Christian Soutou
Avec la contribution de Frédéric Brouard

Avec 30 exercices corrigés inspirés de cas réels

Modélisation des bases de données

UML et les modèles entité-association

4e édition

EYROLLES

Modélisation des bases de données

4e édition

Concevoir une base de données à l'aide d'UML ou d'un formalisme entité-association

S'adressant aux architectes logiciels, chefs de projet, analystes, développeurs, responsables méthode et étudiants en informatique, cet ouvrage explique comment créer un diagramme conceptuel pour concevoir une base de données optimisée via le langage SQL. La démarche est indépendante de tout éditeur de logiciel et aisément transposable, quel que soit l'outil de conception choisi.

Le livre décrit d'abord la construction d'un modèle conceptuel à l'aide de règles de validation et de normalisation. Tous les mécanismes de dérivation d'un modèle conceptuel dans un schéma relationnel sont clairement commentés à l'aide d'exemples concrets. Le modèle logique peut être ensuite optimisé avant l'écriture des scripts SQL. Les règles métier sont implémentées par des contraintes SQL, déclencheurs, ou dans le code des transactions. La dernière étape consiste à définir les vues pour les accès extérieurs. Le livre se clôt par une étude comparative des principaux outils de modélisation sur le marché.

En grande partie réécrite pour prendre en compte les formalismes entité-association tels que Merise ou Barker, cette quatrième édition est commentée par Frédéric Brouard, expert SQL Server et auteur de nombreux ouvrages et articles sur le langage SQL. Émaillée d'une centaine de schémas et d'illustrations, elle est complétée par 30 exercices inspirés de cas réels.

À qui s'adresse ce livre ?

- Aux étudiants en IUT, master et écoles d'ingénieur, ainsi qu'à leurs professeurs
- Aux professionnels souhaitant s'initier à la modélisation de bases de données
- À tous les concepteurs de bases de données

Maître de conférences rattaché au département Réseaux et Télécoms de l'IUT de Blagnac, **Christian Soutou** intervient en licence et master professionnels. Il est aussi consultant indépendant chez Orsys et auteur de nombreux ouvrages aux éditions Eyrolles.

MVP SQL Server depuis plus de dix ans, **Frédéric Brouard** dirige la société SQL Spot qui fournit des services d'assistance, de conseil, d'audit et de formation sur SQL Server et l'architecture de données. Enseignant dans différentes écoles d'ingénieur, il a écrit plusieurs livres consacrés à SQL et rédigé de nombreux articles, notamment sur le site developpez.com.

Au sommaire

www.editions-eyrolles.com

Christian Soutou
Avec la contribution de Frédéric Brouard

Modélisation des bases de données

4e édition

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

DU MÊME AUTEUR

C. Soutou. – **Programmer avec MySQL (5e édition).**
N°67379, 2017, 522 pages.

C. Soutou. – **SQL pour Oracle (7e édition).**
N°14156, 2015, 672 pages.

C. Soutou, F. Brouard, N. Souquet et D. Barbarin. – **SQL Server 2014.**
N°13592, 2015, 890 pages.

AUTRES OUVRAGES

F.-X. Bois et A. Lies Benhenni. – **Bases de données orientées graphes avec Neo4j.**
N°13804, 2016, 182 pages.

R. Bruchez. – **Les bases de données NoSQL et le Big Data (2e édition).**
N°14155, 2015, 332 pages.

P. Roques. – **Mémento UML 2.5 (3e édition).**
N°14356, 2015, 14 pages.

P. Roques. – **UML 2 par la pratique (7e édition).**
N°12565, 2009, 396 pages.

P. Roques. – **UML 2 par la pratique (6e édition).**
N°13344, 2011, 370 pages (format semi-poche).

P. Roques et F. Vallée. – **UML 2 en action (4e édition).**
N°12104, 2007, 396 pages.

H. Balzert. – **UML 2.**
N°11753, 2006, 88 pages.

F. Vallée. – **UML pour les décideurs**.
N°11621, 2005, 282 pages.

H. Bersini. – **La programmation orientée objet (7e édition)**. *Cours et exercices en UML 2, Python, PHP, C# , C++ et Java (y compris Android).*
N°67399, 2017, 696 pages.

-

 ISBN : 978-2-212-67487-3

Table des matières

Avant-propos

Le but de cet ouvrage est de démystifier le processus de conception d'une base de données relationnelle normalisée. Sur la base d'exemples concrets et variés, nous vous expliquerons comment construire un modèle conceptuel avec un formalisme de type entité-association (Barker ou Merise), ou à l'aide d'un diagramme de classes UML. La démarche et les mécanismes décrits dans ce livre sont indépendants de tout éditeur de base de données et aisément transposables quel que soit l'outil de conception que vous adopterez.

Les deux premières éditions de cet ouvrage n'étaient consacrées qu'au formalisme d'UML. La troisième édition a élargi le champ du conceptuel en développant les formalismes de type entité-association qui sont toujours très répandus au travers des outils de modélisation comme PowerAMC, Data Modeler (Oracle), MySQL Workbench, TOAD Data Modeler, etc. Depuis la troisième édition, je donne la parole à Frédéric Brouard (alias SQLPro) qui ne se prive pas de commentaires. Cette quatrième édition est mise à jour et enrichie de nouveaux exercices.

Les pictogrammes

Ce pictogramme introduit une définition, un concept ou une remarque importante.

Ce pictogramme indique une astuce ou un conseil personnel.

Ce pictogramme introduit une remarque, un avis divergent, un complément ou un coup de gueule de Frédéric Brouard.

Avant de rentrer dans le vif du sujet, rappelons brièvement pourquoi les bases de données relationnelles occupent toujours une part prépondérante du marché alors que le big data est en pleine croissance.

Évolution des modèles de données

On peut recenser sept familles de modèles de données qui correspondent aux différents systèmes de stockage existants jusqu'à présent.

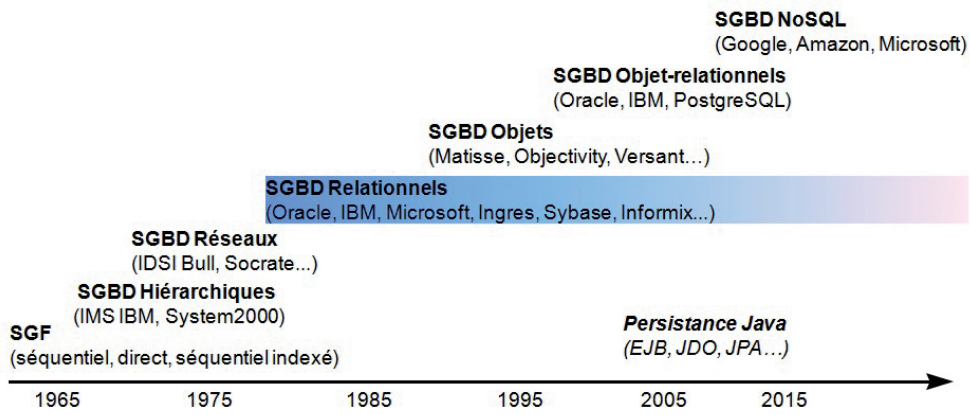

***Figure 1**. Historique des systèmes de stockage de données*

1. Les fichiers COBOL *(Common Business Oriented Language)* furent les plus utilisés par l'informatique de gestion jusque dans les années 1980. Au début des années 2000, le Gartner Group estimait que cette technologie était utilisée encore par plus de la moitié des applications. Bien que la structure d'un fichier de données s'apparente à celle d'une table (suite de champs de types numériques ou alphanumériques), l'inconvénient majeur de COBOL est la forte dépendance qui existe entre la structure des données stockées et les traitements. En effet, chaque programme contient la structure des fichiers utilisés, ce qui impose une maintenance lourde suite à la modification de la structure d'un fichier.
2. Le modèle hiérarchique impose de déterminer une arborescence de données où l'accès à un enregistrement de niveau inférieur n'est pas possible sans passer par le niveau supérieur. Promus par IBM et toujours utilisés dans le domaine bancaire, les SGBD hiérarchiques souffrent toutefois de nombreux inconvénients. Les plus récurrents sont toujours la forte dépendance entre les données stockées et les méthodes d'accès ; les chaînages internes impliquent une programmation complexe, des requêtes inadaptées et des mises à jour hasardeuses. Les modèles XML et JSON qui sont employés par le monde NoSQL (voir plus loin) rappellent certains inconvénients du modèle hiérarchique.

3. C. Bachman a proposé un modèle brisant les hiérarchies avec le modèle CODASYL pour les bases de données réseau. Bien que résolvant quelques limitations du modèle hiérarchique et annonçant des performances en lecture honorables, le modèle réseau fut bien trop complexe à mettre en œuvre avec de trop nombreux pointeurs et il n'existe plus de tels SGBD. Les modèles en graphe qui sont utilisés par le monde NoSQL (voir plus loin) rappellent certains inconvénients du modèle réseau.
4. E. Codd publie ses écrits [COD 70] posant les bases du modèle relationnel. Toutes les limitations précédentes sont résolues et les apports principaux concernent l'indépendance données/traitements, la non-redondance des données, l'intégrité et la capacité d'extraction par requêtes. Le modèle relationnel est à l'origine du succès que connaissent aujourd'hui les grands éditeurs de SGBD, à savoir Oracle, IBM, Microsoft et Sybase.
5. Les bases de données ayant adopté le modèle objet permettaient de stocker des données structurées et d'y adjoindre des méthodes (au sens de sous-programmes). Ce paradigme, qui a connu le succès qu'on connaît (Java, C#, etc.), n'a pas obtenu les faveurs du marché frileux à l'idée de transformer ses tables dans des structures de données plus riches mais moins évolutives.
6. Les éditeurs de bases relationnelles ont proposé des extensions objet suite à l'adoption généralisée du concept objet dans les langages pour proposer des modèles de données objet-relationnels. Encore une fois, ces extensions, toujours présentes dans les moteurs, ne sont pas utilisées pour les mêmes raisons que les bases objet.
7. Les modèles de données NoSQL sont nés avec l'arrivée du big data pour gérer les montées en charge. Les grands acteurs, tels Google, Amazon, LinkedIn et Facebook, furent amenés à créer leurs propres systèmes (BigTable, Dynamo, Cassandra). Des implémentations d'architectures open source comme Hadoop et des SGBD comme HBase, Redis, Riak, MongoDB, CouchDB… ont permis de démocratiser ce nouveau domaine de l'informatique répartie. Dans ces modèles, c'est à l'application cliente de comprendre et de traiter la structure de données. On distingue plusieurs modèles de données : clé-valeur, orienté colonnes (table dénormalisée), document (structure de données hiérarchiques comme XML ou JSON) et graphe (structure de données en réseau). Plus le modèle est complexe, moins le système est apte à évoluer rapidement du fait de montées en charge.

Tables vs documents

Afin de mieux appréhender les différences fondamentales entre le modèle de données relationnel et un modèle de données plus structuré comme XML et JSON, considérons un exemple concret : les passagers d'un vol. Il faudra un document par vol qui permettra de lister les passagers transportés. Dans le cas des tables, il en faudra probablement plusieurs pour s'assurer de la cohérence des données (notamment que le vol en question existe et qu'il est aussi prévu ce jour).

Tables

VOLS

VOL_ID	AEO_DEP	H_DEP	MIN_DEP	AEO_ARR	H_ARR	MIN_ARR
AF4143	TLS	19	10	ORY	20	30

VOLS_JOUR

VOL_ID	JOUR_VOL
AF4143	28/05/2017

PASSAGERS

VOL_ID	JOUR_VOL	PNC_ID	NOM_PAX	PRENOM_PAX	SIEGE_ID	PRIX_PAX
AF4143	28/05/2017	1690352877	Diffis	Gerard	01A	158
AF4143	28/05/2017	1100371007	Blanchet	Guy	01B	190

Figure 2. *Tables du modèle de données relationnel*

Document

```
<?xml version="1.0" encoding="UTF-8"?>
<vol_jour vol_id="AF4143" jour_vol="2017-05-28">
    <aero_dep  h_dep="19.10">TLS</aero_dep>
    <aero_arr  h_arr="20.30">ORY</aero_arr>
    <passagers>
        <pax pax_id="1690352877">
            <nom>Diffis</nom>
            <prenom>Gerard</prenom>
            <siege>01A</siege>
            <prix>158</prix>
        </pax>
        <pax pax_id="1100371007">
            <nom>Blanchet</nom>
            <prenom>Guy</prenom>
            <siege>01B</siege>
            <prix>190</prix>
        </pax>
    </passagers>
</vol jour>
```

Figure 3. *Document XML*

Les tables du modèle relationnel

La force de ce modèle de données réside dans le fait qu'il repose sur des principes simples et permet de modéliser des données complexes. Les liens entre lignes sont réalisés non pas à l'aide de pointeurs physiques, mais à l'aide des valeurs des clés (étrangères vers les primaires, par exemple le numéro des vols journaliers doit se trouver dans la table de référence des vols). Pour cette raison, le modèle relationnel est dit « modèle à valeurs ». Le langage adapté à ce modèle de données est SQL, qui est normalisé depuis 1986.

On trouve une application du modèle relationnel dans différents domaines.

- OLTP *(OnLine Transaction Processing)* où les mises à jour des données sont fréquentes, les accès concurrents et les transactions nécessaires. C'est ce domaine sur lequel nous nous focaliserons.
- OLAP *(Online Analytical Processing)* où les données sont multidimensionnelles (cubes), les analyses complexes et l'informatique décisionnelle. Les schémas de données (modèle en étoile et en flocon) se rapprochent du modèle relationnel en y incluant volontairement des redondances entre des tables de dimensions et des tables de faits (mesures).
- Systèmes d'informations géographiques (SIG) où la majorité des données sont exprimées en 2D ou 3D avec des fonctions bien spécifiques (calculs de distances, d'aires, etc.) et suivant des variations temporelles.

Si le modèle est normalisé, l'accès aux données sera optimal de même que l'indexation (ici la liste des passagers d'un vol et la liste des vols d'un passager concerneront la même table). La théorie des ensembles (algèbre relationnelle) donne un cadre formel à l'interrogation des

tables. Dans le cas de jointures (requêtes multitables), le choix du chemin est assuré par l'optimiseur du SGBD. Les index accéléreront les requêtes comme un index vous sert à aller à une page d'un livre en particulier lorsque vous recherchez un terme en particulier.

Par ailleurs, la redondance des informations ne concerne que les colonnes clés, ce qui rend toute mise à jour plus robuste et efficace. L'intégrité référentielle par les clés étrangères ajoute de la cohérence (exemple : un passager ne peut être ajouté à un vol non prévu et un vol prévu ne peut exister que s'il se trouve dans la table de référence des vols).

Vous allez découvrir tout au long de cet ouvrage comment concevoir de telles tables

Le modèle document

La force de ce modèle de données réside dans le fait qu'un grand nombre d'informations sont regroupées et correctement structurées (enfin, il faut l'espérer) dans un même enregistrement : un fichier (le plus souvent XML ou JSON). Aucun lien entre fichiers n'est possible, ce qui implique que la cohérence des données n'est jamais assurée. Rien n'empêchera qu'un vol inexistant au catalogue soit associé à des passagers. Le langage adapté à ce modèle de données est bien souvent XPath (pour XML) qui est aussi normalisé.

On trouve une application du modèle document dans le domaine du big data avec les bases NoSQL (Mongo DB, Couch DB, Riak…).

L'accès aux données sera optimal seulement si la structure du document est en adéquation avec la requête (ici la liste des passagers d'un vol). En revanche, la liste des vols d'un passager impliquera le parcours de tous les fichiers avec pour chacun d'entre eux une recherche à l'intérieur... On retombe dans les inconvénients du modèle hiérarchique. Pas de jointures possibles mais des mécanismes d'index peuvent être mis en œuvre.

Par ailleurs, la redondance d'informations, en général omniprésente dans des documents XML, rend hasardeuse la plupart des mises à jour.

- Comment ajouter un client s'il n'est pas passager (on imagine qu'une table client sera mise en place avec une base relationnelle), à moins d'ajouter un vol fictif.
- La suppression peut se révéler dangereuse : un vol disparaît, alors de fait ses passagers aussi.
- La modification est souvent problématique : les incohérences proviennent des redondances (par exemple les heures de départ et d'arrivée de chaque vol) qu'il faudra vérifier et répéter pour chaque document.

Enfin, pour conclure cette comparaison, sachez qu'une seule requête SQL servira à générer un document XML aussi complexe soit-il à partir de jointures et de données de plusieurs tables. En revanche, il ne sera pas possible d'automatiser à grande échelle la transformation de documents XML vers des tables à créer.

Quel formalisme utiliser ?

Depuis plus de 30 ans, la conception des bases de données relationnelles s'appuie sur différents formalismes graphiques (Bachman, Chen, IDEF1X, Object-role, etc.). Chacun d'entre eux a son look particulier mais, pour tous, il s'agit de relier graphiquement des boîtes (appelées entités ou classes) contenant des champs (appelés attributs ou propriétés) par des liens (appelés associations). Tous ces formalismes graphiques peuvent être de type entité-association (incluant celui que Merise avait adopté en son temps). Un cas particulier peut être trouvé avec le diagramme des classes d'UML qui ne parle pas d'entité mais de classe.

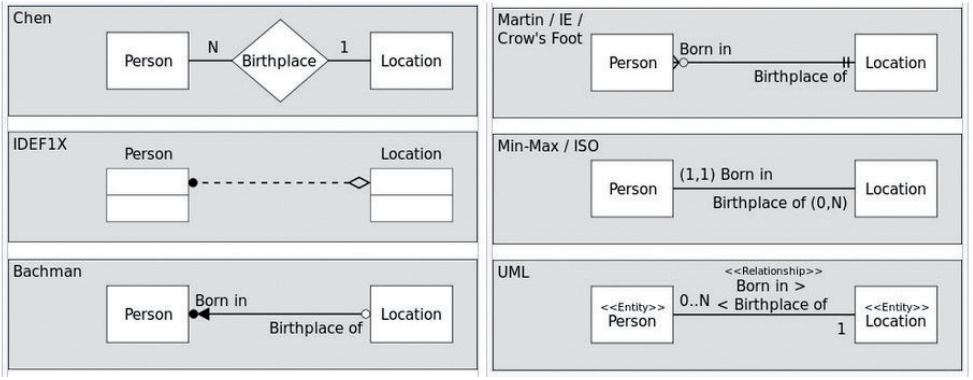

Figure 4. *Quelques formalismes extraits de en.wikipedia.org/wiki/Entity-relationship_model*

Le formalisme de Barker

Les outils du marché (voir chapitre 5) n'ont souvent emprunté qu'une partie d'un formalisme en particulier pour ajouter un petit quelque chose qui fait souvent différer un même schéma entre deux outils distincts. Par exemple, le modèle de Barker qu'Oracle utilise avec l'outil Data Modeler, s'inspire de la notation Crow's foot. Je trouve ce formalisme compact, précis et sobre, c'est pour cela que je l'ai adopté dans cet ouvrage.

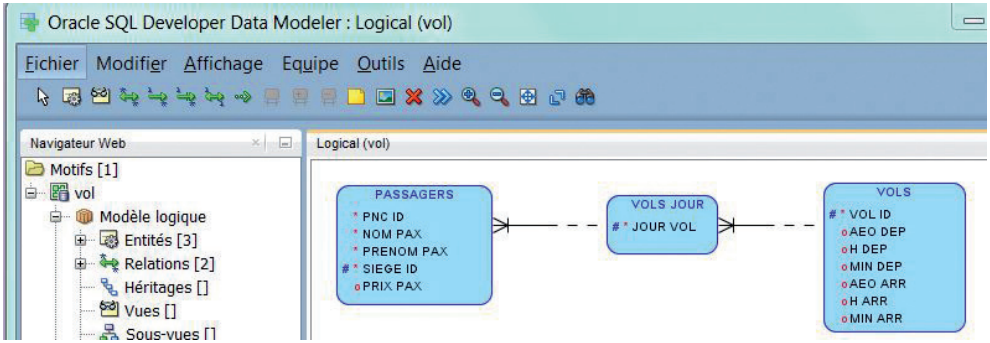

Figure 5. *Formalisme de Barker*

Le diagramme de classes d'UML

La notation UML s'est imposée depuis quelques années pour la modélisation et le développement d'applications écrites dans un langage objet (C++ et Java principalement). Ce succès s'explique par l'adoption unanime des concepts objet, qui ont des avantages indéniables (réutilisabilité de composants logiciels, facilité de maintenance, prototypage et extension des applications, etc.). Parmi les différents diagrammes d'UML, seul celui de classes est pertinent (concernant la modélisation d'une base de données). En effet, les concepts relatifs à la modélisation de données (entités, associations, attributs et identifiants) sont intégrés au diagramme de classes. De plus, d'autres mécanismes (classes-associations, agrégats et contraintes) permettront d'enrichir un schéma conceptuel. Je trouve ce formalisme bien adapté, c'est pour cela que je l'ai aussi adopté dans cet ouvrage.

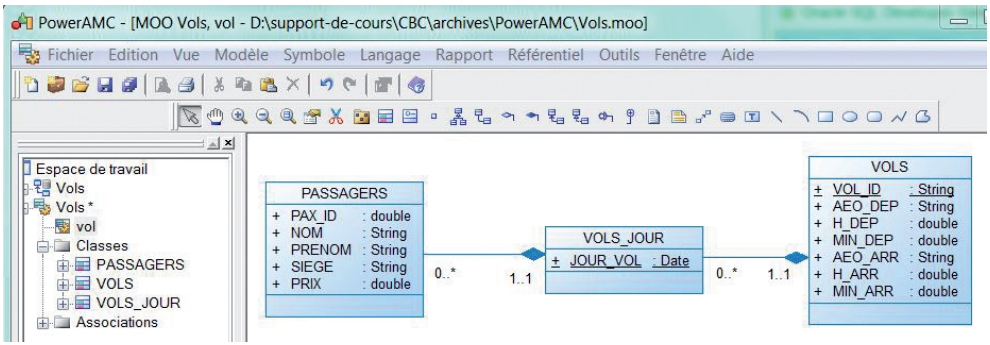

Figure 6. *Formalisme d'UML*

La démarche à adopter

Cet ouvrage s'organise en une introduction de Frédéric Brouard suivie de cinq chapitres qui décrivent les étapes de conception puis d'implémentation illustrées par la figure suivante. Pour toute conception, le départ est le système à modéliser via un schéma conceptuel (souvent nommé *logical* par les outils). Ce dernier est ensuite dérivé dans un modèle de données relationnel (*relational* ou *physical*, selon les outils) qui sera ensuite optimisé avant l'écriture ou la génération de scripts SQL implémentant les tables, clés et index. Il s'agira ensuite d'implémenter des règles métier en ajoutant des contraintes SQL ou en programmant des déclencheurs. Ce troisième niveau est souvent appelé « physique ». La dernière étape consistera à définir des vues *(views)*, véritables interfaces de la base aux utilisateurs. Ce dernier niveau est souvent appelé « externe ».

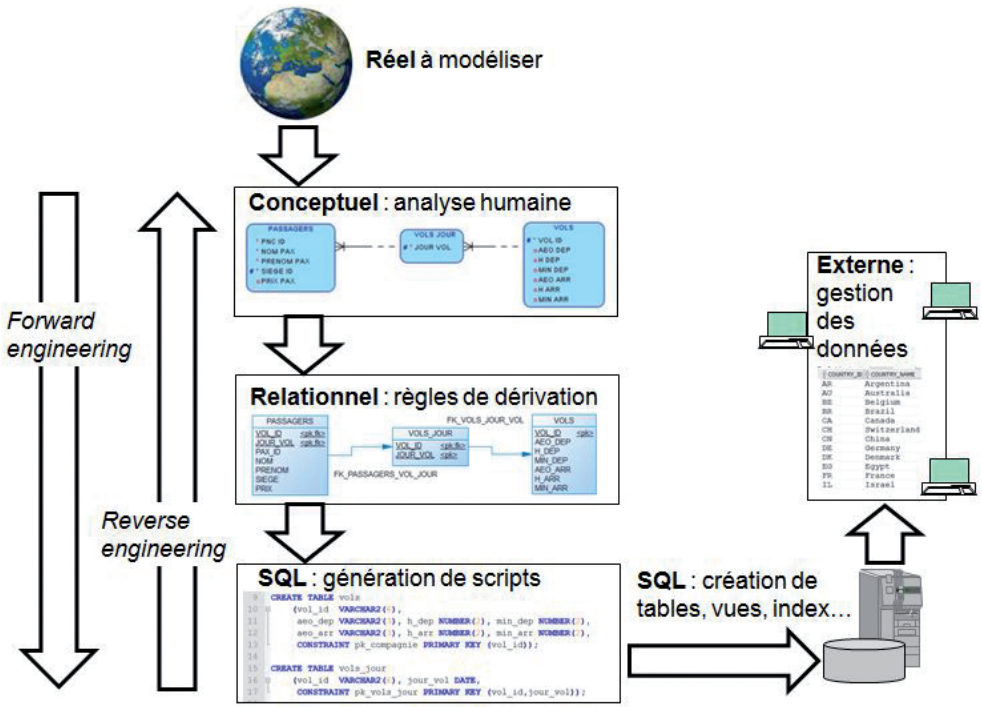

Figure 7. *Niveaux de conception et d'implémentation*

Le processus de conception depuis l'analyse du système à modéliser est nommé *forward engineering* (aussi *top-down*). Quand il s'agit de retravailler à partir de données existantes pour faire évoluer la base de données, on parle de *reverse engineering* (ou *bottom-up*). Alors que l'automatisation est quasiment assurée par les outils du marché entre le niveau conceptuel et SQL, il n'en

va pas de même de l'élaboration du schéma conceptuel qui va conditionner la suite. Ici, seule une action humaine permettra de traduire des faits et événements en un schéma graphique.

Niveau conceptuel

Le chapitre 1 décrit la première étape du processus de conception d'une base de données, à savoir la construction d'un schéma conceptuel. De nombreux exemples et règles de bonne conduite vous guideront. Ne vous inquitez pas d'obtenir un modèle différent de celui de votre collègue qui aura travaillé sur le même sujet. Comme le dit Wikipédia, bien que des techniques de modélisation existent et guident chaque concepteur, il n'est pas rare de constater différents résultats :

> Several techniques have been developed for the design of data models. While these methodologies guide data modelers in their work, two different people using the same methodology will often come up with very different results.

Figure 8. *Extrait de https://en.wikipedia.org/wiki/Data_modeling*

À quoi sont dues ces différences ? À l'interprétation humaine de tout système : le point de vue. Concernant notre exemple, un individu verra un vol comme un ensemble de sièges à une date donnée, chaque siège étant associé à un client. Un autre individu considérera qu'un vol concerne plusieurs clients pouvant acheter plusieurs places (pensons au violoncelliste qui n'acceptera pas de mettre en soute son instrument plusieurs fois centenaire).

Bien que plusieurs schémas conceptuels puissent convenir à une même modélisation, chaque schéma offre ses avantages et ses inconvénients. Les plus simples vont bien souvent minimiser le nombre de tables, mais des contraintes devront être implémentées ultérieurement. Les schémas plus complexes vont faciliter l'implémentation de règles métier mais seront plus surchargés et mettront initialement plus de tables en œuvre.

Niveau relationnel

Le chapitre 2 décrit les concepts puis présente les règles qui permettent d'établir un schéma relationnel avec ses clés à partir d'un schéma conceptuel. Les mécanismes de normalisation qui optimisent le schéma et les calculs de volumétrie sont également décrits.

Niveau SQL

Le chapitre 3 décrit la syntaxe d'écriture de scripts SQL associée à un schéma relationnel (définition des tables, clés primaires et étrangères et index) ainsi que l'implémentation d'éventuelles règles métier par des contraintes ou déclencheurs.

Niveau externe

Le chapitre 4 est consacré à la définition des différents types de vues (relationnelles, matérialisées et structurées) qui agiront comme des fenêtres sur les tables de la base de données.

Les outils du marché

Le chapitre 5 confronte l'offre des principaux outils UML du marché (MagicDraw, MEGA Designer, Modelio, Objecteering, PowerAMC, Rational Rose, Visual Paradigm et Win'Design). Chaque outil est évalué sur différents critères (saisie d'un schéma conceptuel, génération d'un modèle relationnel puis d'un script SQL). Le reverse engineering est également évalué.

Annexes

Les annexes contiennent les corrigés détaillés des exercices, une webographie et une bibliographie. L'index propose les termes utilisés dans la définition des concepts et de certaines instructions SQL.

À qui s'adresse cet ouvrage ?

Cet ouvrage s'adresse à toutes les personnes qui s'intéressent à la modélisation et à la conception des bases de données.

- Les architectes, chefs de projet, analystes, développeurs et responsables méthode habitués au modèle entité-association y trouveront les moyens de raisonner avec le diagramme de classes UML.
- Les novices découvriront une méthode de conception, des règles de normalisation et de nombreux exercices mettant en jeu tous les niveaux du processus d'une base de données.

Contact avec l'auteur

Si vous avez des remarques à formuler sur le contenu de cet ouvrage, n'hésitez pas à me faire part de vos remarques (christian.soutou@gmail.com). Vous trouverez d'éventuels errata sur le site d'accompagnement de cet ouvrage accessible via www.editions-eyrolles.com.

Avant d'aborder la théorie

Beaucoup de développeurs sont persuadés des méfaits du respect des formes normales. N'est-il pas préférable de placer tous ses « champs » dans une même table pour acquérir de bonnes performances ? Certains internautes l'ont même écrit, mesures de performances à l'appui !

Or il n'en est rien, et le principal problème reste l'incompréhension du modèle relationnel. Bon nombre de développeurs ont succombé au NoSQL, croyant résoudre leurs problèmes alors qu'ils en créaient de pires sans le savoir.

Le modèle relationnel reste encore à ce jour le moyen le plus efficace à tout point de vue, lecture comme écriture, pour la manipulation des données de l'informatique de gestion, les transactions, et les recherches complexes. Encore faut-il en comprendre l'esprit.

C'est donc avec pragmatisme que je vais tenter de l'expliquer sans jamais passer par les « formes normales » ou les « dépendances fonctionnelles ». Seuls des éléments de base de la science mathématique seront utilisés dans cette démonstration.

Les origines

En fait, l'art de la modélisation repose sur quelques règles fondamentales facilement compréhensibles et sur la notion de relation, rarement bien comprise.

Lorsque Frank Edgar Codd invente la théorie de l'algèbre relationnelle dans les années 1970, il prétend résoudre toutes les problématiques des systèmes précédents (bases de données hiérarchiques ou en « réseau », entre autres…) aussi bien sur le plan pratique que sur le plan logique. Force est de constater que sa théorie a fort bien réussi et domine toujours la marché des bases de données de gestion. Il en va tout autrement du traitement des données documentaires dont le big data s'empare actuellement, dans l'indécision d'une technologie unique dont les tenants sont regroupés au sein de ce que l'on appelle désormais le NoSQL…

À partir de 1980, les premiers systèmes voient le jour (IBM System R et Oracle de Relationnal Software). Les SGBD relationnels sont donc un succès depuis plus de 35 ans… Et régulièrement, on nous annonce leur fin sans que cela n'arrive jamais. Cependant, la connaissance

du métier de la modélisation se meurt. En effet, depuis que l'objet a vu le jour, on tente de contourner le relationnel sans se rendre compte qu'il faudrait agir avec lui et non contre lui !

Il convient donc de reprendre les choses à la source pour bien les faire comprendre… C'est beaucoup plus simple qu'il n'y paraît, mais de nombreux enseignants préfèrent l'approche théorico-mathématique hyper sophistiquée à l'approche pragmatique, dégoûtant ainsi les étudiants à l'avance…

Cette introduction n'a donc pour but que de vous familiariser avec les concepts afin qu'ils vous apparaissent lumineux.

Au début était la relation

Lorsqu'on interroge les développeurs – qui pour la plupart ont déjà suivi un cours de modélisation – sur ce qu'est une relation, la réponse qui prédomine, et sur laquelle convergent les avis, est la notion de « lien ». Or il n'en est rien.

Une relation a été définie par Codd comme un objet mathématique porteur de données.

Si nous consultons un dictionnaire comme le Larousse, la relation est définie comme suit : « Action de rapporter en détail ce dont on a été le témoin ou dont on a eu connaissance ; récit qu'on en fait. »

En fait, la relation « relate » les faits… Et une base de données c'est de l'information, de la sémantique !

Pour Codd, la définition de la relation est celle-ci :

- un objet mathématique porteur d'informations ;
- contenant un ou plusieurs attribut(s) valué(s) ;
- possédant une clef ;
- doté d'un nom unique au sein de la base de données.

Il précise en outre que les attributs doivent :

- avoir un nom unique au sein de la relation ;
- contenir une information atomique ;
- posséder une valeur prise dans un domaine.

Codd part de la théorie des ensembles et ses relations doivent être considérées comme telles, chacun des éléments de l'ensemble ayant une valeur pour chaque attribut.

Exemple 1 – Une relation

La relation `Employé` constituée des attributs `matricule`, `nom`, `prénom`, `date de naissance` ayant pour clef le seul attribut `matricule`.

On note habituellement comme ceci :

```
Employé : matricule, nom, prénom, date de naissance
```

Le ou les attribut(s) clef(s) étant soulignés.

Complétons par la notion de tuple ou n-uplet. Il s'agit d'un ensemble de valeurs pour chacun des attributs de la relation, décrivant un élément particulier de l'ensemble.

Exemple 2 – Un tuple d'une relation

```
XD1247, DUPONT, Frédéric, 21/04/1960
```

L'élément Frédéric DUPONT, né le 21 avril 1960, dont le matricule est XD1247, est un élément de la relation `Employé` présentée dans l'exemple 1.

Remarque : dans les ensembles, au sens mathématique du terme, il n'y pas d'ordre naturel des choses.

Voici par exemple le contenu d'une relation, représentée par un diagramme de Venn plus communément appelé « patate ».

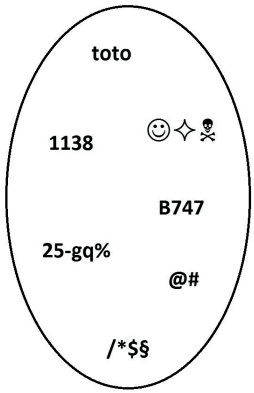

Figure I-1. *Diagramme de Venn*

Remarquez-vous un quelconque ordre des éléments ? Par exemple, une question sans réponse que l'on peut souvent lire dans les forums est la suivante : « Comment récupérer la dernière ligne que j'ai mis dans une table ? » Cette question n'a évidemment aucun sens, d'une part en raison de la concurrence d'accès, deux utilisateurs pouvant insérer une ligne au même moment et d'autre part, il n'y a pas de rangement particulier des lignes dans la table.

Puis vint l'algèbre relationnelle

À partir de la relation, Codd définit des opérateurs relationnels, c'est-à-dire des traitements mathématiques qui transforment les relations en d'autres relations.

Oui, je sais, le terme « algèbre » vous donne des boutons. Pourtant, c'est assez simple. On peut définir une algèbre comme un ensemble d'opérateurs opérant sur des objets mathématiques clairement définis.

Si je vous parle de l'algèbre en nombres entiers, cela vous parle déjà plus... Vous savez que 2 + 2 = 4, soit un nombre entier, tout comme 2 × 3 = 6, encore un entier ! Mais que dire de 7/2 ? Le tout est de savoir si vous voulez rester dans l'univers des entiers ou pas. Si oui, cette dernière opération peut avoir trois résultats différents :

- aucun ;
- 3 ;
- 3 reste 1.

À vous de vous mettre d'accord sur la règle à adopter.

Ce faisant, en restant dans l'univers des entiers, vous opérez ce que l'on appelle une « fermeture », c'est-à-dire que vous vous enfermez dans l'univers des entiers. Ceci possède un intérêt majeur... Celui de pouvoir étendre et combiner les opérations à l'infini !

Eh bien, l'algèbre relationnelle de Monsieur Codd est une fermeture. Chacune des opérations relationnelles donne à nouveau une relation en sortie qui possède une clef et des attributs atomiques tous valués.

Les opérations relationnelles définies par Codd sont les suivantes.

- La **restriction** qui ne conserve que certains tuples de la relation ayant des caractéristiques spécifiques décrites par le biais d'un prédicat (par exemple, un nom commençant par la lettre « D »).
- La **projection** qui ne retient dans la relation résultante que certains attributs et pas d'autres.
- L'**union** qui consiste à « concaténer » des relations aux caractéristiques similaires (par exemple, des clients et des prospects).
- L'**intersection** qui renvoie les éléments communs aux relations.
- La **différence** qui renvoie les éléments de l'une des relations qui n'existent pas dans l'autre.
- Le **produit cartésien** (vulgairement appelé multiplication) qui associe à tout élément d'une relation, chacun des éléments de l'autre (par exemple, une relation contenant les couleurs du jeu de cartes et une autre les figures, permettant ainsi de recomposer un jeu complet).
- La **division** qui est l'opération inverse du produit cartésien.
- Et enfin la **jointure** qui permet d'associer une relation à une autre par le biais d'un prédicat (par exemple le poids d'une lettre et la tarification d'affranchissement).

Vous noterez que certaines opérations n'utilisent qu'une seule relation (restriction, projection). On les appelle opérations unaires ou monadiques. Tandis que les autres portent sur deux relations et sont appelées opérations binaires ou dyadiques.

Et si vous vous demandez ce qu'est un prédicat, sachez que c'est une expression qui peut prendre les valeurs « vrai » ou « faux ». Quelques exemples : « les poissons n'ont pas d'os », « cette phrase compte six mots », ou « ce que vous lisez est irréel ».

Les notions de base

Détaillons maintenant tour à tour quelques-unes des notions précédemment évoquées.

Les noms

Il n'y a aucune ambiguïté sur les noms des relations comme sur les noms des attributs, dans le sens où deux relations au sein de la même base ne peuvent avoir le même nom tout comme deux attributs au sein de la même relation.

Exemple 3 – Noms des éléments

Dans l'exemple 1, le nom de la relation est « *R*employé » et les noms des attributs « nom », « prénom », « date de naissance » et « matricule ».

La notion de valeur

Pour Codd, tout attribut doit être valué. Cela sous-entend qu'un attribut est obligatoirement renseigné. Vous avez sans doute entendu parler du NULL. Cela n'existe pas dans la théorie de Codd. Tous les attributs concourant à une même relation, ou plus exactement à un même tuple (ou n-uplet) doivent posséder une valeur et pas n'importe laquelle… Une valeur vraie ! Pas des mensonges ou de valeurs qui ne veulent rien dire comme une chaîne vide, une date à zéro…

Exemple 4

Les valeurs associées au matricule XD1247 de la relation `Remploye` sont :

```
XD1247, Frédéric, DUPONT, 21/04/1960
```

Comme vous le constatez, chacun des attributs (matricule, nom, prénom et date de naissance) est valué.

La notion de clef

La clef est un moyen d'identifier un élément et un seul au sein de l'ensemble. C'est une information composée de(s) valeur(s) d'un ou plusieurs attributs qui nous permet avec certitude de retrouver un et un seul élément de l'ensemble.

Exemple 5 – Une clef

La clef de la relation `Remploye` présentée à l'exemple 1 est composée d'un seul attribut de nom `matricule`. Pour un certain Frédéric DUPONT né le 21 avril 1960, la valeur de cette clef est `XD1247`.

Si rien n'est précisé au sujet de la clef, alors, par défaut, ce sont tous les attributs de la relation qui composent cette clef.