Base de données

Cours 6 - Jointures, sous-requêtes et transactions

Table des matières

Les jointures (JOINS)

Les jointures permettent de combiner des données provenant de plusieurs tables en fonction d'une relation logique entre ces tables. Les jointures sont essentielles pour interroger des bases de données relationnelles normalisées.

Les jointures - INNER JOIN

L'INNER JOIN retourne uniquement les lignes qui ont des **correspondances dans les deux tables**. C'est comme une intersection entre deux ensembles.

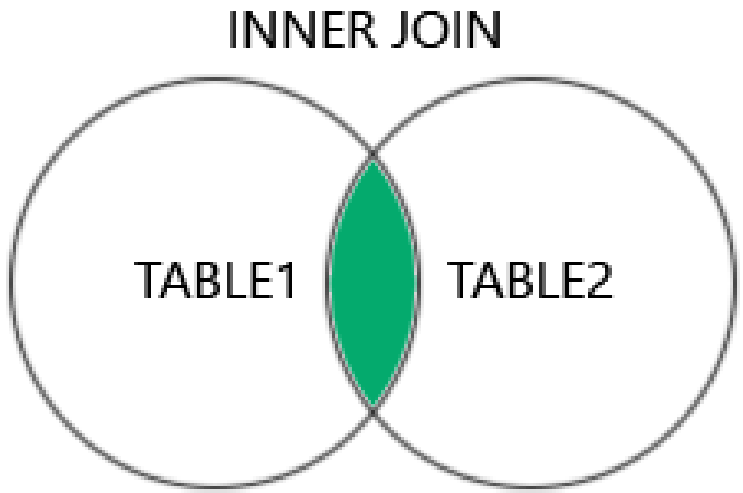

Figure: `https://www.w3schools.com/sql/sql_join_inner.asp`

Les jointures - INNER JOIN

```sql
-- Inner Join: Professors with valid department_id
SELECT professor.*, department.department_name AS department_name
FROM professor
INNER JOIN department
ON professor.department_id = department.department_id;

-- The INNER JOIN keyword selects records
-- that have matching values in both tables.
```

Listing: `https://www.postgresql.org/docs/current/sql-select.html`

Les jointures - INNER JOIN

Voici le résultat de la commmande :

	professor_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	101	Alice	Smith	1	Computer Science
2	102	Bob	Johnson	2	Mathematics
3	103	Carol	Williams	3	Physics
4	105	Eve	Davis	1	Computer Science

Les jointures - LEFT JOIN

Le LEFT JOIN retourne **toutes les lignes de la table de gauche (première table)**, même si elles n'ont pas de correspondance dans la table de droite (deuxième table). Les colonnes de la table de droite seront remplies avec NULL s'il n'y a pas de correspondance.

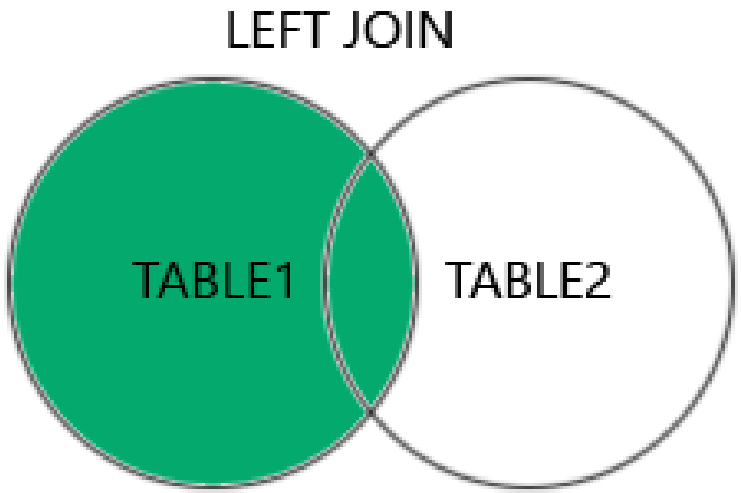

Figure: `https://www.w3schools.com/sql/sql_join_left.asp`

Les jointures - LEFT JOIN

```sql
-- Left Join: All professors, including those without a department
SELECT professor.*, department.department_name AS department_name
FROM professor
LEFT JOIN department
ON professor.department_id = department.department_id;

-- The LEFT JOIN keyword returns all records
-- from the left table (table1), and the matching records
-- from the right table (table2).
-- The result is 0 records from the right side, if there is no match.
```

Listing: `https://www.postgresql.org/docs/current/sql-select.html`

Les jointures - LEFT JOIN

Voici le résultat de la commmande :

	professor_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	101	Alice	Smith	1	Computer Science
2	102	Bob	Johnson	2	Mathematics
3	103	Carol	Williams	3	Physics
4	104	David	Brown	[null]	[null]
5	105	Eve	Davis	1	Computer Science

Les jointures - RIGHT JOIN

Le RIGHT JOIN est **l'opposé du LEFT JOIN**. Il retourne **toutes les lignes de la table de droite**, même si elles n'ont pas de correspondance dans la table de gauche. Les colonnes de la table de gauche seront NULL s'il n'y a pas de correspondance.

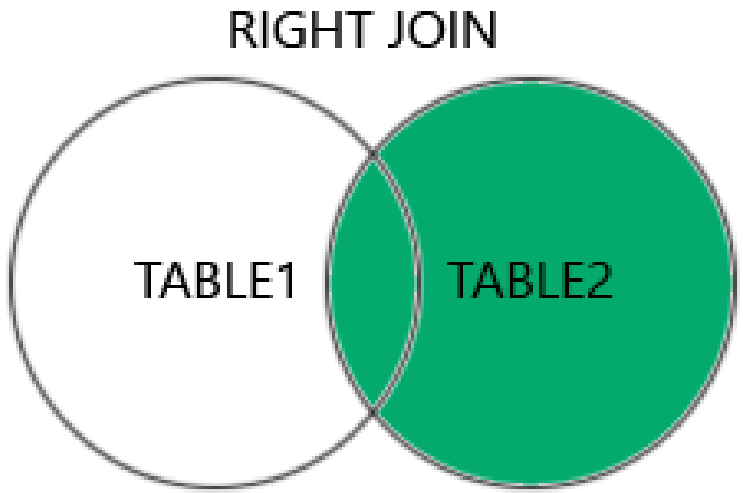

Figure: `https://www.w3schools.com/sql/sql_join_right.asp`

Les jointures - RIGHT JOIN

```sql
-- Right Join: All departments, including those without professors
SELECT professor.*, department.department_name AS department_name
FROM professor
RIGHT JOIN department
ON professor.department_id = department.department_id;

-- The RIGHT JOIN keyword returns all records
-- from the right table (table2), and the matching records
-- from the left table (table1).
-- The result is 0 records from the left side, if there is no match.
```

Listing: `https://www.postgresql.org/docs/current/sql-select.html`

Les jointures - RIGHT JOIN

Voici le résultat de la commmande :

	professor_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	101	Alice	Smith	1	Computer Science
2	102	Bob	Johnson	2	Mathematics
3	103	Carol	Williams	3	Physics
4	105	Eve	Davis	1	Computer Science
5	[null]	[null]	[null]	[null]	English Literature
6	[null]	[null]	[null]	[null]	History

Les jointures - FULL JOIN

Le FULL JOIN (ou FULL OUTER JOIN) **combine les résultats du LEFT JOIN et du RIGHT JOIN**. Il **retourne toutes les lignes des deux tables**, avec des NULL pour les colonnes sans correspondance.

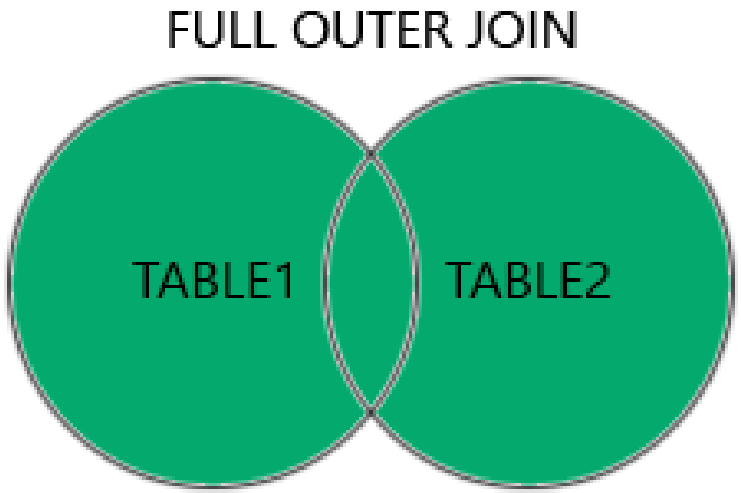

Figure: `https://www.w3schools.com/sql/sql_join_full.asp`

Les jointures - FULL JOIN

```sql
-- Full Join: All professors and departments, including unmatched rows
SELECT professor.*, department.department_name AS department_name
FROM professor
FULL OUTER JOIN department
ON professor.department_id = department.department_id;

-- The FULL OUTER JOIN keyword returns all records when
-- there is a match in left (table1) or right (table2) table records.

-- Tip: FULL OUTER JOIN and FULL JOIN are the same.
```

Listing: `https://www.postgresql.org/docs/current/sql-select.html`

Les jointures - FULL JOIN

Voici le résultat de la commmande :

	professor_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	101	Alice	Smith	1	Computer Science
2	102	Bob	Johnson	2	Mathematics
3	103	Carol	Williams	3	Physics
4	104	David	Brown	[null]	[null]
5	105	Eve	Davis	1	Computer Science
6	[null]	[null]	[null]	[null]	English Literature
7	[null]	[null]	[null]	[null]	History

Sous-requêtes et ensembles - Sous-requêtes

Une sous-requête (ou requête imbriquée) est une requête SQL incluse à l'intérieur d'une autre requête. Elle permet de réaliser des opérations complexes en plusieurs étapes.

Sous-requêtes

La sous-requête la plus courante est celle qui se trouve dans la clause WHERE.

L'objectif de celle-ci est de **filtrer** les résultats de la requête principale en fonction du résultat d'une autre requête.

```sql
-- Find students enrolled in course with ID 101
SELECT first_name, last_name
FROM student
WHERE student_id IN (
    SELECT student_id
    FROM student_course
    WHERE course_id = 101
);
```

Listing: https://www.postgresql.org/docs/current/sql-select.html

Opérations ensemblistes

Les opérations ensemblistes permettent de combiner les résultats de plusieurs requêtes SQL. Les principales opérations ensemblistes sont:

- UNION
- INTERSECT
- EXCEPT

Opérations ensemblistes - UNION

UNION **combine les résultats de deux requêtes** et élimine les doublons.

```sql
-- All students and instructors (no duplicates)
SELECT first_name, last_name FROM student
UNION
SELECT first_name, last_name FROM instructor;
```

Listing: https://www.postgresql.org/docs/current/queries-union.html

Opérations ensemblistes - INTERSECT

INTERSECT retourne uniquement les lignes qui sont **présentes dans les deux requêtes**.

```sql
-- All students who are ALSO instructors
SELECT first_name, last_name FROM student
INTERSECT
SELECT first_name, last_name FROM instructor;
```

Listing: https://www.postgresql.org/docs/current/queries-union.html

Opérations ensemblistes - EXCEPT

EXCEPT retourne les lignes de la première requête **qui ne sont PAS dans la deuxième**.

```
1  -- All students who are NOT instructors
2  SELECT first_name, last_name FROM student
3  EXCEPT
4  SELECT first_name, last_name FROM instructor;
```

Listing: https://www.postgresql.org/docs/current/queries-union.html

Performance des requêtes

L'optimisation des performances est cruciale pour garantir des temps de réponse rapides, surtout avec de grandes volumes de données.

Performance des requêtes - Index

Un index est une structure de données qui permet de trouver rapidement des lignes spécifiques dans une table.

Les index améliorent considérablement la vitesse des opérations de lecture, mais peuvent ralentir les opérations d'écriture (INSERT, UPDATE, DELETE) car l'index doit être mis à jour.

```sql
-- Creation of an index
CREATE INDEX idx_student_last_name
ON student (last_name);

-- Using the index
SELECT * FROM student WHERE last_name = 'Doe';
-- the database automatically uses the index for optimization
```

Listing: https://www.postgresql.org/docs/current/sql-createindex.html

Performance des requêtes - Vues

Une vue est une requête SQL sauvegardée qui se comporte comme une table virtuelle. Elle ne stocke pas physiquement les données, mais affiche des données provenant d'une ou plusieurs tables sous-jacentes.

Les vues simplifient les requêtes complexes et peuvent améliorer la lisibilité du code.

```sql
-- Creation of a view
CREATE VIEW part_time_student_view AS
SELECT first_name, last_name, email
FROM student
WHERE enrollment_type = 'part-time';

-- Using the view
SELECT * FROM part_time_student_view;
```

Listing: https://www.postgresql.org/docs/current/sql-createview.html

Transactions

Les transactions garantissent l'intégrité des données en groupant plusieurs opérations en une seule unité de travail.

```
1  -- Start a transaction
2  BEGIN TRANSACTION;
3
4  UPDATE course
5  SET course_code = '420'
6  WHERE department_id = 1;
7
8  UPDATE course
9  SET course_name = 'Advanced Databases'
10 WHERE course_id = 3;
11 -- If everything is fine, commit the transaction
12 COMMIT;
13 -- If there was an error, rollback the transaction
14 -- ROLLBACK;
```

Listing: https://www.postgresql.org/docs/current/tutorial-transactions.html

Verrous

Les verrous gèrent l'accès concurrent aux données.

```sql
BEGIN TRANSACTION;
-- Lock a specific row in the course table
-- This prevents other transactions from modifying
-- or deleting this row until the current transaction is completed.
SELECT * FROM course WHERE course_id = 1FOR UPDATE;

UPDATE course
SET course_name = 'Database Systems'
WHERE course_id = 1;

-- Commit the transaction to release the lock
COMMIT;
```

Listing: https://www.postgresql.org/docs/current/explicit-locking.html

Bibliographie

- https://www.postgresql.org/docs/current/sql-select.html
- https://www.postgresql.org/docs/current/queries-union.html
- https://www.postgresql.org/docs/current/sql-createindex.html
- https://www.postgresql.org/docs/current/sql-createview.html
- https://www.postgresql.org/docs/current/tutorial-transactions.html
- https://www.postgresql.org/docs/current/explicit-locking.html