Deux relations en liaison

Le tableau 2.2 décrit les cardinalités maximum potentielles des associations d'agrégation à partir de deux relations en liaison. Nous avons numéroté les schémas relationnels en lettres majuscules italiques (*A, B, C*). Le cas *C* traite des références inverses, par conséquent ce schéma peut exprimer soit deux associations d'agrégation distinctes (cas 5, 6, 7, 8) soit une seule association d'agrégation (cas 9).

Tableau 2.2 Cardinalités maximales potentielles des associations d'agrégation à partir de deux relations

Schéma relationnel		Cardi. max. A1/E3	Condition sur les données	N°
A1[a1#,b1#, d1...]		N-1	(a1, b1) non unique dans E3	1
E3[c1 ,c2...,(a1, b1)#]	*(A)*	1-1	(a1, b1) unique dans E3	2
A1[a1# ,b1#, d1..., c1#]		1-N	c1 non unique dans A1	3
E3[c1, c2...]	*(B)*	1-1	c1 unique dans A1	4
A1[a1#, b1#, d1..., c1#]		1-N	c1 non unique dans A1	5
		N-1	(a1, b1) non unique dans E3	
E3[c1, c2...,(a1, b1)#]		1-1	(a1, b1) unique dans E3	6
		N-1	(a1, b1) non unique dans E3	
		1-N	c1 non unique dans A1	7
		1-1	(a1, b1) unique dans E3	
		1-1	c1 unique dans A1	8
		1-1	(a1, b1) unique dans E3	
	(C)	1-1	(a1, b1, c1) se retrouvent simultanément dans E3 et A1	9

Trois relations en liaison

Deux attributs dans la clé primaire

Le tableau décrit les cardinalités maximales potentielles des associations d'agrégation à partir de trois relations en liaison avec le degré des clés primaires inférieur à trois (nombre d'attributs composant la clé primaire). Les cas *F* et *G* traitent des références inverses, par conséquent ces schémas peuvent exprimer soit deux associations d'agrégation distinctes, soit une seule association d'agrégation (cas 18 pour le cas *F* et 23 pour le cas *G*).

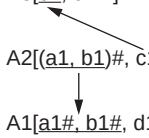
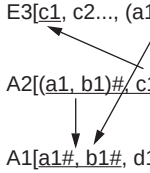
Tableau 2.3 Cardinalités maximales potentielles des associations d'agrégation à partir de trois relations

Schéma relationnel	Cardi. max. A1/E3	Condition sur les données	N°
E3[c1, c2..., (a1, b1)#] A2[(a1, b1)#, p1...] A1[a1#, b1#, d1...] *(D)*	N-1	(a1, b1) non unique dans E3	10
	1-1	(a1, b1) unique dans E3	11
E3[c1, c2...] A2[(a1, b1)#, c1#, p1...] A1[a1#,b1#, d1...] *(E)*	1-N	c1 non unique dans A2	12
	1-1	c1 unique dans A2	13
E3[c1, c2..., (a1, b1)#] A2[(a1, b1)#, c1#, p1...] A1[a1#, b1#, d1...] *(F)*	1-N	c1 non unique dans A2	14
	N-1	(a1, b1) non unique dans E3	
	1-1	(a1, b1) unique dans E3	15
	N-1	(a1, b1) non unique dans E3	
	1-N	c1 non unique dans A2	16
	1-1	(a1, b1) unique dans E3	
	1-1	c1 unique dans A2	17
	1-1	(a1, b1) unique dans E3	
	1-1	(a1, b1, c1) se retrouvent simultanément dans E3 et A2	18
E3[c1, c2..., (a1, b1)#] A2[(a1, b1)#, c1#, p1...] A1[a1#, b1#, d1...] *(G)*	1-N	c1 non unique dans A2	19
	N-1	(a1, b1) non unique dans E3	
	1-1	(a1, b1) unique dans E3	20
	N-1	(a1, b1) non unique dans E3	
	1-N	c1 non unique dans A2	21
	1-1	(a1, b1) unique dans E3	
	1-1	c1 unique dans A2	22
	1-1	(a1, b1) unique dans E3	
	1-1	(a1, b1, c1) se retrouvent simultanément dans E3 et A2	23

Trois attributs dans la clé primaire

Le tableau suivant décrit les cardinalités maximales potentielles des associations d'agrégation à partir de trois relations en liaison avec le degré des clés primaires égal à trois (nombre d'attributs composant la clé primaire). Le cas *I* traite de références inverses, par conséquent ce schéma peut exprimer soit deux associations d'agrégation distinctes soit une seule association d'agrégation (cas 36).

Tableau 2.4 Cardinalités maximales potentielles des associations d'agrégation à partir de trois relations

Schéma relationnel	Cardi. max. A1/E3	Condition sur les données	N°
 *(H)*	N-N	c1 non unique dans A2 pour (a1,b1) donné et (a1,b1) non unique dans A2 pour c1 donné	24
	N-1	c1 unique dans A2 pour (a1,b1) donné et (a1,b1) non unique dans A2 pour c1 donné	25
	1-N	c1 non unique dans A2 pour (a1,b1) donné et (a1,b1) unique dans A2 pour c1 donné	26
	1-1	c1 unique dans A2 pour (a1,b1) donné et (a1,b1) unique dans A2 pour c1 donné	27
 *(I)*	N-1	(a1,b1) non unique dans E3	28
	N-N	cf. 24	
	N-1	(a1,b1) non unique dans E3	29
	N-1	cf. 25	
	N-1	(a1,b1) non unique dans E3	30
	1-N	cf. 26	
	1-N	(a1,b1) non unique dans E3	31
	1-1	cf. 27	
	1-1	(a1,b1) unique dans E3	32
	N-N	cf. 24	
	1-1	(a1,b1) unique dans E3	33
	N-1	cf. 25	
	1-1	(a1,b1) unique dans E3	34
	1-N	cf. 26	
	1-1	(a1,b1) unique dans E3	35
	1-1	cf. 27	
	1-1	(a1,b1,c1) se retrouvent simultanément dans E3 et A2	36

Le tableau 2.5 décrit les cardinalités maximales potentielles des associations d'agrégation avec la relation qui représente l'association d'agrégation contenant deux clés étrangères composées.

Tableau 2.5 Cardinalités maximales potentielles des associations d'agrégation à partir de trois relations

Schéma relationnel	Cardi. max. A1/E3	Condition sur les données	N°
A1[a1#, b1#, d1...] A3[(a1, (b1)#, c1)#, m1...] A2[(b1, c1)#, p1...] *(J)*	N-1	c1 unique dans A3 pour (a1, b1) donné et (a1, b1) non unique dans A3 pour c1 donné	37
	1-N	c1 non unique dans A3 pour (a1, b1) donné et (a1, b1) unique dans A3 pour c1 donné	38
	1-1	c1 unique dans A3 pour (a1, b1) donné et (a1, b1) unique dans A3 pour c1 donné	39
A1[a1#, b1#, d1...] A3[(a1, (b1)#, c1)#, m1...] A2[(b1, c1)#, p1...] *(K)*	N-N	c1 non unique dans A3 pour (a1, b1) donné et (a1, b1) non unique dans A3 pour c1 donné	40
	N-1	cf. 37	41
	1-N	cf. 38	42
	1-1	cf. 39	43

Je ne parle pas des cardinalités minimales car elles n'influencent pas la structure des relations.

Du conceptuel à l'objet

Nous décrivons dans cette section le processus de passage d'un schéma conceptuel (entité-association ou UML) au modèle objet. Pour chaque association du schéma conceptuel, nous préconisons des transformations dans le modèle objet, qui faciliteront par la suite la déclaration du script SQL3 de la base de données.

Transformation des entités/classes

Classes du schéma navigationnel

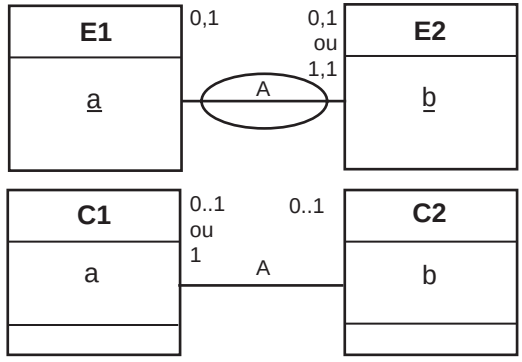
Chaque entité du schéma conceptuel entité-association devient une classe du schéma navigationnel.

Chaque classe UML, excepté les classes-associations, devient une classe du schéma navigationnel.

Transformation des associations un-à-un

On recense les quatre possibilités de transformation d'une association *un-à-un* du modèle conceptuel dans le modèle objet.

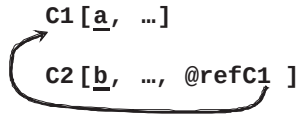
Figure 2-63 *Associations un-à-un*



Solutions avec une référence

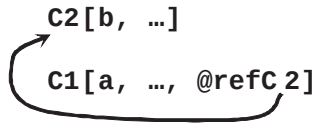
La première solution de transformation met en œuvre une référence entre les deux classes. Dans l'exemple 2-64, ce lien est `refC1`. La référence pourra être nulle (traduction de la multiciplé minimale `0`).

Figure 2-64 *Une référence*



La deuxième solution est symétrique de la précédente.

Figure 2-65 *L'autre référence*



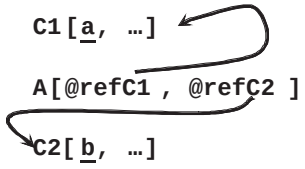
Solution avec deux références

La troisième solution met en œuvre une référence inverse.

Figure 2-66 *Référence et référence inverse*

La dernière solution est basée sur l'utilisation d'une troisième classe. Cette classe porte le nom de l'association et contient deux références.

Figure 2-67 *Solution universelle*

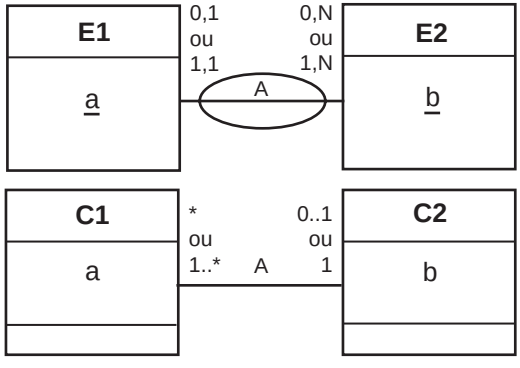


Cette solution présente un avantage : elle est valable pour tous les types d'associations (on la dit ainsi universelle). La structure de la base n'a pas à être modifiée si les cardinalités sont modifiées (activation ou désactivation de contraintes SQL `UNIQUE` au niveau des références).

Associations un-à-plusieurs

On recense quatre possibilités de transformation d'une association *un-à-plusieurs* du modèle conceptuel dans le modèle objet.

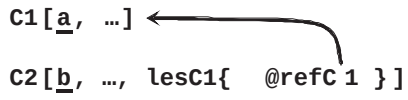
Figure 2-68 *Associations un-à-plusieurs*



Collection de références

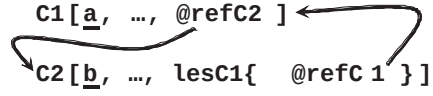
La première solution de transformation met en œuvre une collection de références. Dans l'exemple 2-69, la collection s'appelle `lesC1`, elle contient une référence `refC1`.

Figure 2-69 *Collection de références*



La deuxième solution met en œuvre la première solution avec une référence inverse (du fils vers le père). Dans notre exemple, la référence inverse est `refC2`.

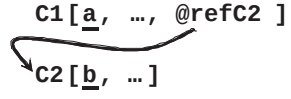
Figure 2-70 *Collection de références avec référence inverse*



Solutions sans collection

La troisième solution est analogue au modèle relationnel : il s'agit de recourir uniquement à une référence du fils vers le père.

Figure 2-71 *Une référence*

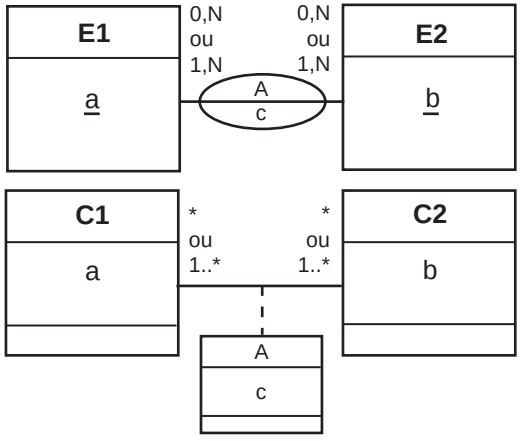


La quatrième alternative est la solution universelle, basée sur l'utilisation de deux références (figure 2-67).

Associations plusieurs-à-plusieurs

On recense deux possibilités de transformation d'une association *plusieurs-à-plusieurs* du modèle conceptuel dans le modèle objet.

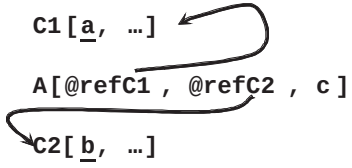
Figure 2-72 *Associations plusieurs-à-plusieurs*



Première solution (solution universelle)

La solution universelle met en œuvre une classe contenant les attributs de l'association et deux références.

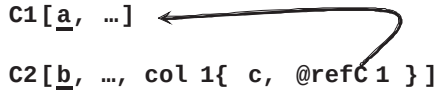
Figure 2-73 *Solution universelle*



Deuxième solution : une collection de références

La deuxième solution met en œuvre une collection qui contient les attributs de l'association et une référence. L'accès aux données est privilégié par la classe qui héberge la collection. Dans l'exemple 2-74, la collection `coll` contient l'attribut `c` et la référence `refC1`.

Figure 2-74 *Transformations par une collection de références*



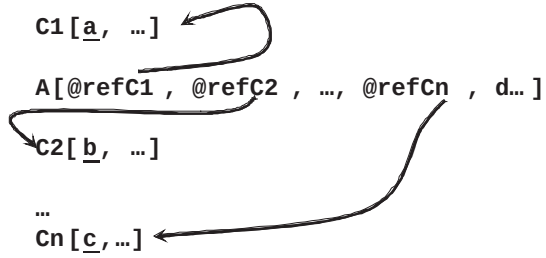
Associations n-aires

On recense *n*+1 possibilités de transformation d'une association *n*-aire du modèle conceptuel dans le modèle objet. Il est possible de privilégier chaque entité/classe qui compose l'association *n*-aire ou de n'en privilégier aucune (par la solution universelle).

Première solution (solution universelle)

La solution universelle consiste à mettre en œuvre une *n*+1e classe qui porte le nom de l'association, et contient les attributs de l'association et *n* références (figure 2-75).

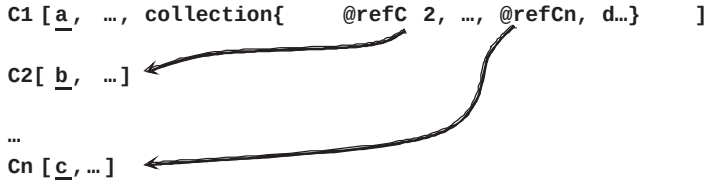
Figure 2-75 *Solution universelle pour les associations n-aires*



Collection de références

Chacune des autres solutions privilégie une des classes qui contiendra une collection incluant les attributs de l'association et *n*-1 références. L'exemple 2-76 illustre la traduction de l'association *n*-aire `A` qui contient l'attribut `d` et est reliée à *n* entités/classes. Ici la classe `C1` est privilégiée.

Figure 2-76 *Solution dans laquelle la classe `C1` est privilégiée*



Bilan

On peut se demander à juste titre quelle solution utiliser pour chacune des associations du schéma conceptuel qu'il faut implanter dans la base de données.

Avantages et inconvénients

Les collections privilégient l'accès aux données *via* une table au détriment d'une autre. Considérons l'exemple 2-76, l'accès aux données est privilégié par la classe C1. La requête qui extrait les attributs des classes `C2` à `Cn` en fonction d'un objet `C1` donné sera immédiate. L'utilisation de références simplifie l'expression des requêtes dans le langage d'interrogation. En ce sens, il est intéressant d'utiliser la solution universelle, même si ce n'est pas la panacée.

Le problème des références, c'est qu'elles ne fournissent pas encore totalement les fonctionnalités des clés étrangères. L'intégrité référentielle est à programmer explicitement. De plus, il n'est pas encore possible d'exprimer certaines contraintes sur une référence (comme définir un index de type clé primaire sous Oracle). L'avenir nous dira comment vont évoluer ces techniques quand même prometteuses.

Les solutions les plus relationnelles

Le tableau 2.6 indique la solution du modèle objet la plus proche d'une solution relationnelle classique. Par exemple, dans le cadre d'une association *un-à-plusieurs*, le modèle relationnel impose l'utilisation d'un attribut de type clé étrangère dans la table *fils* référençant la table *père*. La solution la plus proche au niveau navigationnel est la troisième, c'est-à-dire celle qui met en œuvre une référence.

Tableau 2.6 Correspondances objet/relationnel

Association	Modèle navigationnel
un-à-un	solution 2
un-à-plusieurs	solution 3
plusieurs-à-plusieurs	solution 1
n-aire	solution 1

Contraintes du niveau conceptuel

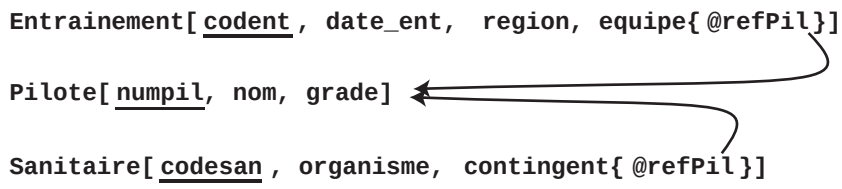
Ces contraintes seront réellement prises en compte au niveau de l'implémentation des méthodes. Il est difficile de déterminer à ce niveau la classe qui hébergera la méthode. Tout dépend du SGBD qui rendra certains cas de figure impossibles. Par exemple, Oracle (version 10*g* R2) ne permet pas de programmer un déclencheur (*trigger*) sur une collection (*nested table*). En conséquence, des méthodes seront probablement déplacées dans d'autres classes lors de l'implantation.

Considérons la contrainte de partition qui impose à tout pilote d'être affecté soit à un exercice d'entraînement, soit à une mission sanitaire. Lors de la création (ou modification) d'un objet `Pilote`, il faudra s'assurer que ledit objet est relié soit à un objet `Sanitaire`, soit à un objet `Entrainement`. Ici deux associations *un-à-plusieurs* sont à contraindre.

Collections de références

L'exemple 2-77 illustre le premier cas de transformation avec une collection de références. La collection s'appelle `contingent` dans la classe `Sanitaire` et `equipe` dans la classe `Entrainement`.

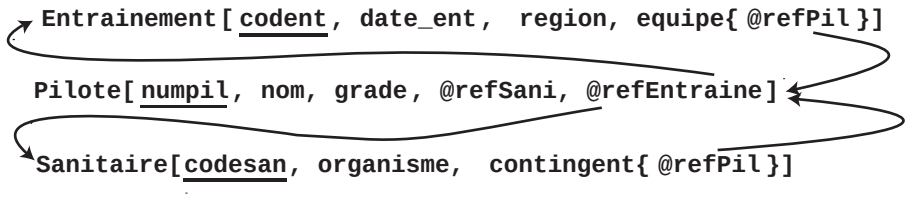
Figure 2-77 *Collection de références*



La contrainte de partition impose de vérifier que chaque référence de la classe `Pilote` (OID) ne doit apparaître qu'une fois au plus dans l'union des collections `contingent` et `equipe`.

Le deuxième cas de transformation d'une association *un-à-plusieurs* fait intervenir une collection de références et une référence inverse. La contrainte de partition porte alors sur les trois

Figure 2-78 *Deuxième solution de transformation*



classes. Il faut s'assurer que :

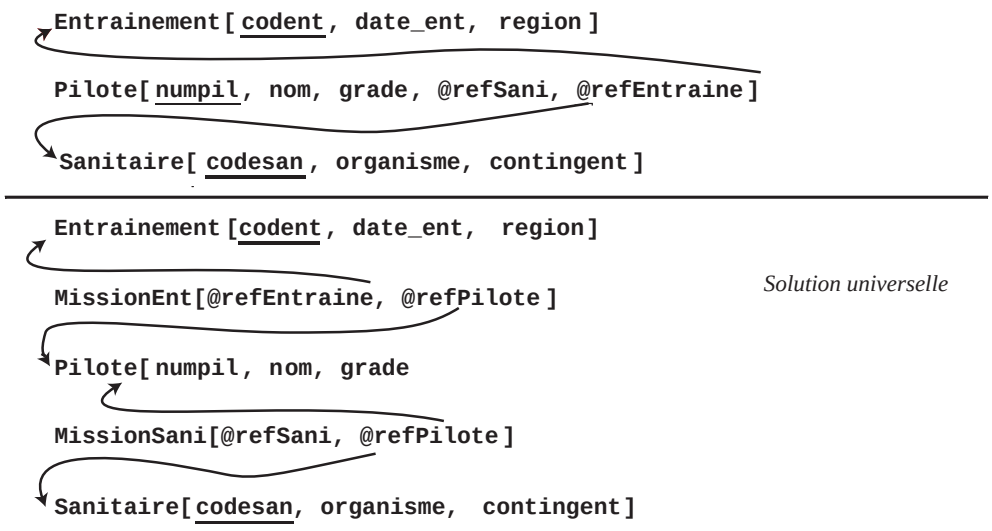
- chaque référence de la classe `Pilote` (OID) ne doit apparaître qu'une fois au plus dans l'union des collections `contingent` et `equipe` ;
- pour chaque objet `Pilote`, l'une des références `refSani` et `refEntraine` est nulle.

Solutions sans collection

Les autres transformations mettent en œuvre des références sans collection. La figure 2-79 décrit ces deux solutions. Dans le premier cas, il faut s'assurer que, pour chaque objet `Pilote`, l'une des références `refSani` et `refEntraine` est toujours nulle.

Le second cas correspond à la solution universelle. Les références `refPil` doivent être toutes distinctes et non nulles.

Figure 2-79 *Solutions sans collection*

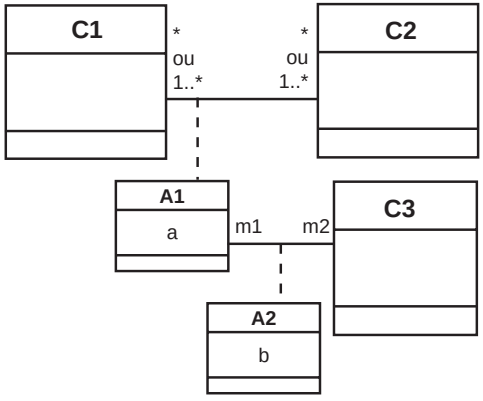


Le même type de raisonnement vaut pour toute autre contrainte (exclusivité, totalité, inclusion, unicité) qu'il faudra programmer en fonction de la nature de l'association (*un-à-un*, *un-à-plusieurs*, etc.).

Classes-associations UML

Nous étudions ici la transformation d'une classe-association reliée à une autre classe-association (classe A2 de la figure 2-80). Ce cas se présente lors de la présence de contraintes d'inclusion ou d'unicité sur une association *n*-aire. Bien que nous considérions un exemple générique (*n* = 3), le même raisonnement vaut pour un degré d'association supérieur.

Figure 2-80 *Classes-associations UML*



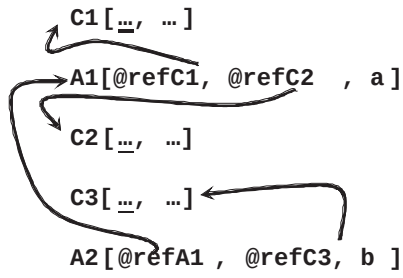
La classe-association A2 traduit une contrainte d'inclusion. Plusieurs multiplicités m1 et m2 de l'association A2 peuvent se présenter. Raisonnons sur le chiffre maximal des multiplicités (1 pour les multiplicités 1 et 0..1, * pour les multiplicités * et 1..*). La multiplicité maximale 1 permet de traduire une contrainte d'unicité.

En fonction des multiplicités, plusieurs schémas objet seront possibles. Nous en retenons deux : la solution universelle et une solution basée sur les collections. Nous modéliserons donc une association *plusieurs-à-plusieurs* (A1) et une association binaire (entre A1 et C3) en fonction des multiplicités m1 et m2. Il faudra programmer la contrainte en fonction des valeurs m1 et m2 sur les références du schéma navigationnel concernées par l'association A2.

Solution universelle

La solution universelle est illustrée à la figure 2-81, la contrainte va porter sur les références refA1 et refC3.

Figure 2-81 *Solution universelle pour les classes-associations*



La contrainte d'inclusion induite par l'association A2 ne requiert pas de contrainte explicite sur les liens, car la classe A2 est directement reliée à la classe A1 par la référence refA1. Les conditions induites par la contrainte d'unicité sont décrites dans le tableau 2.7.

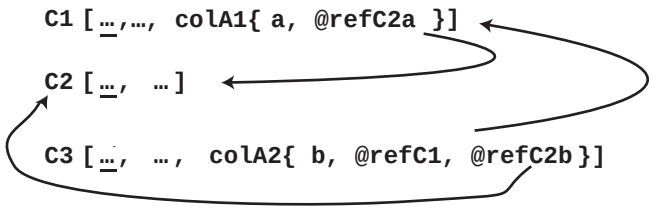
Tableau 2.7 Contraintes à respecter avec la solution universelle

m1	m2	Contrainte
1	1	Pour tous les objets de la classe A2, les références refA1 et refC3 sont toutes distinctes.
1	*	Pour tous les objets de la classe A2, le lien refC3 est distinct. Plusieurs objets de la classe A2 peuvent avoir le même lien refA1.
*	1	Pour tous les objets de la classe A2, le lien refA1 est distinct. Plusieurs objets de la classe A2 peuvent avoir le même lien ref3.
*	*	Aucune : plusieurs objets de la classe A2 peuvent avoir le même lien refC3 ou refA1.

Solution par collections

L'exemple 2-82 représente l'association *plusieurs-à-plusieurs* (A1) et une association binaire entre A1 et C3. L'association A1 se compose d'une collection de deux références, et l'association A2 d'une structure et de trois références. La contrainte va porter sur toutes les références.

Figure 2-82 *Collections pour les classes-associations*



La contrainte d'inclusion induite par l'association A2 s'énonce ainsi : pour tout objet A2, la référence refC1 pointe un objet C1 qui possède une référence refC2a sur un objet C2 d'OID *x*, et la référence refC2b doit pointer l'objet d'OID *x*. Cette contrainte est notée ci. Les conditions sont décrites dans le tableau 2.8.

Tableau 2.8 Contraintes à respecter pour la solution avec structures

m1	m2	Contraintes
1	1	Pour tous les objets de type A2, les couples de références (refC1, refC2b) sont distincts deux à deux et on n'utilise pas la collection colA2. La contrainte ci doit aussi être respectée.
1	*	On n'utilise pas la collection colA2 et la contrainte ci doit être respectée.
*	1	Pour tous les objets A2, les couples de références (refC1, refC2b) sont distincts deux à deux et on utilise la collection colA2. La contrainte ci doit aussi être respectée.
*	*	On utilise la collection colA2 et la contrainte ci doit être respectée.

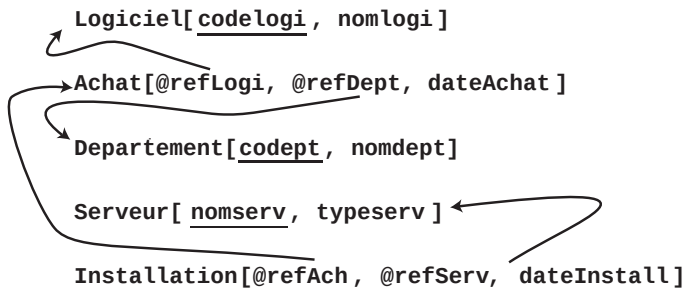
Exemple

Considérons les installations de logiciels avec la contrainte d'unicité (un logiciel d'un département ne doit être installé que sur un seul serveur) et la contrainte d'inclusion (une installation est valide seulement si le logiciel a été acheté par le département). L'exemple 2-83 met en œuvre la solution universelle. Le tableau 2.9 décrit les contraintes avec les multiplicités m1=* et m2=1.

Tableau 2.9 Contraintes à respecter pour la solution universelle

m1	m2	Contrainte
*	1	Pour tous les objets Installation, la référence refAch est distincte. Plusieurs objets Installation peuvent avoir la même référence refServ.

Figure 2-83 *Solution universelle*



L'exemple 2-84 met en œuvre des collections. Le tableau 2.10 décrit les contraintes avec les mêmes multiplicités.

Figure 2-84 *Solution basée sur les collections*

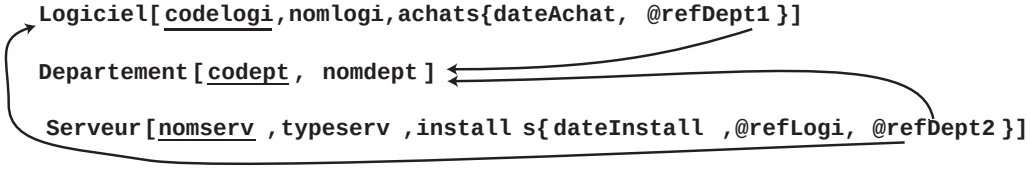


Tableau 2.10 Contraintes à respecter pour la solution avec collections

m1	m2	Contraintes
*	1	**Inclusion** : pour tous les éléments de la collection `installs`, la référence `refLogi` pointe un objet `Logiciel` qui possède une référence `refDept1` sur un objet `Département` d'OID *x*, et la référence `refDept2` pointe l'objet d'OID *x*. **Unicité** : pour tous les éléments de la collection `installs`, les couples de références (`refLogi`, `refDept2`) sont distincts deux à deux.

Agrégations UML

Une agrégation peut être modélisée de plusieurs manières :

- par des collections ou des structures dans la classe composite ;
- par une classe composite reliée aux différentes classes composantes par des références ;
- une combinaison de ces deux approches.

La première solution convient si aucune autre classe que la classe composite n'est reliée à des objets des structures de la classe composite. La seconde solution semble mieux adaptée au partage d'objets de classes composantes entre différentes classes composites. La dernière solution permet de mélanger à la demande les deux premières approches.

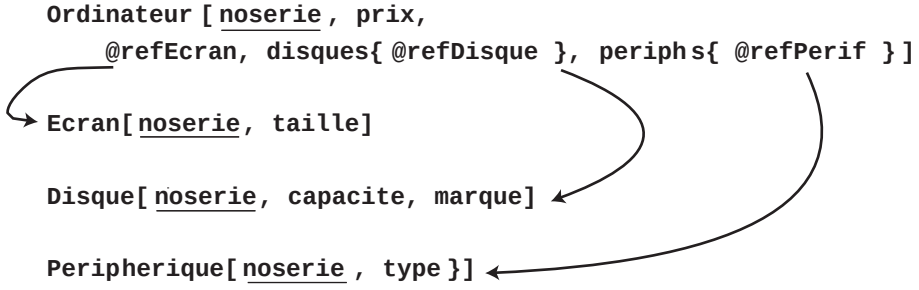
L'exemple 2-85 met en œuvre une composition avec les composants `ecran`, `disques` et `peripheriques`, parties intégrantes de la classe `Ordinateur`. Les valeurs composantes n'ont pas d'existence propre, elles doivent appartenir à un objet de la classe `Ordinateur`.

Figure 2-85 *Composition*

```
Ordinateur[noserie, prix, ecran(noserie, taille),
           disques{noserie, capacite, marque},
           peripheriques{noserie, type}]
```

Supposons qu'un objet composant puisse appartenir à différentes compositions. La figure 2-86 illustre le schéma objet, qui utilise deux collections de références. Des objets des classes `Disque`, `Ecran` et `Peripherique` peuvent avoir une existence distincte (indépendamment d'un objet de la classe `Ordinateur`). La contrainte de composition stricte est abandonnée.

Figure 2-86 *Agrégation partagée*



Il sera possible de combiner ces deux approches (composition stricte et agrégation partagée) en ne partageant que certains objets.

Exercices

Exercice 2.1 Dépendances fonctionnelles

On suppose les hypothèses suivantes :

- Les professeurs portent tous un nom différent.
- Des élèves peuvent avoir le même nom.
- Il y a un seul téléphone par bureau.
- Un bureau héberge plusieurs enseignants.
- Un enseignant n'est affecté qu'à un seul bureau.
- Un contrôle peut avoir lieu dans différentes salles en même temps.
- On ne stockera que le premier prénom.
- Un étudiant peut s'inscrire à 3 UV au maximum.
- Il y a plusieurs contrôles par UV.
- Un enseignant écrit un rapport pour chaque contrôle qu'il surveille, même s'il passe dans plusieurs salles.
- Plusieurs enseignants peuvent surveiller un contrôle.
- On désire savoir combien de temps l'enseignant est resté dans chaque salle.

Les attributs sont les suivants : (`note` d'un étudiant à un contrôle, `noteUv` : moyenne de l'UV pour l'étudiant, `moyUv` : moyenne de l'UV).

Classification des DF

Classifier chaque dépendance suivante (fonctionnelle, élémentaire). Parmi les dépendances fonctionnelles, préciser aussi celles qui sont directes.

1. `codeEtu` → `nom`
2. `nom` → `nInsee`
3. `nInsee` → `adresse`, `prenom`
4. `nInsee` → `codeEtu`
5. `codeEtu` → `nInsee`
6. `telEns` → `nomEns`
7. `nomEns` → `telEns`
8. `codeEns` → `nomEns`
9. `codeEns` → `telEns`
10. `codeEtu` → `codeUv`
11. `codeEtu`, `codeUv` → `nom`
12. `codeEtu`, `codeUv` → `moyUv`
13. `codeEtu`, `codeUv` → `titreUv`
14. `codeEtu` → `noteUv`
15. `codeEtu`, `codeUv` → `note`

16. `codeEtu`, `codeUv` → `note_uv`
17. `nCont` → `codeUv`
18. `codeUv` → `nCont`
19. `nCont` → `note`
20. `nCont`, `codeEtu` → `note`
21. `codeEns` → `rappSurveillance`
22. `codeSalle`, `nCont` → `rappSurveillance`
23. `codeEns`, `codeSalle`, `nCont` → `rappSurveillance`
24. `codeEns`, `codeSalle`, `nCont` → `tempsPasse`

Élaboration du schéma relationnel

Définir le schéma relationnel en troisième forme normale à partir de l'ensemble de DF. Utiliser l'approche par synthèse.

Exercice 2.2 Propriétés des dépendances fonctionnelles

Démontrer les implications suivantes :

- Soient les deux DF $x \rightarrow y$ et $y \rightarrow z$ alors $x \rightarrow y,z$
- Soient les deux DF $x \rightarrow y$ et $z \rightarrow w$ alors $x,z \rightarrow y,w$
- Soit la DF $x \rightarrow y$ alors $x,z \rightarrow y,z$

Exercice 2.3 Dépendances fonctionnelles

Déduire les DF de la situation suivante :

- Une bibliothèque dispose d'ouvrages (`numlivre`, `titre`, `editeur`, `anneedit`, `nbexemplaires`). Les ouvrages qui existent en plusieurs exemplaires ont le même titre mais pas le même numéro.
- Les prêts sont limités à 3 livres par étudiant (`codetu`, `nometu`, `adretu`) et le bibliothécaire souhaite mémoriser la dernière date du dernier prêt pour chaque livre emprunté (`datederpret`).
- Un titre est unique et n'est édité que par un seul éditeur.
- Si un livre est écrit par plusieurs auteurs (`codeauteur`, `nomauteur`), il faut connaître l'ordre (`ordreauteur`) d'apparition des auteurs sur la couverture du livre.
- Le bibliothécaire désire mémoriser les prêts (`datepret`) de manière à connaître les emprunts en cours.

Constuire le diagramme UML équivalent.

Comment stocker l'historique des emprunts ?

La modélisation obtenue permet-elle à un emprunteur de louer plusieurs fois, dans la même année, le même ouvrage ?

Exercice 2.4 Dépendances fonctionnelles

Déduire les DF de la situation suivante.

Soient les dépendances fonctionnelles suivantes :

- (1) $a \rightarrow b$
- (2) $b \rightarrow c$
- (3) $a \rightarrow e$
- (4) $a \rightarrow c$
- (5) $b \rightarrow f$
- (6) $b,a \rightarrow \delta$
- (7) $d \rightarrow g$
- (8) $a,b \rightarrow c$
- (9) $a,b,f \rightarrow h$
- (10) $a,i \rightarrow j$

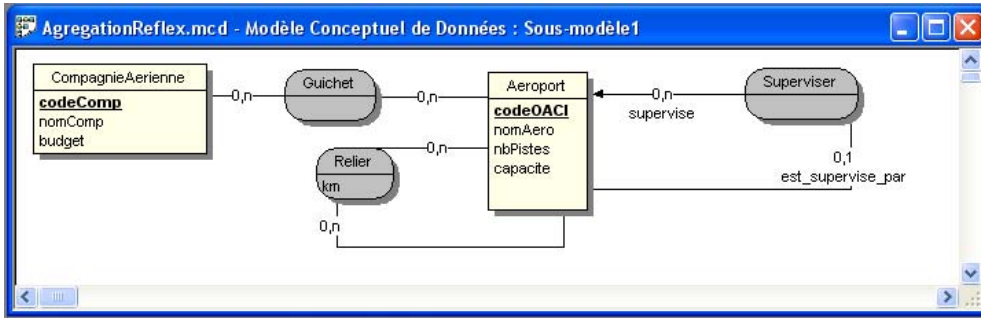
Déterminer le schéma relationnel en utilisant l'approche par synthèse.

Déterminer le MCD Merise et le diagramme UML équivalents.

Exercice 2.5 Associations réflexives

Décrire le schéma relationnel à partir de la modélisation Merise suivante.

Figure 2-87 *MCD Merise*

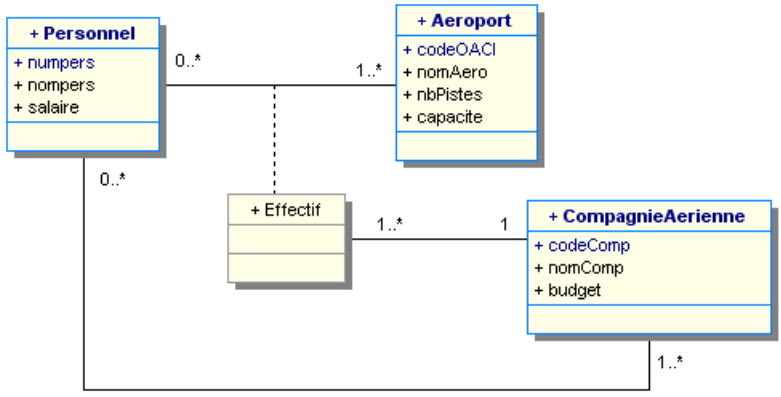


Déduire les dépendances fonctionnelles issues des trois associations.

Exercice 2.6 Classe-association

Décrire le schéma relationnel à partir de la modélisation UML suivante.

Figure 2-88 *Classe association UML*

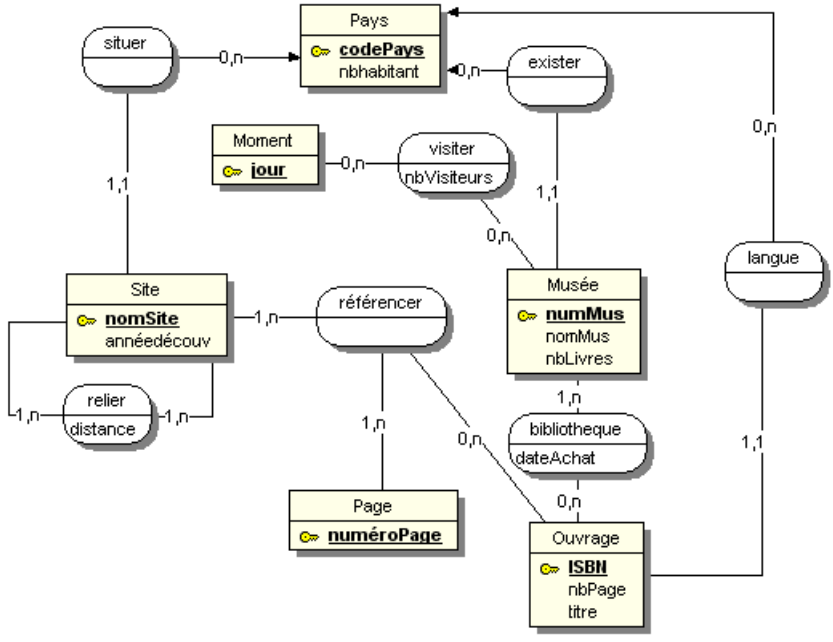


Déduire les dépendances fonctionnelles issues des deux associations.

Exercice 2.7 Association *n*-aire

Décrire le schéma relationnel à partir de la modélisation Merise suivante.

Figure 2-89 *Association* n-*aire Merise*



Cette modélisation permet-elle de répondre aux assertions suivantes :

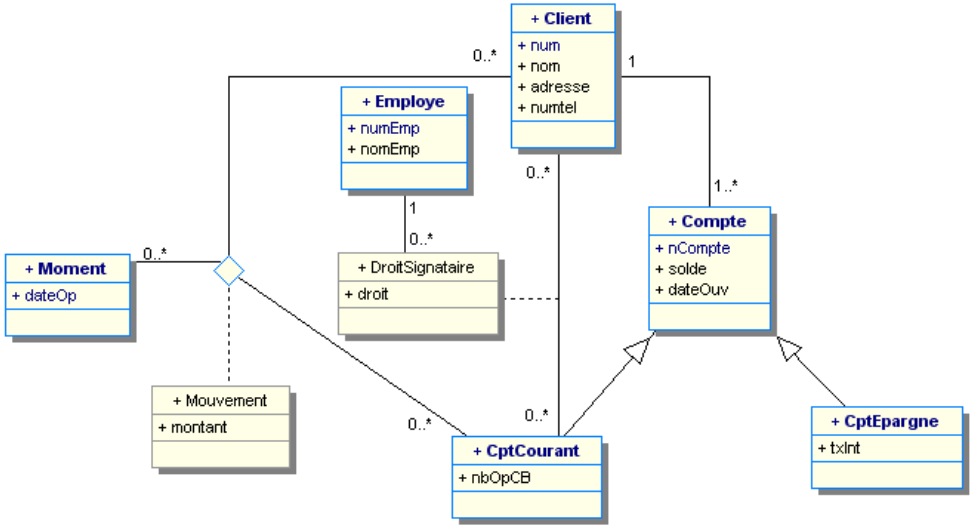
- Chaque musée est dédié à un pays.
- Le nombre de visiteurs par site dépend du jour.
- Une page d'un ouvrage ne référence qu'un site.
- Un site est référencé par plusieurs livres d'une bibliothèque.

Déduire toutes les dépendances fonctionnelles qui concernent le site.

Exercice 2.8 Héritage

Décrire un schéma relationnel à partir de la modélisation UML suivante. On suppose qu'un compte courant ne peut pas être rémunéré.

Figure 2-90 *Association* n-*aire UML*



Expliquer en quelques phrases la situation décrite.

Cette modélisation permet-elle de répondre aux assertions suivantes ?

- Un compte courant permet d'alimenter d'autres comptes courants.
- Un compte courant permet d'alimenter un compte épargne.
- Un client possède un seul compte épargne.
- Un compte courant est géré par différents clients.
- Un client peut avoir différents droits sur un compte courant.

Déduire toutes les dépendances fonctionnelles qui concernent le compte courant.

Exercice 2.9 Du conceptuel au logique

Décrire le schéma relationnel et objet à partir de certaines modélisations du chapitre 1 (exercices 1.5, 1.6 et 1.7).

1. Affrètement d'avions

Concernant le schéma objet, privilégiez l'accès aux affrètements par l'immatriculation de l'avion. Il sera intéressant de connaître rapidement les affrètements qu'un avion a effectués plutôt que la liste des affrètements de tous les avions confondus.

2. Castanet Télécoms

Concernant le schéma objet, privilégiez l'accès aux fournisseurs d'abonnement par le type de formule.

3. Voltige aérienne

Compétition journalière

Concernant le schéma objet, privilégiez l'accès aux données aux figures et aux notes d'un vol par le compétiteur.

Historique des compétitions

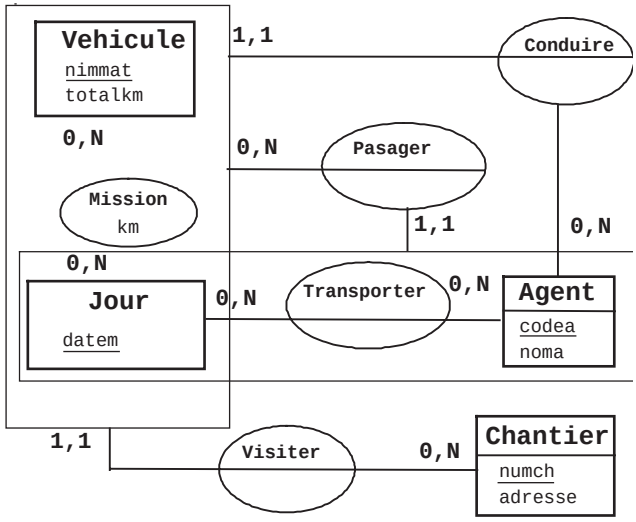
Concernant le schéma objet, privilégiez l'accès :

- aux figures et aux notes (d'un vol) par le compétiteur et le type de programme ;
- à la liste des figures (données aux juges) par le compétiteur.

Exercice 2.10 Associations d'agrégation

Dériver le schéma relationnel et un schéma objet à partir du diagramme conceptuel de la figure 2-91. Concernant le schéma objet, on privilégie l'accès aux données par la date de mission pour les personnes transportées, les chantiers et les véhicules concernés.

Figure 2-91 *Exemple d'associations d'agrégation à traduire au niveau logique*



Décrire le diagramme de classes UML équivalent.

Chapitre 3

Le niveau physique : de SQL2 à SQL3

Ce que j'ai fait, balbutia-t-il, je te le jure, aucune bête n'aurait pu le faire…
Plus tard, Guillaumet lui expliqua.
Je t'ai vu, figure-toi, mais toi tu ne pouvais me voir.
Mais comment as-tu su que c'était moi ? lui demanda Saint-Ex.
Personne d'autre n'aurait osé voler si bas.

Antoine de Saint-Exupéry, Laboureur du ciel,
C. Cate, G.P. Putnam's Sons, 1970

Le niveau physique que nous décrivons ici correspond à la définition des structures de données et à la programmation SQL nécessaires à mettre en œuvre. Nous décrivons dans ce chapitre les transformations à effectuer afin de dériver un schéma logique relationnel ou objet à l'aide du langage du SGBD.

Nous utiliserons une syntaxe SQL2 (Oracle et MySQL) pour les scripts s'appliquant aux bases de données relationnelles, et une syntaxe de type SQL3 (Oracle) pour les scripts s'appliquant aux bases de données objet-relationnelles.

Nous n'abordons pas ici des aspects relatifs à la structure interne de la base (types de fichiers, blocs et pages, chemins d'accès aux données, types d'index, chaînage, etc.). Nous n'abordons pas non plus les aspects liés à l'optimisation.

Le langage SQL

C'est IBM, avec System-R [AST 76], qui a mis en œuvre le premier le modèle relationnel à travers le langage SEQUEL (Stuctured English as QUEry Language), rebaptisé par la suite SQL (Structured Query Language). SQL provient également du langage SQUARE (Specifying Queries As Relational Expressions), développé avant System-R, qui était plus structuré que SQL, mais utilisait moins de commandes.

SQL est inclus dans des logiciels commerciaux dès la fin des années 1970 avec la première version d'Oracle et de SQL/DS d'IBM. Les derniers SGBD *open source* du marché (MySQL, PostgreSQL, MaxDB, Firebird) ont adopté ce langage âgé de plus de 30 ans.

Les normes

SQL1

Le langage SQL est normalisé depuis 1986. Cette norme s'est enrichie au fil du temps. Cette première mouture, appelée SQL86 ou SQL1 (norme décrite en une centaine de pages), est le résultat de compromis entre constructeurs, bien que l'empreinte IBM soit forte. En 1989, d'importantes mises à jour sont faites en matière d'intégrité référentielle.

SQL2

La norme SQL2 (appelée aussi SQL92 [MAR 94] est présentée en 600 pages) est finalisée en 1992. Elle définit quatre niveaux de conformité : le niveau d'entrée (*entry level)*, les niveaux intermédiaires (*transitional* et *intermediate levels)* et le niveau supérieur (*full level)*. Les langages SQL des principaux éditeurs sont tous conformes au premier niveau, et ont beaucoup de caractéristiques relevant des niveaux supérieurs.

SQL3

Les groupes de travail X3H2 de l'ANSI et WG3 de l'ISO se penchent à partir de 1993 sur les extensions à apporter à la précédente norme. Les optimistes prévoient SQL3 pour 1996 mais rien ne se passe comme prévu. C'est en 1999 que naît SQL:1999, appelé aussi SQL3 (présenté dans un volume de 1 600 pages). Ce retard est probablement dû aux nombreux protagonistes (Oracle, IBM, Microsoft, Digital, Computer Associates…) qui rechignent à remettre en cause leur mode de pensée, au risque de ne pouvoir assurer la compatibilité des bases de leurs clients.

Les évolutions de l'ancienne norme ne sont pas limitées qu'aux extensions objet. Bien d'autres mécanismes sont introduits dans SQL:1999 (géographie, temps réel, séries temporelles, multimédia, OLAP, données et routines externes).

La dernière version de SQL est référencée SQL:2003. Ses apports concernent l'auto-incrément des clés, de nouvelles fonctions d'agrégation, de ranking et de calculs statistiques, les colonnes