Cursus JAVA

M2I Formations

Jean-Christophe DOMINGUEZ

Formation

MODULE SQL

1.

Base de données : les concepts

Persistance de la donnée

En programmation, la gestion de la persistance des données et parfois des états d'un programme réfère au mécanisme responsable de la sauvegarde et de la restauration des données. Ces mécanismes font en sorte qu'un programme puisse se terminer sans que ses données et son état d'exécution ne soient perdus.

Ces informations de reprise peuvent être enregistrées sur disque, éventuellement sur un serveur distant (un serveur de bases de données relationnelles, par exemple).

Base de données

- **Ensemble structuré d'informations** (*d'une entreprise ou organisation*), **mémorisé** sur une machine (*serveur*).

- **Données stockées** et **organisées** sous forme de fichiers ou ensemble de fichiers.

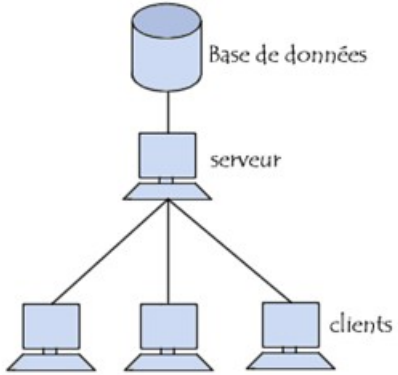
- Une BD sert à **créer**, **enregistrer**, **récupérer** et **manipuler** des **données communes**.

SGBD

- **Système de Gestion de Base de Données** (ou DBMS: Data Base Management System)

- **Ensemble cohérant** de **services** (*logiciels*) permettant aux utilisateurs **d'accéder**, **mettre à jour** ou **administrer** une DB

- Fonctionne sur le modèle **client/serveur** (requêtes/traitements)



SGBD

Pourquoi utiliser un SGBD ? Quels objectifs ?

- Indépendance physique
- Indépendance logique
- Accès / partage des données
- Administration centralisée
- Non redondance des données
- Cohérence des données
- Sécurité des données
- Résistance aux pannes

SGBD

Vocabulaire

	Relational Database	SQL	Other
Collection	Relation	Table	File
Instance	Tuple	Row	Record
Detail	Attribute	Column	Field

SGBD

- **Installation du SGBD**

2.

Base de données : modélisation

Modélisation

- **Pourquoi modéliser ?**
  - Avoir une représentation graphique de la structure
  - Connaître les propriétés attendues d'une données
  - Connaître les relations entre les données

- **Comment modéliser ?**
  - Effectuer un design conceptuel
  - Insérer des cardinalités
  - Effectuer un modèle logique

Modélisation : savoir extraire l'information

Lorsque vous allez recevoir un cahier des charges, il faudra tout d'abord réussir à en extraire les informations qui nécessitent un stockage persistant. Ces information seront regroupées dans un documents appelé le "**dictionnaire de données**".

Celui-ci se présente généralement sous forme d'un tableau comme suit :

Nom donnée	Type donnée	Référence	Commentaire
nom_client	TEXT	Client	Contient le nom des clients

L'objectif est d'obtenir le contenu de la structure de nos **tables** pour notre futur base de données.

Modélisation : la clef primaire

Lorsque vous allez stocker une information dans votre base de données, il est intéressant de mettre en place une information **unique** et **simple** qui permettra de retrouver cette donnée dans la base aisément. Cette information est appelé la **“clef primaire”.**

En général celle-ci est au format numérique et se trouve à l’intérieur d’une colonne nommée “**id**”.

id	prenom	nom
1	Chandler	Bing
2	Phoebe	Buffay
3	Monica	Geller
4	Ross	Geller
5	Chandler	Bing

Modélisation : les relations

Une relation a été créé entre deux tables, ce qui veut dire que leurs données sont désormais liées.

Il existe 3 types de relations :

- un à un (one-to-one)
- un à plusieurs (one-to-many) ou plusieurs à un (many-to-one)
- plusieurs à plusieurs (many-to-many)

Chacun de ces types engendre une conséquence différente sur le modèle de données.

Modélisation : les relations

Afin d'éviter les doublons de données il est possible de mettre en place des **relations** entre nos différentes **tables/entités**.

Par exemple :

id	nom	modele	annee_sortie
1	Jean Manchzeck	Kawasaki 750	2015
2	Edouard Bracame	Kawasaki 750	2015

Pourrait devenir :

id	nom
1	Jean Manchzeck
2	Edouard Bracame

0,1
0,1

id	modele	annee_sortie
1	Kawasaki 750	2015

Modélisation : les relations

One-to-One : Une relation one-to-one implique la création d'une **clé étrangère** dans l'une des deux tables. Cette clé représente la référence de la seconde table. Si on reprend l'exemple précédent :

id	nom
1	Jean Manchzeck
2	Edouard Bracame

id	modele	annee_sortie	conducteur
1	Kawasaki 750	2015	1
2	Kawasaki 750	2015	2

Modélisation : les relations

One-to-Many : Une relation one-to-many implique également la création d'une **clé étrangère** dans l'une des deux tables. Par contre dans ce cas nous n'avons pas le choix de la table qui portera la référence. Si on reprend l'exemple précédent :

id	nom
1	Jean Manchzeck
2	Edouard Bracame

0,1

n

id	modele	annee_sortie	conducteur
1	Kawasaki 750	2015	1
2	Kawasaki 750	2015	2

Modélisation : les relations

Many-to-Many : Une relation many-to-many engendrera la création d'une table de correspondance. Si on reprend l'exemple précédent :

id	nom
1	Jean Manchzeck
2	Edouard Bracame

conducteur	moto
1	1
2	2

id	modele	annee_sortie
1	Kawasaki 750	2015
2	Kawasaki 750	2015

Modélisation : les règles à respecter

- **Normalisation des tables**
  - Ne contient pas d'espace, d'accents ni de caractères spéciaux
  - Tout doit être écrit en lowercase (minuscule)
  - Les espaces sont remplacés par des underscores : "_"
- **Les relations**
  - En UML l'astuce est de lire les relations avec en les liants avec le verbe "possède" afin de ne pas se tromper dans l'ordre de celles-ci.
- **Un identifiant**
  - Un champs d'identification unique est obligatoire, la clef primaire (souvent nommé "id").

3.

Base de données : Le SQL

SQL: Les requêtes structurelle

Il existe plusieurs instructions possibles sur les **objets** de votre base :

- CREATE **t n** pour créer un objet de type **t** et de nom **n**
- ALTER **t n** pour modifier le schéma d'un objet de type **t** et de nom **n**
- DROP **t n** pour supprimer un objet de type **t** et de nom **n**

Exemple :

- **Création d'une base**
  - CREATE DATABASE <nom_colonne>;
- **Création d'une table**
  - CREATE TABLE <nom_table>(<nom_colonne> <type_colonne> <contraintes>, …);
- **Modification d'une table**
  - ALTER TABLE <nom_table> ALTER COLUMN <nom_colonne> TYPE <type_colonne> <contraintes>;

SQL: Les requêtes (CRUD)

Il existe plusieurs instructions possibles sur les **données** de votre base :

- INSERT pour ajouter des lignes à une table
- UPDATE pour modifier des lignes d'une table
- DELETE pour supprimer des lignes d'une table
- SELECT pour extraire des données à partir de tables existantes

Exemple :

- **Lire des données**
  - SELECT <nom_colonne> FROM <nom_table>
- **Ajouter des données**
  - INSERT INTO <nom_table> VALUES (‘valeur1’, ‘valeur2’, ….)
- **Modifier des données**
  - UPDATE <nom_table> SET <nom_colonne> = ‘valeur’
- **Supprimer des données**
  - DELETE FROM <nom_table> WHERE <condition>

SQL: La clause WHERE

La clause **WHERE** dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

Exemple :

- SELECT <nom_colonne> FROM <nom_table> WHERE id=1;
- UPDATE <nom_table> SET <nom_colonne> = 'valeur' WHERE name="Toto";

Il est possible d'ajouter plusieurs conditions sur une requête via les opérateurs **AND** et **OR**.

Exemple :

- SELECT <nom_colonne> FROM <nom_table> WHERE id=1 AND name="Toto";
- SELECT <nom_colonne> FROM <nom_table> WHERE id=1 OR name="Toto";

SQL: Les jointures

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

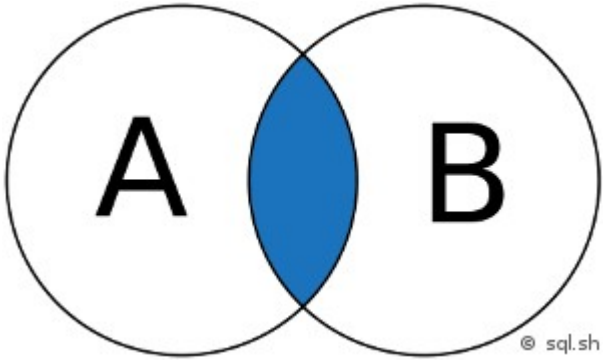
Il existe plusieurs types de jointures :

- INNER JOIN : jointure interne pour retourner les enregistrements quand la condition est vrai dans les 2 tables. C'est l'une des jointures les plus communes.
- CROSS JOIN : jointure croisée permettant de faire le produit cartésien de 2 tables. En d'autres mots, permet de joindre chaque lignes d'une table avec chaque lignes d'une seconde table. Attention, le nombre de résultats est en général très élevé.
- LEFT JOIN (ou LEFT OUTER JOIN) : jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifié dans l'autre table.
- RIGHT JOIN (ou RIGHT OUTER JOIN) : jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifié dans l'autre table.
- FULL JOIN (ou FULL OUTER JOIN) : jointure externe pour retourner les résultats quand la condition est vrai dans au moins une des 2 tables.

SQL: Les jointures INNER JOIN

Voici la syntaxe de l'INNER JOIN :

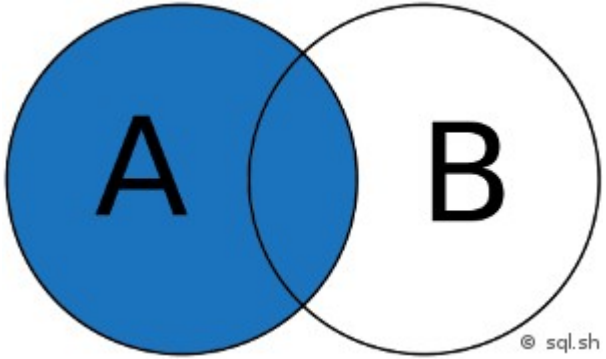
```
SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.fk_id;
```



SQL: Les jointures LEFT JOIN

Voici la syntaxe du LEFT JOIN :

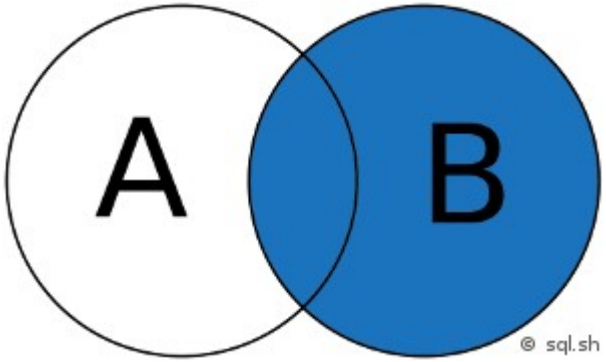
```
SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.fk_id;
```



SQL: Les jointures RIGHT JOIN

Voici la syntaxe du RIGHT JOIN :

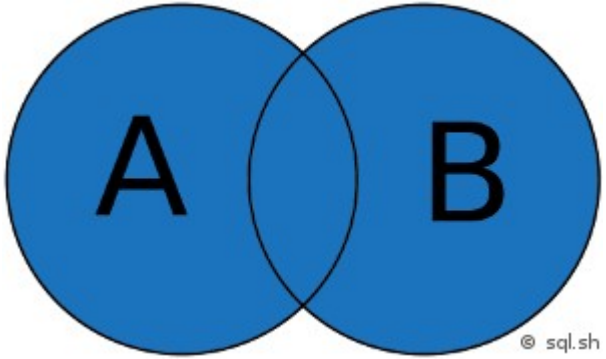
```
SELECT * FROM table1 RIGHT JOIN table2 ON table1.id = table2.fk_id;
```



SQL: Les jointures FULL JOIN

Voici la syntaxe du FULL JOIN :

```
SELECT * FROM table1 FULL JOIN table2 ON table1.id = table2.fk_id;
```



SQL: Les jointures CROSS JOIN

Voici la syntaxe du CROSS JOIN :

SELECT * FROM table1 CROSS JOIN table2;

Attention : effectuer le produit cartésien d'une table A qui contient 30 résultats avec une table B de 40 résultats va produire 1200 résultats (30 x 40 = 1200). En général la commande CROSS JOIN est combinée avec la commande WHERE pour filtrer les résultats qui respectent certaines conditions.

Formation

THE END