SQL DML : Data Manipulation Language 2

Fabien Coelho, Claire Medrala

Mines Paris – PSL, MESR

Décembre 2025

Valeur pour tous les types — NULL

- valeur absorbante ou ignorée par les opérations...
- comparaisons : `d IS NULL`, `e IS NOT NULL`
- **attention** `x = NULL` vaut `NULL`, donc faux !
  mais `NULL OR TRUE` vaut `TRUE`
- `COALESCE(t,u,...)`: premier *non null*

a	*b*	*a=b*	*a!=b*	*a = NULL*	*a != NULL*	*a IS NULL*	*a IS NOT NULL*
0	0	t	f			f	t
1	0	f	t			f	t
	0					t	f
	1					t	f
						t	f

Types — BOOLEAN INTEGER

booléens `BOOLEAN` valeurs `TRUE FALSE`... et `NULL` pour **inconnue**

- opérations `AND OR NOT` : `a>10 AND NOT b>100`
- comparaisons directes de valeurs `=` `!=` `<>`
- double comparaison `a BETWEEN b AND c`

entiers `SMALLINT INTEGER NUMERIC`...

- constantes `5432` `-10` `10000000000000000000000`
- opérations unaires `+` `-`, binaires `+` `-` `*` `/`, modulo `%`
- comparaisons classiques `<` `>` `<=` `>=`
- priorités classiques, division entre entiers **entière** !
- fonctions `MOD(3,2)=1` `ABS(-1)=1`...

Types ... suite — FLOAT TEXT

flottants `FLOAT4 REAL FLOAT8 FLOAT DOUBLE PRECISION`

- constantes `2.71828182845905` `-1.23E2`
- précision arbitraire `DECIMAL(10,2)` (précision, échelle)
- arithmétique `+` `-` `*` `/`, comparaisons...
- fonctions `POW`(2,3) `LN`(10) `EXP`(3.1) `LOG`(123) `COS`(1.3) `SIN`(1.0) `ATAN`(2.0) `SQRT`(4) `CBRT`(27.0) `PI`() `RANDOM`()...

textes constantes *quotées* `'données' 'l''artiste'`

- opérateur *concat.* `||`
- fonctions : `LENGTH`('hi')
  `CONCAT`('hello', 'world')
  `LOWER`('Hi') `UPPER`('hello')
  `TRIM`(' hobbes ') `SUBSTRING`('Calvin',2,4)...

Opérateurs — LIKE SIMILAR TO ~

- expressions régulières : comparaison texte et **motif**
  caractères réguliers et **spéciaux** pour joker, répétition...
- reconnaissance en temps **linéaire** par **automates**
- 3 versions : SQL92, SQL99 et POSIX

`[NOT] LIKE` _ un caractère, % une chaîne
`[NOT] SIMILAR TO` *idem* plus | `* + [a-z] (...)`
opérateur ~ expressions régulières standard POSIX

```
-- films commençant par C
SELECT titre, auteur
FROM les_films
WHERE titre LIKE 'C%';
```

titre	*auteur*
Citizen Kane	Wells
City Lights	Chaplin

Exemples — SIMILAR TO


```
-- titres sans deux majuscules
SELECT auteur, titre
FROM les_films
WHERE titre NOT SIMILAR TO '%[A-Z]%[A-Z]%';
```

auteur	*titre*
Ozu	Ohayô

```
-- titres accentués
SELECT titre, auteur, année
FROM les_films
WHERE titre SIMILAR TO '%[^a-zA-Z ]%';
```

titre	*auteur*	*année*
Le Procès	Wells	1963
Ohayô	Ozu	1959

Autres opérateurs — DATE TIME INTERVAL...

temps `DATE TIME` `TIMETZ` `TIMESTAMP` `INTERVAL`

- constantes `'1970-03-20'::DATE` `TIME '12:30:45'`
- opérateurs arithmétiques `+ - * /` et `OVERLAPS`
  `DATE 'today' + INTERVAL '1 month'`
- fontions `NOW()` `CURRENT_DATE` `CURRENT_TIME`
  extraction `EXTRACT(YEAR FROM une_date)`
  avec `CENTURY` `DECADE` `DOW` `HOUR` `MINUTE` `SECOND`...
  cf `AGE` `JUSTIFY_HOURS` `JUSTIFY_DAYS` `TO_CHAR`...

Autres opérateurs — BIT BOX POLYGON INET...

binaires `BIT(12)` `VARBIT(31)` `BYTEA`

géométrie 2D `BOX` `CIRCLE` `LINE` `POINT` `POLYGON` `PATH`...

- constantes `'(1,1),(2,3),(0,0)'::POLYGON`
- nombreuses opérations : contient, rotation, distance...
  `POINT '(2,0)' @ BOX '(-1,-1),(1,1)'`

réseau `INET` `MACADDR`
`INET '192.168.1.5' << INET '192.168.1/24'`

extensions ajoutables dynamiquement !
bibliothèque `ISSN` `ISBN`

Expression conditionnelle — CASE

CASE
WHEN cond1 THEN res1 condition et valeur
WHEN cond2 THEN res2 condition et valeur...
ELSE resdef END valeur finale par défaut

```
SELECT nom,
  CASE
    WHEN décès IS NULL THEN 'vivant'
    ELSE 'décédé'
  END AS "actuel."
FROM Personnes;
```

nom	*actuel.*
Chaplin	décédé
Wells	décédé
Ozu	décédé
Allen	vivant
Carné	décédé
Tarentino	vivant

Expression conditionnelle ...suite — CASE

CASE expr calcule une expression
WHEN val1 THEN res1 énumération de valeurs
ELSE def END valeur par défaut

```
SELECT nom,
  CASE EXTRACT(CENTURY FROM naissance)
    WHEN 21 THEN 'XXIème'
    WHEN 20 THEN 'XXème'
    WHEN 19 THEN 'XIXème'
    ELSE 'ne sait pas'
  END AS "né au"
FROM Personnes;
```

nom	*né au*
Chaplin	XIXème
Wells	XXème
Ozu	XXème
Allen	XXème
Carné	XXème
Tarentino	XXème

3 types de jointures — JOIN

cartésienne *(cross join)*, rarement utile !
interne *(inner join)* condition ON / USING / NATURAL / WHERE
chaque tuple apparaît **si correspondance**
éviter NATURAL et WHERE...
externe *(outer join)* condition ON / USING / NATURAL
3 sortes LEFT / RIGHT / FULL
tuples apparaissent **même sans correspondance** !
contient la jointure interne

prénoms

prénom	*id*
Albert	1
Allan	2
Kurt	3

noms

id	*nom*
1	Einstein
1	Camus
3	Gödel
4	Church

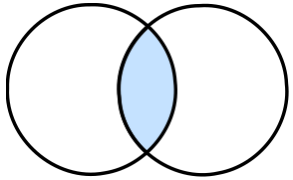
Jointure cartésienne : 2 syntaxes — CROSS JOIN

```
SELECT p.prénom, n.nom
FROM Prénoms AS p CROSS JOIN Noms AS n;

SELECT p.prénom, n.nom
FROM Prénoms AS p, Noms AS n;
```

prénom	*nom*
Albert	Einstein
Albert	Camus
Albert	Gödel
Albert	Church
Allan	Einstein
Allan	Camus
Allan	Gödel
Allan	Church
Kurt	Einstein
Kurt	Camus
Kurt	Gödel
Kurt	Church

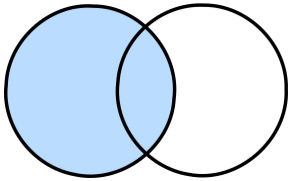
Jointure interne : 4 syntaxes [INNER] JOIN


```sql
SELECT p.prénom, n.nom -- ON condition
FROM Prénoms AS p
INNER JOIN Noms AS n ON p.id=n.id;

SELECT p.prénom, n.nom -- USING colname
FROM Prénoms AS p
INNER JOIN Noms AS n USING (id);

SELECT p.prénom, n.nom -- NATURAL = même colname
FROM Prénoms AS p
NATURAL INNER JOIN Noms AS n;

SELECT p.prénom, n.nom -- X et WHERE condition
FROM Prénoms AS p, Noms AS n
WHERE p.id=n.id;
```



prénom	*nom*
Albert	Einstein
Albert	Camus
Kurt	Gödel

Jointure externe gauche LEFT [OUTER] JOIN

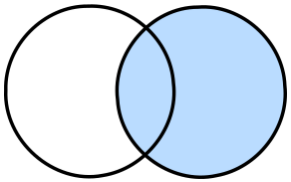


- **tous** les tuples de la table **gauche** apparaissent
- ajout d'un tuple **droit** NULL si nécessaire
- pas de condition de **jointure** dans le WHERE (ambiguïté)

```sql
SELECT p.prénom, n.nom
FROM Prénoms AS p
LEFT JOIN Noms AS n USING (id);
```

prénom	*nom*
Albert	Einstein
Albert	Camus
Allan	
Kurt	Gödel

Jointure externe droite RIGHT [OUTER] JOIN

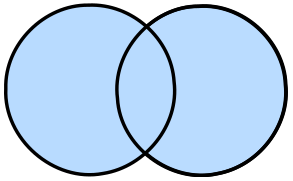


- **tous** les tuples de la table **droite** apparaissent
- ajout d'un tuple **gauche** NULL si nécessaire

```sql
SELECT p.prénom, n.nom
FROM Prénoms AS p
RIGHT JOIN Noms AS n USING (id);
```

prénom	*nom*
Albert	Einstein
Albert	Camus
Kurt	Gödel
	Church

Jointure externe totale FULL [OUTER] JOIN



- **tous** les tuples apparaissent, ajout tuples NULL

```sql
SELECT p.prénom, n.nom
FROM Prénoms AS p
FULL JOIN Noms AS n USING (id)
```

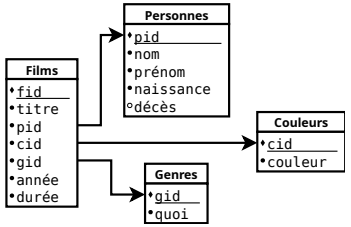
prénom	*nom*
Albert	Einstein
Albert	Camus
Allan	
Kurt	Gödel
	Church

Faisons notre cinema avec 4 tables

- modèle normalisé, pas de redondance

Couleurs

cid	*couleur*
1	N&B
2	couleur



Genres

gid	*quoi*
1	Drame
2	Comédie
3	Comédie dramatique
4	Action

Personnes

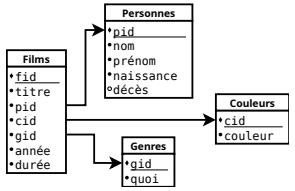
pid	*nom*	*prénom*	*naissance*	*décès*
1	Chaplin	Charles	1889-04-16	1977-12-25
2	Wells	Orson	1915-05-06	1985-10-10
3	Ozu	Yasujiro	1903-12-12	1963-12-12
4	Allen	Woody	1935-12-01	
5	Carné	Marcel	1909-08-18	1996-10-31
6	Tarentino	Quentin	1963-03-27	

Exemples

Films

fid	*titre*	*pid*	*cid*	*gid*	*année*	*durée*
1	Citizen Kane	2	1	1	1936	01:59:00
2	The Dictator	1	1	2	1940	02:07:00
3	Modern Times	1	1	2	1936	01:27:00
4	City Lights	1	1	3	1931	01:27:00
5	Ohayô	3	2	2	1959	01:37:00
6	Le Procès	2	2	1	1963	01:58:00
7	Kill Bill v1	6	2	4	2003	01:51:00
8	Monsieur Verdoux	1	1	3	1947	02:04:00
9	Limelight	1	1	3	1952	02:21:00
10	King in New York	1	1	3	1957	01:50:00
11	Alice	4	2	2	1990	01:42:00
12	The Trial	2	1	1	1962	01:58:00

Jointure externe pour passage au complémentaire `LEFT JOIN`




```sql
-- auteurs sans films avec NULL...
SELECT prénom, nom
FROM Personnes AS p
LEFT JOIN Films AS f USING(pid)
WHERE f.fid IS NULL;
```

prénom	*nom*
Marcel	Carné

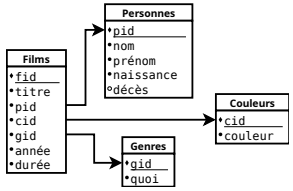
Opérations ensemblistes `UNION INTERSECT EXCEPT ALL`

- opérateurs entre tables `UNION INTERSECT EXCEPT`
- éventuellement non ensembliste (duplications) `ALL`
- tables compatibles ! attributs de même types

```sql
SELECT 1 AS nb
UNION ALL
SELECT 2 AS nb
UNION ALL
SELECT 1 AS nb;
```

nb
1
2
1

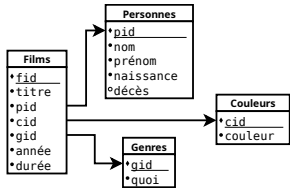
Opérations ensemblistes — INTERSECT




```
SELECT id FROM Prénoms
INTERSECT
SELECT id FROM Noms;
```

id
3
1

Opérations ensemblistes — EXCEPT




```
-- les auteurs sans films...
-- tous les auteurs
SELECT nom, prénom
FROM Personnes
-- moins
EXCEPT
-- ceux avec des films
SELECT DISTINCT nom, prénom
FROM Personnes AS p
JOIN Films AS f ON p.pid=f.pid;
```

nom	*prénom*
Carné	Marcel

Sous requêtes *(subqueries)* — SELECT ... SELECT

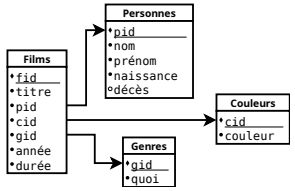
- **algèbre relationnelle** : requête **dans** une requête
- à la place d'une table dans une clause FROM

```
SELECT ...
FROM
  (SELECT ...
   FROM ...
   WHERE ...) AS alias...
WHERE ...;
```

- expressions booléennes, opérateurs EXISTS IN ANY ALL
  sous-requête SELECT simplifié (pas de ORDER...)

```
SELECT ... FROM ...
WHERE attr NOT IN
  (SELECT ... FROM ... WHERE ...);
```


Sous-requête dans un FROM — SELECT ... SELECT




```
-- plus petite durée moyenne par auteur
SELECT MIN(moy) AS durée
FROM
 (SELECT nom, AVG(durée) AS moy
  FROM Films AS f
  JOIN Personnes AS p ON f.pid=p.pid
  GROUP BY nom) AS avdur;
```

durée
01:37:00

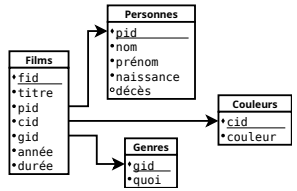
Sous-requête dans une condition WHERE — SELECT ... SELECT

```
-- films courts
SELECT titre
FROM Films
WHERE durée < ANY(SELECT AVG(durée) FROM Films);
```

titre
Modern Times
City Lights
Ohayô
Kill Bill v1
King in New York
Alice

Sous-requête + UNION — 1/3

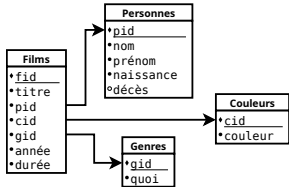
```
-- nb de films pour tous les auteurs
SELECT nom, SUM(nfilms) AS nfilms
FROM
 (-- nb films des auteurs actifs
  SELECT nom, COUNT(*)
  FROM Personnes AS p
  JOIN Films AS f USING (pid)
  GROUP BY nom
  UNION
  -- tous les auteurs
  SELECT nom, 0
  FROM Personnes) AS f(nom,nfilms)
GROUP BY nom
ORDER BY nfilms DESC, nom ASC;
```



nom	*nfilms*
Chaplin	6
Wells	3
Allen	1
Ozu	1
Tarentino	1
Carné	0

Plus simple avec UNION + LEFT JOIN — 2/3

```
-- auteurs avec films, jointure interne
SELECT nom, COUNT(*) AS nfilms
FROM Personnes JOIN Films USING (pid)
GROUP BY nom
UNION
-- auteurs sans films, jointure externe
SELECT nom, 0
FROM Personnes LEFT JOIN Films USING (pid)
WHERE fid IS NULL
-- ordre commun
ORDER BY nfilms DESC, nom ASC;
```

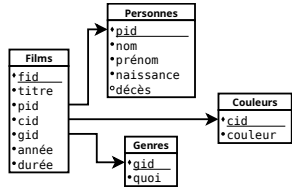


nom	*nfilms*
Chaplin	6
Wells	3
Allen	1
Ozu	1
Tarentino	1
Carné	0

Encore plus simple avec Postgres — 3/3

- jointure externe pour **tous**
- `COUNT(NULL)` vaut 0 !

```
-- nb films tous auteurs (PGSQL!)
SELECT nom, COUNT(f.fid)
FROM Personnes AS p
LEFT JOIN Films AS f USING(pid)
GROUP BY nom;
```



nom	*count*
Carné	0
Chaplin	6
Tarentino	1
Ozu	1
Allen	1
Wells	3

Conseils pour le développement des requêtes

1. déterminer les tables utiles
   - résultats ou conditions
   - tables de liaisons
   - tables *multiples* ?
2. créer le `SELECT FROM` et les jointures `ON USING`
3. ajouter les conditions `WHERE`
4. ajouter les aggrégations `GROUP BY`
   et condition sur aggrégation `HAVING`
5. ajouter les tris `ORDER BY`
6. décomposer si nécessaire (sous requêtes, ensembles) ?
7. simplifier les requêtes (multiples...) avec des vues ?

Réfléchir au problème suivant

- soient deux tables de même schéma *(id INT PK, val TEXT NN)*
- identifier (id) et caractériser (insert/update/delete) les différences pour passer le T1 à T2

Relation T1

id	*val*
1	un
2	deux
3	trois

Relation T2

id	*val*
1	one
2	deux
4	quatre

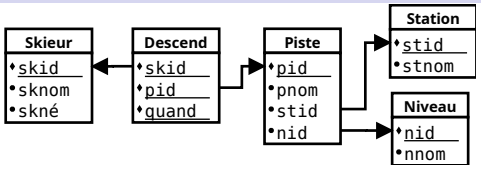
Différences

id	*opération*
1	UPDATE
3	DELETE
4	INSERT

Exercices

- quand tombe $2^{31} - 1$ secondes après le 1 janvier 1970 ?
- les cinéastes vivants
- les titres des films d'action
- les cinéastes nés au XIXème siècle
- les cinéastes par ordre de naissance
- les titres des comédies en couleurs de *Charles Chaplin*
- les cinéastes avec des comédies en noir et blanc
- la durée de vie de tous les cinéastes, en ordre décroissant
- les cinéastes décédés plus jeunes que la moyenne
- les films des cinéastes nés avant 1910
- le(s) cinéaste(s) dont les films sont en moyenne les plus longs

Exercice



- Quelles sont les pistes *noir*es de *Courchevel* par ordre alphabétique ?
- Quelles pistes de quelles stations a descendu *Susie* en *2018*, ordonnées ?
- Combien de pistes *bleue*s a descendu *Calvin* à *Val d'Isère* en *2020* ?
- Quelles pistes, par stations, n'ont jamais été descendues, ordonnées ?
- Quels skieurs ont descendu la piste la plus fréquentée en *2018*, ordonnés ?
- Pour toutes les pistes, donner leur nombre de skieurs différents, ordonnés.
- Pour tous les skieurs, de 2018 à 2020, le nombre de descentes, ordonné.
- La fréquentation (nombre de skieurs) de toutes les stations en 2018.

Insertions d'une ligne — INSERT

- commande `INSERT INTO ... VALUES (...), ...`
- donne la table et (éventuellement) la liste de colonnes
  il est préférable de préciser cette liste (modif du schéma)
- si colonne non spécifiée, valeur par défaut ou `NULL`
- types des valeurs doivent être compatibles !

```
INSERT INTO Personnage(id, nom, quand) VALUES
  (5, 'Calvin', DATE '2015-03-15'),
  (6, 'Hobbes', CURRENT_DATE),
  (7, 'Moe', DATE 'yesterday')
;
```


Insertion de lignes — INSERT ... SELECT

- remplissage d'une table à partir d'une requête
- permet de transférer des données entre tables
- voir aussi `CREATE TEMPORARY TABLE ... AS ...`

```
INSERT INTO Relation
SELECT ... FROM ... WHERE ... ;
```


Effacement de lignes — DELETE

- commande `DELETE FROM ... WHERE ...`
- précise la table concernée
- condition similaire à un `SELECT`
- **attention** peut toucher plusieurs lignes
- voir aussi `TRUNCATE TABLE ...`

```
-- efface un tuple
DELETE FROM Personnage WHERE id = 123;

-- efface TOUT !
DELETE FROM Personnage;
```

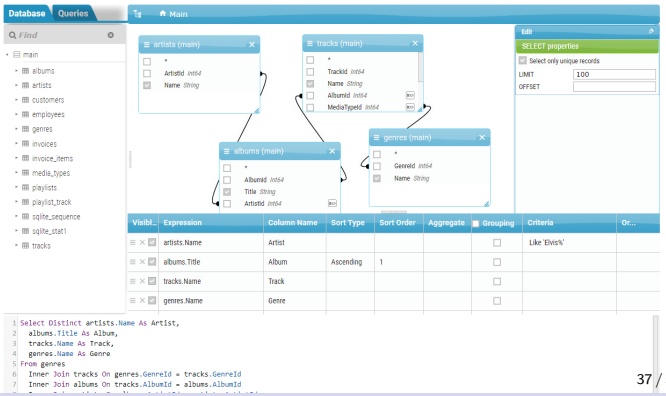

Mise à jour de lignes — UPDATE

- commande `UPDATE ... SET ...=... WHERE ...`
- modifications d'un ou plusieurs attributs
- condition similaire à un `SELECT`
- **attention** peut toucher plusieurs lignes

```
UPDATE Personnage
  SET ami = 'Hobbes', age = 6
  WHERE nom = 'Calvin';
```

Requêtes graphiques

- interface graphique de fabrication d'un SELECT
- limitations éventuelles pour des requêtes avancées
  sous-requêtes, ensembles, fonctions, types de jointures, expressions. . .



Embellissement de requêtes

https://paste.depesz.com/
http://sqlformat.darold.net/

Visualisation de requêtes complexes

http://revj.sourceforge.net/

```
SELECT B.Brand, G.Country, SUM (F.Units_Sold)
FROM Fact_Sales F
JOIN Dim_Date D ON F.Date_Id = D.Id
JOIN Dim_Store S ON F.Store_Id = S.Id
JOIN Dim_Geography G ON S.Geography_Id = C.Id
JOIN Dim_Product P ON F.Product_Id = P.Id
JOIN Dim_Product_Category C ON P.Product_Category_Id = C.Id
JOIN Dim_Brand B ON P.Brand_Id = B.Id
WHERE D.Year = 1997 AND C.Product_Category = 'tv'
GROUP BY B.Brand, G.Country;
```