



atsuhikoMochizuki

**OGROD**

**Olympic Games Results Open Database**



Conception et mise en œuvre d'une application d'insertion en base de données de résultats olympiques à l'aide des outils UML



**Projet individuel**

Sous la direction de Jacques BACH

05/07/23

[https://github.com/atsuhikoMochizuki/OGROD\\_light.git](https://github.com/atsuhikoMochizuki/OGROD_light.git)



# Cahier des charges

A l'occasion des futurs JO de Paris 2024, votre société a été sélectionnée pour réaliser une application permettant de naviguer aisément dans les résultats de toutes les épreuves sportives depuis la création des Jeux Olympiques modernes.

A ce stade vous ne disposez pas encore du cahier des charges avec l'expression de besoin du client mais vous pouvez d'ores-et-déjà vous occuper d'intégrer les données des résultats JO. Vous disposez à cet effet d'un fichier CSV contenant l'ensemble de ces résultats (271 000 lignes).

Vous allez devoir concevoir cette application en commençant par :

- Identifier les entités métier
- Réaliser le diagramme de classes,
- Réaliser le modèle physique de données
- Réaliser un diagramme de séquences pour un traitement décrit un peu plus loin.
- Dans ce TP il n'y a pas de développement à réaliser. Cependant cela n'est pas interdit, vous gérez votre temps comme vous le souhaitez.
- 

## Tâches à réaliser

- Analyser le fichier CSV afin d'identifier les différentes entités métier (i.e. le modèle métier).

Exemple : Athlete, Epreuve, etc.

- Réaliser un dossier technique rassemblant :
  - Le diagramme de classes des entités métier
  - Le modèle physique de données (i.e. les tables)
  - Le diagramme de séquences pour le traitement d'insertion des athlètes (cf ci-dessous).
- Créer un dépôt GitHub appelé projet-jo

---

**OGROD - Olympic Games Results Open Database**

**Réalisation des MCD et MLD de la base de données relationnelle**

- Créer un répertoire conception et déposez-y votre dossier technique au format PDF ou Word
- Le diagramme de séquences à réaliser :
- L'idée est de réaliser le diagramme de séquence pour le traitement suivant : ce traitement parse le fichier et met en base de données uniquement les athlètes. Comme ce traitement doit être rejouable il n'est pas question de créer les athlètes en double.
  - La classe qui lance le traitement est une classe exécutable nommée InsertionAthlete.
  - L'extraction des athlètes du fichier doit être réalisée par une classe de service appelée AthleteService.

Cette classe possède une méthode getAthletes qui prend en paramètre le path d'accès au fichier et retourne la liste d'athlètes.

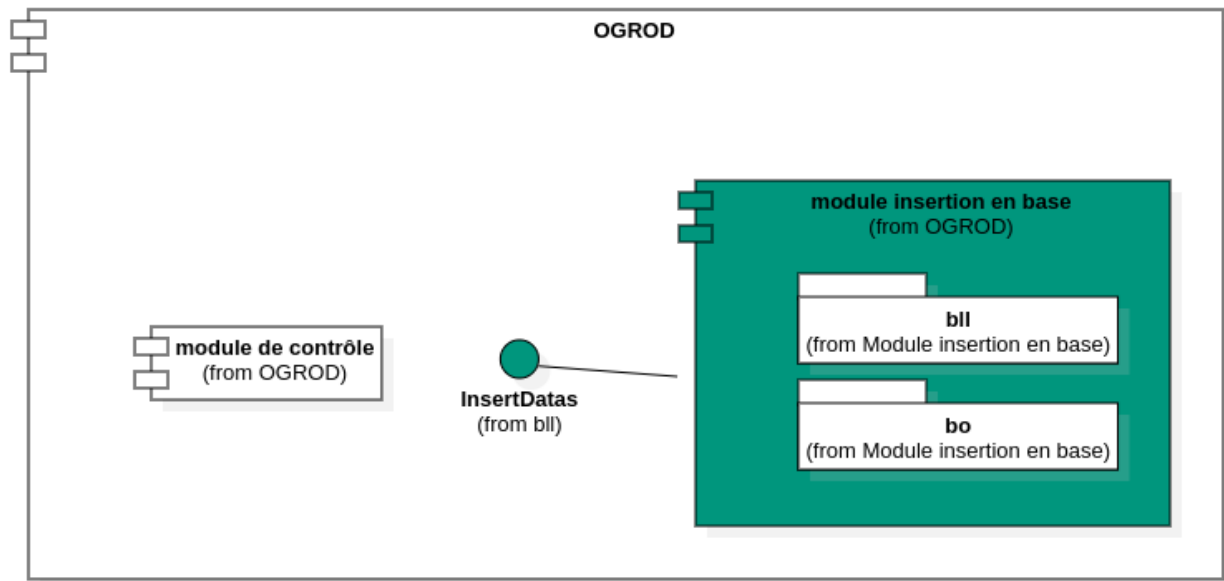
- Les échanges avec la base de données sont réalisés via une classe type DAO appelée AthleteDao. Cette DAO possède 3 méthodes :
  - Une méthode qui vérifie si un(e) athlète existe en base ou non. Cette méthode prend en paramètres ce qui permet d'identifier un athlète de manière unique et retourne un booléen.
  - Une méthode qui permet de mettre en base de données l'athlète si il/elle n'existe pas.
  - Une méthode qui permet de mettre à jour les informations de l'athlète si il/elle existe

# Table des matières

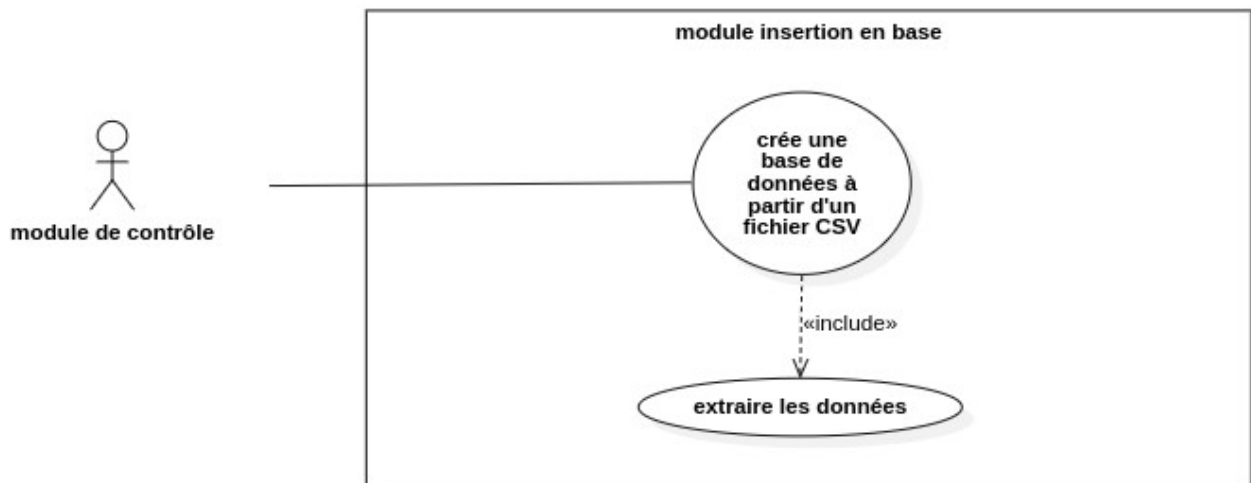
1. Analyse.....	6
1.1. Situation du module d’insertion au sein du logiciel.....	6
1.2. Diagramme de cas d’uilisation du module d’insertion.....	7
1.3. Diagramme de classe des entités métiers (modèle conceptuel des données persistantes de l’application).....	8
2. Mapping vers le modèle physique des données.....	9
2.1. Rappel sur le passage du mapping vers le modèle relationnel en UML.....	9
2.1.1.1. Modèle déduit.....	9
2.1.2. Diagramme des classes de l’ensemble du module d’insertion.....	10
2.1.3. Diagramme de séquence de la fonction d’insertion (Athlete).....	10
3. Annexes.....	11
3.1. Mapping du modèle conceptuel des données en modèle relationnel.....	11
3.1.1. Différencier conceptuel et relationnel.....	11
3.1.2. Définir le mapping de classes.....	14
3.1.3. Faire le mapping des associations binaires.....	18
3.1.4. Utiliser le mapping d'un héritage.....	24
3.1.5. Appliquer le mapping d'agrégats.....	28
3.1.6. Mettre en place des contraintes.....	33
3.1.7. Faire le bilan du modèle relationnel.....	38

# 1. Analyse

## 1.1. Situation du module d'insertion au sein du logiciel

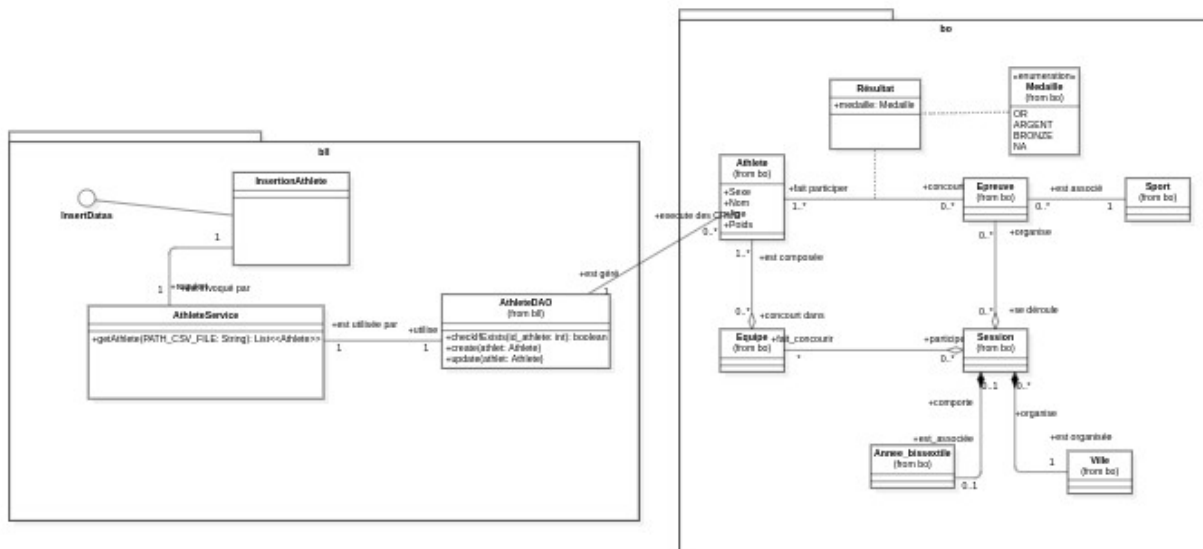


## 1.2. Diagramme de cas d'utilisation du module d'insertion

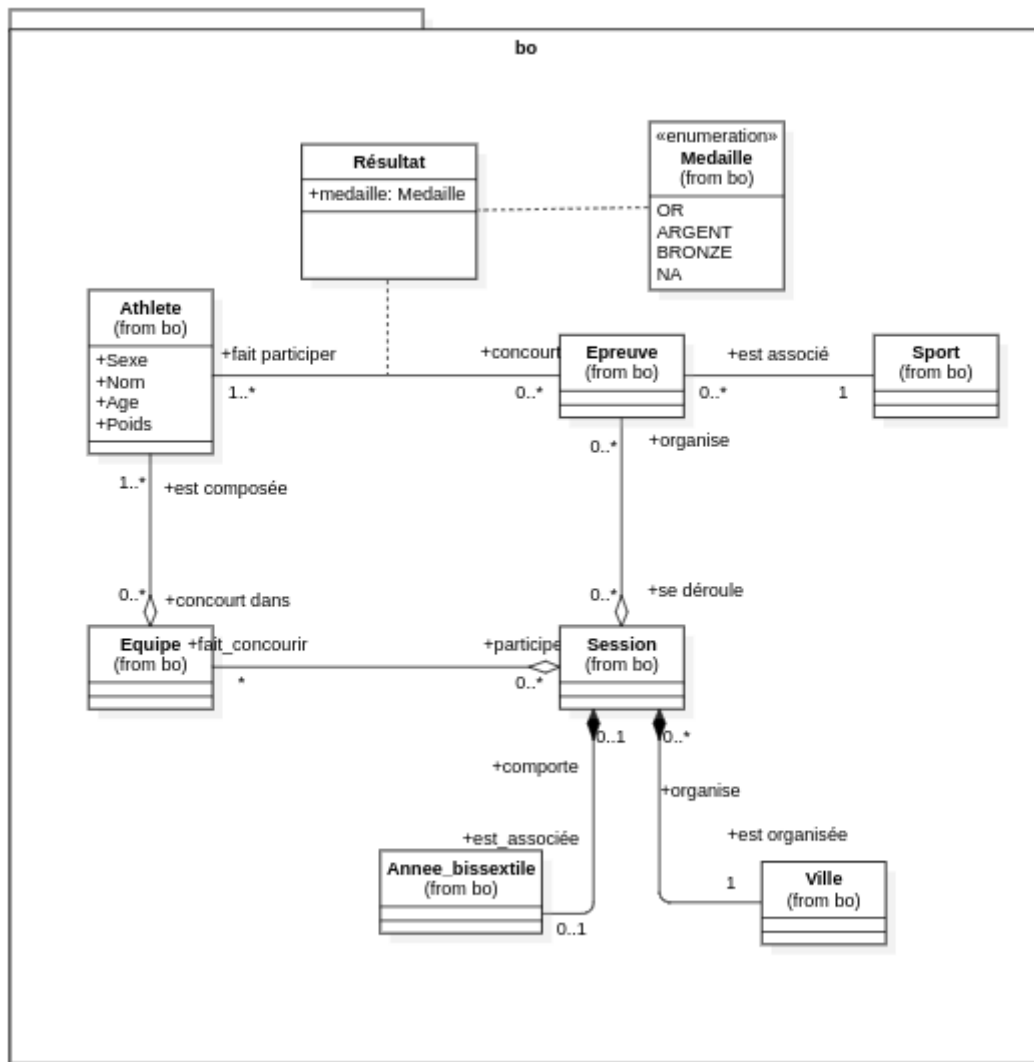


## 1.3. Diagramme des classes à mettre en œuvre

### 1.3.1. Modèle du domaine



#### 1.3.1.1. Diagramme de classe des entités métiers (modèle conceptuel des données persistantes de l'application)



## 2. Mapping vers le modèle physique des données

### 2.1. Rappel sur le passage du mapping vers le modèle relationnel en UML

Voir Annexes



### **2.1.1.      Modèle physique de données**

### **2.1.2.      Diagramme de séquence de la fonction d'insertion (Athlete)**

### **3. Annexes**

## 3.1. Mapping du modèle conceptuel des données en modèle relationnel

### 3.1.1. Différencier conceptuel et relationnel

Alors avant d'aborder la partie consacrée à la transformation d'un diagramme de classes en tables, et à la déduction des clés primaires et des clés étrangères, quelques mots à propos de diagramme de classes que vous serez peut-être en mesure de rencontrer un jour.

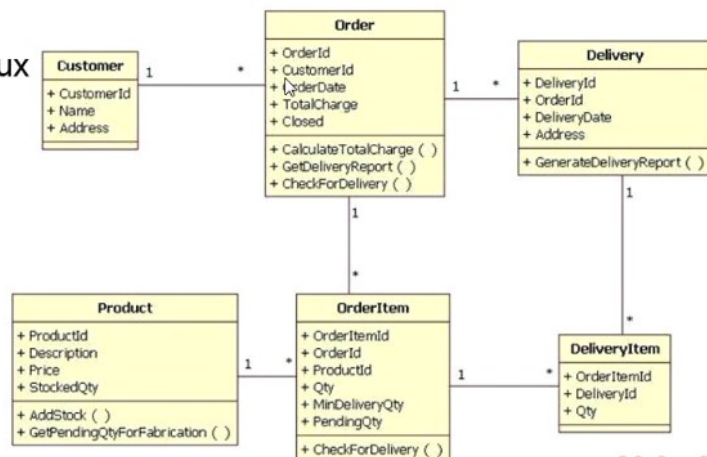
Des diagrammes de classes qui décrivent, en fait, des tables au lieu de décrire simplement des classes et des associations. Alors conceptuel n'est pas relationnel.

## Conceptuel n'est pas relationnel

- Présence d'un attribut à deux endroits :

➤ Modèle relationnel

➤ Modèle conceptuel hasardeux



Ce n'est pas parce que vous voyez un diagramme de classe d'UML qu'il s'agit d'un niveau conceptuel de modélisation. La présence d'un attribut à plusieurs endroits, ça peut vous indiquer que vous êtes en présence soit d'un modèle relationnel, soit d'un modèle conceptuel qui est un peu mal conçu parce qu'on a dupliqué des champs à différents endroits et vous savez qu'il faut pas le faire. Dans cet exemple, il y a des attributs qui sont dupliqués ici, le **CustomerId**, qui intervient en tant qu'identifiant ici et puis ici, en tant que champ. Il y a également le **ProductId** qui

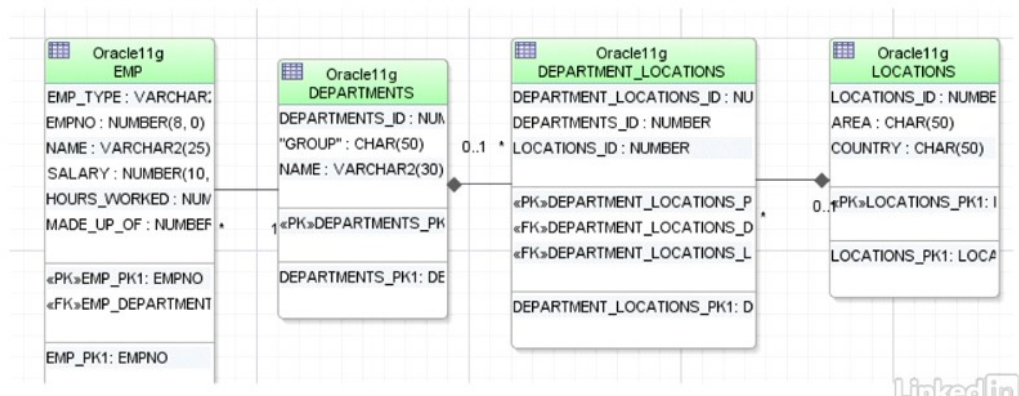
intervient à cet endroit là. Il y a le numéro de commande, l'OrderId qui est ici. Il y a une duplication d'attributs. Et ses attributs sont identifiants.

En fait, on se rend compte, eh bien, qu'on est en présence ici d'un modèle relationnel qui décrit des tables et des clés étrangères, où ne commande, c'est un client qui la passe puisqu'on a cette notation de 1. Et on a la présence de la clé étrangère ici, du code client, dans la commande. Et alors, c'est un mix de modélisation entre l'UML avec son 1 et son \* et puis le fait, que ce champ soit en fait une clé étrangère, qui n'apparaît pas ici clairement comme telle, mais qui est vraiment une clé étrangère.

Alors un autre exemple de modèle.

## Conceptuel n'est pas relationnel

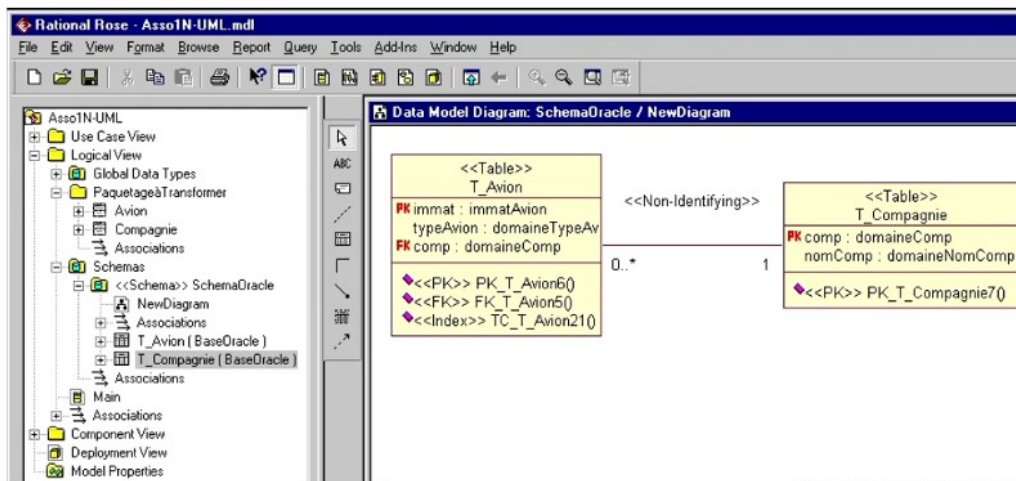
- Présence d'un attribut à deux endroits :
  - Modèle relationnel
  - Modèle conceptuel hasardeux



Là il s'agit d'un diagramme établi avec l'outil JDeveloper d'Oracle, qui est consacré plus à la génération de code Java. Et ici, il y a des champs qui se répètent, notamment le locations\_id, qui est ici, qui décrit un lieu ici. On le voit aussi ici. Il y a le champ department\_id qui aussi se répète. On est encore en présence d'un modèle relationnel puisque ce sont des identifiants qui se dispatchent tout en utilisant une notation aussi UML. On est en présence d'un modèle relationnel avec le nom des contraintes clés primaires et puis le nom aussi des index secondaires dans chacune des tables puisqu'il est en fait il s'agit ici de quatre tables.

# Conceptuel n'est pas relationnel

- Modèle relationnel ou modèle conceptuel hasardeux ?

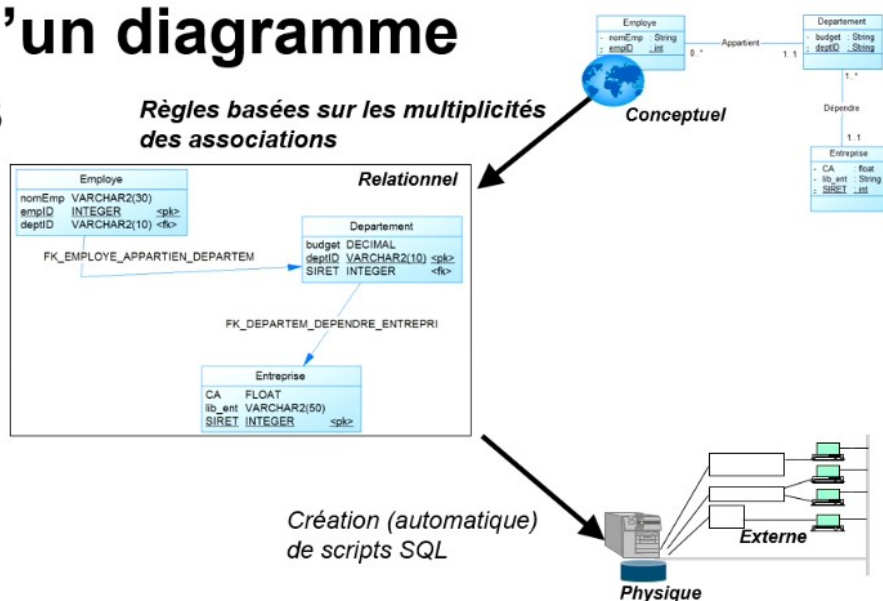


Et pour finir un autre diagramme établi avec l'outil Rational qui présente une notation UML, qui présente une association 1 à plusieurs, 1 d'un côté puis \* de l'autre. Et il y a aussi une duplication d'attributs. Ici, c'est l'attribut comp qui est identifiant de la table compagnie et il se retrouve ici.

C'est en fait une clé étrangère donc on a utilisé cet outil qui utilise Uml à tous les niveaux, au niveau conceptuel mais aussi au niveau relationnel.

### 3.1.2. Définir le mapping de classes

## Mapping d'un diagramme de classes



Dans cette quatrième partie qui est consacrée à la transformation d'un modèle de diagramme de classe en un modèle relationnel, je vais vous apprendre toutes les règles qui régissent le passage justement des associations et des classes en tables et en clés étrangères.

De telle sorte que vous puissiez comparer ce que vous génère l'outil que vous avez utilisé, avec ce que vous attendiez. Et si vous n'avez pas d'outils à votre disposition, vous pourrez quand même opérer ces transformations à la main.

Plus tard, nous verrons la génération et la vérification des scripts SQL qu'il faudra écrire pour créer les tables avec les contraintes et les index. Ces tables qui serviront ensuite aux développeurs à interroger la base et à manipuler la base au travers de vues.

Ce sont essentiellement les associations, les diagrammes de classe qui vont piloter les clés étrangères, le niveau relationnel. Mais surtout les multiplicité maximales.

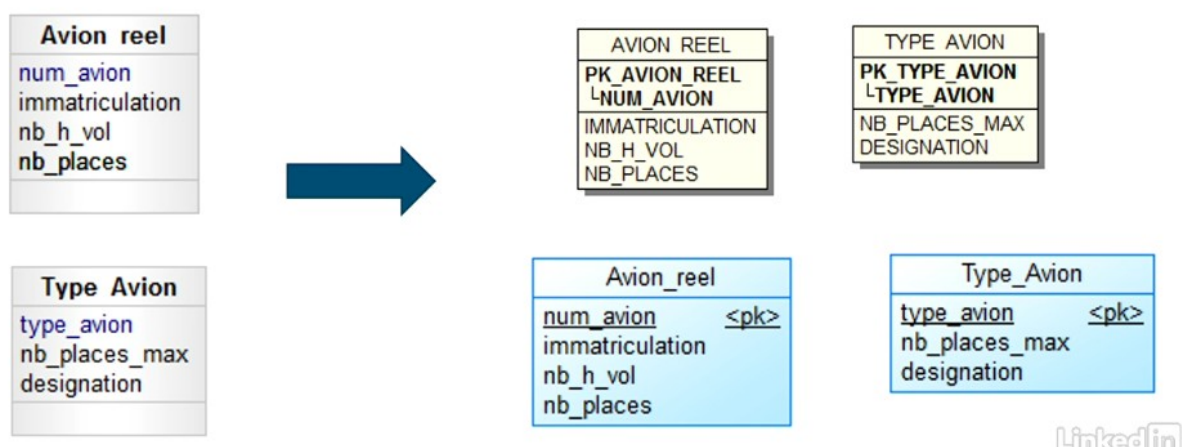
Vous n'avez pas le droit à l'erreur concernant le sens de lecture de vos modèles. Sinon, vos schémas relationnels seront erronés.

Commençons à présent par étudier la transformation des classes. Et puis ensuite, on s'intéressera aux associations aux classes associations, à l'héritage et à l'identification relative.

Concernant chacune de vos classes, elle doit devenir une relation qui deviendra une table plus tard.

## Mapping des classes

- Chaque classe doit se transformer en une relation
  - l'identifiant de la classe devient la clé primaire



Cette table est dotée d'une clé primaire qui n'est autre que l'identifiant de la classe. C'est pour ça que c'est important d'avoir un identifiant sérieux, minimal et stable pour chaque classe, pour qu'il devienne une bonne clé primaire par la suite.

J'ai utilisé ici deux outils différents pour vous montrer que seul le look du résultat change. On retrouve le fait que le numéro d'avion, qui était l'identifiant soit la clé primaire et que le type d'avion qui était l'identifiant de la classe des avions génériques, des familles d'avions soit aussi clé primaire qui est notée ici pk et qui est dans cet encadré.

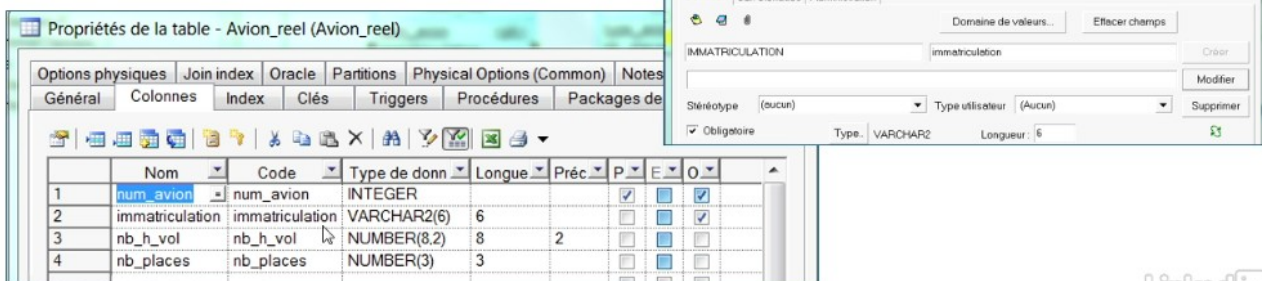
Certains outils dénotent aussi la présence d'identifiants secondaires. Je pense par exemple à l'attribut immatriculation qui pourrait jouer le rôle d'un identifiant secondaire pour un avion.

Et on aurait pu voir ak comme alternate key qui est autre qu'un index unique sur la colonne.

A ce stade, vous pouvez modifier les noms de colonnes et mettre nombre de places disponibles, à ajouter des colonnes, la date de mise en service. Rien n'est figé et il n'est jamais trop tard pour bien faire.

## Mapping des classes

- Statuer sur
  - le caractère obligatoire des autres attributs
  - le type de chaque colonne

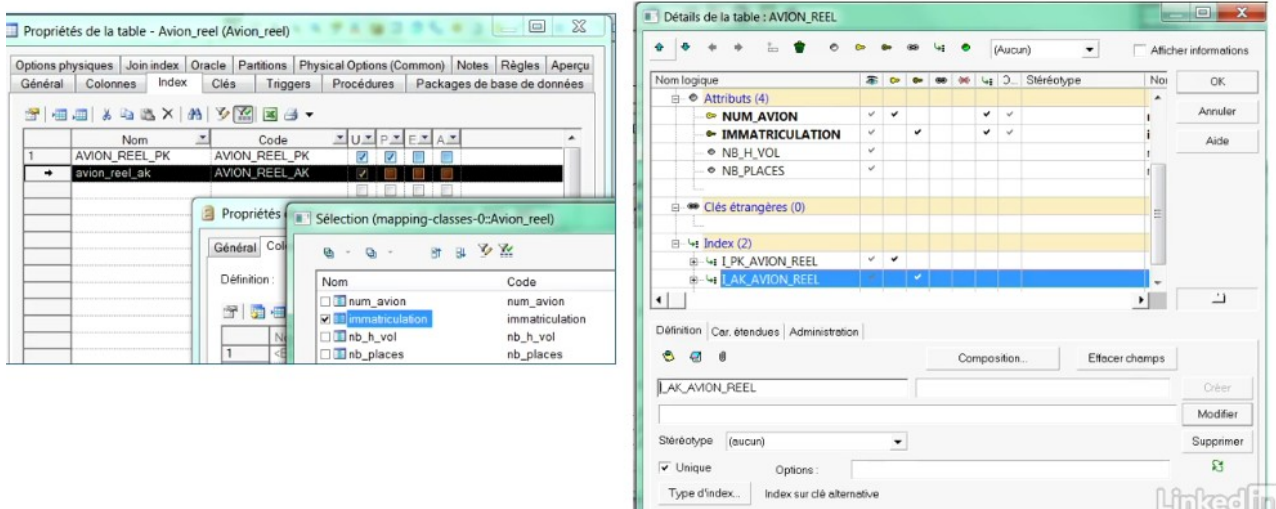


Suivant les outils que vous utiliserez, c'est intéressant de typer évidemment le plus précisément chaque colonne de la table qui sera générée en termes de noms, de taille, et aussi rendre obligatoire certains champs. Ici, je pense par exemple à l'immatriculation d'un avion qui peut être obligatoire ou pas. C'est toujours à discuter.



# Mapping des classes

- Chaque identifiant secondaire doit devenir unique (par un index)



J'évoquais précédemment les identifiants secondaires.

Ils se traduisent toujours par des index uniques au niveau du modèle relationnel et chaque outil dispose de sa stratégie de définition.

Ici, vous devrez choisir le champ immatriculation et définir un index de type unique. Puis, dans un autre cas, il suffira de créer un index en lui associant la colonne associée étant immatriculation. Mais il est important que chaque identifiant alternatif de vos classes soit déclarées en tant que index unique par la suite. Et la vraie différence avec l'index unique de la clé primaire, c'est que l'index unique peut être facultatif. Que vous n'êtes pas obligé de cocher "obligatoire" au niveau de la colonne de l'index unique. Dans ce cas là, ça voudrait dire que chaque avion doit avoir un numéro, obligatoirement, pour être stocké dans la base. Par contre, l'immatriculation on va dire, c'est unique, mais on ne la rend pas obligatoire. On achète l'avion, il n'est pas encore immatriculé ce n'est pas obligatoire ça convient. Et si dans votre outil, vous ne trouvez pas ce genre d'option, vous devrez ajouter manuellement un index dans votre script. C'est ce que l'on verra dans les vidéos consacrées à SQL.

### 3.1.3. Faire le mapping des associations binaires

Après avoir transformé les classes, on va s'intéresser à la transformation des associations binaires.

## Mapping des associations binaires

- Trois cas
  - one-to-many
  - many-to-many
  - one-to-one
- Création de clés étrangères
  - identifiants de la classe « parent »
- Création d'une table supplémentaire
  - many-to-many
  - peut convenir aussi aux autres cas



Il y a trois types d'associations binaire principales.

- Les 1 à plusieurs
- les plusieurs à plusieurs
- les 1 à 1.

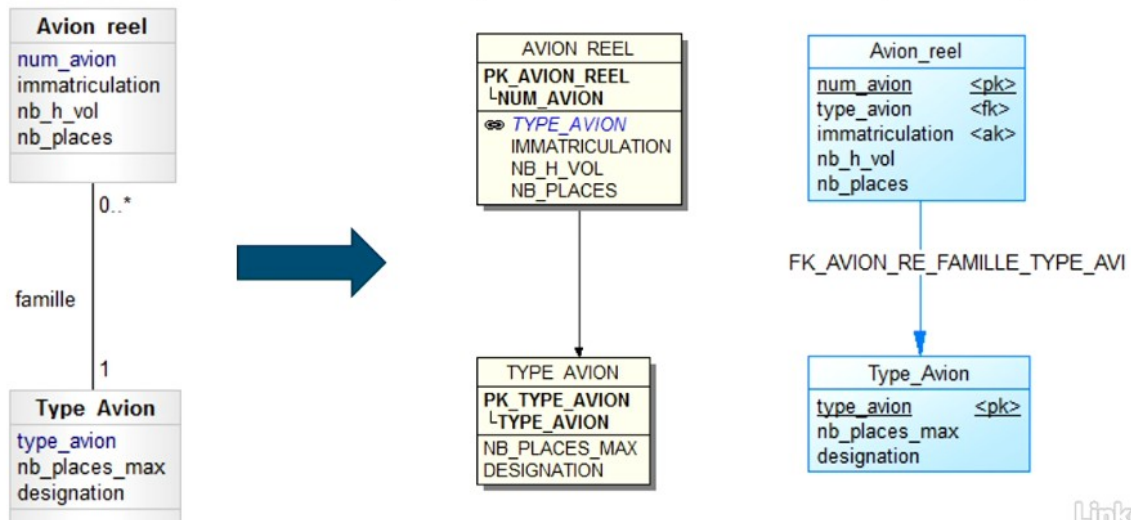
Dans ces trois cas, il y aura l'apparition d'une clé étrangère et dans un des cas, le plusieurs à plusieurs, il faudra créer une table supplémentaire pour faire le lien finalement entre les deux classes à relier.

Alors cette création d'une table supplémentaire ça peut également convenir aussi aux 1 à plusieurs et 1 à 1. C'est la solution que j'appelle universelle et on terminera la vidéo sur ce point en particulier.

Concernant les associations à plusieurs qui sont très, très courantes.

# Mapping des one-to-many

- Création d'une clé étrangère (identifiant de la classe « parent »)



LinkedIn

Il y a une classe qui est dite parent et une classe qui est dite enfant. La classe qui est dite parent, c'est celle qui est du côté du 1. Un type d'avion correspond à plusieurs avions réels.

C'est l'identifiant de cette classe de parents qui va devenir une clé étrangère, qui va être propagée dans l'autre table.

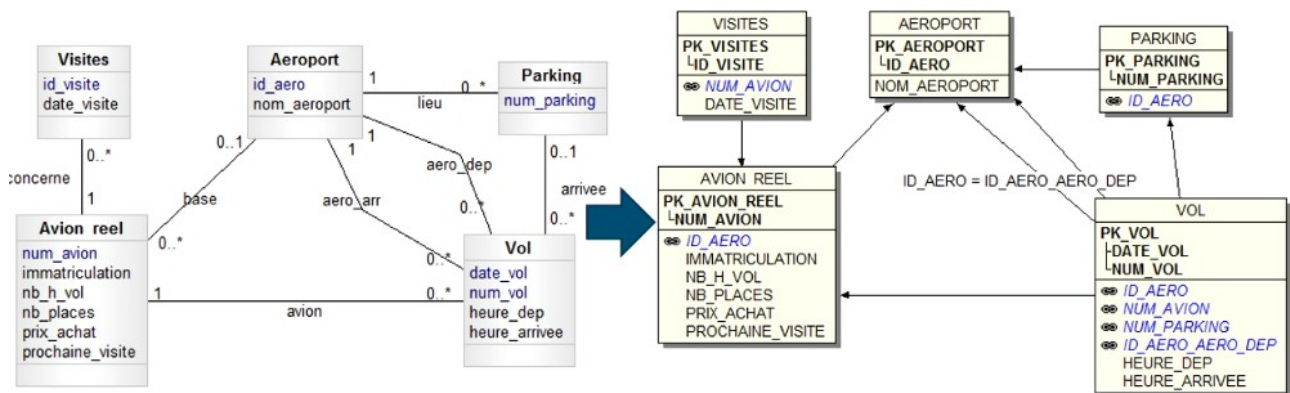
Dans ce cas, on a deux classes Avion\_reel et Type\_avion qui donnent lieu à deux tables. Avion\_reel et type\_avion. J'ai utilisé deux outils différents pour vous montrer que seul le design change mais on retrouve l'identifiant de la classe qui devient clé primaire, le numéro d'avion, l'identifiant de la classe type\_avion également.

Et puis, le lien s'appelle famille ici, devient une clé étrangère avec la colonne type avion qui apparaît dans la table avion\_reel, type\_avion ici, là. Type\_ avion là.

Alors ici c'est dénoté fk comme foreign key. Ici, c'est symbolisé par un petit lien de chaîne et une différence de couleur. En tous les cas, la table de référence, c'est la cible de la flèche. Et dans certains cas, l'outil inscrit déjà le nom de la contrainte SQL.

# Mapping des one-to-many

- Clé étrangère non nulle pour les multiplicités 1..1



LinkedIn

C'est un exemple où il y a plusieurs associations à plusieurs qu'il faut traduire.

Donc je vais voir deux choses à dire. Ici, je vais vous parler du fait de plusieurs liens qui arrivent entre deux mêmes classes et puis du fait d'avoir une multiplicité minimale à 0 ou à 1.

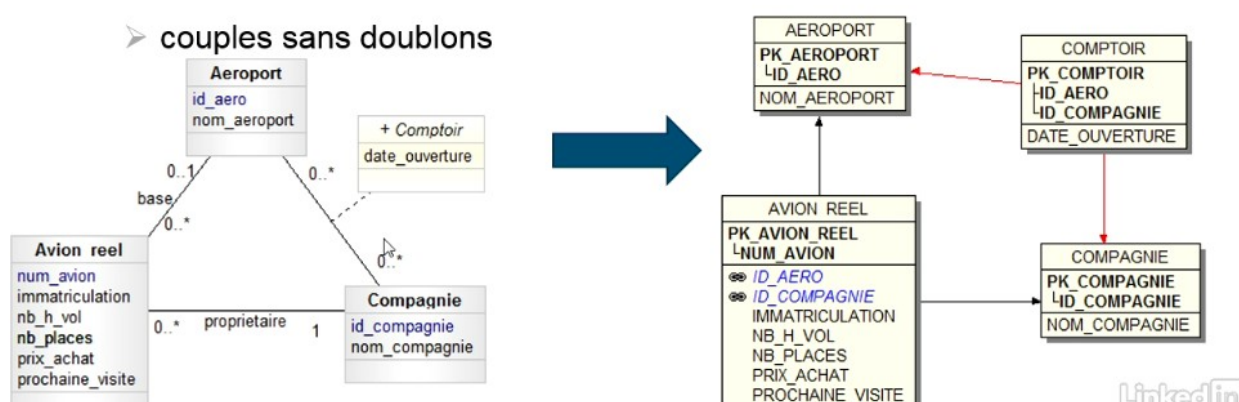
Quand plusieurs associations concernent les deux mêmes classes, le principe est identique. Ici, un vol est au départ d'un aéroport donc il va y avoir le côté id\_aero qui va migrer dans cette table en tant que étrangère. Ça c'est le départ, puis l'arrivée ce sera pareil. Il y aura id\_aero qui va migrer dans cette table.

Mais 2 colonnes ne peuvent pas porter le même nom, 2 colonnes distinctes, donc il va falloir renommer une des colonnes, une des clés étrangères, de manière distincte. Peut-être 1 au départ et l'autre à l'arrivée. Ça illustre aussi le fait qu'une clé étrangère ne porte pas forcément le même nom que la clé primaire référencée. id\_aero ici, c'est le même nom mais l'aéroport de départ, il s'appelle ID\_AERO\_AERO\_DEP. Pour autant, il référence aussi la même colonne en cible. Pour les autres associations, on trouve les clés étrangères qui migrent de manière un petit peu classique. Ici, avion\_reel doit aller dans visite puisque le 1 est du côté avion\_reel donc on a le num\_avion qui porte le même nom parce qu'il n'y a pas d'ambiguïté.

Pour les multiplicités 1 à soit 1 tout court, c'est la même chose, la clé étrangère devra être not null. Ici, un parking est impérativement situé dans un aéroport. Cette colonne sera avec une contrainte not null, ce qui ne sera pas le cas de la clé étrangère de cette association, qui est entre parking et vol. Cette colonne NUM\_PARKING pourra être null parce qu'on ne sait peut-être pas sur quel parking le vol va arriver.

## Mapping des many-to-many

- L'association (ou la classe-association) devient une table
  - la clé primaire est composée des deux identifiants
  - chaque identifiant devient en général clé étrangère
  - couples sans doublons



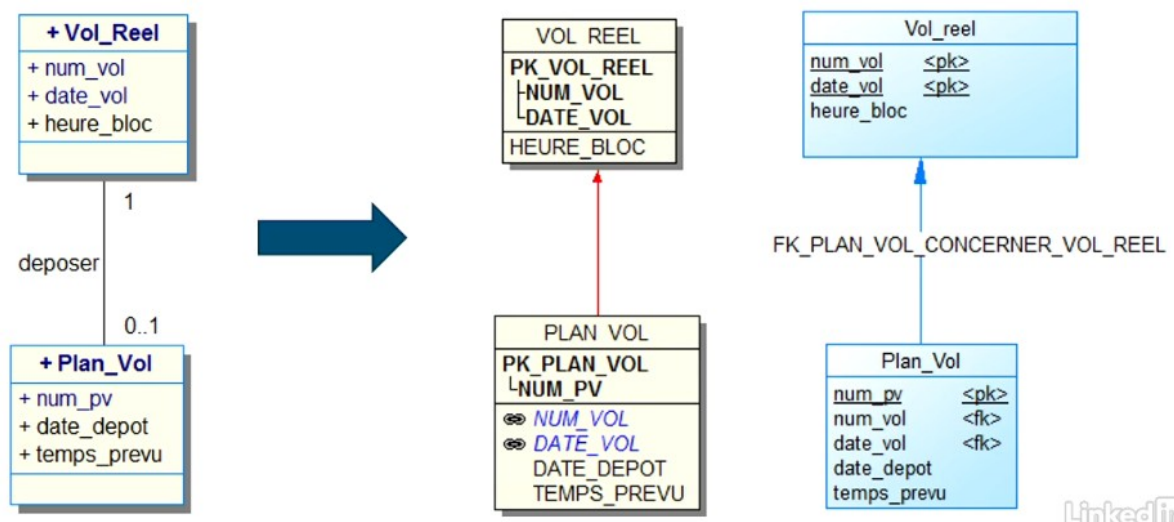
Pour les associations plusieurs à plusieurs, ici il y en a une, qu'il soit associé ou non à une classe association, le processus est un peu plus complexe, c'est-à-dire que ce lien va devenir une table, comptoir, et puis, cette table association va hériter des deux identifiants des classes connectées pour composer la clé primaire. Et chaque composant étant aussi une clé étrangère puisque un comptoir, ça référence un aéroport qui doit exister et une compagnie qui doit exister. Alors les liens sont ici indiqués en rouge avec cet outil. pour montrer que ce sont deux liens not null et qu'ils font partie de la clé primaire.

Ce genre de modélisation modélise des couples sans doublon. Ici, DATE\_OUVERTURE se retrouve en champ qui n'est pas clé, mais s'il n'y avait pas eu de date d'ouverture dans ce comptoir et s'il n'y avait pas eu de classe association, on aurait quand même eu une table d'association comptoir qui référence aéroport et compagnie, sans avoir de colonne date d'ouverture.

Alors je dis que chaque identifiant devient, de manière générale, une clé étrangère aussi, il se peut que dans certains cas vous ayez à constituer des couples de classe, avec une classe qui ne sert que pour le couple et à ce moment-là, on n'aura pas de classe extérieure et seules les colonnes seront dans la clé pour composer un couple. Et dans ce cas-là, une seule sera clé étrangère.

## Mapping des one-to-one

- Création d'une clé étrangère : identifiant de la classe 1..1 (ou 1)



Les associations 1 à 1 sont relativement rares mais c'est intéressant de considérer ce cas en particulier. C'est la même chose que pour une association 1 à plusieurs, à savoir une clé étrangère va migrer d'une classe vers une table. Mais il est préférable que ça soit la clé de la classe qui est du côté du 1 en minimum. I

ci, c'est l'identifiant de vol réel qui va venir dans la table plan de vol. C'était préférable que de migrer le numéro de plan de vol dans la table vol réel, ça permet d'éviter d'avoir des valeurs nulles dans la base de données. Le lien est donc not null, ce qui explique qu'il soit rouge avec cet outil. Ici, on ne voit pas le fait que ces colonnes soient de type obligatoire mais elles le sont.

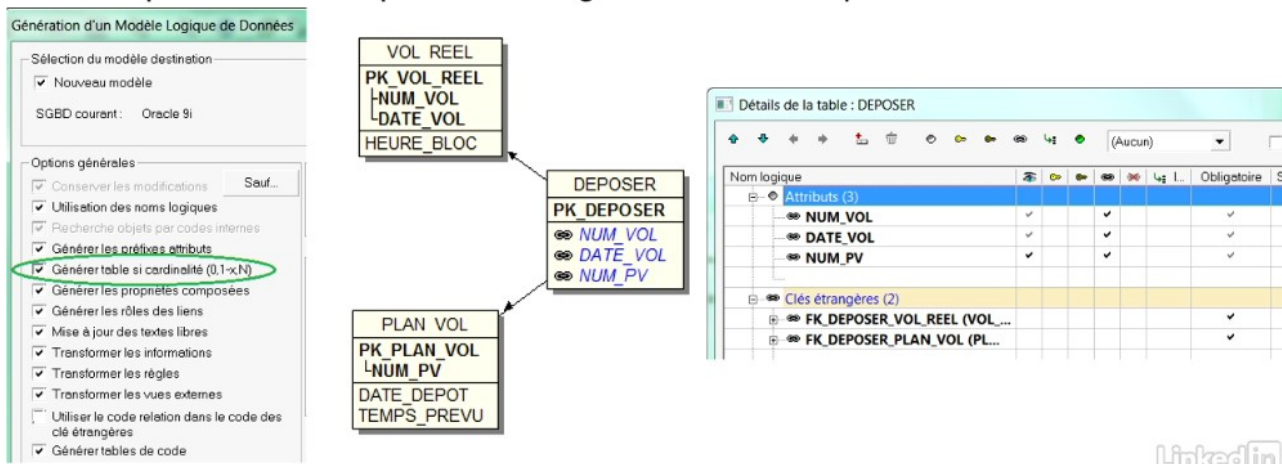


Alors, il y a une possibilité pour transformer une association binaire, quelle qu'elle soit.

## Mapping à l'aide d'une table

- Solution « couteau suisse »

➤ permet de s'adapter aux changements des multiplicités



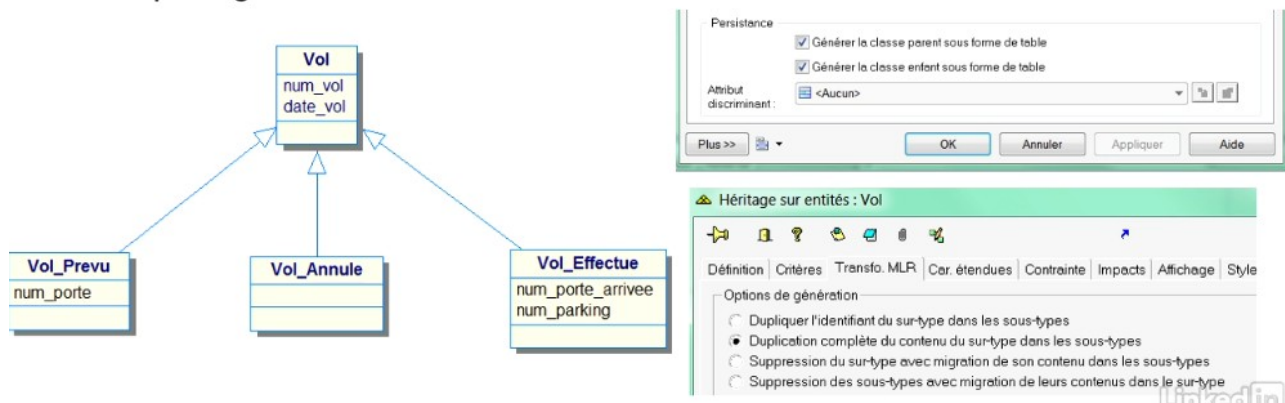
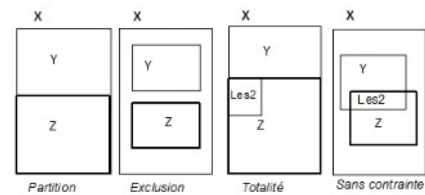
Qu'elle soit 1 à plusieurs, plusieurs à plusieurs ou 1 à 1, en associant toujours une table d'association entre les deux tables de référence. Ici, je choisis de modéliser le même exemple que tout à l'heure, à savoir qu'un plan de vol est associé à un vol réel et un vol réel n'avait qu'un plan de vol. Sans clé étrangère d'un côté ou d'un autre, c'est la table d'association entre les deux qui contient les identifiants des classes concernées. Et seules des contraintes sur ces identifiants, le numéro de vol, le date\_vol et le numéro de plan de vol, que ça soit unique ou pas unique, nous permettra de dire "je suis dans le cas 1 à 1", "je suis dans le cas 1 à plusieurs " ou "je suis dans le cas plusieurs à plusieurs". A moindre frais, cette modélisation permettra d'évoluer en termes de cardinalité dans le temps, sans changer la structure des tables, simplement en ajoutant des contraintes. Si je veux reproduire l'exemple précédent, il faudra que je détermine la clé primaire de cette table pour faire du 1 à 1, je dirais que c'est le numéro de plan de vol qui est la clé primaire et que ces deux champs-là sont not null. Alors ce n'est pas facile à trouver des outils qui automatisent ce processus, et bien souvent, il faudra que vous fassiez ça à la main, c'est-à-dire remplacer le lien de clé étrangère initialement par une table en plus avec deux clés étrangères.

### 3.1.4. Utiliser le mapping d'un héritage

Trois alternatives s'offrent à vous pour traduire les associations d'héritage que vous aurez mises en œuvre dans votre modèle.

## Mapping de l'héritage

- Distinction : autant de tables que de classes
- Push-down : migration des colonnes dans les sous-classes
- Push-up : migration des colonnes dans la sur-classe



Ces trois cas, nous allons les traiter les uns après les autres.

Il s'agit de

- la distinction
- d'une conception descendante
- d'une conception ascendante.

Ces cas sont programmés dans certains outils et si vous utilisez un outil qui ne dispose pas de ces alternatives, vous devrez programmer votre transformation à la main. Dans cet outil, le fait d'avoir coché ces deux cases exprime que je veux dériver une table vol et puis, je veux aussi dériver des tables, des sous-classes. On peut combiner différentes solutions.

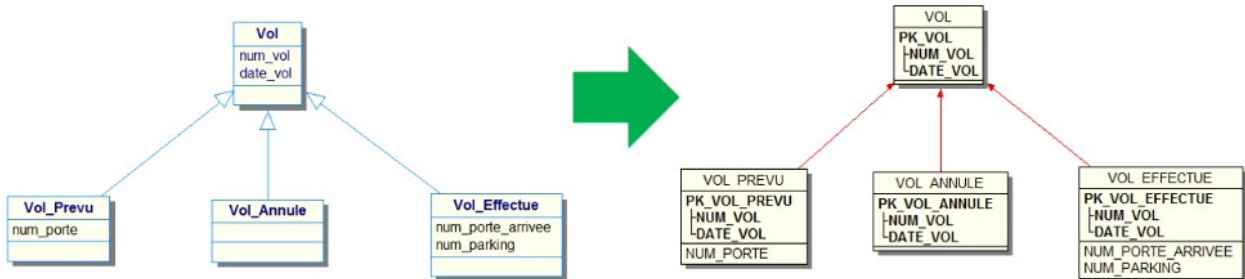
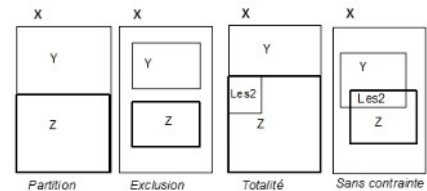
Dans un autre outil, c'est exprimé différemment. Et ici, je veux dupliquer complètement les données de cette classe dans les sous-classes et éventuellement, je veux supprimer cette classe principale.



# La solution par défaut

- Distinction : autant de tables que de classes

- seul l'identifiant migre
- la clé primaire de chaque table dérivée est aussi clé étrangère



LinkedIn

Parallèlement à ça, chaque héritage est contraint par une des familles suivantes, et vous devez savoir dans quel cas votre héritage se positionne, de telle sorte à choisir la meilleure transformation possible.

Il n'y a pas de solution miracle, même si la solution par défaut est celle qui convient à tous les cas.

Néanmoins, vous devrez absolument connaître, suivant le système que vous modélisez, dans quelle famille de contraintes vous vous trouvez.

Ici, si vous supposez qu'un vol peut être à la fois prévu puis effectué, vous serez dans un de ces deux cas de contrainte. Et si vous dites, il peut y avoir des vols qui ne sont ni prévus, ni annulés ni effectués, dans la base de données, on sera dans ce type de contrainte.

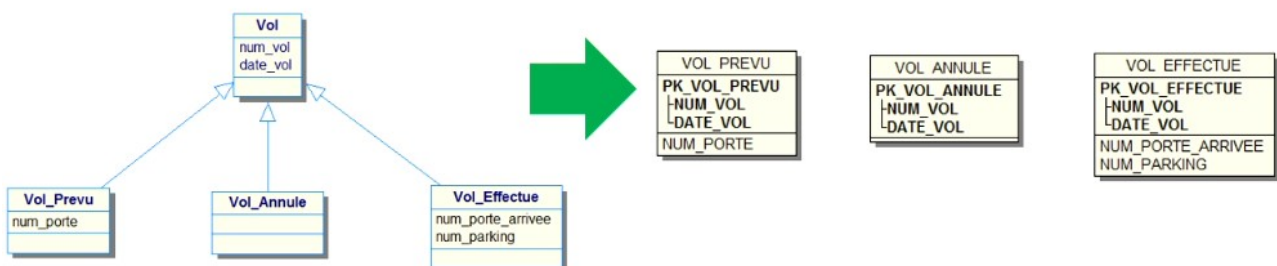
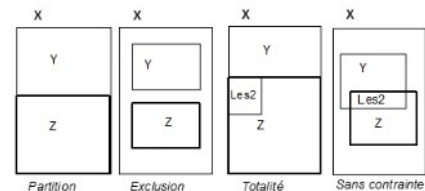
La décomposition par distinction, c'est souvent la solution par défaut qui convient à tous les cas de contrainte, qui est la plus polyvalente. Ici, seul l'identifiant de la surclasse migre dans les sous-tables. Et puisqu'il migre, eh bien, forcément il devient aussi clé étrangère mais aussi identifiant de chaque table dérivée. Cette solution correspond parfaitement à tous les cas possibles, notamment le cas du vol réel, du vol qui peut être juste dans cette table et ne pas être présent dans les autres.

Et puis, il peut y avoir certains vols qui sont à la fois référencés ici et là et qui, bien sûr, existent en tant que vol au niveau le plus haut. Mais cette structure convient également pour les autres cas de contrainte, sous réserve de faire une programmation supplémentaire à l'aide de déclencheurs ou de procédures.

## La solution push-down

- Pas de « sur-table » : que des tables dérivées

- l'identifiant est partout le même
- l'exclusion ou l'absence de contraintes n'est pas réaliste...



LinkedIn

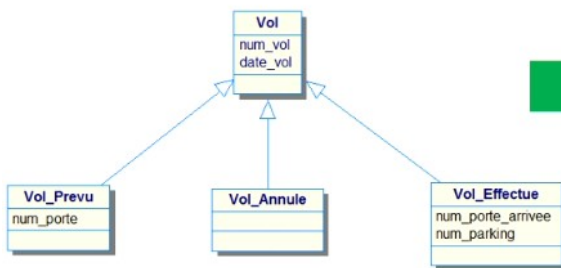
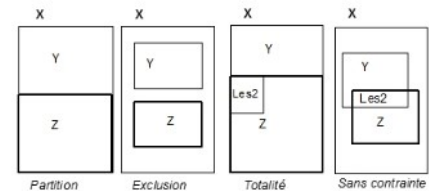
La transformation descendante est la plus exclusive.

Elle se caractérise par le fait qu'il n'y a plus de table qui implémente la surclasse, tous les champs de la classe sont dispatchés dans les sous-classes qui deviennent des tables. Ainsi, il n'est pas possible d'exprimer, par exemple ici et là, l'exclusion ou l'absence de contrainte puisqu'on n'a plus de table pour stocker les vols qui ne sont ni prévus, ni annulés, ni effectués.

C'est une solution qu'il est conseillé de réserver à l'héritage par partition, qui dans le temps d'ailleurs, ne devrait pas évoluer vers un autre cas.

# La solution push-up

- Seule la « sur-table » généralise le tout
  - la totalité ou l'absence de contraintes est possible mais coûteuse



VOL	
PK_VOL	
NUM_VOL	
DATE_VOL	
NUM_PORTE	
NUM_PORTE_ARRIVEE	
NUM_PARKING	

LinkedIn

La transformation ascendante est la plus simple puisqu'au final une seule table est générée et cette table contient toutes les colonnes qui sont extraites des différentes sous-classes. Elle est simple mais elle est problématique par la suite au traitement parce qu'il y a certaines colonnes qui devront être évaluées à null pour caractériser tel cas ou tel autre cas. Par exemple, pour les vols qui ne sont ni prévus, ni annulés, ni effectués, eh bien, toutes ces colonnes seront évaluées à null

Mais comment différencier un vol annulé d'un vol réel ? Puisque le vol annulé n'a pas de caractéristiques supplémentaires. Donc il y aura du traitement à faire, des colonnes évaluées à null, ce qui implique des index qui ne sont pas efficaces et des requêtes un peu plus compliquées à écrire.

Cette solution peut être envisagée si toutes les sous-tables contiennent des données propres et puis, en ajoutant un champ de classification en général c'est ce qu'on fait. On ajoute un champ de classification à cette table.

### 3.1.5. Appliquer le mapping d'agrégats

Alors, vous avez vu comment se traduisent des classes puis des associations, puis l'héritage. Et maintenant, on va étudier comment se traduisent les agrégats.

## Mapping des agrégats

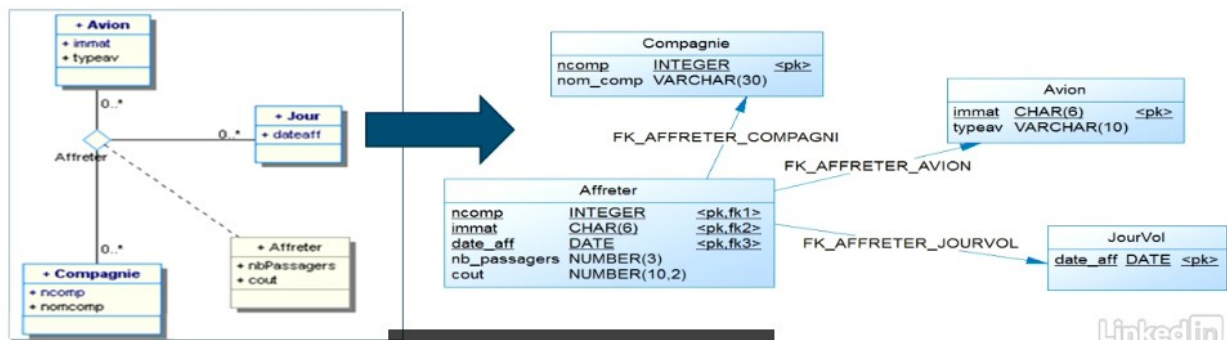
- Trois cas
  - *n*-aires
  - classes-associations
  - composition
- Création d'une table supplémentaire
  - *n*-aires
  - classes-associations
- Migration d'un identifiant
  - composition



Et ces agrégats peuvent provenir d'associations *n*-aires, de classes-associations, ou d'associations de composition, qui permet de programmer l'identification relative. Pour ces trois cas, il faudra créer une table supplémentaire concernant les associations *n*-aires et les classes-associations, et migrer un identifiant concernant la composition. Commençons par les associations *n*-aires qui décrivent un regroupement de plusieurs classes.

# Les associations *n*-aires

- L'association devient une table
  - clé primaire composée des identifiants des classes connectées
  - chaque identifiant devient clé étrangère
  - aucun contrôle...



Et là, le raisonnement est le même que le raisonnement pour les classes-associations, à savoir une nouvelle table doit être créée et cette table réalise finalement chaque regroupement. Ici, il s'agit de composer des triplets compagnie/avion/jours pour permettre de savoir quels avions ont été programmés ou affrétés par quelle compagnie à un jour donné. L'identifiant de la table d'association, c'est tous les identifiants des classes qui sont connectées et la table d'association peut contenir aussi des attributs propres à la classe-association. Ici, le nombre de passagers et le coût du vol.

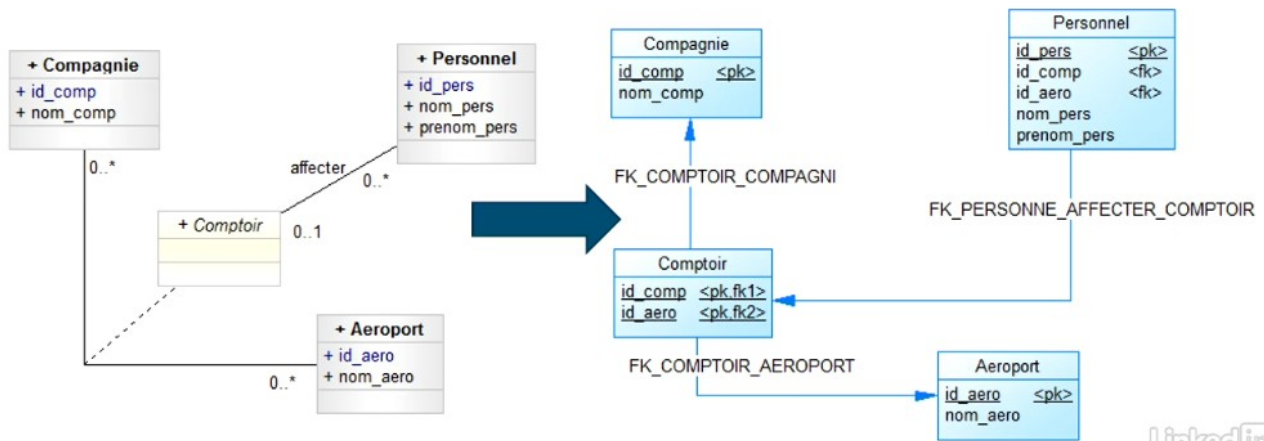
Cette construction du diagramme de classes est plutôt à éviter.

J'en avais parlé auparavant. Parce qu'aucun contrôle ne sera possible a posteriori dans la base de données. Ici, les seuls contrôles qu'on peut faire ce sont les contrôles de clés étrangères à savoir qu'un affrètement peut être possible si la compagnie existe, si l'avion existe et si le jour est référencé comme un jour ouvrable. Mais rien ne dit que la compagnie a le droit d'affréter un avion, en particulier. Ici, il aurait été plus judicieux de composer des couples avion/compagnie. Quelle compagnie peut utiliser ou affréter quel avion ? Et ensuite, relier ces couples à plusieurs jours pour composer les affrètements.

# Classes-associations

- La classe-association devient une table (couples sans doublons)

➤ peut être reliée à d'autres tables



Ce même mécanisme, je l'utilise pour cet exemple, où j'ai privilégié le couple fort compagnie/aéroport en termes de présence.

Chaque aéroport peut accueillir plusieurs compagnies et une compagnie peut être présente sur différents aéroports. Et ce sont ces couples réalisés par la classe-association que je peux relier à d'autres classes. Ici, la classe des personnels. La traduction de ce modèle fait intervenir la table d'associations comptoir puisqu'elle dérive de la classe-association.

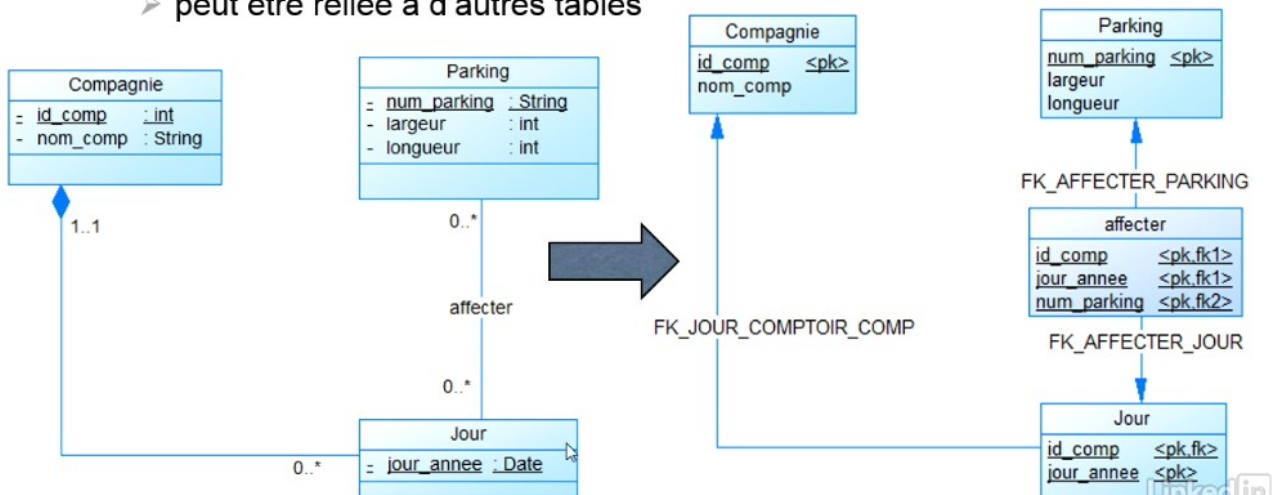
L'identifiant de cette table c'est les identifiants des classes connectées. Ici, tous les couples sans doublon seront présents. Ici, il y aura vraiment l'existence de la présence des compagnies dans chaque aéroport. Ces couples peuvent être reliés à d'autres classes. Ici, pour un comptoir plusieurs personnes peuvent être présentes. Et une personne n'est présente que dans un comptoir. Cette association 1 à plusieurs se traduit naturellement par une clé étrangère. La clé étrangère de la classe parent, ici comptoir, qui dispose de plusieurs personnels. La clé étrangère de la classe comptoir va migrer dans la relation enfant, personnel, et cette clé étrangère est composée, bien sûr, de l'identifiant comptoir à savoir, le numéro de compagnie et le numéro d'aéroport. Cette table personnelle, finalement, c'est elle qui compose l'agrégat entre personnel, compagnie et aéroport, à la condition que le comptoir existe bien dans l'aéroport. Cette cohérence a priori est réalisée à l'aide de ce couple.

Enfin, les couples sans doublon peuvent aussi être implémentés à l'aide de l'association de composition. C'est ce que nous allons voir.

## L'identification relative

- L'identifiant migre vers la table du côté « plusieurs » : table de couples

➤ peut être reliée à d'autres tables



Ici, l'agrégat est au niveau de la classe ce jour puisque l'on va composer des couples sans doublon entre compagnie et jour. L'identifiant de compagnie va migrer dans cette classe, pour devenir la table jour, muni de deux colonnes qui vont composer la clé primaire : l'id compagnie qui référence une compagnie qui existe et le jour de l'année qui référence un jour ouvrable. Un couple peut être connecté à une classe tierce, ici parking. On exprime par l'association affecter qu'une compagnie à un jour donné peut utiliser plusieurs parking. Et un parking peut être utilisé à différents jour, par différentes compagnies. Cette association de plusieurs à plusieurs se traduit naturellement par une table d'association qui référence chaque classe connectée. Ici parking, par son numéro, et puis jour par l'identifiant de jour qui est le couple compagnie/jour.

L'identifiant de cette table, la clé primaire, est composée de ces trois colonnes. Ici, l'agrégat est composé à trois, sous réserve que la compagnie travaille bien ce jour-là. La cohérence est ici apportée, a priori. On a décrit les différentes transformations



entre les différents éléments d'un diagramme de classe, des classes, des associations, l'héritage, l'identification relative.

On va maintenant s'intéresser à la mise en place des contraintes.



### 3.1.6. Mettre en place des contraintes

## Les contraintes au niveau relationnel

- Inclusion
  - clés étrangères
- Exclusion
  - table d'association
- Dénombrer et ordonnancer
  - compteur
  - contraintes de vérification et d'unicité



Au niveau relationnel, il est possible de définir certaines contraintes simples sans faire une programmation complexe pour répondre à des règles métier et à des contraintes déjà prédéfinies d'UML qui existent, qui sont ordered ou subset, par exemple.

La première d'entre elles, c'est l'inclusion. On va utiliser un mécanisme de clé étrangère qu'on va rajouter à une transformation initiale. L'inclusion c'est le sous-ensemble, c'est subset.

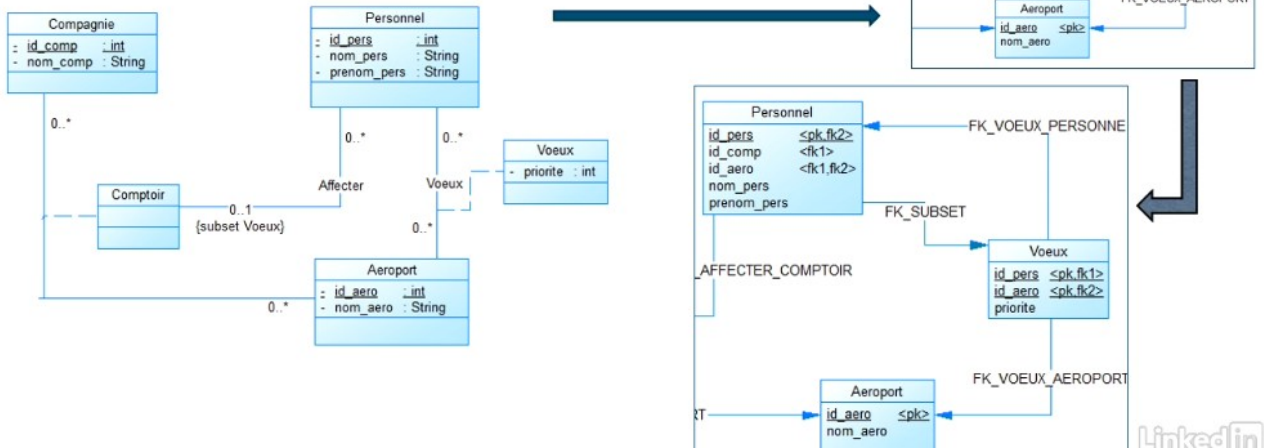
L'exclusion, je pense par exemple à l'ordonnance du patient qui ne doit pas prendre en compte un médicament interdit. Donc là, on utilisera une table d'association et on fera une sorte de clé étrangère à l'envers. Il ne faudra pas que le couple médicament/patient soit présent dans la table des médicaments interdits.

Et puis, l'ordonnancement et le dénombrement, notamment je pense à la contrainte prédéfinie ordered d'UML pour ordonner les choses, eh bien, il faudra rajouter probablement, une colonne de type compteur et puis, des contraintes de vérification et d'unicité, pour contraindre par exemple, un employé à ne pas avoir plus de 3 mails, un avion à ne pas faire 4 vols par jour etc.

On va avoir donc des exemples pour chacun de ces cas.

## Inclusion

- Chaque employé émet des vœux d'affectation
  - pas d'affectation sans une demande préalable



Alors, je considère à nouveau l'exemple qui décrit les embauches des employés dans les comptoirs de chaque aéroport. On l'a vu, il y avait un couple nécessaire pour composer la présence de l'aéroport. A ce couple, on rattachait l'existence de plusieurs personnels. Et puis, on ajoute le fait que des employés ont émis des vœux pour travailler sur certains aéroports avec une certaine priorité. Quelqu'un veut travailler à Orly avec une priorité 1, à Agen avec une priorité par exemple. Donc cette nouvelle association va gérer les vœux. Et la contrainte c'est de dire "on affecte une personne à la condition qu'elle ait déjà postulé au préalable sur cet aéroport sinon on ne l'affecte pas". Cette contrainte, c'est celle de l'inclusion qui s'écrit avec la contrainte subset d'UML A droite, c'est la transformation initiale de ce modèle.

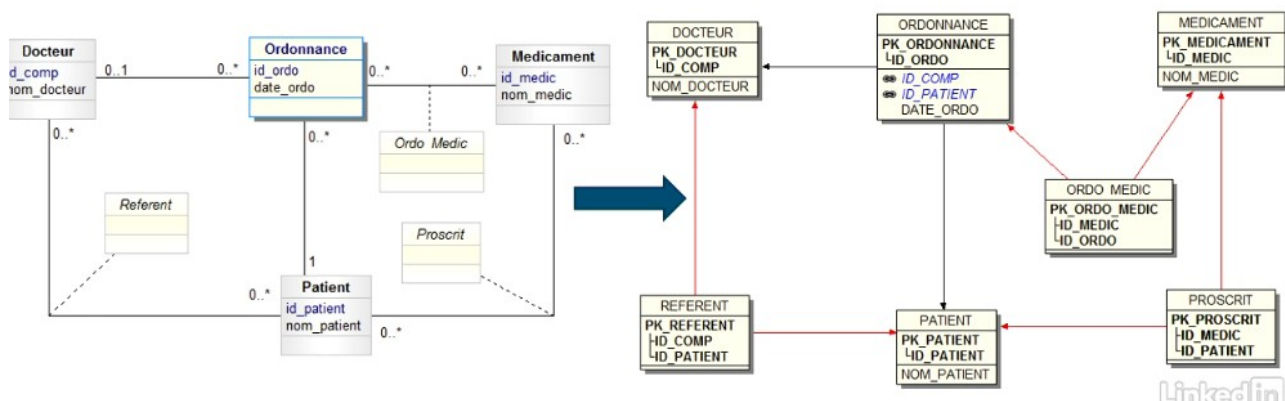
L'association affectée donne lieu à la présence d'une clé étrangère, de comptoir vers personnel, c'est ce qui explique le couple compagnie/aéroport, qui décrit finalement l'affectation de la personne qui vient de l'affectation, l'association 1 à plusieurs. Ensuite, cette association, la classe-association Voeux, va se dériver dans une table d'association qui va lister les couples personnel/aéroport en affectant à chacun d'eux une priorité. Ici, pour l'instant, aucun lien n'est établi entre ces données et il va falloir le faire.

La contrainte d'inclusion se programme tout simplement en ajoutant une clé étrangère entre personnel et vœux, sans aucun impact sur la structure de la table personnelle, et cette clé étrangère entre personnel et vœux est composée des colonnes personnel et aéroport qui se retrouvent en référence ici. FK 2 FK 2, c'est donc la deuxième clé étrangère de cette table. La table personnel aura trois contraintes. Une contrainte clé primaire, qui est l'id personnel, une contrainte clé étrangère qui est le couple compagnie/aéroport, pour dire où travaille la personne et une clé étrangère personne/aéroport qui doit référencer cette table de vœux qui sera préalablement remplie.

Un exemple de contrainte d'exclusion, c'est celle qui concerne l'ordonnance qui ne doit pas contenir un médicament interdit.

## Exclusion

- Chaque ordonnance concerne un patient
  - pas de prescription dangereuse...

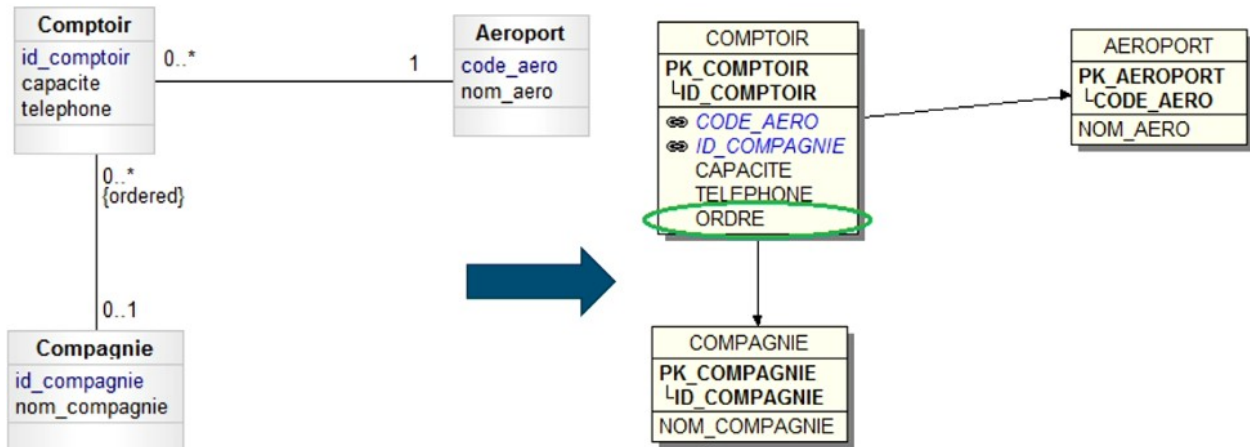


Alors, on avait déjà fait cette modélisation auparavant. Une ordonnance, c'est une liste de médicaments. Un patient dispose d'une liste de médicaments interdits. Et quand on dérive ce modèle, on obtient plusieurs tables d'association. La table d'association proscrit, qui va lister les médicaments interdits pour les patients. Et puis, la table d'association qui va dire quelle ordonnance concerne quels médicaments. La contrainte ne se voit pas aussi directement que tout à l'heure. Néanmoins, il faudra s'assurer que par une jointure avec la table ordonnance, on récupérera le numéro du patient, Et le couple médicament/patient récupéré par jointure ne devra pas se

trouver dans la table proscrits. Ici, le modèle n'assure pas cette contrainte mais le permet par programmation.

## Odonnancer

- Chaque compagnie peut classer chacun de ses comptoirs

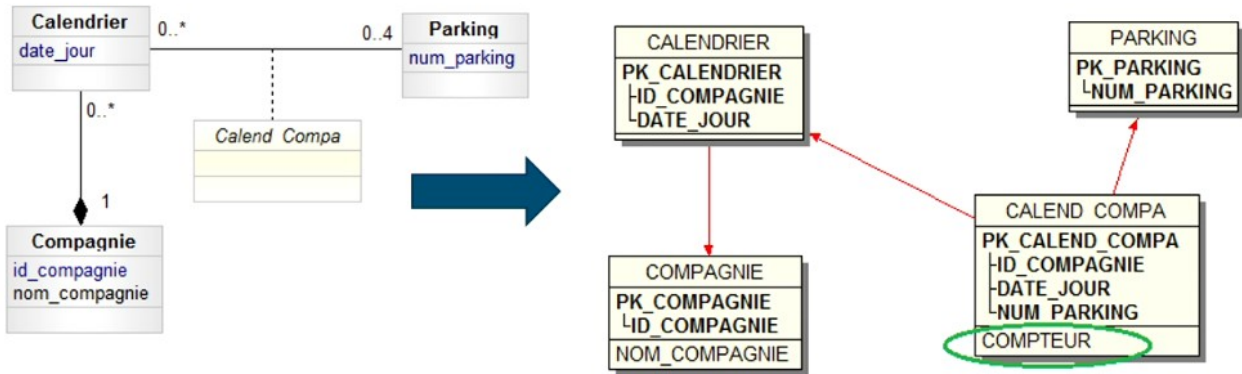


LinkedIn

Concernant l'ordonnancement, en général, il suffit d'ajouter un compteur bien placé dans la bonne table pour répondre aux besoins de vouloir ordonnancement des objets reliés. Ici, chaque compagnie a envie de classer ses comptoirs. Eh bien, le fait d'ajouter une colonne ordre va nous permettre pour chaque comptoir d'associer un entier. Bien sûr, par vérification, il faudra s'assurer que ce numéro est un numéro distinct. Cette notion de distinct sera vue dans le chapitre consacré à SQL à l'aide des contraintescheck.

# Dénombrer

- Chaque compagnie peut réserver 4 parkings au maximum par jour



LinkedIn

Alors, le principe d'ajouter une colonne compteur sert aussi dans tout ce qui est obligation de dénombrer des objets. Ici, si on veut désirer programmer la contrainte que chaque compagnie ne puisse pas réserver plus de quatre parkings par jour. On va ajouter un compteur dans chaque couple. Et puis, ce compteur, il faudra s'assurer qu'il ne dépasse pas quatre. Et également, il faudra ajouter une autre contrainte. On verra ça dans la partie avec SQL.

### 3.1.7. Faire le bilan du modèle relationnel

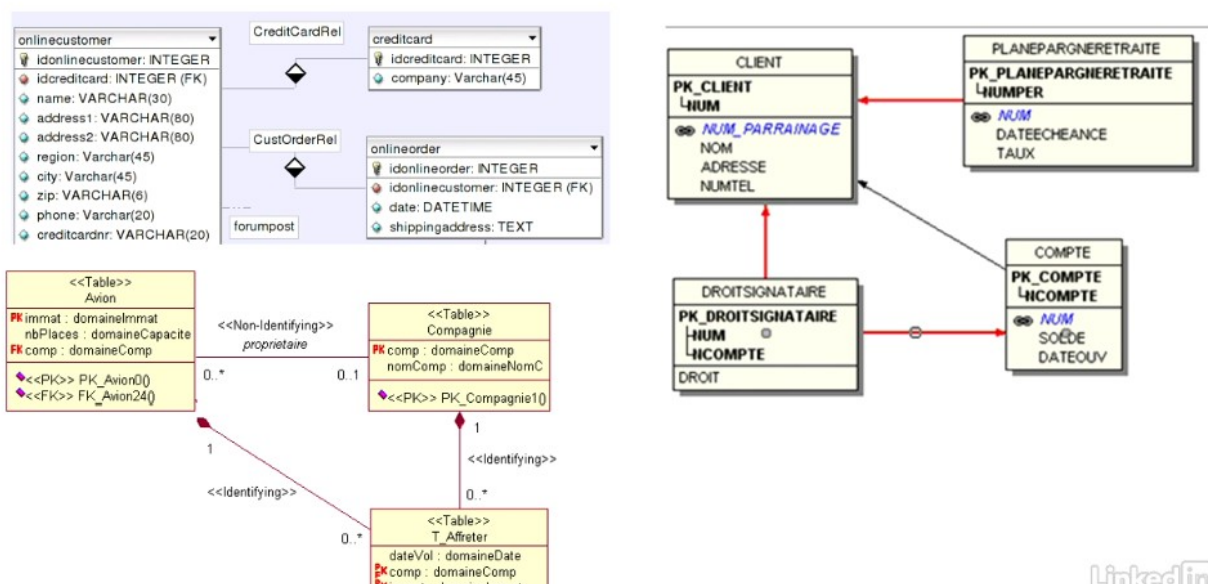
Alors, qu'est-ce qu'on peut faire comme bilan en termes du modèle relationnel ? C'est d'abord qu'il est **indépendant de la base de données**, quelle que soit la base sur laquelle vous avez travaillé. Eh bien, vous vous déchargez de cette contrainte.

Vous faites une **conception en amont qui est indépendante**.

Vous ne pouvez pas vous passer de ce niveau avant de créer vos tables.

Il est intéressant ce niveau parce qu'il permet de **vérifier votre modèle une dernière fois. Renommer des colonnes, ajouter des tables, ajouter des clés, retyper des colonnes**. Et aussi, il va servir à **faire une prévision sur le volume de données qui sera nécessaire pour stocker vos tables**.

## Notation des outils du marché



Les outils ont chacun leur propre formalisme graphique pour représenter les tables, les colonnes et des clés étrangères.

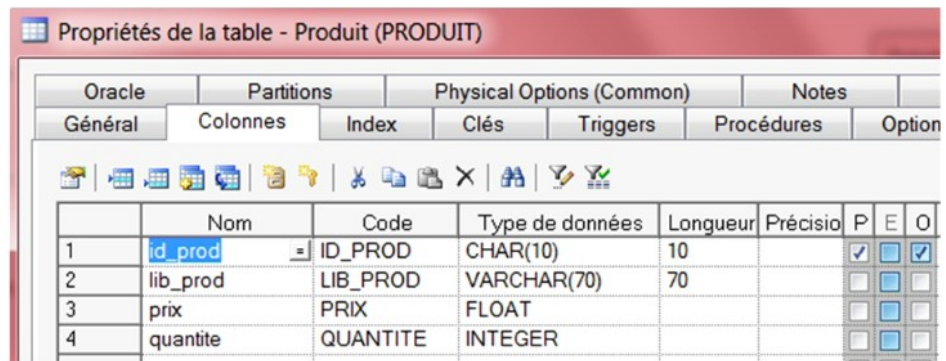
Mais on retrouve toujours les **constantes** : l'**identifiant**, le **lien**, la clé **étrangère**, qui est dénotée bien souvent FK, ici FK. Ici, elle n'est pas dénotée mais elle est soulignée par un petit pictogramme et puis, indiquée dans une couleur différente. Et FK, par le



pictogramme, ici rouge. Donc tous ces modèles sont analogues. L'important c'est de constater la structure des tables et surtout, le sens des clés étrangères.

## Typier les colonnes

- Précision (exactitude)
- Actualité
- Compréhension
- Confiance



Propriétés de la table - Produit (PRODUIT)						
Oracle		Partitions		Physical Options (Common)		Notes
Général	Colonnes	Index	Clés	Triggers	Procédures	Options
	Nom	Code	Type de données	Longueur	Précision	P E O
1	id_prod	ID_PROD	CHAR(10)	10		<input checked="" type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/>
2	lib_prod	LIB_PROD	VARCHAR(70)	70		<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
3	prix	PRIX	FLOAT			<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
4	quantite	QUANTITE	INTEGER			<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

LinkedIn

Bien sûr, typer chaque colonne sera une étape importante. J'en ai déjà parlé. Pour préciser le type de la colonne, sa taille. Est-ce que cette colonne est obligatoire ou pas ? Est-ce que cette colonne peut avoir des décimales ou pas ? Est-ce que cette colonne fait partie de la clé primaire ? Toutes ces caractéristiques sont importantes pour composer un modèle relationnel de qualité. Il y a plusieurs critères qu'il est important de vérifier. La précision, vous ne devez pas avoir des données qui vont entraîner des calculs erronés. Je parle surtout de décimales ou des domaines de valeurs. Eviter le plus possible les valeurs et marqueurs null. L'actualité, c'est quand il est intéressant de remettre à jour ses données. Est-ce que ces données ont une espérance de vie longue ? Le critère de compréhension concerne l'absence d'ambiguïté sur le sens d'une colonne. Si c'est un prix, est-ce que ce prix est hors taxes ou TTC ? Si c'est une longueur, est-ce que cette longueur est en mètre ou en cm ? Peut-être qu'il faudra ajouter une colonne unité. Enfin, le critère de confiance ça dépend toujours de la provenance de la saisie de vos données.

# Calcul prévisionnel d'une volumétrie

- C'est à vous de prévoir le nombre de lignes
- Chaque table d'association est à analyser à part et il ne faut pas faire de multiplication simple

The image shows three screenshots of a database management tool's 'Properties of the table' dialog box. Each dialog has tabs for 'Partitions', 'Physical Options (Common)', and 'Notes'. The 'General' tab is selected in each.

- Pilotes (PILOTES):** Nom: Pilotes, Code: PILOTES, Nombre: 3500, Type dimensionnel: <Aucun>, Type: Relational.
- Type\_Avion (TYPE\_AVION):** Nom: Type\_Avion, Code: TYPE\_AVION, Nombre: 150, Type dimensionnel: <Aucun>, Type: Relational.
- Aptitude (APTITUDE):** Nom: Aptitude, Code: APTITUDE, Nombre: (empty), Type dimensionnel: <Aucun>, Type: Relational.

Between the first and second dialog is a large 'X' symbol, and between the second and third is an '=' symbol, indicating a multiplication operation.

Concernant la volumétrie prévisionnelle, il faudra que pour chaque table vous soyez en mesure d'estimer la longueur de chaque ligne. Avec le type de la colonne, vous devez savoir le faire. Si c'est une chaîne de caractères, sur n-caractères, souvent ça prend n-octets. Les entiers, ça varie de 2 à 8 octets. Les chiffres décimaux, ça varie de 4 à 2 octets. Les dates, c'est aux alentours de 8 octets. Ça sera à vous d'étudier la documentation officielle de la base de données que vous allez utiliser. Certains outils disposent d'aide et il suffit juste d'estimer un nombre de lignes prévisionnel pour ensuite, en déduire un nombre de lignes total si des tables d'association sont à mettre en œuvre. Dans l'exemple présenté, si je suppose que dans ma compagnie j'ai regroupe 150 avions différents, si je fais la multiplication de ces 2 chiffres, j'obtiens toutes les combinaisons qu'il est possible de traiter. Ce n'est peut être pas ce chiffre qu'il faudra retenir, bien sûr. Et si en moyenne un pilote est qualifié sur 5 ou 6 appareils, il faudra multiplier le nombre de pilotes par 5 ou 6 au lieu du nombre total d'appareils pour obtenir le volume de la table d'association entre pilote et type d'avions. Et à ce chiffre final, en faisant ainsi pour toutes les tables, en multipliant par un coefficient entre 2 et 3, on obtient un volume à peu près prévisionnel aux données à venir. Par la suite, il faudra ajouter aussi la taille des index qui n'est pas négligeable.



# Pour conclure avec le relationnel

- Conditionne la structure des tables à venir
- N'oubliez pas le caractère obligatoire (NOT NULL) et les clés métiers (UNIQUE)
- Le processus est automatisé, mais c'est à surveiller
- Les contraintes seront programmées avec SQL
- Le graphisme du modèle aide à l'écriture des requêtes (jointures)
- Base de discussion pour les différents acteurs



Alors pour en conclure avec ce niveau relationnel, ce niveau relationnel il va conditionner complètement la structure des tables que vous aurez dans la base de données. N'oubliez pas de disposer le caractère obligatoire et des clés métier avec des contraintes uniques sur vos tables de références. Beaucoup d'outils automatisent ce processus de passage d'un modèle conceptuel à un modèle logique. Vous avez vu les règles à appliquer, ce sera à vous de surveiller que votre outil fonctionne correctement. Les contraintes simples peuvent être mises en place mais la majorité des contraintes sont implémentées avec SQL et c'est ce que l'on va voir un peu plus tard.