1 Généralités

Le stockage des informations numériques n'est pas un problème ; ce qu'il faut ensuite, c'est retrouver l'information !

Internet propose le modèle « moteur de recherche ». C'est un outil complexe dont voici la définition (source Wikipédia) :

> *Instrument de recherche sur le web constitué de « robots », encore appelés bots, spiders, crawlers ou agents qui parcourent les sites à intervalles réguliers et de façon automatique (sans intervention humaine, ce qui les distingue des annuaires) pour découvrir de nouvelles adresses (URL). Ils suivent les liens hypertextes (qui relient les pages les unes aux autres) rencontrés sur chaque page atteinte. Chaque page identifiée est alors indexée dans une base de données, accessible ensuite par les internautes à partir de mots-clés.*

On note que tout le savoir d'un moteur de recherche est finalement stocké de façon organisée dans une Base de Données. Cela contraste avec le côté « anarchique » des millions de sites web développés indépendamment les uns des autres.

C'est exactement l'objectif d'une **Base de données** : stocker l'information de façon structurée (par exemple, sous forme de tableau lignes/colonnes).

Un **Système de Gestion de Base de Données (SGBD)** sert à exploiter la base de données : il classe les informations (notion d'index ou de clé), permet de rechercher une information en fonction de critères précisés par l'utilisateur, garantit la cohérence des données et leur protection contre les accès frauduleux.

Nous utiliserons ici un SGBD dit « relationnel » (SGBDR). Le mot « relationnel » définit la façon dont l'information est stockée dans la Base de Données : sous forme de table à deux dimensions appelée « ***relation*** ».

Table Relationnelle VEHICULES :

ATTRIBUTS (Colonnes)

IdVeh	marque	couleur	plaque
0001	Renault	Blanche	EL-294-MM
0002	BMW	Noire	MS-189-SA
0004	Alfa Romeo	Rouge	AA-200-BB
0005	Citroën	Grise	XX-555-YY

RELATIONS (Lignes, TUPLES)

La « relation » est un ensemble de lignes structurée : à un type d'information correspond un type de colonne.

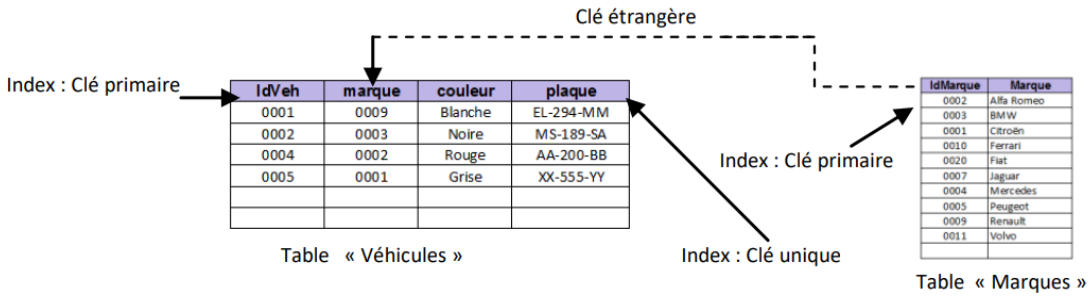
Une ligne correspond à un ensemble d'informations cohérentes entre elles.

Ainsi l'ordre des lignes ou des colonnes importe peu : on retrouve aisément l'information en fonction de son attribut (nom de la colonne). Le SGBDR garantit que la ligne contiendra les bonnes informations complémentaires.

Une base de données peut comporter plusieurs tables (relations) indépendantes.

On peut établir également des liens (notion de clé étrangère) entre 2 tables. Par exemple, ci-dessus la colonne « marque » pourrait faire référence à une autre table qui contiendrait une liste de marques existantes. Ainsi, il ne serait pas possible de mentionner dans notre table « voiture » une marque qui n'existe pas.

La question de l'unicité : Dans notre exemple, on conçoit qu'il n'est pas possible d'avoir 2 véhicules avec la même plaque ! Comment un SGBDR nous aide-t-il à maintenir cette condition ? On peut définir un index spécial, appelé CLE PRIMAIRE ou CLE UNIQUE sur un ou plusieurs attributs, pour garantir leur unicité. Si on cherche à ajouter un *tuple* qui viole cette unicité, le SGBDR refusera l'enregistrement :



La question du tri : Pour retrouver une information le plus rapidement possible, il est conseillé de trier dans un certain ordre. Par exemple, dans une liste de nom, on retrouvera facilement le nom que l'on cherche si la liste est classée par ordre alphabétique. Le SGBDR aussi retrouvera plus facilement une information si on lui demande de trier les *tuples* dans un certain ordre. La difficulté est de décider quelle colonne sera triée. Cela dépend de l'usage que l'on fait de la base de données : Si on a l'habitude de rechercher une voiture par sa plaque, on triera cette colonne : En SGBDR, cela revient à définir un INDEX sur cette colonne.

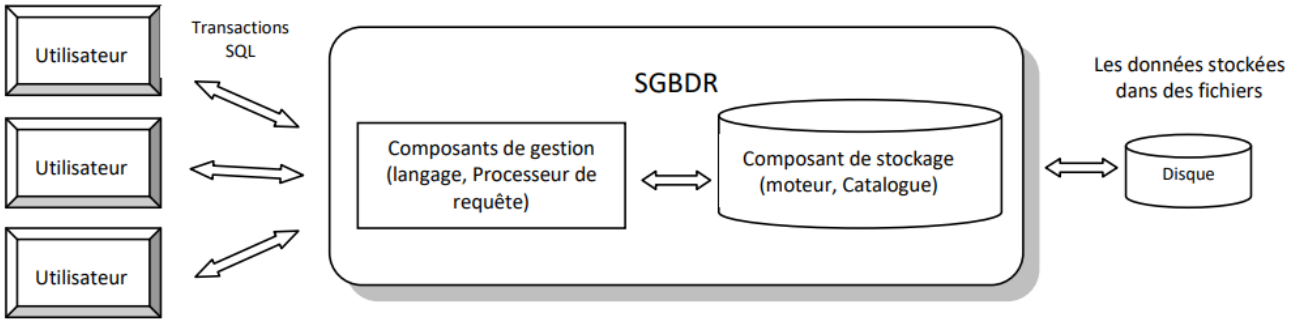
Il faut forcément un langage !

Pour créer une table, un index, une clé, rechercher l'information, insérer et supprimer des données, etc., il faut forcément un langage de dialogue entre l'utilisateur et le SGBDR.

Dans notre cas, nous utilisons un SGBDR de la famille des SQL (*Structured Query Language*) : MySQL.

Le langage sera donc SQL.

Représentation schématique d'un SGBDR :



Le langage SQL permet toutes les opérations sur une base de données.

Les commandes sont regroupées en catégories :

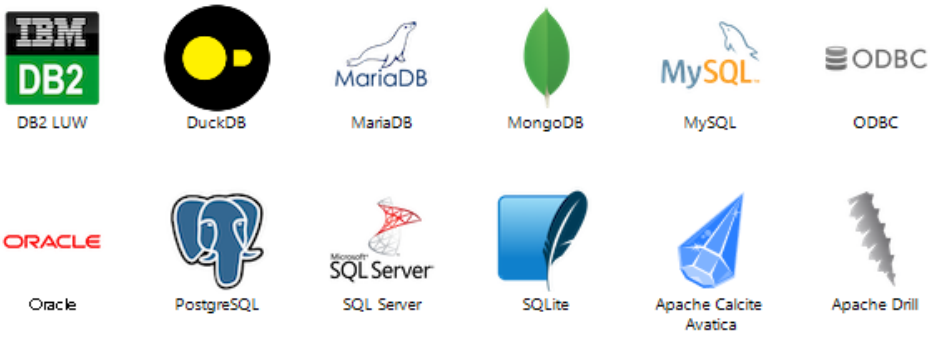
- LMD (Langage de Manipulation de Données) qui regroupe les commandes SQL de l'utilisateur qui exploite la base de données (SELECT, INSERT, …)
- LDD (Langage de Définition des Données) qui regroupe les commandes SQL de l'administrateur de la base de données (CREATE, DROP, ALTER, …)
- LCD (Langage de Contrôle de Données) qui regroupe les commandes SQL de gestion global du SGBD (GRANT, REVOKE, CREATE USER, SHUTDOWN, …)

Nous utiliserons ici les commandes du LMD, ce qui signifie que vous allez exploiter une base de données existante.

Le choix du « moteur » (*engine*) jouera sur les performances de notre SGBD. On peut favoriser les temps d'accès (moteur ISAM ou MyISAM), ou favoriser les fonctionnalités (moteur InnoDB).

Par exemple, les clés étrangères ne sont gérées que par le moteur InnoDB. Le choix du moteur dépendra donc de l'usage.

Les acteurs du marché :



2 Analyse d'un schéma de données : Exemple

L'exemple ci-dessous reprend le cas d'une gestion de base de données d'abonnés à un flux d'information en fonction de ses centres d'intérêts. Un abonné peut avoir plusieurs centres d'intérêt. De plus, le service Marketing impose que chaque abonné choisisse dans une liste ce qui motive son inscription.

De ce qui précède, il se dégage nettement quelques **entités** (futures tables) : abonnés, motivations, information (*newsletter*), centres d'intérêt.

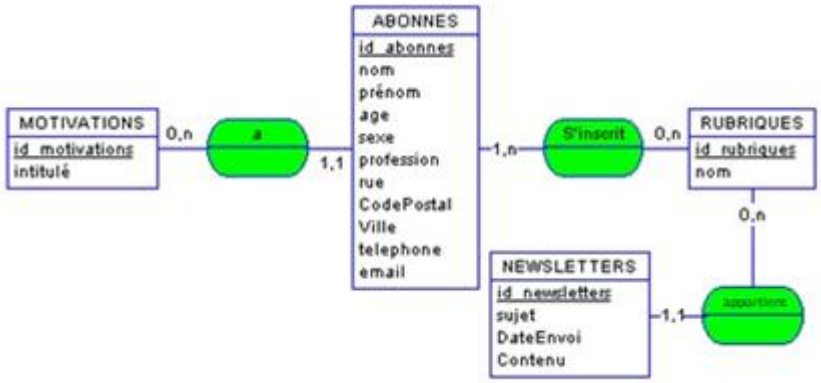
2.1 Cardinalité

La cardinalité définit les liens entre les entités. Dans les méthodes d'analyse Merise ou UML (Diagrammes de classes), la préparation du schéma de la base de données fait apparaître des « connexions » entre les entités. Ces cardinalités détermineront peut-être la création de tables supplémentaires. Les cardinalités sont de type (X,Y) : un à un, zero à un, plusieurs à plusieurs …

Exemple d'analyse Merise :

2.2 Etape du Modèle Conceptuel de Données (MCD - Merise)

On retrouve ici les entités avec leur liste d'attributs. On y ajoute les cardinalités.



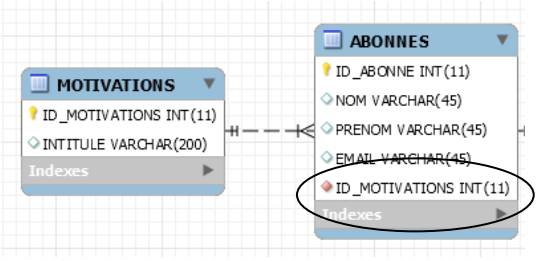
- Un Abonné a ici une et une seule Motivation d'inscription, le marketing ayant imposé un champ obligatoire afin d'avoir cette valeur. On a donc 1 minimum, et 1 maximum. D'où la cardinalité (1;1).
- Une Motivation donnée concerne 0 ou plusieurs Abonnés. On a donc 0 minimum, et n en maximum. D'où la cardinalité (0;n).
- De même, un Abonné s'inscrit à une ou plusieurs Rubriques : (1;n),
- Et une Rubrique est utilisée par 0 ou plusieurs Abonnés : (0;n).
- Enfin, une Rubrique appartient à 0 ou plusieurs Newsletters : (0;n),
- Et une Newsletter appartient à une et une seule Rubrique : (1;1).

2.3 Etape du Modèle Physique de Données (MPD – Merise) :

Dans le MPD, on transforme les relations en tables SQL complètes, incluant le type des champs, et les champs supplémentaires imposés par l'étape précédente du MCD.

La nature des relations entre les entités détermine le champ à ajouter dans une table :

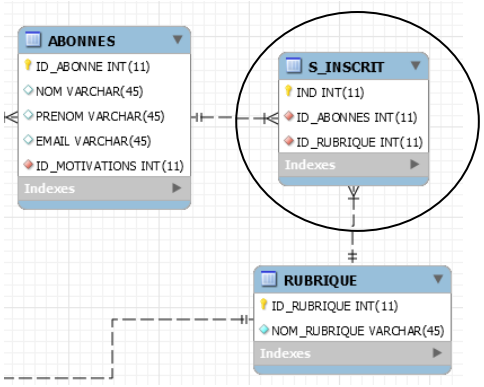
Par exemple, le lien ABONNES vers MOTIVATIONS (1,1 vers 0,n) peut se traiter simplement en ajoutant dans la table ABONNES un champ ID_MOTIVATION qui contiendra une référence à la colonne ID_MOTIVATIONS de la table MOTIVATIONS :



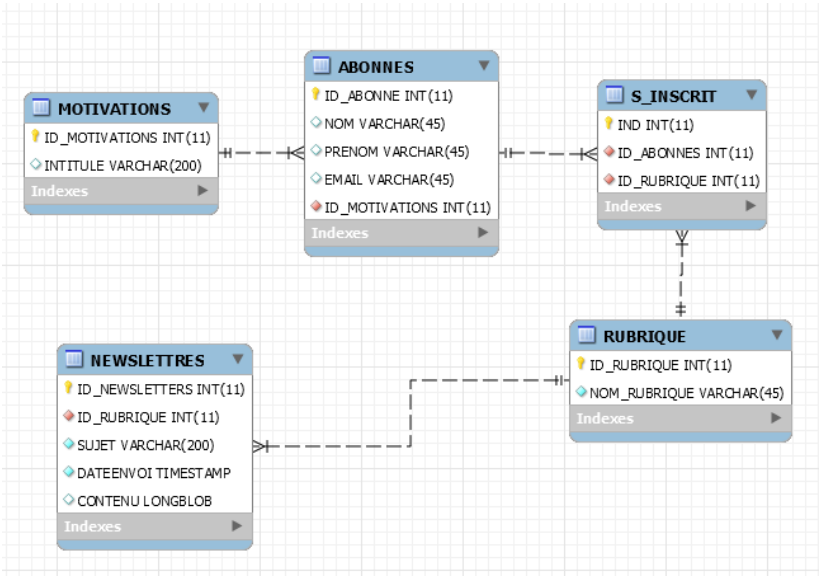
Par contre, la relation entre ABONNES et RUBRIQUES (1,n vers 0,n) pose une difficulté :

Comment intégrer dans la table ABONNES une ou plusieurs rubriques ? Combien de rubriques ? Le nombre de champs à ajouter n'est pas défini !

On règle cette question en créant une **TABLE DE JOINTURE** contenant un champ d'index connecté à l'index de chaque tables à relier. Ici, nous avons nommé cette table S_INSCRIT :



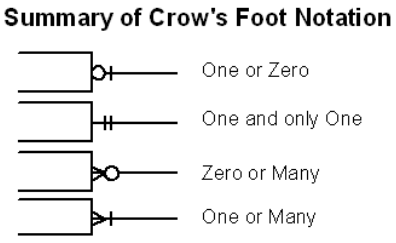
L'analyse Merise (MCD et MPD) donne donc le résultat complet suivant :



Les champs servant de liaison entre les tables sont appelé **Clés étrangères.**

On remarque la notation « Crow's foot » sur les liaisons entre les tables :

Ces notations complètent les cardinalités définies lors du MCD :



3 Synthèse des clés

3.1 Les index

Note : Les clés sont des index particulier.

Lors de la recherche d'informations dans une relation, MySQL parcours la table correspondante dans n'importe quel ordre. Dans le cas d'un grand nombre de lignes, cette recherche est très longue du fait du parcours de TOUTE la table.

Pour y remédier, une optimisation possible et FORTEMENT recommandée, est d'utiliser des index.

La création d'un index associé à un attribut ou à un ensemble ordonné d'attributs va créer une liste ordonnée des valeurs de ces attributs et de l'adresse de la ligne associée. C'est sur les valeurs de cette liste que se fera les recherches et les tris. Les algorithmes de recherche et de tri sur des ensembles ordonnés sont énormément plus rapides !

Ainsi, d'une recherche à coût prohibitif, on passe à une recherche sur un ensemble déjà trié. On gagne donc énormément en temps d'accès aux informations. Bien que cela ralentisse les mises à jour (insertion, suppression, modification de clé).

On choisira de créer des index sur les attributs qui seront les plus sollicités par les recherches ou utilisés comme critère de jointure. Par contre, on épargnera les attributs qui contiennent peu de valeurs différentes les unes des autres et ceux dont les valeurs sont très fréquemment modifiées.

Syntaxe :

INDEX *index* (*liste des attributs***)**

Exemple, pour créer un index sur les 3 premiers caractères seulement de l'attribut *nom* :

INDEX *idx_nom* (*nom(3)***)**

Exemple, pour créer un index sur le couple *nom* / *prénom* :

INDEX *idx_nom_prenom* (*'nom', 'prénom'*)

Un index peut porter sur 15 colonnes maximum.
Une table peut posséder au maximum 16 index.
Un index peut avoir une taille d'au maximum 256 octets et ne doit porter que sur des attributs NOT NULL.
Un index peut être combiné avec UNIQUE.
[ASC] ou [DESC] défini l'ordre de tri (optionnel).

3.2 Clé primaire, clé unique

Une relation a besoin d'une clé primaire (PRIMARY KEY ou clé principale) dès qu'il s'agit d'assurer l'unicité d'un *tuple* (éviter les doublons).

Cette clé est composée d'un ou plusieurs attributs. Par exemple, une clé formée des attributs (nom, prénom, date de naissance) pourrait être suffisante pour éviter les doublons dans une table contenant des coordonnées d'individus. Mais si l'attribut « Num_INSEE » existe, il pourrait à lui seul servir de clé primaire.

Une clé primaire est forcément NOT NULL sur tous ses attributs.

Un attribut déclaré clé UNIQUE accepte la valeur NULL. Dans ce cas, le SGBD s'assure seulement qu'aucun *tuple* ne contient la même valeur, sans tenir compte d'une valeur NULL.

Déclaration de clé primaire : PRIMARY KEY (attibut1, attribut2, ...). A insérer lors de la création de la table ou par la commande ALTER TABLE.

Déclaration de clé unique : UNIQUE (attibut1, attribut2, ...). A insérer lors de la création de la table ou par la commande ALTER TABLE.

Pour résumer : UNIQUE+*NotNull* = PRIMAIRE

3.3 Clé étrangère

Lorsqu'un champ d'une table A ne doit contenir que des informations provenant d'une autre table B, on parle alors de **clé étrangère**.

```
FOREIGN KEY (colonne1, colonne2, ...)
                    REFERENCES  Nom_de_la_table_etrangere(colonne1,colonne2,...)
```

Les colonnes de la table A doivent être déclarées comme INDEX simple.

La clé étrangère simplifie le travail du concepteur de l'interface utilisateur/Base de données :

Les tests de cohérence de la base sont assumés par le SGBDR, et non par le programmeur. Il est en effet impossible d'insérer un *tuple* contenant (dans son attribut déclaré clé étrangère) une valeur n'existant pas dans la table « mère » (celle sur laquelle pointe la clé étrangère). Ce problème pourrait être géré sans clé étrangère si le programmeur force l'utilisateur à choisir une valeur parmi celles existant dans la table « mère ».

Les options CASCADE et RESTRICT sont par contre plus compliquées à implémenter en langage structuré (C++ pour SQL, PHP, Perl, ASP…). Il s'agit de définir le comportement du SGBDR en cas d'effacement ou de mise à jour d'un *tuple* de la table « mère ».

Si l'option CASCADE est choisie, une modification/suppression d'un *tuple* de la table « mère » entrainera la suppression/modification de tous les *tuples* liés, dans les tables « filles », par la clé étrangère supprimée/modifiée.

Si l'option RESTRICT est choisie, une modification/suppression d'un *tuple* de la table «mère » ne se fera que si aucune table « fille » n'utilise le *tuple* en question.

Dans l'exemple ci-dessus, dans la table S_INSCRITS, on accepte de supprimer (DELETE) les *tuples* correspondant à un abonné si celui-ci est supprimé de la table « Abonnés ». Pour les modifications (UPDATE) d'un *tuple* de la table « Abonnés », elle sera répercutée sur la table « s_inscrits ».

Par contre, le SGBDR refusera de supprimer une rubrique d'intérêt dans la table « Rubriques » si celle-ci est utilisée dans la table « s_inscrits » par au moins un abonné.

La clé étrangère représente donc une **contrainte** du point de vue du LMD. C'est d'ailleurs par ce nom qu'elle est aussi appelée.

Les contraintes ON DELETE et ON UPDATE sont appelés des **gâchettes** (***trigger***).

Elles peuvent prendre également comme valeur SET NULL, SET DEFAULT *valeur*.

- *SET NULL* : place la valeur NULL dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé
- *SET DEFAULT* place la valeur par défaut (qui suit ce paramètre) dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé

4 Aperçu du langage de manipulation de données (LMD)

4.1 Prise en main du SGBD

`SHOW DATABASES;` Liste des bases disponibles

`USE basededonnées;` Utiliser *basededonnées*

`SELECT DATABASE();` Quelle base est en cours d'utilisation

`SHOW TABLES;` Liste des tables de la base en cours d'utilisation

`DESCRIBE table;` Schéma de la table

4.2 La commande SELECT

`SELECT * FROM table;` Affiche la table entière

`SELECT * FROM table LIMIT 10;` Affiche les 10 premières lignes de la table

`SELECT attribut1, attribut2 FROM table;` Affiche deux attributs de la table

`SELECT attribut1, attribut2 FROM table WHERE attribut1='toto' AND attribut4='5';` Affiche les *tuples* satisfaisant aux 2 conditions.

`SELECT * FROM table WHERE attribut1 LIKE 'A%';` Affiche les tuples dont l'attribut1 commence par la lettre A. Le caractère _ remplace 1 seul caractère dans l'expression. Plus d'option avec l'option REGEXP.

`SELECT * FROM table ORDER BY attribut2;` Affiche la table entière classée par ordre l'attibut2 croissant.

`SELECT * FROM table GROUP BY attribut3;` Affiche la table entière en regroupant les tuples dont l'attribut3 est identique.

`SELECT attribut3 FROM table GROUP BY attribut3;` Comme on ne souhaite que l'attribut3, affiche uniquement la liste des attribut3 sans doublon.

`SELECT DISTINCT attibut3 FROM table;` Idem. Affiche la liste des atttibut3 sans doublon

`SELECT attribut1, COUNT(*) FROM table;` Indique le nombre de lignes affiché

`SELECT attribut3, COUNT(*) FROM table GROUP BY attribut3;` Liste des attribut3 avec le nombre de fois qu'ils apparaissent dans la table.

`SELECT MAX(attribut1) FROM table;` Affiche la valeur maxi d'attribut1 (MIN() pour la + petite)

`SELECT AVG(attribut1) FROM table;` Affiche la valeur moyenne d'attribut1

Astuce : BINARY : Pour obliger les requêtes à respecter la « casse » (majuscule/minuscule) :

`SELECT * FROM table WHERE BINARY(attribut1) = "toTo";` On recherche exactement toTo

Pour ignorer la casse, on peut comparer des valeurs en les forçant en minuscule :

`SELECT * FROM table WHERE LOWER(ville) LIKE "aix%"` On recherche les villes commençant par Aix

REQUETES IMBRIQUEES:

`SELECT * FROM table WHERE attribut1 = (SELECT MAX(attribut1) FROM table);`

Affiche les tuples dont l'attribut1 à la valeur maximum (imbrications de select). Dans ce cas, le 2eme SELECT ne doit renvoyer qu'une seule valeur.

`SELECT * FROM table WHERE attribut1 = ANY (SELECT MAX(attribut1) FROM table GROUP BY attribut2);`

Le mot clé ANY permet de répéter automatiquement le 1er SELECT pour chaque attribut1 renvoyé par le 2eme SELECT.

REQUETES AVEC JOINTURE :

```
SELECT champ1Table1, champ1Table2 FROM Table1, Table2
                                        WHERE Table1.Champ2Table1 = Table2.Champ2Table2;
```

Affiche 2 attributs venant de 2 tables différentes, mais les tuples de ces attributs ont en point commun un champ de valeur identique.

Si la jointure automatique n'apporte pas le résultat voulu, utiliser les mots clés LEFT JOIN ou RIGHT JOIN :

```
SELECT champ1Table1, champ1Table2 FROM Table1
                            LEFT JOIN Table2 ON Table1.Champ2Table1 = Table2.Champ2Table2;
```

4.3 La commande INSERT

La commande INSERT ajoute des *tuples* dans une table. La syntaxe générale est :

```
INSERT INTO table (attribut1,attibut2, …) VALUES ('valeur1', 'valeur2', …);
```

Si les valeurs sont données dans le même ordre que les attributs, on peut écrire :

```
INSERT INTO table  VALUES (valeur1, valeur2);
```

On indiquera NULL pour les attributs de type AUTO_INCREMENT, puisque c'est le SGBD qui affectera une valeur.

La fonction LAST_INSERT_ID() retourne le dernier numéro AUTO_INCREMENT affecté : `SELECT LAST_INSERT_ID();`

4.4 Remplir une table avec les données d'une autre table

Exemple : On veut copier les colonnes CP et VILLE de la TABLE1 vers TABLE2, dans les colonnes CODEP et NOMVILLE sans les doublons :

```
INSERT INTO TABLE2 (NOMVILLE, CODEP) SELECT distinct VILLE, CP FROM TABLE1;
```

4.5 La commande UPDATE

```
UPDATE table SET attribut1="nouvelle_valeur" WHERE attribut2="xxxx";
```

La nouvelle valeur remplacera l'ancienne dans les *tuples* remplissant la condition WHERE. Cela peut donc affecter plusieurs *tuples*.

Si la modification ne doit concerner qu'un *tuple* précis, on utilisera un test sur la clé primaire (ou unique) de la table dans la clause WHERE.

4.6 La commande DELETE

```
DELETE from table WHERE attribut1="Valeur";
```

Tous les *tuples* remplissant la condition WHERE seront supprimés.

Si la modification ne doit concerner qu'un *tuple* précis, on utilisera un test sur la clé primaire (ou unique) de la table dans la clause WHERE.