

Langage SQL

SQL : structured query langage = langage de requêtes structuré est un langage informatique de dialogue avec une base de données relationnelle (un fichier qui organise les données sur une ou plusieurs tables).

Une **requête** est une question posée à une base de données. Nous allons voir comment sont écrites les requêtes de base en SQL. Une requête est la traduction d'une relation écrite en algèbre relationnelle.

L'algèbre relationnelle est un outil créé par le chercheur E. Codd (1970) pour manipuler les tables dans le modèle relationnel. Ses principales opérations sont : **SELECTIONNER** certaines colonnes (Projection) ou certaines lignes (selection) d'une table, mais aussi de combiner 2 tables.

Chaque opération SQL prend en entrée une ou deux tables et renvoie une table.

Voyons dans un premier temps les requêtes utiles pour **CREER**, **SUPPRIMER**, **REMPLIR** et **MODIFIER** la table

1.1 CREATE TABLE "nom_table" (schema relation);

Créons la table :

Nom_eleve	Classe	Math	Anglais	Info
Kevin	209	16	17	18
Zoe	209	5	15	17
Toto	210	4	6	NULL

```
1 CREATE TABLE IF NOT EXISTS Table_notes (  
2     Nom_eleve      TEXT PRIMARY KEY,  
3     Classe INTEGER,  
4     Math          REAL,  
5     Anglais       REAL,  
6     Info          REAL);
```

CREATE TABLE va permettre d'indiquer le schéma d'une relation, avec ses **Attributs** et **Domaines**. Les types (ou domaines) les plus courants sont parmi :

- INTEGER, entier positif ou nul
- TEXT, CHAR(n), chaîne de caractères
- DATE, format de date du type YYYY-MM-DD
- REAL, une valeur décimale

On peut ajouter des contraintes (constraints) en paramètre : PK (Primary Key), UNIQUE (dans une colonne, les valeurs doivent être uniques à chaque ligne), AUTOINCREMENT, NOT NULL (doit comprendre une valeur), DEFAULT 'Not Applicable' (la valeur mise par défaut si vide).

```
1 CREATE TABLE etat_civil (  
2     id INTEGER PRIMARY KEY,  
3     name TEXT UNIQUE,  
4     date_of_birth TEXT NOT NULL,  
5     date_of_death TEXT DEFAULT 'Not Applicable';)
```

1.2 INSERT INTO "nom_table" (Attributs) VALUES (n-uplet1), (n-uplet2), ...;

On **insère** les valeurs dans la table :

```
1 INSERT INTO Table_notes VALUES
2 ("Doe", 16, 17, 18),
3 ("Zoe", 12, 15, 17),
4 ("Toto", 4, 6, NULL);
```

("Doe", 16, 17, 18) est un argument contenant les valeurs à insérer.

Remarque : si un enregistrement n'existe pas, il faudra mettre NULL

1.3 UPDATE <table> SET <attribut = valeur> WHERE <attribut = valeur>

Pour **modifier des valeurs**

```
1 UPDATE Table_notes
2 SET Math = 18
3 WHERE Nom = "Doe";
```

1.4 DELETE FROM <table> WHERE <condition>

Supprimer des lignes

```
1 DELETE FROM Table_notes
2 WHERE Math <= 14;
```

1.5 ALTER TABLE <table> ADD COLUMN <Attribut Domaine>

pour **modifier la table**

```
1 ALTER TABLE Table_notes ADD COLUMN Biologie INTEGER;
```

pour **Renommer une colonne** :

```
1 ALTER TABLE nom_table
2 RENAME COLUMN ancien_nom TO nouveau_nom
```

pour **Renommer la table** :

```
1 ALTER TABLE table_name
2 RENAME TO new_table_name
```

1.6 DROP TABLE <table>

Supprimer une table

```
1 DROP TABLE Table_notes;
```

Les CLAUSES SQL

Les mots-clefs SELECT, FROM, WHERE, GROUP BY, HAVING et ORDER BY sont appelés des clauses.

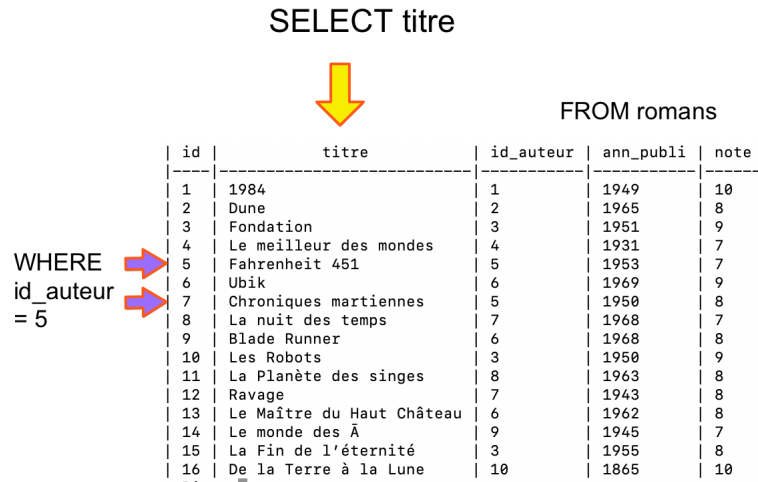


FIGURE 1 – clauses SQL sur la tables des romans

2.1 SELECT <attributs> FROM nom_table

SELECT est la commande qui retourne la table selon les **attributs** choisis. Il s'agit d'une **Projection** comme type d'*opération relationnelle*.

Une **projection** est un type de sélection où seulement une partie des attributs des tables choisies est retenue pour le résultat.

FROM est la *table concernée*.

Pour avoir toutes les colonnes, faire :

```
1 SELECT * FROM Table_notes
```

ou bien seulement certaines :

```
1 SELECT Math, Nom_eleve AS Nom FROM Table_notes
```

On peut renommer une colonne avec l'alias AS, et choisir un nom plus pratique.

SELECT DISTINCT retourne les éléments de manière **unique**.

Et limiter le nombre de lignes à 10 par exemple :

```
1 SELECT * FROM Table_notes LIMIT 10
```

2.2 WHERE <condition>

La clause WHERE permet de SELECTIONNER des lignes à partir de conditions dans les requêtes, mais aussi de réaliser des JOINTURES (voir plus loin)

```
1 SELECT Nom_eleve
```

```
2 FROM Table_notes
3 WHERE Math > 14 OR Info >= 18;
```

Exercice : Représenter la table `Table_notes` (En-tête et contenu de la colonne) qui sera retournée par cette requête.

2.3 AVG SUM MIN MAX COUNT

Ce sont des fonctions d'**agregation**, qui retournent un resultat évalué sur la colonne issue de la SELECTION

Exemple1 : Combien d'élèves ont plus de 15 en math dans la table `Table_notes` ?

```
1 SELECT COUNT(*) FROM Table_notes WHERE Math > 15
```

Exemple2 : Combien de romans de l'auteur `id_auteur = 5` y-a-t-il dans la table `romans` ?

```
1 SELECT count(*) FROM romans WHERE id_auteur = 5
```

- COUNT(*) va compter le nombre de lignes pour la selection donnée
- SUM(attribut) va faire la somme pour les valeurs de l'attribut et pour la selection donnée
- AVG(attribut) calcule la moyenne
- MIN(attribut) retourne la valeur min, MAX(attribut) la valeur max

Exercice : Ecrire la requête qui calcule la moyenne de math pour tous les élèves de la table `Table_notes`

2.4 GROUP BY <uneTable.attribut1>

Permet de regrouper des lignes les unes avec les autres.

Pour regrouper des lignes d'une table selon un attribut, il faut que ces lignes aient la **même** valeur pour cet attribut.

GROUP BY s'utilise avec des **agrégats**.

Exemple : Quel total de notes figure dans la table `romans` pour chaque auteur ?

```
1 SELECT SUM(note) FROM romans GROUP BY id_auteur
```

Exercice : Ecrire une requête qui calcule la moyenne des notes obtenues par écrivain sur la table `Romans`

2.5 ORDER BY <colonne> ASC [DESC]

Triions les lignes par ordre croissant des notes en math :

```
1 SELECT Nom, Math
2 FROM Table_notes
3 ORDER BY Math ASC;
```

Sinon par ordre descendant : ORDER BY Math DESC

Exercice : Représenter le tableau retourné par la requête ci-dessus.

Exercice : Ecrire une requête qui calcule la moyenne des notes obtenues par écrivain sur la table Romans et qui classe ces auteurs par ordre de note moyenne décroissante.

Remarque : On peut aussi rajouter une *clause* sur le nombre de lignes retournées avec LIMIT <nombre> Très pratiques pour de grandes tables.

2.6 WHERE <colonne> LIKE <motif> et WHERE <colonne> IN <tuple>

2.6.1 LIKE

On peut faire une recherche dans une table selon certains motifs. Pour cela, on utilise WHERE ... LIKE <suivi de caractères spéciaux>.

Caractères spéciaux :

- le caractère % représente 0, 1 ou plus de caractères
 - %s permet d'obtenir toutes chaînes qui finissent par s, mais aussi le caractère unique s.
- le caractère _ représente un caractère unique
 - ro_e permet d'obtenir robe ou bien rose

Exercice : Dans la relation **Eleves** de schéma ((prenom, TEXT), (nom, TEXT), (nais, DATE), (rue, TEXT), (numero, INTEGER), (CP, INTEGER), (ville, TEXT), (email, TEXT)) : 1. Définir une requête qui permet d'obtenir tous les élèves nés en 2002 2. Définir une requête qui permet d'obtenir tous les élèves habitant dans le département des Alpes Maritimes. 3. Définir une requête qui permet d'obtenir tous les élèves ayant un compte e-mail chez l'opérateur laposte (du type pseudo@laposte.net)

2.6.2 IN

Pour faire une recherche dans une table selon une liste de critères, on utilise le mot clé IN On peut par exemple rechercher les élèves qui habitent Nice, Saint-Laurent ou Cagnes-sur-mer dans la table Eleves :

```
1 SELECT nom, prenom
2 FROM Eleves
3 WHERE ville IN ('Nice', 'Saint-laurent', 'Cagnes-sur-mer')
```

Remarque : Cette syntaxe peut être ajoutée à l'opérateur NOT, ce qui fait NOT IN.

On peut avoir besoin d'extraire un groupe de lettres du texte. On utilise alors la fonction substr. Les paramètres attendus sont :

- nom de l'attribut
- rang du premier caractere (commence à 1)
- rang du dernier caractere.

Exemple : Recherche des élèves dont les noms commencent par 'a', 'b' ou 'c' :

```
1 SELECT nom, prenom
2 FROM Eleves
3 WHERE substr(nom, 1, 1) IN ('a', 'b', 'c')
```

Exercice : Ecrire une requête sur la table Eleves qui retourne les noms des élèves dont la famille habite dans les départements ('06', '83', '05')

2.7 Jointure

Pour éviter les redondances, on a vu qu'il était préférable de répartir les données sur plusieurs tables, chaque table modélisant UNE seule entité.

La jointure permet de mettre en relation plusieurs tables, par l'intermédiaire des liens qui existent en particulier entre la **clé primaire** de l'une et la **clé étrangère** de l'autre. La jointure est une opération de sélection car elle permet de ne retenir que les enregistrements pour lesquels la valeur de la **clé primaire** d'une table correspond à la valeur de la **clé étrangère** d'une autre table.

Product		
Num	Label	Category
1	Produit 1	1
2	Produit 2	1
3	Produit 3	2

Category	
Num	Label
1	Categorie 1
2	Categorie 2

SELECT * FROM Product, Category

Product.Num	Product.Label	Product.Category	Category.Num	Category.Label
1	Produit 1	1	1	Categorie 1
1	Produit 1	1	2	Categorie 2
2	Produit 2	1	1	Categorie 1
2	Produit 2	1	2	Categorie 2
3	Produit 3	2	1	Categorie 1
3	Produit 3	2	2	Categorie 2

La requête retourne un nombre de lignes égal au produit du nombre de lignes de chaque table

SELECT * FROM Product, Category WHERE
Product.Category = Category.Num

Product.Num	Product.Label	Product.Category	Category.Num	Category.Label
1	Produit 1	1	1	Categorie 1
2	Produit 2	1	1	Categorie 1
3	Produit 3	2	2	Categorie 2

La requête retourne une table issue de la JOINTURE des 2 tables

FIGURE 2 – exemple de jointure entre 2 tables

Pour joindre 2 tables complètes :

- 1ere méthode :

```
1 SELECT *
2 FROM orders
3 JOIN customers
```

```
4 ON orders.customer_id = customers.customer_id;
```

à la 4e ligne : on declare comment les 2 tables sont combinées : on veut faire correspondre la colonne `id_customers` de la table `orders` avec celle `id_customers` de la table `customers`.

Comme le nom d'une colonne va se retrouver dans de nombreuses tables, on utilisera la syntaxe : `table_name.column_name`

- 2e méthode :

```
1 SELECT *
2 FROM orders, customers
3 WHERE orders.customer_id = customers.customer_id;
```

Exercice : Voici les tables `order`, `subscriptions` et `customers` d'un service d'abonnements à des magazines, à New York :

orders (a table with information on each magazine purchase)

order_id	customer_id	subscription_id	purchase-date
1	2	3	2017-01-01
2	2	2	2017-01-01
3	3	1	2017-01-01

subscriptions (a table that describes each type of subscription)

subscription_id	description	price_per_month	length
1	Politics Magazine	5	12 months
2	Fashion Magazine	10	6 months
3	Sports Magazine	7	3 months

customers (a table with customer names and contact information)

customer_id	customer_name	address
1	John Smith	123 Main St
2	Jane Doe	456 Park Ave
3	Joe Schmo	798 Broadway

Question 1 : Quelle est la clé primaire de la table `order`. Quelles colonnes de la table `order` sont des clés étrangères, en vue du lien possible avec les autres relations ?

Question 2 : Ecrire la requête qui renvoie la table suivante, par jointure des tables `orders` et `customers` :

order_id	'customer_name'
1	Jane Doe
2	Jane Doe

order_id	'customer_name'
3	Joe Schmo

Donner la requête pour chacune des 2 méthodes vues plus haut.

Question 3 : chercher les objets de la table jointe selon la valeur d'une clé : Que renvoie la requête suivante ?

```
1 SELECT *
2 FROM orders
3 JOIN subscriptions
4 ON orders.subscription_id = subscriptions.subscription_id
5 WHERE subscriptions.description = 'Fashion Magazine';
```

Partie 3

Exercices

3.1 Connaissances

3.1.1 Quel outil permet à un utilisateur de rechercher une information spécifique dans une base de données ?

- Une recherche
- Une requête
- Une clause
- Un filtre

3.1.2 Qu'est ce que SQL

- Un langage permettant de construire des bases de données
- Un langage permettant de d'effectuer des recherches dans une base de données
- Un langage permettant de déboguer des bases de données
- Un langage permettant de manipuler des fichiers Excel

3.1.3 Quelle commande SQL permet d'ajouter une nouvelle ligne ?

- INSERT
- INSERT INTO
- ADD
- PUT

3.1.4 Quelle commande permet d'extraire des lignes d'une table ?

- GET
- OPEN
- EXTRACT
- SELECT

3.1.5 Quelle commande permet d'extraire TOUTES les lignes de la table élève ?

- SELECT [all] FROM Eleves
- SELECT * .Eleves
- SELECT ALL Eleves
- SELECT * FROM Eleves

3.1.6 Quelle clause SQL permet de sélectionner certaines lignes de la table ?

- HAVING
- SORT
- SELECT
- WHERE

3.1.7 Les jointures SQL sont utilisées pour combiner :

- les colonnes d'une table
- deux tables seulement
- plusieurs tables
- les lignes d'une table

3.1.8 Représenter l'architecture du modèle client-serveur incluant une base de données et une application web.

Partie 4

Exercice de Bac 2022 metropole1 : (Exercice 2)

Cet exercice porte sur les bases de données.

On pourra utiliser les mots clés SQL suivants : SELECT, FROM, WHERE, JOIN, ON, INSERT, INTO, VALUES, UPDATE, SET, AND.

Nous allons étudier une base de données traitant du cinéma dont voici le schéma relationnel qui comporte 3 relations :

- la relation individu (id_ind, 'nom, prenom, naissance)
- la relation realisation (id_rea, titre, annee, type)

- la relation `emploi` (`id_emp`, `description`, `#id_ind`, `#id_rea`)

Les clés primaires sont **en gras** et les clés étrangères sont précédées d'un **#**.

Ainsi `emploi.id_ind` est une clé étrangère faisant référence à `individu.id_ind`.

Voici un extrait des tables `individu` et `realisation` :

extrait de <code>individu</code>				extrait de <code>realisation</code>			
<code>id_ind</code>	<code>nom</code>	<code>prenom</code>	<code>naissance</code>	<code>id_rea</code>	<code>titre</code>	<code>annee</code>	<code>type</code>
105	'Hulka'	'Daniel'	'01-06-1968'	105	'Casino Imperial'	2006	'action'
403	'Travis'	'Daniel'	'10-03-1968'	325	'Ciel tombant'	2012	'action'
688	'Crog'	'Daniel'	'07-07-1968'	655	'Fantôme'	2015	'action'
695	'Pollock'	'Daniel'	'24-08-1968'	950	'Mourir pour attendre'	2021	'action'

FIGURE 3 – extrait tables

4.1 Question 1

On s'intéresse ici à la récupération de données dans une relation.

a A. Écrire ce que renvoie la requête ci-dessous :

```
1 SELECT nom, prenom, naissance
2 FROM individu
3 WHERE nom = 'Crog';
```

B. Fournir une requête SQL permettant de récupérer le titre et la clé primaire de chaque film dont la date de sortie est strictement supérieure à 2020.

4.2 Question 2

Cette question traite de la modification de relations.

A. Dire s'il faut utiliser la requête 1 ou la requête 2 proposées ci-dessous pour modifier la date de naissance de Daniel Crog. Justifier votre réponse en expliquant pourquoi la requête refusée ne pourra pas fonctionner.

Requête 1 :

```
1 UPDATE individu
2 SET naissance = '02-03-1968'
3 WHERE id_ind = 688 AND nom = 'Crog' AND prenom = 'Daniel';
```

Requête 2 :

```
1 INSERT INTO individu
2 VALUES (688, 'Crog', 'Daniel', '02-03-1968');
```

B. Expliquer si la relation `individu` peut accepter (ou pas) deux individus portant le même nom, le même prénom et la même date de naissance.

4.3 Question 3

Cette question porte sur la notion de clés étrangères.

- A. Recopier sur votre copie les demandes ci-dessous, dans leur intégralité, et les compléter correctement pour qu'elles ajoutent dans la relation emploi les rôles de Daniel Crog en tant que James Bond dans le film nommé 'Casino Impérial' puis dans le film 'Ciel tombant'.

```
1 INSERT INTO emploi
2 VALUES (5400, 'Acteur(James Bond)', ... );
3 INSERT INTO emploi
4 VALUES (5401, 'Acteur(James Bond)', ... );
```

- B. On désire rajouter un nouvel emploi de Daniel Crog en tant que James Bond dans le film 'Docteur Yes'.

Expliquer si l'on doit d'abord créer l'enregistrement du film dans la relation realisation ou si l'on doit d'abord créer le rôle dans la relation emploi.

4.4 Question 4

Cette question traite des jointures.

- A. Recopier sur votre copie la requête SQL ci-dessous, dans son intégralité, et la compléter de façon à ce qu'elle renvoie le nom de l'acteur, le titre du film et l'année de sortie du film, à partir de tous les enregistrements de la relation emploi pour lesquels la description de l'emploi est 'Acteur(James Bond)'.

```
1 SELECT ...
2 FROM emploi
3 JOIN individu ON ...
4 JOIN realisation ON ...
5 WHERE emploi.description = 'Acteur(James Bond)';
```

- B. Fournir une requête SQL permettant de trouver toutes les descriptions des emplois de Denis Johnson (Denis est son prénom et Johnson est son nom). On veillera à n'afficher que la description des emplois et non les films associés à ces emplois.

Partie 5

TP Exoplanètes

Le site <http://exoplanet.eu/catalog/> référence toutes les exoplanètes découvertes à ce jour.

Voici un extrait de la table planetes_es :

Planet	Mass (M_{Jup})	Radius (R_{Jup})	Period (day)	a (AU)	e	i (deg)	Ang. dist. (arcsec)	Discovery
Kepler-89 e	0.11	0.585	54.32031	0.3046	0.019	89.76	—	2013
Kepler-89 d	0.334	1.005	22.342989	0.1684	0.022	89.871	—	2013
Kepler-89 c	0.049	0.385	10.423648	0.1013	0.43	88.36	—	2013
Kepler-89 b	0.033	0.153	3.743208	0.05119	0.25	89.3	—	2013
TOI-201 b	0.42	1	52.97818	0.3	0.28	—	—	2021
GJ 486 b	0.00887	0.11643	1.467119	0.01734	0.05	88.4	—	2021
TOI-1685 b	0.01189	0.152	0.6691403	—	—	84.74	—	2021
KMT-2020-BLG-0414 b	0.00381	—	—	1.79	—	—	—	2021
KMT-2020-BLG-0414 c	23.3	—	—	0.16	—	—	—	—
LHS 1478 b	0.00733	0.1106	1.9495378	0.01848	—	87.45	—	2021
OGLE-2018-BLG-1428L b	0.77	—	—	3.3	—	—	—	2021
RIK 72 b	56.1	3.06	97.84	—	0.131	—	—	2019
GJ 740 b	—	—	2.37756	0.029	0.24	—	—	2021
HD 76920 b	—	—	415.89	1.187	0.8782	—	—	2017
HD 5278 b	0.0245	0.2186	14.33902	0.12	0.08	89.27	—	2021

FIGURE 4 – premières entrées de la table *planetes_es*

La table contient les *Attributs* suivants : (seuls certains sont représentés dans l'image ci-dessus)

- Planet
- Mass
- Radius
- Period
- Discovery : année de découverte
- Detection_type : méthode de détection (RadialVelocity, PrimaryTransit, Microlensing, Imaging,...)
- Molecules : molécules détectées
- Star_name : nom de l'étoile
- Sp.type : type de l'étoile (O, B, A, F, G, K M)
- Star_distance : distance entre l'étoile et le système solaire en parsecs (1pc = distance depuis laquelle une UA est vue sous un angle de 1")
- Star_mass : masse de l'étoile en masses solaires
- Star_radius : rayon de l'étoile en rayons solaires
- Star_age : âge de l'étoile en milliards d'années
- Star_teff : température de l'étoile en Kelvins

Ecrire les requêtes permettant de retourner :

- 5.1 Toutes les données relatives à l'étoile *Kepler-89 b*
- 5.2 Tous les noms des planètes dont la masse est plus de 10 fois supérieure à celle de Jupiter (Indiquer quelles sont ces planètes dans l'extrait de la table)
- 5.3 Les planètes ainsi que le nom de l'étoile hôte, dont l'étoile est située à moins de 40 pc de la Terre.
- 5.4 Les noms des 20 étoiles les plus proches de la Terre où des planètes ont été détectées.
- 5.5 Nombre total de planètes dans la table (utiliser COUNT(*))
- 5.6 Nombre total d'étoiles dans la table (utiliser COUNT(DISTINCT ...))
- 5.7 Table des planètes qui n'ont été détectées ni par "Radial Velocity", ni par "Primary Transit"
- 5.8 Le nombre de planètes qui n'ont été détectées ni par "Radial Velocity", ni par "Primary Transit"
- 5.9 Moyenne des distances entre le Soleil et les différentes étoiles de la table (utiliser AVG). *Attention à ne compter qu'une seule fois chacune des étoiles.*
- 5.10 Table des planètes où le type de l'étoile est G ... (utiliser LIKE "G%")
- 5.11 Nom des planètes et distances au soleil, où de l'eau a été détectée (utiliser LIKE)