Cours sur l'algorithmique, la POO, MERISE et SQL

14/07/2023
Baptiste MEGEVAND

Table des matières

I- Algorithmique

Les variables sont des emplacements mémoire sur le système, définies par le développeur et elles permettent de stocker et utilisé des données.

- Les variables "classiques", les types les plus utilisés que vous retrouverez également en base de données :
    - Integer : les entiers donc tous les chiffres sans virgules (il possède également une limite de grandeur)
    - String (VarChar) : tableau de charactère permettant de composer des textes
    - Float (Decimal, Long) : emplacement mémoire plus important que les entiers cela permet donc de stocker des chiffres à virgules ou très grands (BigInt peut être utilisé également pour ce cas)
    - Boolean : variable binaire soit True (1) soit False (0)
- Les tableaux :
    - De données, tableaux simples dans lequel nous pouvons stocker des données ex : i = [1,2], il faut utiliser un index pour accéder aux valeurs (i[0] -> 1), ils peuvent-être imbriqués
    - Clés/valeurs, à utiliser quand les clés sont ou seront connus, ces tableaux permettent d'utiliser n'importe quelle valeur même d'autres tableaux, en revanche il est recommandé d'utiliser des valeurs simples suivant les cas, il est possible de les imbriquer. Attention : suivant les langages certain type ne seront pas supporté par le format JSON

Les méthodes, sont des définitions de code qui peuvent-être appelé par le programme principal. Elles servent à ne pas dupliquer le code mais également à isolés les logiques algorithmiques afin de rendre les modifications (autant techniques que métiers) plus simple et moins impactante dans le programme principal. Il existe deux types de méthodes :

- Les procédures, pas de retour attendu à part (1 j'ai fonctionné, 0 je n'ai pas fonctionné) elles vont souvent être utilisé pour traiter les données, « les procédures sont des fonctions sans retours ».
- Les fonctions, les fonctions retourne obligatoirement une donnée, ça peut-être un paramètre d'entrée retravaillé, ou une donnée totalement différente, « les fonctions sont des procédures avec retours ».

Les méthodes peuvent récupérer des paramètres d'entrée, ce sont des variables dont nous ne connaissons pas la valeur mais nous connaissons le type et l'utilisation ce qui permet de tout de même développer le comportement attendu. Les paramètres d'entrée peuvent prendre des valeurs par défaut afin de faciliter les développements, d'assurer de la non-régression, éviter de fausses erreurs etc…
Les boucles, elles permettent aux développeurs d'apporter de la logique algorithmique :

- If : Si Alors, si les conditions sont remplies alors nous codons un comportement
- For i : boucle permettant principalement de parcourir les tableaux simple, i devant commencer à 0 et il faut ajouter 1 à chaque itération et il faudra prévoir la sortie (souvent indexé sur la taille du tableau - 1)
- Foreach : boucle permettant de parcourir tous types tableaux, la boucle s'arrête d'elle-même à la fin du tableau (attention dans les clés valeurs, il faudra préparer la récupération des deux variables)
- While do : tant que la condition dans while est vrai le code boucle
- Do while : execute le code puis boucle tant que la condition while est vraie

Les exceptions, elles permettent la gestion des erreurs avec les mots clés :

- Try, permet de coder à l'intérieur le code qui aura l'erreur connue
- Catch, permet d'attraper les erreurs et d'ajouter un cas particulier en fonction de ces dernières
- Finally, ajout d'un comportement qui sera réalisé après le try réussis ou après le catch

Attention, chaque clause try duplique l'entièreté du code à l'intérieur du try, afin de lever l'erreur sans arrêter le langage, il faut donc être extrêmement minutieux à l'utilisation de cette fonctionnalité et le faire au plus "bas" possible dans votre code, ex : réaliser le try catch dans une partie de la fonction plutôt que d'entourer cette même fonction du try catch.

II - Programmation Orientée Objet (POO)

La programmation orientée objet (POO) est une méthode de programmation qui permet de structurer le code de manière plus efficace et flexible. Au lieu de structurer le code autour de procédures et de fonctions, comme c'est le cas dans la programmation procédurale, la POO structure le code autour d'entités appelées objets. La POO permet également de donner une cohérence métier aux programmes informatique.
Un objet découle d'une une classe. Une classe peut être considérée comme un modèle ou un plan à partir duquel des objets sont créés. Les objets représentent des entités du monde réel et ont des attributs (propriétés) et des méthodes (fonctions et procédures).

La POO embarque donc plusieurs concepts :

- L'objet : L'objet est une entité qui possède un état et un comportement. Par exemple, une voiture est un objet qui a des propriétés comme la couleur, la marque et le modèle, et des comportements comme démarrer, freiner et accélérer.
- Le constructeur : Le constructeur est une méthode spéciale d'une classe qui est appelée automatiquement lorsque vous créez un nouvel objet de cette classe. Il permet d'initialiser l'objet et d'attribuer les valeurs initiales aux attributs de l'objet. Dans la plupart des langages de programmation, le constructeur a le même nom que la classe.
- L'accessibilité (Getter et Setter) : Les getters et setters sont des méthodes utilisées pour contrôler l'accès aux attributs d'un objet. Les getters sont utilisés pour obtenir la valeur d'un attribut, tandis que les setters sont utilisés pour définir ou modifier la valeur d'un attribut. Ils permettent de contrôler comment les attributs sont accessibles et modifiables.

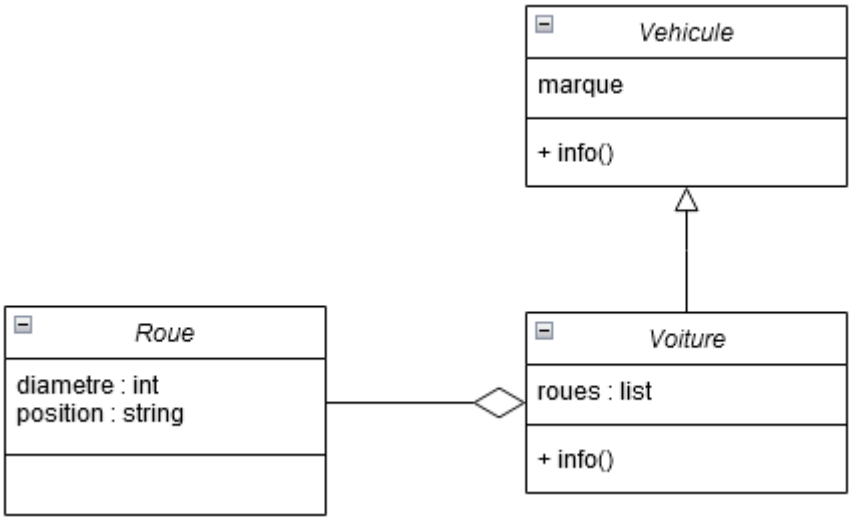
Il est possible de créer des liens entre les classes, que nous pouvons résumer avec les concepts d'agrégation et de composition.

- L'Agrégation : L'agrégation est une relation entre deux classes où une classe peut exister indépendamment de l'autre. Par exemple, une classe Voiture et une classe Roue ont une relation d'agrégation : une voiture a des roues, mais une roue peut exister sans la voiture.
- La Composition : La composition est une relation entre deux classes où une classe ne peut pas exister indépendamment de l'autre. Par exemple, une classe Moteur et une classe Voiture ont une relation de composition : un moteur fait partie intégrante d'une voiture, et ne peut pas exister indépendamment d'elle.

En ce qui concerne l'importation d'objets, il existe deux cas principaux à considérer : l'import unique d'un objet et la collection d'objets en cas d'import multiple.

- L'import unique est utilisé lorsqu'il y a besoin d'une seule instance d'un objet dans votre code. Par exemple, une classe Voiture importera qu'une seul classe Moteur.
- L'import multiple est utilisé lorsqu'il y a besoin de plusieurs instances d'un objet dans votre code. Par exemple, une classe Voiture importera plusieurs classes Roue, ces classes seront stockés dans un tableau (simple) d'objets.

L'héritage, est un des principes fondamentaux de la programmation orientée objet. Il permet à une classe d'hériter des attributs et des méthodes d'une autre classe. La classe qui hérite est appelée classe enfant ou sous-classe, et la classe dont elle hérite est appelée classe parent ou super-classe. L'héritage permet d'éviter la redondance du code en permettant aux classes enfants de réutiliser le code de leur classe parent, tout en ayant la possibilité de modifier ou d'ajouter des comportements spécifiques. Avec l'héritage vient également le concept de surcharge, c'est la possibilité de modifier une fonction existante dans la classe parente afin d'ajouter un comportement supplémentaire à celui d'origine. L'héritage facilite également la maintenance et l'évolution du code, car les modifications apportées à une classe parent se répercutent automatiquement sur les classes enfants.



L'implémentation d'un objet est le processus de création d'une instance d'une classe. Pour implémenter un objet, vous devez d'abord écrire une classe, puis créer une instance de cette classe après l'avoir importé dans fichier de lancement ou code principal. Cette instance devient alors un objet qui possède les attributs et les méthodes que vous lui avez définis.

III - Les bases de données

Nous allons aborder le processus global de modélisation des données avec MERISE et sa mise en œuvre avec SQL, ce comprend donc les étapes suivantes :

1. Définition des entités, des relations et des cardinalités avec MERISE.
2. Conversion de ces éléments en tables, colonnes et clés étrangères dans une base de données SQL.
3. Création de la structure de la base de données avec les commandes SQL CREATE et ALTER.
4. Insertion, mise à jour, sélection et suppression des données avec les commandes SQL INSERT, UPDATE, SELECT et DELETE.
5. Utilisation de procédures, de fonctions et de vues pour automatiser les opérations et simplifier l'accès aux données.

En maîtrisant ces concepts et ces étapes que nous allons aborder ci-dessous, vous serez en mesure de créer et de gérer efficacement une base de données relationnelle.

1 - MERISE

Merise est une méthode de modélisation des systèmes d'information très utilisée, notamment en France. Elle s'articule autour de plusieurs modèles, dont le Modèle Conceptuel de Données (MCD). Le MCD est une représentation graphique et conceptuelle de la structure des données d'une application ou d'un système d'information.

Dans un MCD, on distingue trois types d'objets : les entités, les relations et les cardinalités.

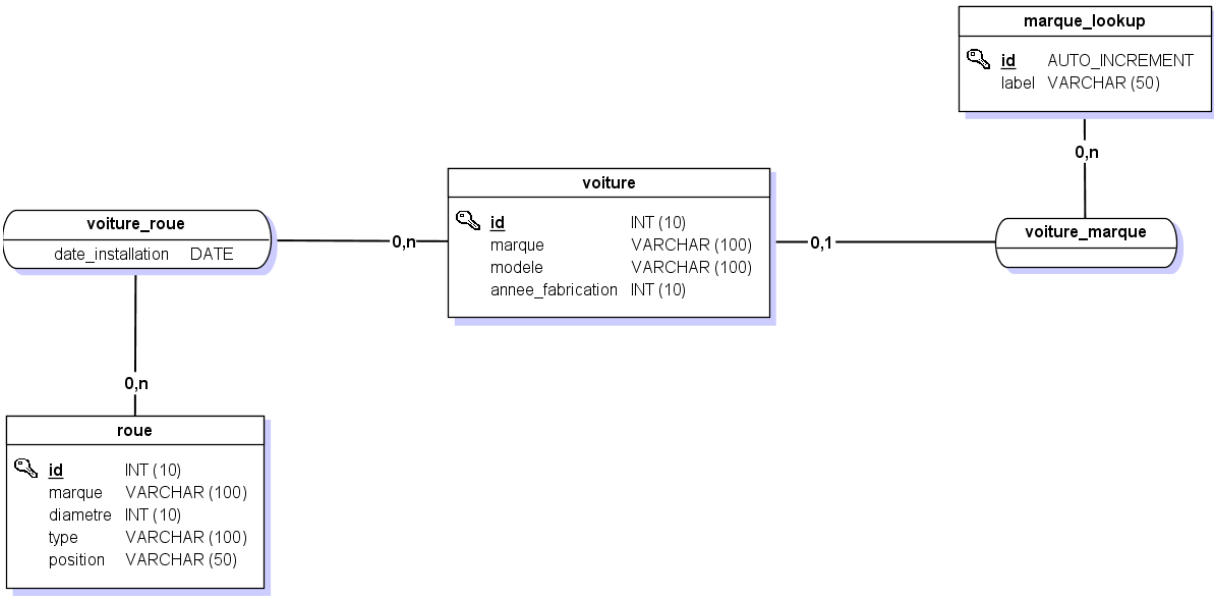
- Entité : Une entité représente un concept ou un objet du monde réel qui sera traité par le système d'information. Par exemple, dans une base de données pour une bibliothèque, les entités pourraient être 'Livre', 'Auteur', 'Adhérent', etc. Chaque entité a des attributs qui la décrivent. Par exemple, pour l'entité 'Livre', les attributs pourraient être 'titre', 'année de publication', 'genre', etc.
- Relation : Une relation est un lien logique qui existe entre deux entités. Par exemple, entre l'entité 'Livre' et l'entité 'Auteur', il pourrait y avoir une relation 'écrit par'.
- Cardinalité : La cardinalité décrit le nombre d'occurrences d'une entité qui peuvent être associées à une occurrence d'une autre entité via une relation. Par exemple, une cardinalité '1,N' entre Voiture et Roue signifie qu'une voiture peut avoir plusieurs roues, mais qu'une roue n'aura qu'une seul voiture durant son cycle d'existence.

Il existe plusieurs types de cardinalités, mais peu importe le type le premier chiffre annoncera toujours si le lien entre les deux objets est requis (1) ou non (0). Le deuxième chiffre annonce le nombre de fois qu'une entité se retrouvera en lien avec une autre, 1 indique une seule, n plusieurs. Voici donc les quatre types de cardinalités :

- 1,1 : Une occurrence d'une entité est associée à une et une seule occurrence d'une autre entité.
- 0,1 : Une occurrence d'une entité peut être associée à aucune ou une seule occurrence d'une autre entité.
- 1,N : Une occurrence d'une entité peut être associée à une ou plusieurs occurrences d'une autre entité.
- 0,N : Une occurrence d'une entité peut être associée à aucune, une ou plusieurs occurrences d'une autre entité.

Les attributs dans un MCD jouent un rôle très important dans la conception de votre base de données et définissent les caractéristiques d'une entité. Chaque entité dans le MCD est définie par un ensemble d'attributs. Ces attributs deviennent ensuite les colonnes de la table lors de la transcription vers le MLD.

- NULL / NOT NULL : Lors de la création d'une table, chaque attribut doit avoir une propriété NULL ou NOT NULL. Si un attribut est déclaré comme NOT NULL, cela signifie qu'il doit toujours avoir une valeur, c'est-à-dire qu'il ne peut pas être laissé vide lors de l'insertion ou la mise à jour de données dans la table. En revanche, si l'attribut est déclaré comme NULL, il peut ne pas avoir de valeur.
- PRIMARY KEY : Une clé primaire est un attribut ou un ensemble d'attributs qui identifie de manière unique chaque ligne dans une table. Chaque entité doit avoir une clé primaire et cette clé primaire ne peut pas être NULL, car elle est utilisée pour identifier de manière unique chaque enregistrement.
- UNIQUE : La contrainte UNIQUE est une contrainte qui garantit que toutes les valeurs dans une colonne sont différentes. Cela signifie que vous ne pouvez pas insérer deux enregistrements dans la table avec la même valeur pour cette colonne. Il est possible d'avoir plusieurs contraintes UNIQUE dans une table, mais une seule clé primaire.
- INDEX : Un index est utilisé pour accélérer les requêtes dans une base de données. En d'autres termes, il aide la base de données à retrouver les données plus rapidement. Les index sont particulièrement utiles lorsque vous interrogez de grandes quantités de données. Cependant, il faut noter que les index nécessitent de l'espace de stockage et peuvent ralentir les opérations d'insertion, de mise à jour et de suppression. Par conséquent, il est nécessaire de trouver un bon équilibre entre le nombre d'index et le type de requêtes que vous exécutez sur votre base de données.



Le MCD doit ensuite être transcrit en Modèle Logique de Données (MLD) qui est plus proche de la structure des bases de données relationnelles. Dans le MLD, les entités deviennent des tables, les attributs deviennent des colonnes dans ces tables et les relations sont représentées par des clés étrangères. Pour traduire le MCD en MLD il faut suivre les règles suivantes :

- Relation 0,1,1 : Dans une relation 1,1 entre deux entités A et B, l'entité qui est la plus souvent dépendante contiendra la clé primaire de l'autre en tant que clé étrangère. En pratique, ce choix peut être arbitraire ou dépendre du contexte.
- Relation 0,N – 1,1 ou 1,N – 1,1 : Dans ce cas, l'entité du côté le plus faible "1" contient la clé primaire de l'entité du côté "N" comme clé étrangère.

- Relation 0,N – 0,N ou 1,N – 0,N : Une table intermédiaire est créée pour gérer cette relation. Cette table contiendra les clés primaires des deux entités en tant que clés étrangères et sa clé primaire sera l'ensemble de ces clés étrangères.

2 - SQL, Structured Query Language

Le SQL (Structured Query Language) est un langage standardisé utilisé pour manipuler les bases de données relationnelles. Il se divise en deux grands axes : la manipulation de la structure de la base de données (DDL pour Data Definition Language) et la manipulation des données elles-mêmes (DML pour Data Manipulation Language).

Les fonctions les plus importantes de la modification de la structure de la base de données (DDL) :

- CREATE : La commande CREATE est utilisée pour créer de nouveaux éléments dans la base de données, comme une table ou une base de données. Par exemple, "CREATE DATABASE Voiture" créera une nouvelle base de données nommée "Voiture".
- ALTER : La commande ALTER est utilisée pour modifier la structure d'une table existante, elle permet par exemple d'ajouter une colonne, de la supprimer ou de modifier son type. Par exemple, "ALTER TABLE Voiture ADD annee_fabrication INT" ajoutera une nouvelle colonne de type entier à la table "Voiture".

Une fois que la structure de la base de données est définie, vous pouvez commencer à travailler avec les données elles-mêmes. Pour ce faire, il faut utiliser les fonctions les plus importantes de la manipulation des données (DML) :

- INSERT : Cette commande est utilisée pour insérer de nouvelles données dans une table. Par exemple, "INSERT INTO Voiture (marque, annee) VALUES ('Tesla', 1999)" ajoutera une nouvelle ligne à "Voiture".
- UPDATE : Cette commande est utilisée pour mettre à jour les données existantes dans une table. Par exemple, "UPDATE Voiture SET marque = 'WW' WHERE modele = 'POLO'" mettra à jour "marque" dans les lignes dont le modèle correspond à "POLO".
- DELETE : Cette commande est utilisée pour supprimer des données d'une table. Par exemple, "DELETE FROM Voiture WHERE marque ='Tesla'" supprimera toutes les lignes de "MaTable" qui répondent à "condition".

- SELECT : Cette commande est utilisée pour sélectionner et récupérer des données de la base de données. Par exemple, "SELECT * FROM Voiture WHERE marque = 'WW'" renverra toutes les colonnes des lignes de "Voiture" qui ont le modèle "POLO".

Le SELECT est un peu particulier parmi les requêtes SQL, en effet il permet beaucoup plus de chose qu'une simple sélection. Cette instruction peut être combinée avec plusieurs clauses, dont WHERE pour filtrer les données, GROUP BY pour regrouper les données, et HAVING pour filtrer après un regroupement. Il est également possible d'ajouter des fonctions d'agrégation afin de réaliser des calculs avant la sortie du SELECT.

La clause GROUP BY est utilisée pour regrouper des lignes qui ont des valeurs identiques dans des colonnes spécifiées. Elle est souvent utilisée avec des fonctions d'agrégation (COUNT, MAX, MIN, SUM, AVG) pour regrouper les données en fonction des valeurs de certaines colonnes et appliquer une opération sur chaque groupe. Par exemple, si vous voulez compter le nombre de voitures par marque, vous pouvez utiliser GROUP BY comme suit :

```
SELECT marque, COUNT(*) AS nb_voiture
FROM voiture, marque_lookup AS marque
WHERE marque_lookup.id = voiture.id_marque_lookup
GROUP BY marque
```

La clause HAVING est souvent utilisée avec GROUP BY pour filtrer les groupes de données. Contrairement à la clause WHERE qui filtre les lignes de données avant le regroupement, HAVING filtre les groupes de données après le regroupement. Supposons que vous ne voulez afficher que les marques qui ont plus de 10 voitures. Vous pouvez ajouter une clause HAVING à la requête précédente :

```
SELECT marque, COUNT(*) AS nb_voiture
FROM voiture, marque_lookup AS marque
WHERE marque_lookup.id = voiture.id_marque_lookup
GROUP BY marque
HAVING nb_voiture > 10;
```

Les fonctions d'agrégation sont utilisées pour effectuer un calcul sur un ensemble de valeurs.

- COUNT : Compte le nombre d'éléments dans un ensemble. Par exemple, COUNT(*) compte le nombre total de lignes, tandis que COUNT(column) compte le nombre de lignes où column n'est pas NULL. Attention, si vous voulez compter toutes les occurrences d'un table il est préférable d'utiliser COUNT(id) car le caractère * demande d'abord la sortie de tous les champs avant de réaliser le COUNT.
- SUM : Calcule la somme des valeurs. Par exemple, SUM(column) donne la somme des valeurs de column.
- AVG : Calcule la moyenne des valeurs. Par exemple, AVG(column) donne la moyenne des valeurs de column.
- MIN et MAX : Retourne respectivement la valeur minimum et maximum d'une colonne.

Il faut noter qu'un SELECT bien configuré vous permettra de gagner énormément de temps de réponse surtout sur des base remplies de plusieurs millions de lignes.

Exemple : Si vous voulez connaître la moyenne de diamètre des roues de chaque marque de voiture, vous pouvez utiliser AVG comme suit :

```
SELECT ml.label AS marque, AVG(r.diametre) AS moy_diametre
FROM voiture AS v
JOIN voiture_roue AS vr ON v.id = vr.id
JOIN roue AS r ON vr.id_roue = r.id
JOIN marque_lookup ml ON v.id_marque_lookup = ml.id
GROUP BY ml.label;
```

De plus, Il est possible d'automatiser des comportements sur les bases données, avec des concepts sur le SGBDR que ressemblent à ceux que nous pouvons trouver dans les langages de programmation, ce sont les suivants :

- Les procédures : Une procédure est un ensemble de commandes SQL qui sont stockées et exécutées sur le serveur. Vous pouvez les utiliser pour encapsuler une logique complexe et la réutiliser. Il est également possible d'ajouter des triggers afin de les déclencher par exemple avant l'insertion d'une table.
- Les fonctions : Une fonction est similaire à une procédure, mais elle renvoie une valeur. Vous pouvez les utiliser pour créer des opérations complexes qui sont utilisées fréquemment dans les procédures ou les vues.
- Les vues : Une vue est une table virtuelle basée sur le résultat d'une instruction SELECT. Une vue contient des lignes et des colonnes, tout comme une vraie table. La différence est qu'une vue n'existe pas physiquement, elle est définie par une requête qui est exécutée lors de son utilisation. Les vues sont utilisées pour simplifier les requêtes complexes, pour sécuriser les données en limitant l'accès à certaines colonnes ou lignes, ou pour restructurer les données. Il est également possible de dénormaliser une partie de vos données afin de faciliter par exemple les recherches par mots clés.

Une base de données SQL est organisée en tables. Une table est un ensemble d'informations et est structurée en colonnes et en lignes. Chaque colonne correspond à un type de donnée spécifique et chaque ligne (ou enregistrement) contient un ensemble de valeurs pour ces colonnes.

La puissance du SQL vient en grande partie de son typage fort. Chaque colonne d'une table a un type de données spécifique, ce qui garantit que toutes les données stockées dans cette colonne respectent un certain format. Les types de données courants comprennent INT pour les nombres entiers, VARCHAR pour les chaînes de caractères, DATE pour les dates, etc.

Le typage fort garantit l'intégrité des données en s'assurant que les données stockées dans la base de données respectent un format prédéfini. Par exemple, vous ne pouvez pas stocker une chaîne de caractères dans une colonne de type INT. Cela protège également la base de données contre certains types d'erreurs qui peuvent survenir lors de l'exécution de requêtes SQL.

Le typage fort peut aider à optimiser les performances de la base de données. Par exemple, les données numériques sont généralement plus rapides à analyser et à trier que les données textuelles. Ainsi, le fait d'utiliser le type de données approprié pour chaque colonne peut contribuer à améliorer l'efficacité de vos requêtes SQL. Cette optimisation poussée à l'extrême découle sur le concept de table de correspondance (ou lookup_table) que nous avons abordé pendant la présentation.

Une clé étrangère est un concept essentiel des bases de données relationnelles. Il s'agit d'un attribut (ou un ensemble d'attributs) d'une table qui fait référence à la clé primaire d'une autre table. Le but de ce concept est de relier les tables entre elles en récupérant l'id de la cardinalité la plus faible, si

une table doit récupérer plusieurs fois une autre table, la clé ne peut pas être placée sur la table de récupération table.

La clé étrangère dans la table de récupération correspond à la clé primaire de la table référence. Cette correspondance est ce qui crée la relation entre les deux tables. L'intégrité référentielle, qui est un concept important en SQL. En d'autres termes, pour chaque enregistrement dans la table de récupération, il doit y avoir un enregistrement correspondant dans la table de référence.

Les clés étrangères sont particulièrement utiles pour modéliser des relations complexes entre les entités. Par exemple, elles peuvent être utilisées pour créer des relations "plusieurs à plusieurs". Dans ce cas, une table de liaison est utilisée pour créer une relation entre deux autres tables.

Reprenons notre exemple de voiture, disons qu'une voiture a toujours exactement quatre roues. Pourtant, chaque roue pourrait être utilisée sur différentes voitures au fil du temps (par exemple, lorsque les roues sont changées ou réutilisées). Nous avons donc une relation "plusieurs à plusieurs" entre les Voiture et les Roue.

Voici comment vous pourriez créer ces tables en SQL :

```
CREATE TABLE marque_lookup (
   id INT PRIMARY KEY,
   label VARCHAR(50) NOT NULL
);

CREATE TABLE roue (
   id INT PRIMARY KEY,
   marque VARCHAR(100),
   diametre INT,
   type VARCHAR(100),
   position VARCHAR(50) NOT NULL
);

CREATE TABLE voiture (
   id INT PRIMARY KEY,
   marque VARCHAR(100),
   modele VARCHAR(100),
   annee_fabrication INT,
   id_marque_lookup INT,
   FOREIGN KEY (id_marque_lookup) REFERENCES marque_lookup(id)
);

CREATE TABLE voiture_roue (
   id_voiture INT,
   id_roue INT,
   date_installation DATE,
   /*PRIMARY KEY (id_voiture, id_roue, date_installation),*/
   PRIMARY KEY (id, id_roue),
   FOREIGN KEY (id_roue) REFERENCES roue(id),
   FOREIGN KEY (id) REFERENCES voiture(id)
);
```

Dans cet exemple, id_voiture et id_roue dans la table voiture_roue sont des clés étrangères qui se réfèrent respectivement à la clé primaire des tables Voiture et Roue. Cela signifie que pour chaque installation d'une roue sur une voiture (un enregistrement dans la table voiture_roue), il doit y avoir une voiture correspondante dans la table Voitures et une roue correspondante dans la table Roues. De plus, en inversant la PRIMARY KEY commenté, la combinaison de id_voiture, id_roue et date_installation peut être définie comme la clé primaire dans voiture_roue, cela signifie qu'une roue spécifique ne peut être installée sur une voiture spécifique qu'une seule fois à une date donnée. Bien évidement cette même roue pourra être réutilisé dans une autre installation car la table Roue est indépendante, c'est bien l'installation d'une sur une voiture qui est unique dans ce cas. Pour nos exemples, nous utiliserons seulement la stratégie de clé primaire étrangère classique afin d'alléger les contraintes sur notre base d'exemple.

Lors de la création de bases de données relationnelles, les données sont généralement réparties entre plusieurs tables pour éviter la redondance et pour optimiser la gestion des données. Cependant, pour extraire des informations significatives de ces données, nous avons souvent besoin de les rassembler. Pour ce faire, SQL propose deux méthodes de jointures : l'utilisation des instructions JOIN ou l'utilisation des conditions dans la clause WHERE.

Les instructions JOIN sont utilisées pour combiner des lignes de deux ou plusieurs tables, en se basant sur une relation entre certaines colonnes dans ces tables :

- INNER JOIN produit une table de résultat pour les valeurs combinées de toutes les tables lorsqu'il y a une correspondance dans TOUS les modèles indiqués dans la condition JOIN.
- LEFT JOIN renvoie toutes les lignes de la table de gauche (la première table) et les lignes correspondantes de la table de droite (la deuxième table). Si aucune correspondance n'est trouvée, le résultat est NULL du côté droit.
- RIGHT JOIN est l'inverse de LEFT JOIN. Elle renvoie toutes les lignes de la table de droite et les lignes correspondantes de la table de gauche. Si aucune correspondance n'est trouvée, le résultat est NULL du côté gauche.
- FULL OUTER JOIN renvoie toutes les lignes pour lesquelles il y a une correspondance dans l'une des tables. Si aucune correspondance n'est trouvée, le résultat est NULL des deux côtés.

La clause WHERE est également utilisable pour joindre les tables. L'utilisation de WHERE pour joindre des tables génère un produit cartésien entre les tables, ce qui signifie que chaque ligne d'une table est combinée avec chaque ligne de l'autre table. Ensuite, les conditions de la clause WHERE sont utilisées pour filtrer les résultats. A noter, le type de jointure que vous devez utiliser dépend fortement du contexte métier, et de la sortie de données que vous attendez.

Quelques exemples de jointures :

- Si nous voulons obtenir toutes les voitures et les détails correspondants des roues, nous pouvons utiliser INNER JOIN :

```
SELECT voiture.id, voiture.marque, roue.diametre
FROM voiture
INNER JOIN voiture_roue ON voiture.id = voiture_roue.id_voiture
INNER JOIN Roue ON voiture_roue.id_roue = roue.id;
```

- Exemple ci-dessus avec la clause WHERE :

```
SELECT voiture.id, voiture.marque, roue.diametre
FROM voiture, voiture_Roue, roue
WHERE voiture.id = voiture_roue.id_voiture
AND voiture_roue.id_roue = roue.id;
```

- Si nous voulons obtenir toutes les voitures et, s'ils existent, les détails de ses roues, nous pouvons utiliser LEFT JOIN :

```
SELECT voiture.id, voiture.marque, roue.diametre
FROM voiture
LEFT JOIN voiture_roue ON voiture.id = voiture_roue.id_voiture
LEFT JOIN roue ON voiture_roue.id_roue = roue.id;
```

Liens utiles :

UML : http://uml.free.fr/cours/i-p13.html
SQL : https://sql.sh/cours/select
SQL Jointures : https://sql.sh/cours/jointures