Chapitre 9 – Bases de données

L'administration, les banques, les assurances, les secteurs de la finance utilisent des bases de données, systèmes d'informations qui stockent dans des fichiers les données nombreuses qui leur sont nécessaires. Une base de données relationnelle permet d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Un logiciel, le système de gestion de bases de données (SGBD), est utilisé pour la gestion (lecture, écriture, cohérence, actualisation...) des fichiers dans lesquels sont stockées les données. L'accès aux données d'une base de données relationnelle s'effectue en utilisant un langage informatique qui permet de sélectionner des données spécifiées par des formules de logique, appelées requêtes d'interrogation et de mise à jour.
L'objectif est de présenter une description applicative des bases de données en langage de requêtes SQL (Structured Query Language). Il s'agit de permettre d'interroger une base présentant des données à travers plusieurs relations.

1. Le modèle relationnel

On considère des éléments d'une même **entité**, sur lesquelles nous avons des informations.
Les propriétés dont on a la valeur pour chaque élément d'une entité et qui forment les données s'appellent les **attributs**
L'ensemble des attributs forme les **descripteurs** de l'entité.
L'ensemble des valeurs possibles d'un attribut s'appelle le **domaine** de l'attribut.
Les domaines au programme sont : **INTEGER** et **TEXT** (mais il existe également DATE, BOOLEAN, REAL, VARCHAR …)

Exemple :
Un commerçant a créé une carte de fidélité pour ses clients, ce qui lui permet de conserver plusieurs informations : le numéro de client, le nom et le prénom, la ville, le montant du dernier achat, le montant total.

Les données avec lesquelles nous travaillerons sont :

Numéro Client	Nom	Prénom	Ville	Montant dernier achat	Montant total
1	Aubry	Alphonse	Amnéville	77	180
2	Bertrand	Barnabé	Ban St Martin	28	206
3	Chevalier	Charlotte	Cuvry	62	172
4	Dupont	Dorothée	Dieuze	60	153

Dans cet exemple, l'entité est un client.
Les attributs sont `NumClient, Nom, Prenom, Ville, DernierAchat, MontantTotal`
Le domaine de `NumClient, MontantAchat` et `MontantTotal` est `INTEGER`
Le domaine de `Nom, Prenom` et `Ville` est `TEXT`

On appelle **schéma de relation** le descriptif de la relation, constitué d'un nom (celui de l'entité), suivi de la liste des attributs et de leur domaine.

Exemple :
Ici le schéma de relation est :
```
Clients=((NumClient : INTEGER), (Nom : TEXT), (Prenom : TEXT), (Ville : TEXT), (DernierAchat : INTEGER), (MontantTotal : INTEGER))
```

On appelle **enregistrement** la liste des valeurs pour une entité donnée.

Exemple :
Pour le client n°2 l'enregistrement est :
`(2, Bertrand, Barnabé, Ban Saint Martin, 28, 206)`

On appelle **relation** ou **table** l'ensemble des données pour tous les éléments.

Table de l'exemple précédent :

NumClient	Nom	Prenom	Ville	DernierAchat	MontantTotal
1	Aubry	Alphonse	Amnéville	77	180
2	Bertrand	Barnabé	Ban Saint Martin	28	206
3	Chevalier	Charlotte	Cuvry	62	172
4	Dupont	Dorothée	Dieuze	60	153

On confondra souvent les termes "attribut" et "colonne" d'une part, et "enregistrement" et "ligne" d'autre part.

Clé primaire : **`PRIMARY KEY`**
Une clé primaire d'une table est un attribut (ou une colonne) qui permet **d'identifier de manière unique** les entités de la table.

Ex : `NumClient` est une clé primaire, mais les autres ne le sont pas (deux clients peuvent avoir le même prénom, le même montant d'achat, …).

Clé étrangère : **`FOREIGN KEY`**
On appelle clé étrangère d'une table toute colonne qui référence la clé primaire d'une autre table.
Cette clé met donc en relation cette deuxième table avec la première.

Exemple :
Supposons que le commerçant propose un service de livraison dont le tarif dépend de l'adresse de livraison :

Ville	Tarif (en euros)
Amneville	5
BanSaintMartin	5
Cuvry	10
Dieuze	15

La colonne `Ville` est une clé primaire de la table `Livraison` et c'est une colonne de la table `Client`.
Ces deux tables sont donc en relation.

Remarque : Les clés étrangères permettent de vérifier la cohérence entre les valeurs des différentes tables.
Les attributs ne portent pas forcément le même nom dans les deux tables.

Ex : Dans l'exemple précédent, le schéma relationnel devient :

```
Clients = ((NumClient : INTEGER), (Nom : TEXT), (Prenom : TEXT), (#Ville
: TEXT), (DernierAchat : INTEGER), (MontantTotal : INTEGER))
Livraison = ((Ville : TEXT), (Tarif : INTEGER))
```

2. Langage SQL

On pourra s'exercer par exemple avec `DB Browser SQLite`

En SQL, on a l'habitude de noter en majuscules les mots-clés du langage et en minuscule le reste. (mais SQL ne fait pas la différence entre majuscules et minuscules).

Création d'une table (avec le premier attribut comme clé primaire) :

```
CREATE TABLE nomtable
     (attribut1     nom_de_domaine,
     …
     attributn      nom_de_domaine,
     PRIMARY KEY(attribut1))
```

Création d'une table avec `attributn` comme clé étrangère, qui est l'`attributnbis` de la table `nomtable2` :

```
CREATE TABLE nomtable
     (attribut1     nom_de_domaine
     …
     attributn      nom_de_domaine
     FOREIGN KEY (attributn) REFERENCES nomtable2(attributnbis)
```

Exemple :

```
CREATE TABLE livraison
        (Ville  TEXT,
         Tarif  INTEGER,
          PRIMARY KEY (Ville))

CREATE TABLE clients
     (NumCient INTEGER  PRIMARY KEY,
          Nom   TEXT,
     Prenom  TEXT,
          Ville TEXT,
          DernierAchat  INTEGER,
          MontantTotal INTEGER,
          FOREIGN KEY (Ville) REFERENCES livraison(ville))
```

Dans ce cas, tous les enregistrements de la table Clients devront obligatoirement avoir un attribut 'Ville' qui appartient à la liste des attributs 'Ville' de la table Livraison.

Essentiel : A Retenir

Sélection de données dans une table (ou projection) :
Pour sélectionner une ou plusieurs colonnes dans une table :
`SELECT attribut1 FROM nomtable;`
`SELECT attribut1, attribut2 FROM nomtable;`
`SELECT * FROM nomtable;` (sélection de toutes les colonnes)

Sélection avec conditions (ou **restriction**)
Pour sélectionner uniquement les enregistrements qui vérifient une certaine condition, on utilise :
`SELECT` … **`FROM`** … **`WHERE condition`**

Remarques :
`condition` est un booléen
On peut utiliser les opérateurs de test suivants : **`=, <>`** (différent)**, `<, <=, >, >=`**, **`IN`**
On peut utiliser les opérateurs logiques : **`AND OR NOT`**
Enfin, on peut utiliser les opérateurs arithmétiques : `+, -, *`

Exemple :
Rechercher les noms et prénoms des clients qui ont dépensé en tout moins de 200 euros.

```
SELECT Nom,Prenom FROM clients WHERE MontantTotal<=200;
```

Renvoie :

```
Nom       Prenom
Aubry     Alphonse
Chevalier Charlotte
Dupont    Dorothee
```

Ajout d'un enregistrement dans une table :
`INSERT INTO nomtable VALUES (element1, element2,…)`
insère une ligne dans la table `nomtable` de valeurs (`element1`, …)
(Attention au nombre et à l'ordre. Attention également aux guillemets autour du texte).
On ne peut pas ajouter une ligne si la valeur n'existe pas pour la clé étrangère

Exemple :

```
INSERT INTO clients VALUES (5,'Evrard','Eugène','Etting',48,221);
```

n'est pas accepté par SQL, car 'Etting' n'existe pas dans la table Livraison

```
INSERT INTO clients VALUES (5,'Evrard','Eugène','Cuvry',48,221);
```

fonctionne

Suppression de lignes :
La requête :
`DELETE FROM nomtable WHERE condition`
supprime toutes les lignes vérifiant la condition
`DELETE FROM nomtable`
supprime toute la table

Modification d'une table :
La requête :

UPDATE nomtable SET attribut=valeur WHERE condition
remplace la valeur de l'attribut pour les lignes qui vérifient la condition

Exemple :
```
UPDATE clients SET `Nom`='Dupond' WHERE `Nom`='Dupont';
```

Remplace 'Dupont' par 'Dupond'

Lorsque deux tables sont en relation via une de leur colonne, il est possible de regrouper l'ensemble des données en réalisant une **jointure** :

On suppose que les tables `nomtable1` et `nomtable2` ont des colonnes `attribut1` et `attribut2` qui ont au moins une valeur commune.
La requête :

SELECT * FROM nomtable1 INNER JOIN nomtable2
ON nomtable1.attribut1 = nomtable2.attribut2
commande l'affichage, à la suite, des lignes de la table `nomtable1` et de la table `nomtable2` pour les seuls cas où les champs de la colonne `attribut1` sont égaux à ceux de la colonne `attribut2`.

Exemple :
```
SELECT * FROM clients INNER JOIN livraison ON client.ville=livraison.Ville;
```

renvoie :

NumCient	Nom	Prenom	Ville	DernierAchat	MontantTotal		Ville	Tarif
1	Aubry	Alphonse	Amneville	77	180		Amneville	5
2	Bertrand	Barnabe	BanSaintMartin	28	206		BanSaintMartin	5
3	Chevalier	Charlotte	Cuvry	62	172		Cuvry	10
4	Dupond	Dorothee	Dieuze	60	153		Dieuze	15
5	Evrard	Eugène	Cuvry	48	221		Cuvry	10

Exemple :
Trouver les noms et prénoms des clients pour qui la livraison coûte moins de 10 euros.

```
SELECT Clients.Nom,Clients.Prenom FROM Clients INNER JOIN Livraison ON
Clients.Ville=Livraison.Ville WHERE Livraison.Tarif<=10
```

2.3 Commandes non exigibles

La requête

```
SELECT attribut1 FROM nomtable1
UNION
SELECT attribut2 FROM nomtable2
```

regroupe sous une seule colonne la colonne `attribut1` de la table `nomtable1` et la colonne `attribut2` de la table `nomtable2`.

Remarque : On peut ajouter des conditions à l'aide de `WHERE`

La requête

```
SELECT attribut1 FROM nomtable1
INTERSECT
SELECT attribut2 FROM nomtable2
```

commande l'affichage des données communes aux deux colonnes indiquées.

La requête

```
SELECT attribut1 FROM nomtable1
EXCEPT
SELECT attribut2 FROM nomtable2
```

commande l'affichage des données de la colonne `attribut1` de la table `nomtable1` qui ne sont pas présentes dans la colonne `attribut2` de la table `nomtable2`.

Fonctions d'agrégation :

Si attribut est une colonne de type numérique

```
SELECT SUM(attribut) FROM nomtable
SELECT MIN(attribut) FROM nomtable
SELECT MAX(attribut) FROM nomtable
SELECT AVG(attribut) FROM nomtable
SELECT COUNT(attribut) FROM nomtable
```

affiche respectivement la somme, le minimum, le maximum, la moyenne, et le cardinal de la colonne `attribut`.

Exemple :

```
SELECT AVG(DernierAchat) FROM clients
```

renvoie : 55.0000

La requête

```
SELECT DISTINCT attribut FROM nomtable
```

commande l'affichage de la colonne sans afficher les doublons.

La requête

```
SELECT attribut FROM nomtable ORDER BY attribut2
```

commande l'affichage de la colonne `attribut` en les affichant dans l'ordre croissant de la colonne `attribut2`. (On ajoute **`DESC`** à la fin pour l'ordre décroissant)

Ex : Classer les noms, prénoms et montants totaux dans l'ordre décroissant du montant total des achats

```
SELECT Nom, Prenom, MontantTotal FROM Clients ORDER BY MontantTotal DESC
```