- Création d'une table nommée « matable » dans la base de données :

```
CREATE TABLE matable (
  attribut1 type,
  attribut2 type,
  ...
)
```

- Suppression de la table :

```
DROP TABLE matable
```

- Insertion d'un enregistrement dans une table :

```
INSERT INTO
    matable
VALUES (
    'valeur 1',
    'valeur 2',
    ...
)
```

- Modification d'une ou plusieurs valeurs d'attributs :

```
UPDATE
    matable
SET
    attribut1 = 'valeur',
    attribut2 = 'valeur'
WHERE
    condition
```

- Suppression d'un enregistrement dans une table :

```
DELETE FROM
    matable
WHERE
    condition
```

- Sélection d'un ou plusieurs attributs dans une table :

```
SELECT
    attribut1,
    attribut2
FROM
    matable
```

- Classification des données selon un ou plusieurs attributs :

```
SELECT
    *
FROM
    matable
ORDER BY
    attribut
```

À noter que « * » désigne la totalité des attributs.

- Suppression des doublons sur un attribut :

```
SELECT DISTINCT
    attribut
FROM
    matable
```

Exemple : à l'aide d'une interface graphique (DB Browser pour SQLite ou PhpMyAdmin pour MySQL), il est facile de créer une base de données nommée « gestion » ainsi que deux tables :

→ « restaurants » d'attributs : id, nom, rue, cp, ville, email, tel et id_gerant ;

→ « gerants » d'attributs : id, nom, prenom, email et tel.

- Pour changer l'adresse email de l'enregistrement dont l'id est 1, on écrira :

```
UPDATE
    restaurants
SET
    email = "chezmomo@free.fr"
WHERE
    id = 1;
```

- Pour supprimer tous les enregistrements concernant la ville de Nantes dans la table « restaurants », on écrira :

```
DELETE FROM
    restaurants
WHERE
    ville = "Nantes"
```

4.3.2 - Jointures

Il existe trois types de jointures : INNER (jointure interne), OUTER (jointure externe) et CROSS (produit cartésien).

4.3.2.1 - Jointure interne

La syntaxe pour une jointure interne est la suivante :

```
SELECT
    champ1,
    ...
FROM
    table1
INNER JOIN
    table2
ON
    table1.champA = table2.champB
```

Cette requête joint les deux tables `table1` et `table2` et on choisit un critère de sélection avec « ON ». La condition qui vient après peut être une égalité ou autre chose.

4.3.2.2 - Jointures externes

La syntaxe pour une jointure externe est la suivante :

```
SELECT
    champ1,
    ...
FROM
    table1
[LEFT|RIGHT] OUTER JOIN
    table2
ON
    table1.champA = table2.champB
```

Si aucune correspondance n'est trouvée pour un attribut, la valeur « NULL » est donnée pour l'attribut. Ainsi, si l'on dispose de deux tables `clients` (avec les attributs *nom_client* et *ville_client*) et `distributeurs` (avec les attributs *nom_distrib* et *ville_distrib*), la requête :

```
SELECT
    clients.nom_client,
    distributeurs.nom_distrib,
    clients.ville_client
FROM
    clients
LEFT OUTER JOIN
    distributeurs
ON
    clients.ville_client = distributeurs.ville_distrib
```

donne une table comme ci-dessous :

nom_client	nom_distrib	ville_client
Jean Naimart	CARREFOUR	BORDEAUX
Anne Onime	LECLERC	BORDEAUX
Paul Tronc	AUCHAN	PARIS
Chloé Opieu	NULL	MARSEILLE

La dernière ligne signifie que l'enregistrement correspondant à Chloé Opieu n'est mis en relation avec aucun enregistrement de la table `distributeurs`.

Maintenant, si on teste la requête :

```
SELECT
    clients.nom_client ,
    distributeurs.nom_distrib ,
    clients.ville_client
FROM
    clients
RIGHT OUTER JOIN
    distributeurs
ON
    clients.ville_client = distributeurs.ville_distrib
```

s'afficheront uniquement les entrées de la seconde table qui satisfont la condition, avec éventuellement des « NULL » si certains enregistrements de la table `distributeurs` ne sont mis en relation avec aucun enregistrement de la table « clients ».

Astuce : quand les noms des tables sont « longs », on peut créer des raccourcis. Par exemple, la requête précédente peut s'écrire :

```
SELECT
    C.nom_client , D.nom_client , C.ville_client
FROM
    clients C
RIGHT OUTER JOIN
    distributeurs D
ON
    C.ville_client = D.ville_distrib
```

Remarquez que nous avons mis un raccourci tout de suite après avoir fait appel à une table : « distributeurs D » (donc ici, la lettre D est le raccourci pour cette table) et « clients C ».

4.3.2.3 - Produit cartésien

Pour effectuer un produit cartésien, on utilisera une syntaxe telle que :

```
SELECT
    *
FROM
    clients
CROSS JOIN
    distributeurs
```

Le CROSS JOIN met en relation *tous* les enregistrements de la table de gauche avec tous ceux de la table de droite sans rechercher aucune égalité ni logique particulière.

4.3.3 - Opérations d'agrégation

4.3.3.1 - Ajouter

Dans certains cas, il est nécessaire d'ajouter toutes les entrées d'une colonne (d'un attribut). On pourra le faire à l'aide de la fonction d'agrégation SUM.

Exemple : considérons la table « commandes » suivante :

commandes		
id_commande	id_client	total
1	12	125
2	7	35
3	8	50

Tableau 7.18 – Table « commandes »

La requête :

```
SELECT SUM(montant) AS somme
FROM commandes
```

retourne :

somme
210

où « 210 » représente le total des valeurs de l'attribut `montant` des commandes. Il n'est pas indispensable de spécifier d'alias (`AS`), mais c'est assez pratique dans les requêtes plus complexes. Dans notre exemple, nous avons choisi l'alias « somme » pour désigner la somme obtenue.

4.3.3.2 - Compter

Dans certains cas, savoir combien une colonne comporte d'entrées s'avère utile. On le fera avec la fonction d'agrégation COUNT.

Exemple : prenons à nouveau la table « commandes » du tableau 7.18 de l'exemple précédent.

Si on considère la requête suivante :

```
SELECT COUNT(*) AS nb
FROM commandes
WHERE montant < 100
```

alors, elle affiche :

nb
2

car il y a deux enregistrements où la valeur de l'attribut « montant » est strictement inférieure à 100.

4.3.3.3 - Effectuer une moyenne

Cela se fait à l'aide de la fonction d'agrégation AVG (abréviation de *average*, qui signifie « moyenne » en anglais).

Exemple : en reprenant la table représentée dans le tableau 7.18, la requête :

```
SELECT AVG(montant) AS
    moy
FROM commandes
```

affiche :

moy
66.6667

où « 66,6667 » représente donc la moyenne des commandes.

4.3.3.4 - Maximum et minimum

Connaître le maximum et le minimum des valeurs d'un attribut se fait avec les fonctions d'agrégation MAX et MIN.

Exemple : toujours avec la table représentée par le tableau 7.18, la requête :

```
SELECT
    MIN(C.montant) AS minimum ,
    MAX(C.montant) AS maximum
FROM commandes C
```

affiche :

minimum	maximum
25	125

4.3.4 - Complément : créer une vue

Il sera de temps en temps utile de nommer la table obtenue à l'issue d'une requête ; c'est ce que l'on appelle *créer une vue* de la requête. Prenons par exemple la requête précédente qui permet de calculer le maximum et le minimum, et créons une vue que l'on nomme « mavue »

```
CREATE VIEW mavue AS
    SELECT
        MIN(C.montant) AS minimum ,
        MAX(C.montant) AS maximum
    FROM commandes C
```

On peut ainsi utiliser la table (la vue) dans une autre requête, par exemple pour calculer l'écart entre le maximum et le minimum :

```
SELECT maximum - minimum FROM mavue
```