`# Fiche de révision : BD4

Conception d'un schéma relationnel

Définitions

Concepts de base

- **Domaine** : Un ensemble de **valeurs possibles** pour un attribut (par exemple, les valeurs possibles pour l'âge d'une personne).
- **Produit cartésien** : Combinaison de tous les éléments de plusieurs domaines (par exemple, combiner les valeurs de "Nom" et "Âge").
- **Relation** : Une **table** dans une base de données, composée de lignes (tuples) et de colonnes (attributs).
- **Attribut** : Une **colonne** dans une table, caractérisée par un **nom** et un **domaine**.

Schéma de relation

- **Schéma de relation** : La structure d'une relation, définie par le nom de la relation et les attributs qui la composent.
  - **Intention** : La structure d'une relation, définie par le nom de la relation et les attributs qui la composent.
  - **Extension** : L'ensemble des tuples d'une relation à un moment donné.

Clés

- **Clé** : Ensemble minimal d'attributs qui permet d'identifier un tuple de manière unique
- **Clé primaire** : Clé choisie pour identifier de manière unique chaque tuple d'une relation

Problèmes dans la conception

- **Redondance** : Répétition d'informations dans une base de données, ce qui peut entraîner des anomalies et des incohérences.
- **Anomalies de mise à jour** : Problèmes qui surviennent lorsqu'une modification des données entraîne des incohérences dans la base de données.
- **Anomalies d'insertion** : Problèmes qui surviennent lorsqu'il est impossible d'ajouter des données à la base de données sans ajouter d'autres informations.

Dépendances fonctionnelles

- **Dépendance fonctionnelle** : Relation entre deux attributs dans une relation, où la valeur d'un attribut détermine la valeur de l'autre attribut.
  - Par exemple, X→Y signifie que la valeur de X détermine la valeur de Y. (ex. : "Nom" → "Âge")

Normalisation

> Processus de conception d'une base de données relationnelle pour minimiser la redondance et les anomalies.

Formes normales

- **1NF (Première forme normale)** : Chaque attribut d'une relation doit contenir une valeur atomique (non décomposable).
  - Exemple de violation : Une colonne "Nom" contenant "Prénom" et "Nom de famille", ou "Adresse"
  - Règle de décomposition : Créer une nouvelle relation avec les attributs décomposables et une clé étrangère pour les relier.
    - Exemple : "Nom" → "Prénom", "Nom de famille"
- **2NF (Deuxième forme normale)** : Une relation est en 2NF si elle est en 1NF et si tous ses attributs non-clés dépendent de la clé primaire.
  - Exemple de violation : Une relation "Commande" avec "ID Commande", "ID Produit", "Nom Produit", "Prix Produit" où "Nom Produit" et "Prix Produit" dépendent de "ID Produit" et non de "ID Commande".
  - Règle de décomposition : Créer une nouvelle relation avec les attributs dépendants et une clé étrangère pour les relier.
    - Exemple : "ID Produit" → "Nom Produit", "Prix Produit" (nouvelle relation "Produit")
- **3NF (Troisième forme normale)** : Une relation est en 3NF si elle est en 2NF et si tous ses attributs non-clés ne dépendent pas des autres attributs non-clés.
  - Exemple de violation : Une relation "Employé" avec "ID Employé", "Nom Employé", "Département", "Salaire" où "Département" dépend de "ID Employé" et non de "Nom Employé".
  - Règle de décomposition : Créer une nouvelle relation avec les attributs dépendants et une clé étrangère pour les relier.
    - Exemple : "ID Employé" → "Département" (nouvelle relation "Département")
- **BCNF (Forme normale de Boyce-Codd)** : Une relation est en BCNF si, pour chaque dépendance fonctionnelle X→Y, X est une clé entière et Y est un attribut non-clé.

- Exemple de violation : Une relation "Cours" avec "ID Cours", "ID Professeur", "Nom Professeur" où "Nom Professeur" dépend de "ID Professeur" et non de "ID Cours".

Algorithme de décomposition

Il suffit d'appliquer récursivement les règles de décomposition jusqu'à ce que toutes les relations soient en 3FN.

Algorithme de synthèse

> **Fermeture**: La fermeture d'un ensemble d'attributs X, notée X^+, est l'ensemble de tous les attributs qui sont fonctionnellement déterminés par X.
> Exemple: Si $X = \{A\}$, et que $A \rightarrow B$, $B \rightarrow C$, alors $X^+ = \{A, B, C\}$.
> **Ce sont l'ensemble des attributes atteignables à partir de X.**

1. Trouver une couverture minimale G de F
   i. Initialiser la relation R avec tous les attributs et identifier toutes les dépendances fonctionnelles.
   ii. Simplifier à droite toutes les DF.
      a. Développer les DF $X \rightarrow A_1, ..., A_n$ telles que $X \rightarrow A_1, X \rightarrow A_2, ..., X \rightarrow A_n$.
   iii. Simplifier à gauche toutes les DF.
      a. Pour chaque attribut B de X, si B appartient à la fermeture suivante: $\{X - \{B\}\}^+_{G-\{X \rightarrow A\}}$.

         C'est à dire que B est atteignable par les autres attributs de X.

         Alors on remplace $X \rightarrow A$ par $X - \{B\} \rightarrow A$.
   iv. Supprimer les DF redondantes.
      a. Pour chaque DF $X \rightarrow A$, si A appartient à $X^+_{G-\{X \rightarrow A\}}$, alors on peut supprimer $X \rightarrow A$.

         C'est à dire que A est atteignable à partir des autres DF depuis X.
2. Edition des relations
   i. Pour chaque X de G, créer une relation $\{X, A_1, ..., A_n\}$ avec les DF $X \rightarrow A_1, ..., X \rightarrow A_n$.
   ii. X est la clé de la relation.
3. Fusionner les relations avec les clés équivalentes
   i. Si les fermeture des clés sont équivalentes, alors les relations peuvent être fusionnées.

ii. Exemple: Si $X^+ = Y^+$, alors fusionner les relations $\{X, A_1, ..., A_n\}$ et $\{Y, B_1, ..., B_m\}$ en $\{X, Y, A_1, ..., A_n, B_1, ..., B_m\}$ où X et Y sont 2 clés distinctes.

> Voir le document BD4-ALGO.md pour avoir un exemple.

NoSQL - Oracle 21c, JSON

DDL

CREATE TABLE

- Utilisée pour créer une nouvelle table dans la base de données.
- Exemple :

```
CREATE TABLE client (
    id NUMBER NOT NULL,
    nom VARCHAR(20) NOT NULL,
    prenom VARCHAR(20) NOT NULL,
    adresse JSON NOT NULL,
    tel JSON NOT NULL,
    CONSTRAINT pk_client PRIMARY KEY (id)
);
```

- Fonctionnement : Définit une table avec des colonnes, dont certaines peuvent être de type JSON.

JSON_SERIALIZE

- **Fonctionnement** : Convertit une valeur SQL en une représentation JSON.
- **Exemple** :

```
SELECT JSON_SERIALIZE(CURSOR(SELECT nom, prenom FROM client)) AS json_result FROM du
```

- **Utilisation** : Utilisée pour sérialiser des résultats de requêtes SQL en format JSON.

JSON_VALUE

- **Fonctionnement** : Extrait une valeur scalaire d'une colonne JSON.
- **Exemple** :

```
SELECT JSON_VALUE(adresse, '$.ville') AS ville FROM client;
```

- **Utilisation** : Récupère une valeur spécifique d'un document JSON.

JSON_QUERY

- **Fonctionnement** : Extrait un objet ou un tableau JSON d'une colonne JSON.
- **Exemple** :

```
SELECT JSON_QUERY(adresse, '$.coordonnees') AS coordonnees FROM client;
```

- **Utilisation** : Utilisée pour extraire des sous-ensembles de données JSON.

JSON_MERGEPATCH

- **Fonctionnement** : Fusionne deux documents JSON en appliquant des modifications d'un document à un autre.
- **Exemple** :

```
SELECT JSON_MERGEPATCH('{"nom": "Dupont"}', '{"prenom": "Jean"}') AS result FROM dua
```

- **Utilisation** : Permet de mettre à jour un document JSON avec des modifications spécifiques.

JSON_TRANSFORM

- **Fonctionnement** : Transforme un document JSON en appliquant une série de modifications spécifiées.
- **Exemple** :

```
SELECT JSON_TRANSFORM(adresse,
                      SET '$.ville' = 'NouvelleVille') AS transformed_adresse
FROM client;
```

- **Utilisation** : Utilisée pour modifier la structure ou les valeurs d'un document JSON.

JSON_EXISTS

- **Fonctionnement** : Vérifie l'existence d'une clé ou d'une valeur dans un document JSON.
- **Exemple** :

```
SELECT * FROM client
WHERE JSON_EXISTS(adresse, '$.ville');
```

- **Utilisation** : Utilisée pour filtrer les enregistrements en fonction de la présence de certaines données JSON.

Redis, Key-Value

Bases de données

- `CONFIG GET databases` : Obtient le nombre total de bases de données.
- `INFO keyspace` : Affiche les bases de données contenant des clés.

Manipulation des Strings

Écriture et lecture de clés

- `SET clef valeur` : Stocke une valeur dans une clé.
  - `SET clef valeur NX` : Stocke si la clé n'existe pas.
  - `SET clef valeur XX` : Stocke si la clé existe.
  - `SET clef valeur EX 10` : Stocke avec expiration (en secondes)
- `GET clef` : Récupère la valeur d'une clé.
- `MSET clef1 valeur1 clef2 valeur2` : Stocke plusieurs paires clé-valeur.
- `MGET clef1 clef2` : Récupère plusieurs valeurs.

Opérations sur les Strings

- `INCR compteur` : Incrémente une valeur numérique.
- `DECR compteur` : Décrémente une valeur numérique.
- `INCRBY compteur 5` : Incrémente de 5.
- `DECRBY compteur 2` : Décrémente de 2.
- `SETRANGE clef 6 "Redis"` : Remplace une partie de la chaîne.
- `GETRANGE clef 0 5` : Récupère une sous-chaîne.
- `STRLEN clef` : Retourne la longueur de la chaîne.
- `APPEND clef " ajout"` : Ajoute du texte à une chaîne existante.

Manipulation des Hashes

Écriture et lecture

- `HSET dictionnaire champ "valeur"` : Définit une paire champ-valeur.
- `HGET dictionnaire champ` : Récupère la valeur d'un champ.

- `HMSET utilisateur nom "Dupont" age "30"` : Stocke plusieurs champs.
- `HMGET utilisateur nom age` : Récupère plusieurs champs.

Opérations sur les Hashes

- `HEXISTS dictionnaire champ` : Vérifie si un champ existe.
- `HDEL dictionnaire champ` : Supprime un champ.
- `HLEN dictionnaire` : Nombre de champs.
- `HKEYS dictionnaire` : Liste des champs.
- `HVALS dictionnaire` : Liste des valeurs.
- `HGETALL dictionnaire` : Liste des champs et valeurs.
- `HSCAN dictionnaire 0 MATCH "hp:*"` : Recherche des champs.

Manipulation des Listes

Écriture et lecture

- `LPUSH liste a b c` : Ajoute en tête de liste.
- `RPUSH liste x y z` : Ajoute en fin de liste.
- `LRANGE liste 0 -1` : Récupère tous les éléments.
- `LPOP liste` : Supprime le premier élément.
- `RPOP liste` : Supprime le dernier élément.

Opérations sur les Listes

- `LINSERT liste AFTER b "bb"` : Insère après un élément.
- `LINDEX liste 2` : Récupère l'élément à un index.
- `LSET liste 2 "nouveau"` : Modifie un élément.
- `LREM liste 2 "valeur"` : Supprime les occurrences de "valeur".

Manipulation des Sets

Écriture et lecture

- `SADD ensemble a b c` : Ajoute des éléments à un Set.
- `SREM ensemble c` : Supprime un élément.
- `SMEMBERS ensemble` : Liste tous les éléments.
- `SCARD ensemble` : Nombre d'éléments.

Opérations sur les Sets

- `SISMEMBER ensemble a` : Vérifie si un élément existe.
- `SDIFF set1 set2` : Différence entre deux ensembles.
- `SINTER set1 set2` : Intersection entre ensembles.
- `SUNION set1 set2` : Union de plusieurs ensembles.

MongoDB, Document

Commandes de base

```
help                 # Affiche l'aide
version              # Affiche la version de MongoDB
show dbs             # Liste les bases de données
show collections     # Liste les collections de la BD courante
db                  # Affiche la BD courante
exit                # Quitter MongoDB
```

Gestion des bases de données

```
use mydb             # Crée ou sélectionne la base de données `mydb`
db.dropDatabase()    # Supprime la base de données courante
```

Gestion des collections

```
db.createCollection('profs')  # Créer une collection `profs`
db.profs.drop()               # Supprime la collection `profs`
```

Opérations CRUD

Opérateurs de comparaison

- `$eq` : égal à
- `$ne` : différent de
- `$gt` : supérieur à
- `$gte` : supérieur ou égal à
- `$lt` : inférieur à

- `$lte` : inférieur ou égal à
- `$in` : dans un tableau
- `$nin` : pas dans un tableau
- `$exists` : existe

  Syntaxe: `{ champ: { $op: valeur } }`

  Exemple: `{ age: { $gte: 30 } }`

Opérateurs logiques

- `$and` : ET
- `$or` : OU
- `$not` : NON
- `$nor` : OU exclusif
- `$all` : tous les éléments d'un tableau
- `$elemMatch` : au moins un élément d'un tableau
- `$size` : taille d'un tableau

  Syntaxe: `{ $op: [ { champ1: valeur1 }, { champ2: valeur2 } ] }`

  Exemple: `{ $and: [ { age: { $gte: 30 } }, { age: { $lt: 40 } } ] }`

Insertion

```
db.profs.insertOne({nom: 'Durand', prenom: 'Paul', age: 35})
db.profs.insertMany([{nom: 'Martin', age: 55}, {nom: 'Lefevre', age: 40}])
```

Lecture

```
db.profs.find()                    # Tous les documents
db.profs.find({age: {$gte: 40}})    # Documents où `age` >= 40
db.profs.find({}, {nom: 1, _id: 0}) # Affiche uniquement le champ `nom`
```

Syntaxe: `db.collection.find({filtre}, {projection})`

Mise à jour

Opérateurs de mise à jour

- `$` Agit comme un placeholder correspondant au 1er élément qui correspond au filtre
- `$[]` Agit comme un placeholder correspondant à tous les éléments qui correspondent au filtre
- `$set` : met à jour un champ
- `$inc` : incrémente un champ

- `$push` : ajoute un élément à un tableau
- `$pull` : supprime un élément d'un tableau
- `$pullAll` : supprime plusieurs éléments d'un tableau
- `$pop` : supprime le premier ou le dernier élément d'un tableau
- `$rename` : renomme un champ
- `$unset` : supprime un champ
- `$currentDate` : met à jour un champ avec la date actuelle

  Syntaxe: `{ $op: { champ: valeur } }`

```
db.profs.updateOne({nom: 'Durand'}, {$set: {age: 36}})
db.profs.updateMany({age: {$lt: 30}}, {$inc: {age: 1}})
```

Syntaxe: `db.collection.updateOne({filtre}, {$op: {champ: valeur}})`

Suppression

```
db.profs.deleteOne({nom: 'Durand'})
db.profs.deleteMany({age: {$gt: 50}})
```

Requêtes avancées

Filtres conditionnels

```
db.salles.find({postes: {$gte: 15}})  # Salles avec au moins 15 postes
db.salles.find({postes: 0, places: {$gte: 28}})  # Salles sans postes avec >= 28 places
```

Opérations sur tableaux

```
db.salles.insert({nom: 'TP4', postes: ['TP4-01', 'TP4-02']})
db.salles.find({postes: {$all: ['TP4-01', 'TP4-02']}})
db.salles.updateOne({nom: 'TP4'}, {$pull: {postes: 'TP4-01'}})  # Supprime un élément
```

Tri et limitation

```
db.people.find({}, {_id: 0, firstname: 1}).sort({age: -1}).limit(5)
```

Opérations sur les champs

```
db.people.updateMany({}, {$rename: {technicalId: 'id'}})  # Renommer un champ
db.people.updateMany({age: {$gt: 30}}, {$set: {cars: 2}})  # Ajouter un champ
db.people.updateMany({age: {$gt: 30, $lt: 40}}, {$unset: {email: ''}})  # Supprimer un c
```