# Le langage SQL

## *SQL*

Lors de la définition du modèle relationnel, Edgar F. Codd a développé un ensemble d'opérations mathématiques servant à manipuler les objets, que l'on nomme **algèbre relationnelle**. C'est ces opérations que reposent le langage SQL (*Structured Query Language*) utilisé par les SGBD.
Il existe plusieurs variantes de ce langage. Nous utiliserons SQLite qui présente l'avantage de ne pas nécessiter de serveur et fonctionne avec un utilisateur unique.
La création des tables ne fait pas partie du programme de NSI, mais voici les commandes que nous utiliserons pour nos exemples :

```sql
CREATE TABLE Realisateurs (
    real INTEGER NOT NULL,
    nom TEXT,
    PRIMARY KEY (real)
);

CREATE TABLE Films (
    film INTEGER NOT NULL,
    titre TEXT NOT NULL,
    real INTEGER,
    annee INTEGER,
    duree INTEGER,
    PRIMARY KEY (film),
    FOREIGN KEY (real) REFERENCES Realisateurs (real)
);
```

On peut voir que les types des attributs sont bien indiqués, ainsi que les clefs primaires et étrangères. Il y a également des contraintes de domaine pour les valeurs qui ne peuvent pas être vides.

## *Insertion de valeurs*

Pour insérer des valeurs dans une table, il faut utiliser la commande :

```sql
INSERT INTO nom_table VALUES (val1, val2, ..., valN);
```

Pour obtenir les tables de la feuille sur les bases de données, il faut faire :

```sql
INSERT INTO Realisateurs VALUES (1, "Stanley Kubrick");
INSERT INTO Realisateurs VALUES (2, "Ridley Scott");
INSERT INTO Realisateurs VALUES (3, "Katsuhiro Otomo");

INSERT INTO Films VALUES (1, "2001: A Space Odyssey", 1, 1968, 149);
INSERT INTO Films VALUES (2, "Alien", 2, 1979, 117);
INSERT INTO Films VALUES (3, "Blade Runner", 2, 1982, 117);
INSERT INTO Films VALUES (4, "Akira", 3, 1988, 124);
```

Le language n'est pas sensible à la casse, mais dans la pratique on met les commandes SQL en majuscule pour bien les distinguer du reste.

**Exercice 1 :** Pour les questions suivantes, on prendra le plus petit entier non nul disponible comme clef primaire.

1) Rajouter le réalisateur *James Cameron* dans la table `Realisateurs`.
2) Rajouter le film *Aliens* qu'il a réalisé, qui dure 107 minutes et qui est sorti en 1984.

## *Projection*

La **projection** consiste à créer une nouvelle relation à partir des attributs des autres relations. Elle se fait à l'aide de la commande **SELECT**, qui est la commande la plus utilisée en SQL. Il y a deux syntaxes de base :

```
SELECT * FROM nom_table;
SELECT attribut FROM nom_table;
SELECT attribut1, attribut2, ..., attributN FROM nom_table;
```

La première syntaxe permet d'obtenir tous les attributs de la table alors que la seconde permet de n'en garder que quelques uns.
Voici quelques exemples de commandes avec le résultat correspondant :

**SELECT * FROM Realisateurs**

| réal | nom |
|---|---|
| 1 | Stanley Kubrick |
| 2 | Ridley Scott |
| 3 | Katsuhiro Ôtomo |

**SELECT titre,annee FROM Films**

| titre | annee |
|---|---|
| 2001: A Space Odyssey | 1968 |
| Alien | 1979 |
| Blade Runner | 1982 |
| Akira | 1988 |

**SELECT duree FROM Films**

| duree |
|---|
| 149 |
| 117 |
| 117 |
| 124 |

On peut remarquer que dans le dernier exemple, il y a plusieurs lignes identiques. En effet, **SELECT** peut s'affranchir de la contrainte de l'algèbre relationnelle interdisant les tuples identiques. En utilisant **SELECT DISTINCT** `duree` **FROM** `Films`, on obtient des lignes toutes différentes, comme ci-contre.

| duree |
|---|
| 149 |
| 117 |
| 124 |

**Exercice 2 :** Donner les commandes SQL permettant d'obtenir :

1) Tous les noms de réalisateurs.
2) Tous les titres de films avec leur durée.
3) Toutes les informations sur les films.
4) Tous les titres de films, sans doublons.

La projection n'est pas si intéressante si on ne l'associe pas avec la **sélection** en faisant :

```
SELECT ... FROM nom_table WHERE condition;
```

Les conditions possibles s'expriment à l'aide des opérateurs suivants :

- Comparaisons : = < > <= >= <> (différent)
- Opérateurs logiques : **AND** **OR** **NOT**
- Encadrement : **BETWEEN**
- Motifs de chaînes : **LIKE**, avec "_" pour un caractère quelconque et "%" pour aucun ou plusieurs. Les motifs de chaînes ne sont pas sensibles à la casse, contrairement aux égalités.

Voici quelques exemples de requètes :

```
# Les films sortis aprèis 1980 :
SELECT * FROM Films WHERE annee > 1980;
```

```sql
# Les films commençant par "A" :
SELECT * FROM Films WHERE titre LIKE "A%";

# Les films durant entre 120 et 150 minutes :
SELECT * FROM Films WHERE duree BETWEEN 120 AND 150;

# Les films ne durant pas 117 minutes ou avec un "i" dans le titre :
SELECT * FROM Films WHERE duree <> 117 OR titre LIKE "%i%";
```

**Exercice 3 :** Donner les commandes SQL permettant d'obtenir :

1) La liste des films sortis avant 2000.
2) La liste des films de moins de 90 minutes ou sortis après 1980.
3) La liste des réalisateurs dont le prénom commence par S et le nom de famille commence par K. On supposera que le seul espace dans le nom du réalisateur se trouve entre le prénom et le nom.

## *Ordonner les résultats*

Lorsqu'il y a de nombreux résultats, il est possible de les trier selon une ou plusieurs colonnes en utilisant **ORDER BY**.
Voici quelques exemples :

```sql
# Les films par ordre alphabétique :
SELECT titre FROM Films ORDER BY titre;

# Les films par ordre décroissant de durée :
SELECT titre FROM Films ORDER BY duree DESC;

# Les films de moins de 120 minutes
# par ordre croissant de sortie puis par ordre décroissant de durée :
SELECT titre FROM Films
WHERE duree < 120
ORDER BY annee ASC, duree DESC;
```

Les mots-clefs **ASC** et **DESC** permettent de dire s'il faut trier selon l'ordre croissant ou décroissant par rapport à la colonne indiquée juste avant. Pour l'ordre croissant, **ASC** est optionnel.

**Exercice 4 :** Donner la commande SQL permettant d'obtenir les titres et les durées des films sorties en 2000 par ordre croissant de durée.

## *La jointure interne*

La **jointure** consiste à créer une nouvelle relation en reprenant les attributs de plusieurs tables.
Il existe plusieurs types de jointures, mais nous n'étudierons que la jointure interne qui ne garde que les lignes vérifiant une condition particulière. Elle se fait en utilisant :

```sql
SELECT ... FROM table1 JOIN table2 ON condition;
```

En général, on va vouloir relier la clef primaire d'une table avec une clef étrangère d'une autre table. Si ces attributs ont le même nom, on les distingue en écrivant "*table.attribut*". Il est également possible de rajouter une sélection après la jointure.

- Tous les films et leurs réalisateurs :

```
SELECT * FROM Films JOIN Realisateurs ON Films.real = Realisateurs.real;
```

| film | titre | real | annee | duree | real | nom |
|---|---|---|---|---|---|---|
| 1 | 2001: A Space Odyssey | 1 | 1968 | 149 | 1 | Stanley Kubrick |
| 2 | Alien | 2 | 1979 | 117 | 2 | Ridley Scott |
| 3 | Blade Runner | 2 | 1982 | 117 | 2 | Ridley Scott |
| 4 | Akira | 3 | 1988 | 124 | 3 | Katsuhiro Ôtomo |

- Tous les titres des films de Ridley Scott :

```
SELECT titre FROM Films
JOIN Realisateurs ON Films.real = Realisateurs.real
WHERE nom = "Ridley Scott";
```

| titre |
|---|
| Alien |
| Blade Runner |

Il est possible de faire des jointures en utilisant **WHERE**, mais il est préférable de séparer les conditions de jointure, des conditions de sélection, pour faciliter la lecture.

**Exercice 5 :** Donner la commande SQL permettant d'obtenir le nom de tous les réalisateurs ayant sorti un film en 2000.

## *La mise à jour*

Pour changer un ou des attributs d'un tuple déjà défini, on peut utiliser l'expression suivante :

```
UPDATE nom_table SET attribut1=val1, ..., attributN=valeurN WHERE condition;
```

Pour changer la longueur de Blade Runner, il faut utiliser :

```
UPDATE Films SET duree = 120 WHERE titre = "Blade Runner";
```

**Exercice 6 :** Donner la commande SQL permettant de changer le nom du réalisateur *Joseph McGinty Nichol* par *McG*.

## *La suppression*

Pour supprimer certaines lignes d'une table, il faut faire :

```
DELETE FROM nom_table WHERE condition;
```

Ainsi, pour supprimer Ridley Scott des réalisateurs, on doit utiliser :

```
DELETE FROM Realisateurs WHERE nom = "Ridley Scott";
```

Selon le logiciel utilisé, et les réglages de la base, vous pourrez avoir un message d'erreur avec cette commande. Puisque la table `Films` utilise une clef étrangère provenant de `Realisateurs`, supprimer un réalisateur qui apparaît dans `Films` ne respecte pas les contraintes d'intégrité. Il faut alors supprimer ses films avant de supprimer le réalisateur. Avec d'autres logiciels, au contraire, ses films seront supprimés en même temps que lui. Dans tous les cas, l'intégrité de la base est garantie.

**Exercice 7 :** Donner la commande SQL permettant de supprimer le film *Indiana Jones 4* de la base.