

Base de données

Chapitre 5 : Le langage SQL DML

Joel Cavat

2022

Objectifs

- Familiarisation avec le langage SQL
- Manipulation d'une base de données
 - Requêtes d'extraction de données
 - Requêtes de modification (insertion, mise à jour et suppression)
- Injections SQL

Langage SQL

Le langage SQL (Structured Query Language) est un langage de requêtes déclaratif servant à manipuler des bases de données relationnelles. Il est inspiré de l'**algèbre relationnelle**.

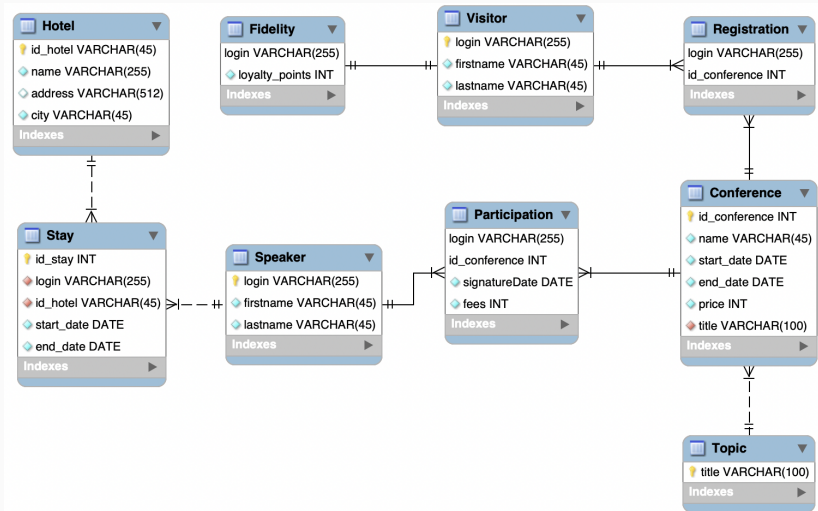
Le langage SQL est divisé en quatre catégories:

- **DML (Data Manipulation Language)**
 - SELECT, INSERT, UPDATE, DELETE
- **DDL (Data Definition Language)**
 - CREATE TABLE, CREATE VIEW, CREATE TRIGGER...
- **DCL (Data Control Language)**
 - GRANT, REVOKE
- **TCL (Transaction Control Language)**
 - COMMIT, ROLLBACK

Quatre opérations de base

- Extraction (SELECT)
- Ajout (INSERT)
- Suppression (DELETE)
- Modification (UPDATE)

Contexte : diagramme relationnel



Commande SELECT

SELECT

Syntaxe générale (simplifiée)

```
1 SELECT [ DISTINCT ] select_expression
2 [ FROM table_references ]
3 [ WHERE where_condition ]
4 [ GROUP BY col_name ]
5 [ HAVING where_condition ]
6 [ ORDER BY col_name ]
7 [ LIMIT [offset, ] row_count ]
```


Sélection et projection

Une expression simple retourne une table

```
1 SELECT 1+1, "COUCOU" -- retourne <2, COUCOU>
```

Lister tous les visiteurs

```
1 SELECT *  
2 FROM Visitor;
```

Sélection et projection

Lister toutes les conférences concernant le thème “databases”

$$\sigma_{title=databases}(Conference)$$

```
1 SELECT *
2 FROM Conference
3 WHERE title = "databases";
```

- L'égalité IS est *null-safe*:

```
1 SELECT 0.1 = null; -- retourne null
2 SELECT 0.1 IS null; -- retourne 0
```

Sélection et projection

```
1 SELECT *  
2 FROM Conference  
3 WHERE title = "databases" OR title = "blockchain";
```

```
1 SELECT *  
2 FROM Conference  
3 WHERE title IN ("databases", "blockchain");
```

Sélection et projection

```
1 SELECT *
2 FROM Conference
3 WHERE title NOT IN ("databases", "blockchain");
```

```
1 SELECT *
2 FROM Conference
3 WHERE title IS NOT "databases";
```

- <> et != sont équivalents
- <> respecte la norme ANSI/ISO
- en SQLITE, <=> n'existe pas (pourtant ANSI). Remplacé par IS.

Comparaison de chaînes de caractères

```
1 SELECT *  
2 FROM Visitor  
3 WHERE login LIKE "%1%";
```

- _ fait correspondre un caractère unique
- % fait correspondre un nombre arbitraire de caractères

Comparaison de chaînes de caractères

- Par défaut, non sensible à la casse
 - GLOB permet d'y être sensible (sqlite)

```
1 SELECT *  
2 FROM Visitor  
3 WHERE login GLOB "jca";
```

Comparaison de chaînes de caractères

Expressions régulières

```
1 SELECT *  
2 FROM Visitor  
3 WHERE login REGEXP "[0-9]$";
```

La projection

$$\pi_{\{login\}}(Visitor)$$

```
1 SELECT login  
2 FROM Visitor;
```

$$\pi_{\{name, title, startDate, endDate\}}(Conference)$$

```
1 SELECT name, title, start_date, end_date  
2 FROM Conference;
```


La projection

$$\pi_{\{name, startDate, endDate\}}(\sigma_{title = Cloud}(Conference))$$

```
1 SELECT name, start_date, end_date
2 FROM Conference
3 WHERE title = "Cloud";
```

```
1 SELECT firstname, lastname, Visitor.login,  
2         Fidelity.login, loyalty_points  
3 FROM Fidelity, Visitor;
```

Produit cartésien

firstname	lastname	login	login	loyalty_points
Alfonso	Gillibrand	agillibrandc	agillibrandc	10
Alfonso	Gillibrand	agillibrandc	asustin1d	20
Alfonso	Gillibrand	agillibrandc	bstempg	40
Alfonso	Gillibrand	agillibrandc	cbalassaj	20
Alfonso	Gillibrand	agillibrandc	ekeasey15	10
Alfonso	Gillibrand	agillibrandc	eshierr	50
Alfonso	Gillibrand	agillibrandc	ggrigolond	20
Alfonso	Gillibrand	agillibrandc	gmattiazzi18	30
Alfonso	Gillibrand	agillibrandc	hasaaf11	20
Alfonso	Gillibrand	agillibrandc	hcoldrickm	50
Alfonso	Gillibrand	agillibrandc	hlarway1b	0
Alfonso	Gillibrand	agillibrandc	lclewa	50
Alfonso	Gillibrand	agillibrandc	memeneyk	40
Alfonso	Gillibrand	agillibrandc	mgodbold19	10
Alfonso	Gillibrand	agillibrandc	ocoutts13	200
Alfonso	Gillibrand	agillibrandc	ptrinder5	200
Alfonso	Gillibrand	agillibrandc	scullip1c	10
Alfonso	Gillibrand	agillibrandc	skonkeh	0
Alfonso	Gillibrand	agillibrandc	tdwerryhouse7	10
Adela	Pendock	apendockq	agillibrandc	10
Adela	Pendock	apendockq	asustin1d	20
...
...

Jointures

Jointure interne

Sélectionne des enregistrements qui ont des valeurs correspondantes dans deux tables

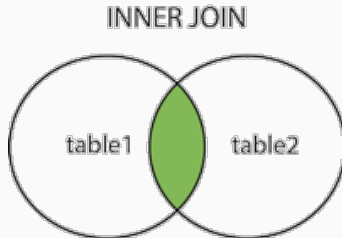


Figure 1: Jointure interne

Equi-jointure

```
1 SELECT firstname, lastname, Visitor.login,  
2         loyalty_points  
3 FROM Fidelity  
4 INNER JOIN Visitor ON Fidelity.login = Visitor.login;
```

Jointure interne

Jointure avec clause USING

```
1 SELECT firstname, lastname, Visitor.login,  
2         loyalty_points  
3 FROM Fidelity INNER JOIN Visitor USING (login);
```

Jointure naturelle

- Si aucune ambiguïté entre les références
- **non autorisée dans ce cours** (source de bugs et non ANSI)

```
1 SELECT firstname, lastname, Visitor.login,  
2         loyalty_points  
3 FROM Fidelity NATURAL JOIN Visitor;
```

Jointure

firstname	lastname	login	loyalty_points
Alfonso	Gillibrand	agillibrandc	10
Alair	Sustin	asustin1d	20
Burg	Stemp	bstempg	40
Corey	Balassa	cbalassaj	20
Esme	Keasey	ekeasey15	10
Emlynne	Shier	eshierr	50
Geraldine	Grigolon	ggrigolond	20
Georgetta	Mattiazzi	gmattiazzi18	30
Harper	Asaaf	hasaaf11	20
Harriet	Coldrick	hcoldrickm	50
Hilario	Larway	hlarway1b	0
Loren	Clew	lclewa	50
Maddie	Emeney	memeneyk	40
Mario	Godbold	mgodbold19	10
Oona	Coutts	ocoutts13	200
Petrina	Trinder	ptrinder5	200
Sande	Cullip	scullip1c	10
Starla	Konke	skonkeh	0
Tanner	Dwerryhouse	tdwerryhouse7	10

19 rows in set (0.00 sec)

La jointure old-school (avant SQL ANSI-92)

```
1 SELECT firstname, lastname, Visitor.login,  
2     loyalty_points  
3 FROM Fidelity, Visitor  
4 WHERE Fidelity.login = Visitor.login;
```

A ne pas utiliser !

- Le join a une signification sémantique différente du where
 - join = jointure
 - where = filtre

```
1  SELECT firstname, lastname, Visitor.login,  
2         loyalty_points  
3  FROM Fidelity  
4  INNER JOIN Visitor ON Fidelity.login = Visitor.login  
5  WHERE loyalty_points > 100;
```

firstname	lastname	login	loyalty_points
Oona	Coutts	ocoutts13	200
Petrina	Trinder	ptrinder5	200

2 rows in set (0.00 sec)

Jointure avec plus que 2 tables

```
1 SELECT
2     Visitor.login, firstname, lastname,
3     loyalty_points, name, Conference.title
4 FROM Fidelity
5 INNER JOIN Visitor ON Fidelity.login = Visitor.login
6 INNER JOIN Registration ON Visitor.login = Registration.login
7 INNER JOIN Conference ON
8     Registration.id_conference = Conference.id_conference;
```

- Il faut simplement suivre le chemin : Fidelity → Visitor → Registration → Conference

Algorithme de jointure (Nested Loop)

Imaginons cette requête

```
1 SELECT * FROM T1
2 INNER JOIN T2 ON T1.key = T2.foreignkey
3 INNER JOIN T3 ON T2.foreignkey = T3.key
4 WHERE P(T1)
```

Supposons qu'elle est opérée dans l'ordre (ci-dessous, le résultat de la commande EXPLAIN)

Table	Join Type
t1	range
t2	ref
t3	ref_eq

Algorithme de jointure (Nested Loop)

Application :

```
1  for each row in T1 such that P1(T1) {  
2      for each row in T2 matching join condition between T1 and T2 {  
3          for each row in T3 matching join condition between T2 and T3 {  
4              add row to the result  
5          }  
6      }  
7  }
```

LEFT JOIN (et RIGHT JOIN)

Sélectionne tous les enregistrements de la table 1 (à gauche) ainsi que les enregistrements correspondants dans l'autre table. Les attributs de la table de droite prennent la valeur NULL s'il n'y a pas de correspondance dans l'autre table.

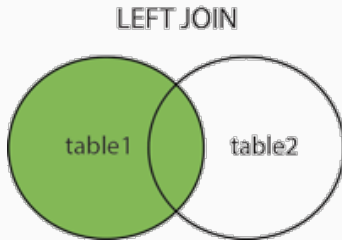


Figure 2: Jointure droite

LEFT JOIN (et RIGHT JOIN)

```
1 SELECT firstname, lastname, Visitor.login,  
2         loyalty_points  
3 FROM Visitor  
4 LEFT JOIN Fidelity ON Fidelity.login = Visitor.login;
```


LEFT JOIN (et RIGHT JOIN)

firstname	lastname	login	loyalty_points
Alfonso	Gillibrand	agillibrandc	10
Adela	Pendock	apendockq	NULL
Alair	Sustin	asustin1d	20
Banky	Glidder	bgliddery	NULL
Bessy	Scroxton	bscroxton1	NULL
Burg	Stemp	bstempg	40
Corey	Balassa	cbalassaj	20
Chancey	Iliffe	ciliffeb	NULL
Calvin	La Vigne	clavignew	NULL
Dieter	Foden	dfoden12	NULL
Elisa	Birtle	ebirtle17	NULL
Esme	Keasey	ekeasey15	10
Esteban	Lidgey	elidgeye	NULL
...
...
...

50 rows in set (0.00 sec)

Variantes du LEFT JOIN (non supportées par sqlite!)

- RIGHT JOIN
- FULL JOIN

Quiz

Que signifie cette requête :

```
1 SELECT firstname, lastname, Visitor.login, loyalty_points
2 FROM Visitor
3 LEFT JOIN Fidelity ON Fidelity.login = Visitor.login
4 WHERE loyalty_points IS NULL;
```

Agrégation

Les agrégations permettent de regrouper des enregistrements en un calcul unique.

Opérations les plus courantes :

- AVG()
- COUNT()
- SUM()
- MIN()/MAX()

```
1 SELECT column_1, aggregate_function(column_2)
2 FROM table
3 GROUP BY column_1;
```

Agrégation

```
1 SELECT Topic.title, COUNT(Conference.id_conference)
2 FROM Topic
3 INNER JOIN Conference ON Topic.title = Conference.title
4 GROUP BY Topic.title;
```

title	COUNT(Conference.id_conference)
blockchain	1
databases	2
lambda calculus	1
microservices	1

4 rows in set (0.00 sec)

Agrégation

```
1 SELECT Topic.title, COUNT(Conference.id_conference)
2 FROM Topic
3 LEFT JOIN Conference ON Topic.title = Conference.title
4 GROUP BY Topic.title;
```

title	COUNT(Conference.id_conference)
blockchain	1
cloud	0
databases	2
lambda calculus	1
microservices	1

5 rows in set (0.00 sec)

Clause HAVING

- Permet de filtrer sur la colonne agrégée
- La clause WHERE ne permet pas de le faire

```
1 SELECT Topic.title, COUNT(Conference.id_conference)
2 FROM Topic
3 LEFT JOIN Conference ON Topic.title = Conference.title
4 GROUP BY Topic.title
5 HAVING COUNT(Conference.id_conference) >= 2;
```

title	COUNT(Conference.id_conference)
databases	2

1 row in set (0.00 sec)

Sans regroupement

- Sommer le nombre total de points accumulés

```
1 SELECT SUM(loyalty_points)
2 FROM Fidelity;
```

Remarques

- La clause `GROUP BY` ne supprime pas les doublons ! elle permet d'agréger des données
 - pour **supprimer des doublons**, utilisez la clause `DISTINCT`
- Il est impératif d'appliquer une fonction d'agrégat lors d'un regroupement
 - `GROUP BY` \implies `f(x)`

Quiz

- Calculez la moyenne de points fidélité des visiteurs en prenant en compte les visiteurs qui n'ont pas de compte.

AS (alias)

Permet d'avoir des alias sur les champs ou les tables

La requête:

```
1 SELECT Topic.title, COUNT(Conference.id_conference)
2 FROM Topic
3 LEFT JOIN Conference ON Topic.title = Conference.title
4 GROUP BY Topic.title
5 HAVING COUNT(Conference.id_conference) >= 2;
```

se simplifie:

```
1 SELECT T.title, COUNT(CF.id_conference) AS C
2 FROM Topic AS T
3 LEFT JOIN Conference AS CF ON T.title = CF.title
4 GROUP BY T.title
5 HAVING C >= 2;
```

Sous-requêtes

Visiteurs qui n'ont pas de compte de fidélité:

```
1 SELECT Visitor.login, firstname, lastname
2 FROM Visitor
3 WHERE Visitor.login NOT IN (
4     SELECT Fidelity.login
5     FROM Fidelity
6 );
```

```
1 SELECT Visitor.login, firstname, lastname
2 FROM Visitor
3 WHERE NOT EXISTS (
4     SELECT *
5     FROM Fidelity
6     WHERE Fidelity.login = Visitor.login
7 );
```

Remarques

- EXISTS s'arrête dès qu'un enregistrement répond au prédicat, liste la requête racine en premier
- IN liste la totalité de la sous-requête, utilise une jointure et l'ordre n'est pas garanti (optimisé par le moteur)
- NOT EXISTS doit par contre comparer toutes les valeurs de la sous-requêtes

Sous-requêtes

Compte le nombre de participants pour chaque conférence:

```
1 SELECT
2     Conference.name,
3     COUNT(Participation.login)
4 FROM Conference
5 LEFT JOIN Participation
6 ON Conference.id_conference = Participation.id_conference
7 GROUP BY Conference.id_conference;
```

Pareil, mais avec une sous-requête:

```
1 SELECT Conference.name, (
2     SELECT COUNT(*)
3     FROM Participation
4     WHERE Conference.id_conference = Participation.id_conference
5 )
6 FROM Conference;
```

Quiz

- Pour chaque conférence, comptez le nombre d'intervenants et le nombre de visiteurs.
- Calculez le pourcentage des points fidélité de chaque visiteur en fonction du nombre total de points

DISTINCT

Thèmes attribués à des conférences (sans doublon)

```
1 SELECT DISTINCT title
2 FROM Conference;
```

title

blockchain

databases

lambda calculus

microservices

DISTINCT

- Tous les topics des conférences du speaker cbaszniak8

```
1 SELECT title
2 FROM Conference
3 INNER JOIN Participation
4     ON Conference.id_conference = Participation.id_conference
5 INNER JOIN Speaker
6     ON Participation.login = Speaker.login
7 WHERE Speaker.login = "cbaszniak8";
```

title

databases

lambda calculus

databases

DISTINCT

```
1 SELECT DISTINCT title
2 FROM Conference
3 INNER JOIN Participation
4     ON Conference.id_conference = Participation.id_conference
5 INNER JOIN Speaker
6     ON Participation.login = Speaker.login
7 WHERE Speaker.login = "cbasznia8";
```

title

databases

lambda calculus

UNION

- UNION combine les résultats de plusieurs SELECT
- Le nombre de champs doit correspondre (idéalement représenter la même chose)

Visiteurs et intervenants (avec doublons)

```
1 SELECT * FROM Visitor
2 UNION ALL
3 SELECT * FROM Speaker;
```

Visiteurs et intervenants (sans doublons)

```
1 SELECT * FROM Visitor
2 UNION
3 SELECT * FROM Speaker;
```

INTERSECT

- INTERSECT retourne les enregistrements qui se trouvent dans les deux requêtes
- **Non standard**: peut être remplacé par un INNER JOIN

Le login des visiteurs qui sont également intervenants:

```
1 SELECT login FROM Visitor
2 INTERSECT
3 SELECT login FROM Speaker;
```

ORDER BY

Visiteurs par login trié alphabétiquement

```
1 SELECT *  
2 FROM Visitor  
3 ORDER BY login;
```

Visiteurs avec compte fidélité trié par nombre de points décroissant

```
1 SELECT *  
2 FROM Visitor INNER JOIN Fidelity  
3 ON Visitor.login = Fidelity.login  
4 ORDER BY loyalty_points DESC;
```

LIMIT

Permet de limiter le nombre de résultat.

- Augmente grandement l'efficacité !
- Résultat parfois arbitraire

LIMIT [offset,] row_count

```
1 SELECT *  
2 FROM Visitor  
3 LIMIT 10;
```

```
1 SELECT *  
2 FROM Visitor  
3 LIMIT 11, 10;
```

WITH - Common Table Expression

Les CTEs permettent de définir un résultat intermédiaire qui peut être utilisé ultérieurement.

L'objectif est d'avoir une meilleure lisibilité sur des requêtes compliquées.

```
1 WITH i AS (  
2     SELECT id_conference, name FROM Conference  
3 )  
4 SELECT * FROM i;
```


WITH - Common Table Expression

Peut être récursive

```
1 WITH RECURSIVE cte (n,m) AS (  
2   SELECT 1,1  
3   UNION ALL  
4   SELECT m, n+m FROM cte WHERE n < 1000  
5 )  
6 SELECT n FROM cte;
```

- <https://dev.mysql.com/>

Commande INSERT (et REPLACE)

INSERT

```
1  INSERT INTO Visitor  
2  VALUES ("myLogin", "myFirstname", "myLastname");
```

INSERT

```
1 INSERT INTO Conference
2 VALUES (6, "SoftShake", "2019-06-01",
3         "2019-06-03", 200, "databases");
```

Ou, sans spécifier de clé:

```
1 INSERT INTO Conference (name, start_date, end_date, price, title)
2 VALUES ("SoftShake", "2019-06-01",
3         "2019-06-03", 200, "databases");
```

De manière générale, insertion de plusieurs enregistrement

```
1 INSERT INTO tbl_name (a,b,c)
2 VALUES(1,2,3), (4,5,6), (7,8,9);
```

INSERT avec critère de recherche

- Le mot-clé VALUES est substitué par un SELECT

```
1  INSERT INTO Fidelity
2  SELECT login, 1000
3  FROM Visitor
4  WHERE Visitor.firstname = "Adela"
5  AND Visitor.lastname = "Pendock";
```

INSERT avec critère de recherche

Idem avec une CTE:

```
1  INSERT INTO Fidelity
2  WITH V (login) AS (
3      SELECT login FROM Visitor
4      WHERE Visitor.firstname = "Adela"
5      AND Visitor.lastname = "Pendock"
6  )
7  SELECT login, 1000 FROM V;
```

INSERT avec critère de recherche

sur jointure

```
1  INSERT INTO Fidelity (login, loyalty_points)
2  SELECT Visitor.login, 1000
3  FROM Fidelity RIGHT JOIN Visitor ON Fidelity.login = Visitor.login
4  WHERE firstname <=> "Adela";
```

INSERT vs REPLACE

- REPLACE
 - insère si inexistant
 - met à jour sinon

```
1 REPLACE INTO Conference
2 VALUES (6, "SoftShake", "2019-07-01",
3         "2019-07-03", 200, "databases");
```


Commande UPDATE et DELETE

UPDATE

Syntaxe:

```
1 UPDATE Table
2 SET column1 = value1, column2 = value2, ....
3 WHERE id_conference = 3;
```

UPDATE

```
1 UPDATE Conference
2 SET name = "Devovx 2017"
3 WHERE id_conference = 3;
```

```
1 UPDATE Conference INNER JOIN Topic
2 ON Conference.title = Topic.title
3 SET name = "Devovx 2017"
4 WHERE Topic.title = "blockchain";
```

(la jointure est inutile, elle est utilisée comme exemple)

DELETE

```
1 DELETE FROM Conference
2 WHERE id_conference = 6;
```

UPDATE & DELETE

Attention:

- ne pas oublier la condition !
 - supprime/modifie tous les enregistrements sinon
- à votre condition
 - si toujours vrai

```
1 DELETE FROM Conference
2 WHERE 3 > 2;
```

Injection SQL

Injectons SQL

```
1 DELETE
2 FROM Visitor
3 WHERE login = '$login';
```

Injectons SQL

input: tartempion

```
1 DELETE
2 FROM Visitor
3 WHERE login = 'tartempion';
```


Injectons SQL

input: tartempion' or 1 = 1; --

```
1 DELETE
2 FROM Visitor
3 WHERE login = 'tartempion' or 1 = 1; -- ';
```

Découvrir une table

input: `x' AND 1=(SELECT COUNT(*) FROM Speaker); --`

```
1 SELECT *  
2 FROM Visitor  
3 WHERE login = 'x' AND 1=(SELECT COUNT(*) FROM Speaker); --';
```

- échoue si Speaker n'existe pas

Injectons SQL

input: tartempion'; DROP TABLE Speaker; --

```
1 SELECT *  
2 FROM Visitor  
3 WHERE login = 'tartempion'; DROP TABLE Speaker; -- ';
```

Prévenir les injections

- Requête pré-compilée

```
1 // En Java
2 PreparedStatement ps = connection.prepareStatement(
3     "SELECT * FROM Visitor WHERE login = ?");
4 ps.setString(1, login);
5 ResultSet rs = ps.executeQuery();
```

A lire: <http://www.unixwiz.net/techtips/sql-injection.html>