

Chapitre 10 - Bases de données - SQL

Le développement de traitements informatiques nécessite la manipulation de données de plus en plus nombreuses. Leur organisation et leur stockage constituent un enjeu essentiel de performance.

1- POURQUOI A-T-ON INVENTE LES BASES DE DONNEES ? :

Pour stocker de manière **persistante** des données on peut utiliser des fichiers. Suivant le format de ce fichier, les données seront stockées de manière plus ou moins organisées et pourront ainsi être récupérées plus ou moins rapidement.

fichier au format .txt ou .csv	fichier au format .xml	fichier au format .json
 nom,prenom,domicile DUPUIS,Louis,Bordeaux MARTIN,Guerre,Marseille EINSTEIN,Albert,Princeton	 <pre><annuaire> <personne> <nom>DUPUIS</nom> <prenom>Louis</prenom> <domicile> Bordeaux</domicile> </personne> <personne> <nom>MARTIN</nom> <prenom>Guerre</prenom> <domicile> Marseille</domicile> </personne> <personne> <nom>EINSTEIN</nom> <prenom>Albert</prenom> <domicile>Princeton</domicile> </personne> </annuaire></pre>	 <pre>{ "annuaire": { "personne": [{ "nom": "DUPUIS", "prenom": "Louis", "domicile": " Bordeaux" }, { "nom": "MARTIN", "prenom": "Guerre", "domicile": " Marseille" }, { "nom": "EINSTEIN", "prenom": "Albert", "domicile": " Princeton" }] }</pre>

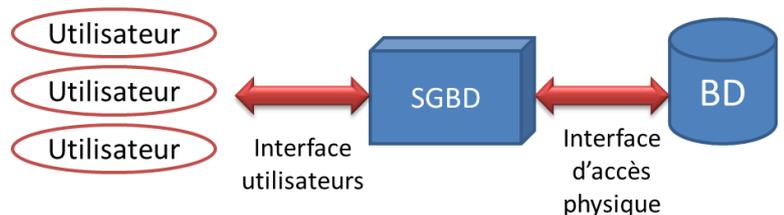
Il est possible dans tous les langages de programmation, d'extraire les données contenues dans ces fichiers pour les placer temporairement dans des listes ou dictionnaires afin de pouvoir les manipuler. Si certaines de ces données sont modifiées, on met à jour le fichier avec des requêtes d'écriture. Cette façon de faire convient lorsque les données sont peu nombreuses. Elle est clairement insuffisante pour répondre aux attentes actuelles :

- souvent le volume des données est gigantesque (voir l'article : [16000 malades oubliés à cause d'Excel](#));
- les requêtes de recherche peuvent être complexes et donc lourdes à traiter;
- les données peuvent être simultanément utilisées par différents programmes ou différents utilisateurs (exemples : sites marchands, réservations en ligne, etc.)

Il est donc nécessaire d'utiliser des solutions plus performantes. L'utilisation de **bases de données relationnelles** est aujourd'hui la solution la plus répandue.

2- LA BDD ET LE SYSTEME DE GESTION DE BASE DE DONNEES (SGBD) ? :

Dans une base de données, l'information est stockée dans des fichiers, mais ceux-ci ne sont en général pas lisibles par un humain : ils nécessitent l'utilisation d'un logiciel appelé **Système de Gestion de Bases de Données** (abrégié SGBD) pour les exploiter (écrire, lire ou encore modifier les données).



Les SGBD sont des logiciels, fonctionnant pour la grande majorité, en mode client/serveur. Parmi les SGBD les plus connus, on peut citer :

- dans le domaine du libre :
 - MariaDB / MySQL
 - PostgreSQL
 - SQLite (qui ne fonctionne pas sur le modèle client/serveur, toute la BDD est stockée dans un fichier)
- dans le monde propriétaire :
 - Oracle Database
 - IBM DB2
 - Microsoft SQL Server.

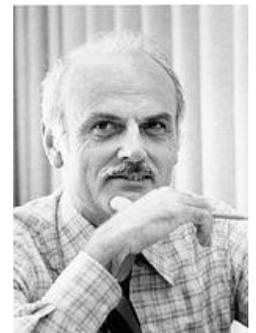
Les SGBD sont conçus pour gérer plusieurs millions, voire milliards d'enregistrements de manière fiable et sécurisée. Les accès en lecture ou écriture sont réalisés très rapidement. Les requêtes correspondantes sont écrites dans un langage spécifique qui s'appelle le SQL (**S**tructured **Q**uery **L**anguage)

Plusieurs personnes ou applications peuvent avoir besoin d'accéder aux informations contenues dans une base données en même temps. Cela peut parfois poser problème, notamment si les 2 personnes désirent modifier la même donnée au même moment (on parle d'accès concurrent), ce qui arrive très souvent : réservation ou achat en ligne d'un même objet, paiements en ligne à partir ou en direction d'un même compte bancaire, etc. Les SGBD arrivent à gérer ce type de situation en utilisant la notion de file d'attente.

3- LE MODELE RELATIONNEL :

Les bases de données sont basées sur ce qu'on appelle le **modèle relationnel**. Il s'agit d'un modèle *logique* défini en 1970 par l'informaticien britannique [Edgard F. Codd](#) (1923-2003), lors de ses travaux chez IBM. Il a reçu le prix Turing en 1981.

Dans ce modèle, on modélise les **relations** existantes entre plusieurs informations et on les ordonne entre elles.



4- EXEMPLE : RELATION SCOOTER



Un site nommé *branlyscoot.fr*, propose un service de location de scooter. Les données relatives à la flotte de scooters et aux clients sont contenues dans une base de données nommée *scootloc*. Cette bdd est hébergée et gérée par un SGBD du serveur qui héberge le site.

α. CREATION DE LA RELATION SCOOTER :

Dans cette bdd, les différents scooters sont répertoriés dans ce que l'on appelle une TABLE ou une RELATION. Cette relation est nommée ici *scooter*. Elle possède 4 attributs : *immatriculation*, *marque*, *annee* et *motorisation*.

Les différents attributs peuvent être mis en évidence dans un diagramme comme celui donné ci-contre.

Le symbole  définit l'attribut *immatriculation* comme **clé primaire**. Cet attribut devra être **unique** pour chacun des scooters répertoriés dans cette table.

Scooter	
immatriculation 	TEXT
marque	TEXT
annee	INT
motorisation	INT

Ces attributs peuvent aussi être mis en évidence par une représentation appelée *schéma relationnel* et donnée ci-dessous. L'attribut qui fera office de **clé primaire** y est souligné :

```
scooter ( immatriculation TEXT , marque TEXT , annee INTEGER , motorisation INTEGER)
```

Pour créer cette table dans la bdd, on dialogue avec le SGBD du serveur du site, en **langage SQL** (*Structured Query Language*).

La requête SQL pour créer cette relation (ou *table*) *scooter* est donnée ci-contre :

```
CREATE TABLE scooter (
    immatriculation INTEGER PRIMARY KEY ,
    marque TEXT,
    annee INTEGER,
    motorisation INTEGER,
);
```

```
CREATE TABLE ..... (
    .....
);
```

b. REMPLISSAGE DE LA RELATION SCOOTER :

Cette relation va permettre de stocker de manière persistante des informations sur toute la flotte de scooters proposés en location. On donne ci-contre un extrait des **enregistrements** contenus dans cette relation *scooter* :

<i>immatriculation</i> 	<i>marque</i>	<i>annee</i>	<i>motorisation</i>
'GV823AV'	'Yamaha'	2024	125
'FZ154BB'	'Peugeot'	2021	50
'GA101EB'	'Honda'	2023	50
'GC098CC'	Honda	2024	125

Concrètement, pour **insérer** ces données dans la table *scooter*, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
INSERT INTO .... VALUES ....
```

```
INSERT INTO scooter VALUES
('GV823AV', 'Yamaha', 2024, 125),
('FZ154BB', 'Peugeot', 2021, 50),
('GA101EB', 'Honda', 2023, 50),
('GC098CC', 'Honda', 2024, 125);
```

c. LECTURE DES DONNEES CONTENUES DANS LA TABLE SCOOTER :

Exemple 1 : « On veut récupérer toutes les données contenues dans la table *scooter* »

Concrètement, pour récupérer ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
SELECT ... FROM ....
```

```
SELECT * FROM scooter;
```

Le SGBD retourne alors généralement une liste de tuples :

```
('GV823AV', 'Yamaha', 2024, 125)
('FZ154BB', 'Peugeot', 2021, 50)
('GA101EB', 'Honda', 2023, 50)
('GC098CC', 'Honda', 2024, 125)
```

Exemple 2 : « On veut récupérer les attributs *immatriculation* et *motorisation* des scooters de marque *Honda* »

Concrètement, pour récupérer ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
SELECT immatriculation , motorisation FROM scooter
WHERE marque = 'Honda';
```

Le SGBD retourne les tuples suivants :

```
('GA101EB', 50)
('GC098CC', 125)
```

```
SELECT ..... FROM ....
WHERE ... = ....
```

Exemple 3 : « On veut récupérer les valeurs de tous les attributs pour les scooters dont l'immatriculation se termine par un 'B' »

Concrètement, pour récupérer ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
SELECT * FROM scooter
WHERE immatriculation LIKE '%B';
```

Le SGBD retourne les tuples suivants :

```
('FZ154BB', 'Peugeot', 2021, 50)
('GA101EB', 'Honda', 2023, 50)
```

```
SELECT ..... FROM ....
WHERE ... LIKE '%...%'
```

Exemple 4 : « On veut récupérer, sans aucun doublon, les marques des scooters de motorisation inférieure à 250 cm³ et d'année supérieure à 1990 , marques triées par ordre alphabétique décroissant »

Concrètement, pour récupérer ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
SELECT DISTINCT marque FROM scooter
WHERE motorisation < 250 AND annee > 1990
ORDER BY marque DESC ;
```

```
SELECT DISTINCT..... FROM ....
WHERE ... > .. AND ...
ORDER BY .... ASC
```

Le SGBD retourne les tuples suivants :

```
('Yamaha',)
('Peugeot',)
('Honda',)
```

Exemple 5 : « On veut récupérer tous les attributs, classés par année en ordre croissant. Si les années sont égales, on classe avec un ordre alphabétique décroissant sur l'immatriculation »

Concrètement, pour récupérer ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
SELECT * FROM scooter
ORDER BY marque ASC , annee DESC;
```

Le SGBD retourne les tuples suivants :

```
('GC098CC', 'Honda', 2024, 125)
('GA101EB', 'Honda', 2023, 49)
('FZ154BB', 'Peugeot', 2021, 50)
('GV823AV', 'Yamaha', 2024, 125)
```

```
SELECT ..... FROM ....
ORDER BY .... ASC , ... DESC
```

Exemple 6 : « On veut réaliser des calculs sur la valeur de certains attributs»

```
COUNT(..)
AVG(..)
SUM(..)
MIN(..)
MAX(..)
```

Concrètement, on dialogue avec le SGBD du serveur en donnant par exemple la requête SQL ci-contre :

```
SELECT COUNT(*), COUNT(DISTINCT annee),
        AVG(DISTINCT annee),MIN(motorisation),
        MAX(motorisation),SUM( annee)
FROM scooter
WHERE motorisation >= 50;
```

Le SGBD retourne le tuple suivant :

```
(4, 3, 2022.66666666666667, 50, 125, 8092)
```

d. MODIFICATION DU CONTENU DE LA TABLE SCOOTER :

Exemple 1 : « On veut modifier la valeur de l'attribut motorisation des enregistrements pour lesquels cet attribut est égal à 50 et l'attribut marque est égal à 'Honda' ».

```
UPDATE ...
SET .. = ..
WHERE .....
```

Concrètement, pour mettre à jour ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL

ci-contre :

```
UPDATE scooter SET motorisation = 49
WHERE motorisation = 50 AND marque = 'Honda'
```

Si on exécute ensuite la requête :

```
SELECT * FROM scooter;
```

Le SGBD retourne les tuples suivants :

```
('GV823AV', 'Yamaha', 2024, 125)
('FZ154BB', 'Peugeot', 2021, 50)
('GA101EB', 'Honda', 2023, 49)
('GC098CC', 'Honda', 2024, 125)
```

Exemple 2 : « On veut supprimer l'enregistrement du scooter immatriculé 'GC098CC'».

Concrètement, pour supprimer ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
DELETE FROM scooter
WHERE immatriculation = 'GC098CC';
```

```
DELETE FROM ...
WHERE ...
```

Si on exécute ensuite la requête :

```
SELECT * FROM scooter;
```

Le SGBD retourne les tuples suivants :

```
('GV823AV', 'Yamaha', 2024, 125)
('FZ154BB', 'Peugeot', 2021, 50)
('GA101EB', 'Honda', 2023, 49)
```

Point Cours : Les commandes SQL suivantes sont à connaître pour le bac Nsi :

- CREATE TABLE (.....)
- INSERT INTO VALUES
- SELECT ... FROM .. WHERE ...
 - Opérateurs après WHERE : < = > != LIKE % IN ...
 - Fonctions COUNT() MIN() MAX() AVG() SUM() SUM(DISTINCT ..)
- UPDATE ... SET WHERE
- DELETE FROM WHERE

Point Cours : Avec WHERE, on peut utiliser les opérateurs suivants :

- Opérateurs de comparaisons :

Opérateur	Description
=	Égal
!= ou <>	Différent
>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	Inférieur ou égal
IS NULL	Valeur est égale à NULL
IS NOT NULL	Valeur n'est pas égale à NULL

- Opérateurs logiques :

```
SQL
SELECT une_colonne FROM une_table WHERE une_condition AND une_autre_condition;
```

```
SQL
SELECT une_colonne FROM une_table WHERE une_condition OR encore_condition;
```

- Intervalles :

```
SQL
SELECT une_colonne FROM une_table WHERE une_colonne BETWEEN "valeur_1" AND "valeur_2";
```

- Comparaison à une liste de valeur :

```
SQL
SELECT une_colonne FROM une_table WHERE une_colonne IN ("valeur_1", "valeur_2", "valeur_3");
```

o Opérateurs LIKE :

Modèle	Commentaire
LIKE "a%"	Recherche toutes les chaînes de caractères qui commencent par le caractère a .
LIKE "%a"	Recherche toutes les chaînes de caractères qui terminent par le caractère a .
LIKE "%a%"	Recherche toutes les chaînes de caractères qui contiennent au moins un caractère a .
LIKE "a%"	Recherche toutes les chaînes de caractères qui commencent par le caractère a et terminent par le caractère b .
LIKE "a_"	Recherche toutes les chaînes de caractères de trois caractères qui commencent par le caractère a .
LIKE "_a%"	Recherche toutes les chaînes de caractères qui possèdent le caractère a en deuxième position .

5- EXEMPLE : RELATION *ENTRETIEN*



La base de données *scootloc* contient aussi une relation nommée *entretien* . Elle permet de gérer les travaux et réparations des différents scooters de la flotte.

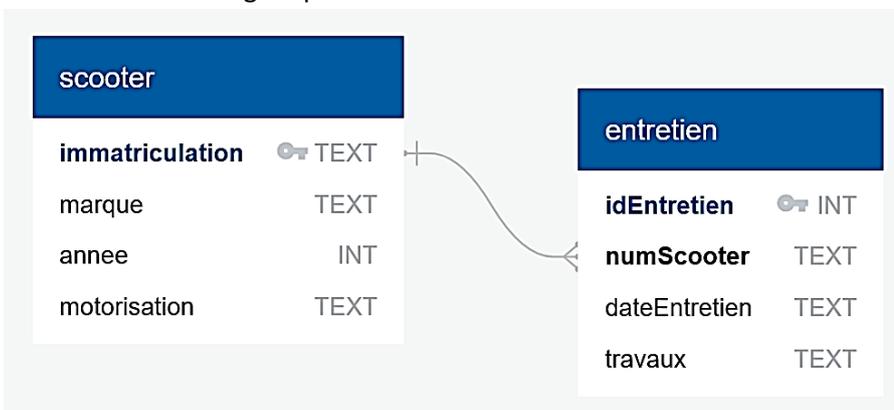
a. CREATION DE LA RELATION ENTRETIEN :

La table *entretien* possède 4 attributs : *idEntretien*, *numScooter*, *dateEntretien* et *travaux*.

Le schéma relationnel de cette table est le suivant :

```
entretien ( idEntretien INTEGER , #numScooter TEXT , dateEntretien TEXT , travaux TEXT )
```

Le symbole # définit l'attribut *numScooter* comme **clé étrangère**. On voit sur le diagramme ci-dessous que cette clé étrangère permet de mettre en relation la table *entretien* et la table *scooter*.



Concrètement, l'attribut *numScooter* sera le numéro d'immatriculation d'un des scooters de la flotte.

En créant cette contrainte de **clé étrangère**, le SGBD s'assurera que, pour tout nouvel enregistrement, l'attribut *numScooter* correspond à un des attributs *immatriculation* existant dans la table *scooter*.

La **clé primaire** de la relation *entretien* est l'attribut *idEntretien* . Cet attribut correspond à un numéro d'identification **unique** pour chaque enregistrement de cette table. Il n'était pas possible de définir l'attribut

numScooter comme clé primaire, car un même scooter peut se retrouver dans plusieurs enregistrements de cette table.

La requête SQL pour créer cette relation (ou table) *entretien* est donnée ci-contre :

```
CREATE TABLE entretien (
    idEntretien INTEGER PRIMARY KEY AUTOINCREMENT,
    numScooter TEXT,
    dateEntretien TEXT,
    travaux TEXT,
    FOREIGN KEY(numScooter) REFERENCES scooter(immatriculation)
);
```

La propriété

AUTOINCREMENT sur

l'attribut *idEntretien*, autorise le SGBD à définir lui-même cet attribut en définissant un nombre entier qui s'incrémente de 1 à chaque nouvel enregistrement.

b. REPLISSAGE DE LA RELATION ENTRETIEN :

Cette relation va permettre de stocker de manière persistante des informations sur les travaux d'entretien réalisés sur les scooters de la flotte. On donne ci-dessous un extrait des **enregistrements** de cette relation *entretien* :

<i>idEntretien</i> 	<i>#numScooter</i>	<i>dateEntretien</i>	<i>travaux</i>
1	'FZ154BB'	'12/11/2021'	'Remplacement bougie d'allumage'
2	'FZ154BB'	'12/11/2023'	'Entretien 10 000 km'
3	'GV823AV'	'03/04/2024'	'Changement pneu avant'

Concrètement, pour **insérer** ces données dans la table *scooter*, on dialogue avec le SGBD du serveur en donnant la requête

SQL ci-contre :

```
INSERT INTO entretien(numScooter,dateEntretien,travaux) VALUES
('FZ154BB','12/11/2021','Remplacement bougie d allumage'),
('FZ154BB','12/11/2023','Entretien 10 000 km'),
('GV823AV','03/04/2024','Changement pneu avant');
```

On aurait pu aussi donner la requête ci-contre qui elle, ne donne pas la main au SGBD pour définir

l'attribut *idEntier*.

```
INSERT INTO entretien VALUES
(17255, 'FZ154BB', '12/11/2021', 'Remplacement bougie d allumage'),
(28736, 'FZ154BB', '12/11/2023', 'Entretien 10 000 km'),
(39002, 'GV823AV', '03/04/2024', 'Changement pneu avant');
```

C. LECTURE DES DONNEES CONTENUES DANS LA TABLE ENTRETIEN :

Exemple 1 : « On veut récupérer toutes les données contenues dans la table *entretien* et celles concernant les scooters qui y sont répertoriés »

```
SELECT ___ FROM table1  
JOIN table2  
ON table1.___ = table2.___
```

Concrètement, pour récupérer ces données, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-dessous :

```
SELECT * FROM entretien  
JOIN scooter ON entretien.numScooter = scooter.immatriculation;
```

Le SGBD retourne les tuples suivants :

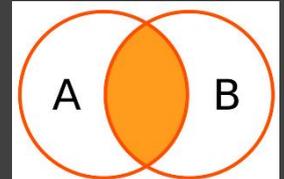
```
(1, 'FZ154BB', '12/11/2021', 'Remplacement bougie d allumage', 'FZ154BB', 'Peugeot', 2021, 50)  
(2, 'FZ154BB', '12/11/2023', 'Entretien 10 000 km', 'FZ154BB', 'Peugeot', 2021, 50)  
(3, 'GV823AV', '03/04/2024', 'Changement pneu avant', 'GV823AV', 'Yamaha', 2024, 125)
```

Si on présente ces données dans un tableau, cela donne :

	idEntretien	numScooter	dateEntretien	travaux	immatriculation	marque	annee	motorisation
1		1 FZ154BB	12/11/2021	Remplacement bougie d allumage	FZ154BB	Peugeot	2021	50
2		2 FZ154BB	12/11/2023	Entretien 10 000 km	FZ154BB	Peugeot	2021	50
3		3 GV823AV	03/04/2024	Changement pneu avant	GV823AV	Yamaha	2024	125

Point Cours :

Lorsque 2 tables A et B sont reliées par une contrainte de clé étrangère, la commande JOIN .. ON .. permet de réaliser une JOINTURE entre ces 2 tables afin de pouvoir récupérer les données qui apparaissent dans A ET dans B



La syntaxe SQL pour réaliser cette jointure est :

```
SELECT ___ FROM tableA  
JOIN tableB ON tableA.___ = tableB.___
```

Remarque : On peut aussi utiliser le terme INNER JOIN à la place de JOIN

Exemple 2 : « On veut récupérer toutes les données contenues dans la table *scooter* et celles contenues dans la table *entretien* »

On peut dialoguer avec le SGBD du serveur en donnant la requête SQL ci-dessous :

```
SELECT * FROM scooter  
JOIN entretien ON scooter.immatriculation = entretien.numScooter;
```

Le SGBD retourne alors les tuples suivants :

```
('FZ154BB', 'Peugeot', 2021, 50, 1, 'FZ154BB', '12/11/2021', 'Remplacement bougie d allumage')  
( 'FZ154BB', 'Peugeot', 2021, 50, 2, 'FZ154BB', '12/11/2023', 'Entretien 10 000 km')  
( 'GV823AV', 'Yamaha', 2024, 125, 3, 'GV823AV', '03/04/2024', 'Changement pneu avant')
```

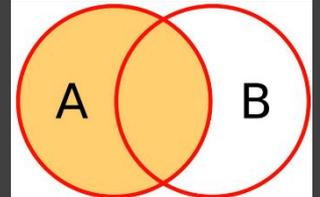
Ce qui donne le tableau suivant :

	immatriculation	marque	annee	motorisation	idEntretien	numScooter	dateEntretien	travaux
1	FZ154BB	Peugeot	2021	50	1	FZ154BB	12/11/2021	Remplacement bougie d allumage
2	FZ154BB	Peugeot	2021	50	2	FZ154BB	12/11/2023	Entretien 10 000 km
3	GV823AV	Yamaha	2024	125	3	GV823AV	03/04/2024	Changement pneu avant

On peut également réaliser une jointure différente en utilisant la commande LEFT JOIN :

Point Cours :

Lorsque 2 tables A et B sont reliées par une contrainte de clé étrangère, la commande LEFT JOIN .. ON .. permet de réaliser une JOINTURE entre ces 2 tables afin de pouvoir récupérer les données qui apparaissent dans A avec celles liées à B *si elles existent*. Si elles n'existent pas, les attributs correspondants prendront la valeur *NULL*.



La syntaxe SQL pour réaliser cette jointure est :

```
SELECT ___ FROM tableA
LEFT JOIN tableB ON tableA.___ = tableB.___
```

En donnant la requête SQL ci-contre :

```
SELECT ___ FROM tableA
LEFT JOIN tableB
ON tableA.___ = tableB.___
```

```
SELECT * FROM scooter
LEFT JOIN entretien
ON scooter.immatriculation = entretien.numScooter;
```

Le SGBD retourne les tuples suivants :

```
('GV823AV', 'Yamaha', 2024, 125, 3, 'GV823AV', '03/04/2024', 'Changement pneu avant')
('FZ154BB', 'Peugeot', 2021, 50, 1, 'FZ154BB', '12/11/2021', 'Remplacement bougie d allumage')
('FZ154BB', 'Peugeot', 2021, 50, 2, 'FZ154BB', '12/11/2023', 'Entretien 10 000 km')
('GA101EB', 'Honda', 2023, 49, None, None, None, None)
('GC098CC', 'Honda', 2024, 125, None, None, None, None)
```

Si on présente ces données dans un tableau, cela donne :

	immatriculation	marque	annee	motorisation	idEntretien	numScooter	dateEntretien	travaux
1	GV823AV	Yamaha	2024	125	3	GV823AV	03/04/2024	Changement pneu avant
2	FZ154BB	Peugeot	2021	50	1	FZ154BB	12/11/2021	Remplacement bougie d allumage
3	FZ154BB	Peugeot	2021	50	2	FZ154BB	12/11/2023	Entretien 10 000 km
4	GA101EB	Honda	2023	49	NULL	NULL	NULL	NULL
5	GC098CC	Honda	2024	125	NULL	NULL	NULL	NULL

d. UTILISATION D'UN ALIAS :

Dans le langage SQL il est possible d'utiliser des **alias** pour renommer temporairement une colonne ou une table dans une requête.

```
SELECT COUNT(*) AS nombre FROM scooter AS s  
JOIN entretien AS e ON s.immatriculation = e.numScooter;
```

COUNT(*) AS

Cette astuce est particulièrement utile pour faciliter la lecture des requêtes. Par exemple la requête ci-dessus, renverra le résultat ci-contre :

	nombre
1	3

6- EXEMPLE : RELATION *CLIENT*



La base de données *scootloc* contient encore une autre relation nommée *clients*. Elle permet de mémoriser les données liées aux clients qui se sont enregistrés sur le site.



a. CREATION DE LA RELATION *CLIENTS* :

La table *clients* possède 7 attributs : *idClient*, *nom*, *prenom*, *genre*, *age*, *email* et *pass*.

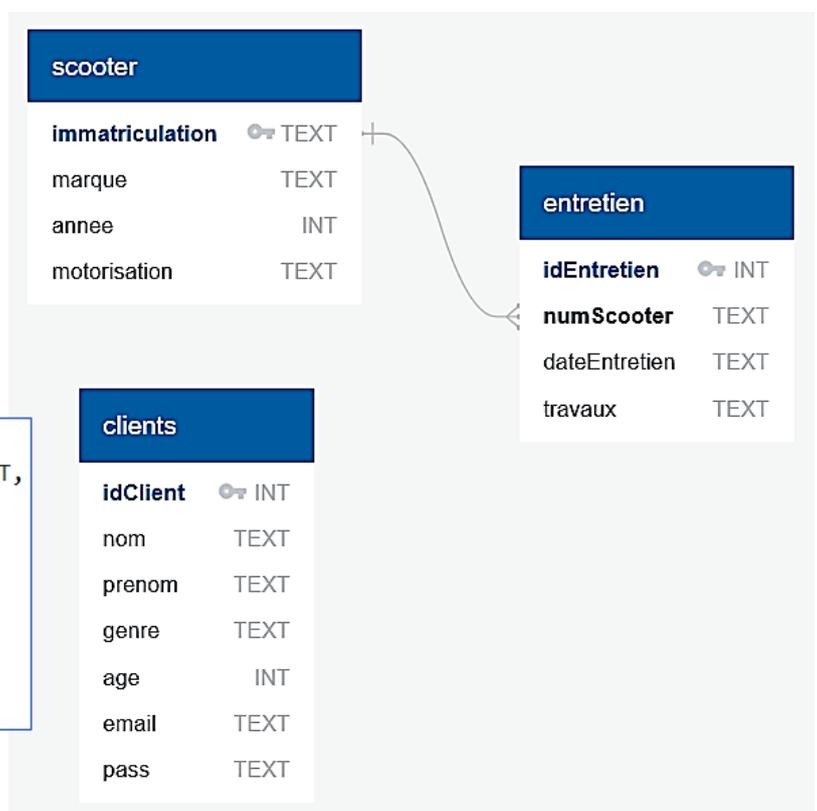
Le schéma relationnel de cette table est le suivant :

```
clients ( idClient INTEGER , nom TEXT , prenom TEXT , genre TEXT , age INTEGER , email INTEGER , pass TEXT )
```

La base de données *scootloc* contiendra ainsi 3 tables dont les attributs sont donnés sur le diagramme ci-contre :

La commande SQL pour créer cette table *clients* est donnée ci-dessous :

```
CREATE TABLE clients (  
  idClient INTEGER PRIMARY KEY AUTOINCREMENT,  
  nom TEXT,  
  prenom TEXT,  
  genre TEXT,  
  age INTEGER,  
  email TEXT,  
  pass TEXT  
);
```



b. REPLISSAGE DE LA RELATION CLIENTS :

On donne ci-contre un extrait des **enregistrements** de cette relation *clients* :

<i>idClient</i>	<i>nom</i>	<i>prenom</i>	<i>genre</i>	<i>age</i>	<i>email</i>	<i>pass</i>
1	'Chouhan'	'Jean'	'homme'	50	'jeanchouhan@gmail.com'	'leRenard69!'
2	'Durand'	'Louis'	'homme'	37	'louisdurand@gmail.com'	'foutMoiLaPaix'
3	'GranJean'	'Alice'	'femme'	45	'aliceg@wanadoo.fr'	' BouffonDu13'
4	'Bobet'	'Louison'	'homme'	27	'louisonbobet@free.fr'	'6paPi!!Llon9'

Concrètement, pour **insérer** ces données dans la table *scooter*, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
INSERT INTO clients(nom,prenom,genre,age,email,pass) VALUES
('Chouhan','Jean','homme',50,'jeanchouhan@gmail.com','leRenard69!'),
('Durand','Louis','homme',37,'louisdurand@gmail.com','foutMoiLaPaix'),
('GranJean','Alice','femme',45,'aliceg@wanadoo.fr','BouffonDu13'),
('Bobet','Louison','homme',27,'louisonbobet@free.fr','6paPi!!Llon9');
```

7- EXEMPLE : RELATION LOCATION



La base de données *scootloc* contient enfin une dernière relation nommée *location*. Elle permet de mémoriser les contrats de location en cours.

a. CREATION DE LA RELATION LOCATION :

La table *location* possède 3 attributs : *immatriculation*, *dateDebut*, *numClient*.

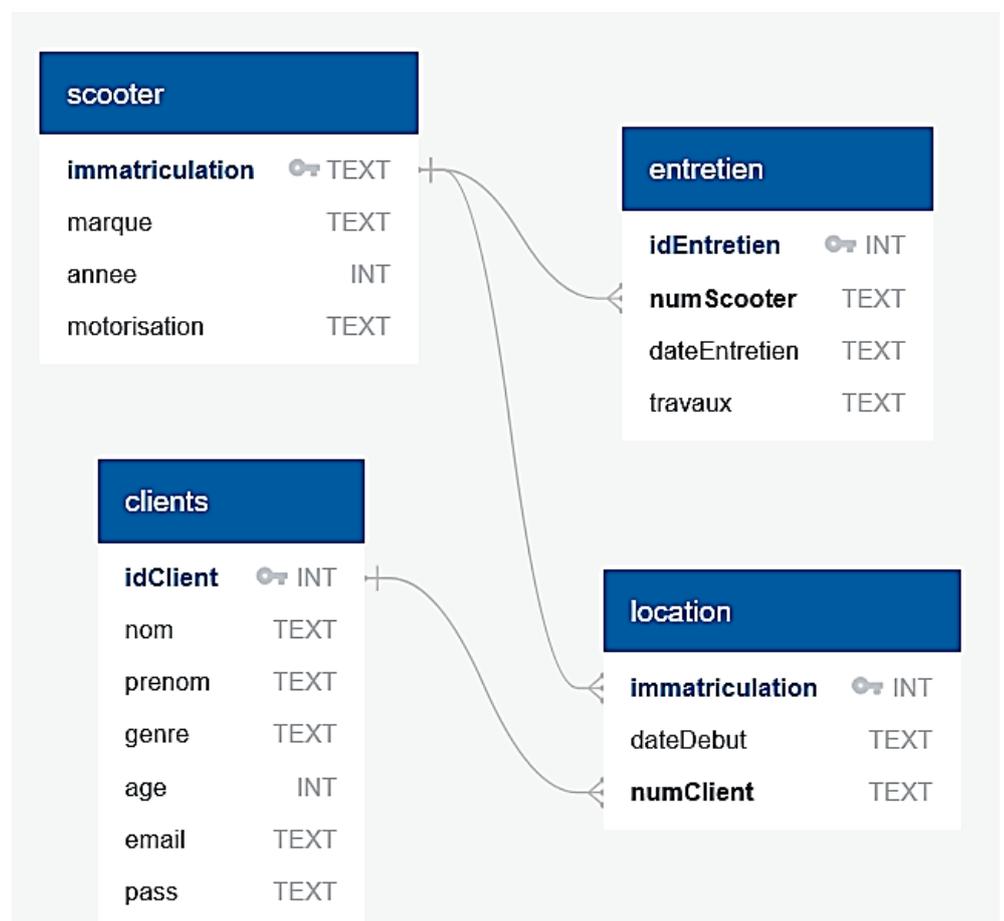
Le schéma relationnel de cette table est le suivant :

location (#immatriculation INTEGER , *dateDebut* TEXT , #*numClient* INTEGER)

La base de données *scootloc* contiendra finalement 4 tables dont les attributs sont donnés sur le diagramme ci-contre :

Remarques :

L'attribut *immatriculation* a été défini comme **clé primaire**. Cela garantit le fait qu'un même scooter ne pourra pas figurer 2 fois dans cette table. En effet un même scooter ne peut pas être loué en même temps à des clients différents.



La commande SQL qui permet de créer cette table est donnée ci-dessous :

```
CREATE TABLE location (
    immatriculation INTEGER PRIMARY KEY ,
    dateD = TEXT,
    numClient INTEGER,
    FOREIGN KEY(immatriculation) REFERENCES scooter(immatriculation),
    FOREIGN KEY(numClient) REFERENCES clients(idClient),
);
```

b. REPLISSAGE DE LA RELATION LOCATION :

On donne ci-contre un extrait des **enregistrements** de cette relation *clients* :

#immatriculation 	dateDebut	#idClient
'GV823AV'	'03/04/2024'	3
'FZ154BB'	'03/04/2024'	1

Concrètement, pour **insérer** ces données dans la table *scooter*, on dialogue avec le SGBD du serveur en donnant la requête SQL ci-contre :

```
INSERT INTO location VALUES
('GV823AV', '03/04/2024', 3),
('FZ154BB', '03/04/2024', 1);
```

c. LECTURE DES DONNEES CONTENUES DANS LA TABLE LOCATION :

Exemple : « On veut récupérer toutes les données concernant un contrat de location et contenues dans les différentes tables de la bdd »

En donnant la requête SQL ci-contre :

```
SELECT * FROM location
JOIN scooter ON location.immatriculation = scooter.immatriculation
JOIN clients ON location.numClient = clients.idClient;
```

Le SGBD retourne les tuples suivants :

```
('GV823AV', '03/04/2024', 3, 'GV823AV', 'Yamaha', 2024, 125, 3, 'GranJean', 'Alice', 'femme', 45, 'aliceg@wanadoo.fr', 'BouffonDu13')
('FZ154BB', '03/04/2024', 1, 'FZ154BB', 'Peugeot', 2021, 50, 1, 'Chouhan', 'Jean', 'homme', 50, 'jeanchouhan@gmail.com', 'leRenard69!')
```

Si on présente ces données dans un tableau, cela donne :

	immatriculation	dateD	numClient	immatriculation	marque	annee	motorisation	idClient	nom	prenom	genre	age	email	pass
1	GV823AV	03/04/2024	3	GV823AV	Yamaha	2024	125	3	GranJean	Alice	femme	45	aliceg@wanadoo.fr	BouffonDu13
2	FZ154BB	03/04/2024	1	FZ154BB	Peugeot	2021	50	1	Chouhan	Jean	homme	50	jeanchouhan@gmail.com	leRenard69!

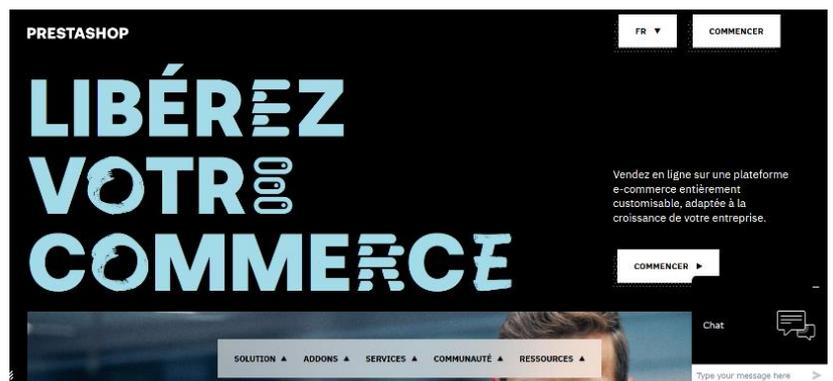
8- CONCLUSION :

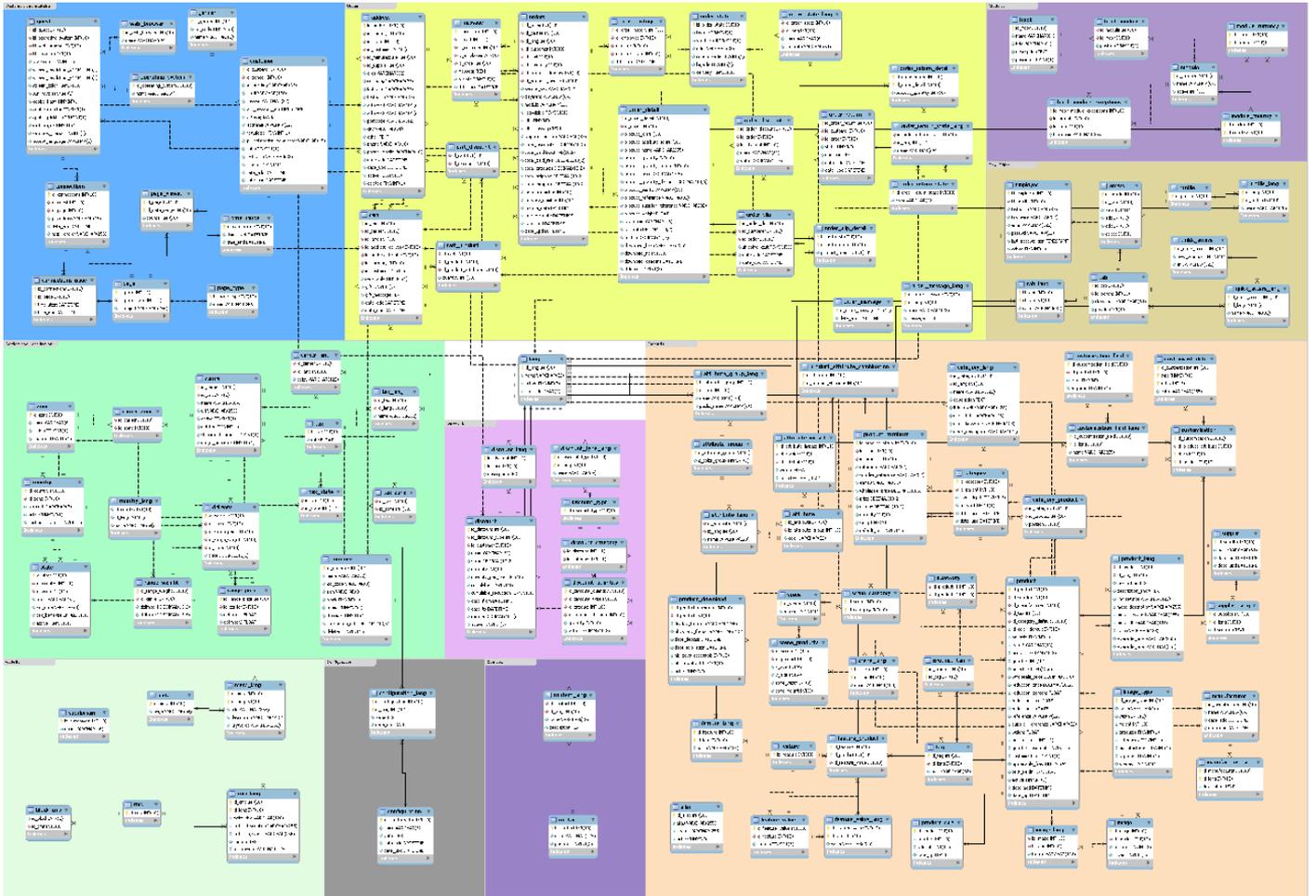
Les données contenues dans une base de données sont réparties dans différentes tables. Cette façon de procéder évite de réécrire plusieurs fois les mêmes données et permet de les récupérer rapidement.

On donne sur la page qui suit, la structure de la base de données qui gère le site de vente en ligne

<https://prestashop.fr/> . Chaque rectangle y représente une table. Les traits entre les différentes tables représentent les relations qu'elles ont entre elles.

Les blocs de couleurs mettent en avant les univers des tables : produits, clients, commandes, transporteurs, etc.





Les **clés primaires** définies dans chacune des tables permettent d'avoir un attribut dont la valeur est **unique** par rapport à tous les enregistrements de la table. Les **clés étrangères** permettent de lier les tables entre elles.

Point Cours :

- Il est possible sur une table, de définir comme clé primaire non pas un attribut particulier, mais un tuple de plusieurs attributs de cette table. En procédant ainsi, le SGBD ne tolèrera que des enregistrements pour lesquels ce tuple est UNIQUE.
- Si une table B est liée à une table A par une clé étrangère. Si on supprime dans A un enregistrement qui est référencé dans B, le SGBD provoquera une erreur. Il faut commencer par supprimer l'enregistrement en question, dans B.