, 10, 14, 15

a) On veut identifier les étudiants ayant eu moins de 10 au premier contrôle de PROG. On veut également calculer la moyenne sur la promotion du second contrôle du module de PROG. Comment doit-on faire ?

b) Est-il facile de rajouter à chaque contrôle un coefficient qui lui est propre ? Quel problème cela pose si l'on veut modifier le coefficient d'un contrôle ?

Est en première forme normale (1NF), une relation (ayant par définition au moins une clé candidate) dont les attributs possèdent tous une valeur sémantiquement atomique. Un attribut est dit « atomique » si aucune subdivision de la donnée initiale n'apporte une information supplémentaire ou complémentaire dans le domaine. Ici ce n'est pas le cas de l'attribut `Notes` qui n'est pas une valeur unique mais une liste de valeurs avec chacune une sémantique intéressante pour le domaine. Mettre des « listes » dans un seul attribut est très souvent une erreur majeure de conception car elle contraint à de pénibles manipulations dans les applications de structure de données de type nombre ou chaîne de caractères alors que l'atomisation des informations dans la base permettrait l'écriture de requêtes beaucoup plus simples intégralement prises en charge par le SGBD.

Exercice 7. Inconvénients du non-respect de la 2NF

Voici une nouvelle base de données usant de deux relations. Cette base permet de résoudre les problèmes précédents. Mais elle en pose d'autres 😉. Voici son schéma relationnel.

```
Résultats(  numEtudiant : int (1),
            @(module : String, contrôle : int)  REF Evaluations(codeModule, numContrôle) (1)
            note : int (NN)
)

Evaluations(
            codeModule : String (1),
            NumContrôle : int (1),
            intituléModule : String
)
```

Et voici un aperçu de ce qu'elle contient.

Résultats

numÉtudiant	module	contrôle	note
11	BCO	1	10
11	BCO	2	11
11	BCO	3	15
8	PROG	1	9
…	…	…	…

Evaluations

codeModule	numContrôle	intituléModule
BCO	1	Base Conception Objet
BCO	2	Base Conception Objet
BCO	3	Base Conception Objet
PROG	1	Programmation à Objets
…	…	…

a) Complétez les deux relations pour retrouver toutes les informations de la base de données de la question 1.
b) Montrez qu'il est ici beaucoup plus facile de réaliser des calculs de moyenne par étudiant, module ou contrôle.
c) Est-il facile de rajouter avec ce nouveau schéma un coefficient à chaque contrôle ?

d) On souhaite modifier dans la base l'intitulé du module de BCO. Que faut-il faire ? Quel problème pose en termes de redondance d'information, d'intégrité des données et de performance la structure actuelle de la base ?

Les 2 relations sont ici en 1NF. Cependant la relation `Évaluations` ne respecte pas la 2NF. En effet l'attribut `intituléModule` est entièrement « fixé » par l'attribut `codeModule` : a un code de module donné ne correspond qu'un et un seul intitulé de module. Formellement on dit que l'attribut `intituléModule` est en *dépendance fonctionnelle* avec l'attribut `codeModule` ce que l'on note : `codeModule` → `intituléModule`. Or la clé primaire de la table `Evaluations` est composée des deux attributs (`codeModule, numContrôle`). L'intitulé du module est donc dépendant d'un sous-ensemble de la clé de la table. Il y a donc un risque de répétition de l'intitulé du module autant de fois qu'apparait le code du module dans la relation. Il peut apparaître plusieurs fois car le code du module n'est pas à lui tout seul clé de la relation : il y aura autant de répétitions du code du module (et donc de son intitulé !) que de contrôle à définir dans un module.

Les dépendances fonctionnelles sont l'outil théorique qui permet d'étudier le niveau de redondance d'une relation. C'est l'outil de base dans la théorie de la normalisation. Les seules dépendances fonctionnelles réellement utiles sont les dépendances *élémentaires* : i) un seul attribut cible ii) l'attribut cible n'est pas dans la source (non trivialité) et iii) la source est minimale (pas de gras !). Depuis les dépendances élémentaires ont peut retrouver toutes les dépendances fonctionnelles à l'aide de propriétés mathématiques pleine de bon sens.

Une relation est dite en 2NF si et seulement si : *elle est en 1NF et tout attribut non membre d'une clé ne dépend jamais d'un sous-ensemble strict des attributs d'une clé*. Dit autrement : elle est 1NF et toutes les dépendances ayant pour source une clé sont élémentaires.

Astuce : une relation n'ayant que des clés candidates formées d'un seul attribut est donc forcément en deuxième forme normale.

Le non-respect de la 2NF pose des problèmes de redondance, des risques de perte d'intégrité et de baisse de performance.

Exercice 8. Inconvénients du non-respect de la 3NF

Voici une nouvelle base de données usant de trois relations. Elle permet de résoudre les problèmes précédents. Mais elle en pose, une fois encore, d'autres 😉. Voici son schéma relationnel.

```
Résultats(  numEtudiant : int (1),
            @(module : String, contrôle : int)  REF Evaluations(codeMod, numContrôle) (1)
            note : int
)

Evaluations(
            @(codeMod : String) REF Modules(codeModule) (1),
            NumContrôle : int (1),
            coefficient : int (NN)
)

Modules(
            codeModule : String (1),
            intituléModule : String
            nomResponsable : String
            bureauResponsable : int
)
```

Et voici un aperçu de ce qu'elle contient. On suppose qu'il n'y a pas d'enseignants qui ont le même nom et qu'un enseignant n'a qu'un et un seul bureau. Plusieurs enseignants peuvent partager le même bureau.

Résultats

numÉtudiant	module	contrôle	note
11	BCO	1	10
11	BCO	2	11
11	BCO	3	15
8	PROG	1	9
…	…	…	…

Evaluations

codeMod	numContrôle	coefficient
BCO	1	2
BCO	2	1
BCO	3	3
PROG	1	1
…	…	…

Modules

codeModule	intituléModule	nomResponsable	bureauResponsable
BCO	Base Conception Objet	Fleurquin R.	101
PROG	Programmation à Objets	Borne I.	105
TLANG	Théorie des Langages	Fleurquin R.	101
…	…	…	…

a) Vérifiez que ce schéma relationnel est bien en 2NF.
b) On veut changer de bureau l'enseignant « Fleurquin R » que faut-il faire ? Quel problème pose en termes de redondance d'information, d'intégrité des données et de performance la structure actuelle de la base ?
c) Est-il possible sans créer de redondance d'ajouter un coefficient à un module ?
d) Est-il possible sans créer de redondance d'ajouter un statut (agrégé, vacataire ou enseignant chercheur) à un responsable de module ? Proposez un nouveau schéma relationnel pour permettre la chose sans redondance.
e) Avec ce nouveau schéma, on veut connaître la liste des étudiants qui ont été notés par les enseignants du bureau 101. Que faut-il faire ?

Les 3 relations sont en 2NF. Cependant la relation `Modules` ne respecte pas la 3NF. En effet l'attribut `bureauResponsable` est entièrement « fixé » par l'attribut `nomResponsable` : à un enseignant donné ne correspond qu'un et un seul bureau. On a donc : `nomResponsable` → `bureauResponsable`. Le bureau qui n'est pas un attribut membre d'une clé est en dépendance avec un autre attribut qui lui-même n'est pas membre d'une clé. Il y aura donc répétition du numéro de bureau d'un enseignant autant de fois qu'il apparait dans la relation. Il va donc apparaître plusieurs fois car l'enseignant peut être responsable de plusieurs modules (il n'est pas une clé de la relation).

Une relation est dite en 3NF si et seulement si : *elle est en 2NF et tout attribut non membre d'une clé ne dépend jamais d'autres attributs non membres d'une clé*. Dit autrement : la relation est 1NF et toutes les dépendances fonctionnelles élémentaires ciblant des attributs non membre de clé ont pour sources des clés.

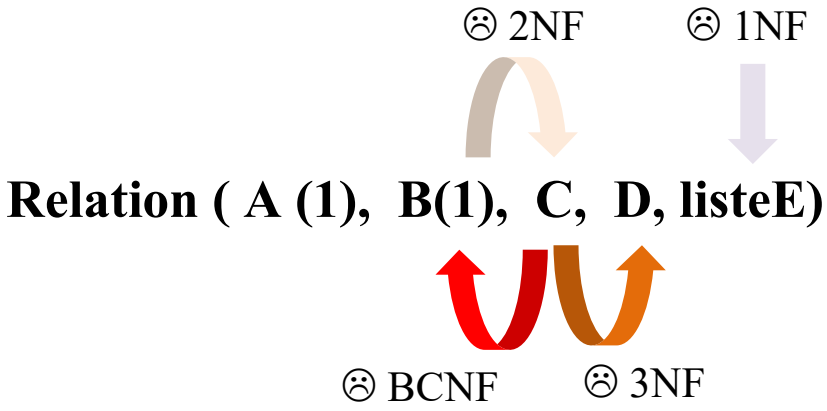
Le non-respect de la 3NF pose des problèmes de redondance, des risques de perte d'intégrité et de baisse de performance. On recherche toujours a minima le respect de la 3NF.

Astuce : une relation 2NF qui a un seul de ses attributs qui n'est pas membre d'une clé est forcément en 3NF.

Il existe des formes normales plus contraignantes encore que la 3NF.

La forme BCNF impose d'être 1NF et que toutes les dépendances élémentaires de la relation soient du type K → `A` (avec K une clé candidate et A un attribut quelconque membre d'une clé ou non). C'est la plus simple à retenir !

Une relation BCNF est toujours 2NF (car toutes les dépendances se font sur l'intégralité des clés !) et 3NF (car un attribut non clé ne peut être à l'origine, à gauche donc, d'une dépendance !). On notera que la BCNF ne fait que rajouter à la 3NF l'interdiction qu'un attribut membre d'une clé dépende d'attributs non clé.



Exercice 9. Reconnaître le niveau de normalisation d'un schéma relationnel

Pour chacun des schémas de relation indiquez s'il est en 1NF, 2NF, 3NF ou BCNF. Identifiez si besoin les redondances potentielles. Toutes les clés candidates sont identifiées. Les dépendances entre les attributs sont celles de « la vraie vie ».

1. Entreprises(numEntreprise (1), nomEntreprise, chiffreAffaires, listeSitesProduction)
2. Entreprises(numEntreprise (1), nomEntreprise, chiffreAffaires, villeSiège)
3. Commandes(numClient (1), numProduit (1), nomProduit, prixProduit, quantitéCommande)
4. Etudiants(numéroEtudiant (1), nom (2), prénom (2), filièreSup, sérieBac, dateNaissance, dateFête)
5. Salariés(numSalarié (1), numEntreprise, salaire, poste, adresseSiègeEntreprise)
6. Salariés(nomSalarié (1), prénomSalarié (1), salaire)
7. Résultats(numRessource (1), numEtudiant (1), numContrôle (1), note, coeffContrôle, coeffRessource)
8. Prêts(numAdhérent (1), numExemplaire (1), dateEmprunt (1), dateRetour)
9. Abonnés(numAdhérent (1), nomAdhérent, départementAdhérent, régionAdhérent)
10. Planning(numClasse(1), numMatière (1), numEnseignant, volumeHoraire). On suppose qu'un enseignant peut intervenir dans plusieurs classes mais toujours dans la même matière.

Il peut vous sembler difficile de normaliser vos schémas de bases de données. Effectivement il faut en théorie faire des études mathématiques parfois pénibles de dépendances entre les données. Vous allez maintenant vous initier à la modélisation de domaines avec UML et à la traduction de ces modèles en schéma relationnel. Ces deux étapes qui viennent en amont de l'implantation d'une base de données relationnelle, si elles sont bien faites, vont vous permettre très souvent et sans effort supplémentaire d'obtenir un schéma relationnel directement en 3NF. Pas besoin, le plus souvent, de faire des mathématiques. Ouf !