Informatique *Cours*
SQL- Requêtes Ipesup

Objectifs

À l'issue de cette section, l'étudiant devra être capable de :
- ok

Types SQL

Le langage SQL propose des types variés :
- entiers (tailles diverses) ;
- flottants ;
- chaînes de longueur $\leq n$;
- booléens ;
- dates ;
- intervalles de temps ;
- types énumérés et tableaux.

Chaque type possède des spécificités (précision, conversions implicites ou non, etc.) et leur définition, souvent incomplète dans la norme, dépend des implémentations. Bien que le programme se limite à des types simples (entiers, flottants, chaînes), le choix des types reste crucial en bases de données.

Nos données étant modélisées, nous abordons leur interrogation. La présentation de SQL se décompose en quatre parties : requêtes simples sur une seule table avec expressions et types de base ; opérateurs ensemblistes ; jointures (opération centrale du modèle relationnel) ; enfin, requêtes avancées, notamment avec agrégats et groupements.

Création de tables en SQL

La création de tables en SQL est *hors programme*, mais nécessaire pour tester des requêtes. Voici les instructions correspondant au schéma considéré :

```
CREATE TABLE Personne (pid INTEGER PRIMARY KEY,
                       prenom VARCHAR(30),
                       nom VARCHAR(30));

CREATE TABLE Film (fid INTEGER PRIMARY KEY,
                   titre VARCHAR(255),
                   annee INTEGER,
                   duree INTEGER,
                   titre_orig VARCHAR(255));

CREATE TABLE Genre (fid INTEGER, genre VARCHAR(25),
                    FOREIGN KEY (fid) REFERENCES FILM(fid));

CREATE TABLE Pays (fid INTEGER, pays VARCHAR(25),
```

```
                    FOREIGN KEY (fid) REFERENCES FILM(fid));

CREATE TABLE Remporte (fid INTEGER, annee INTEGER, hote INTEGER,
                       FOREIGN KEY (fid) REFERENCES FILM(fid),
                       FOREIGN KEY (hote) REFERENCES PERSONNE(pid));

CREATE TABLE Realise (pid INTEGER,
                      fid INTEGER,
                      FOREIGN KEY (fid) REFERENCES FILM(fid),
                      FOREIGN KEY (pid) REFERENCES PERSONNE(pid));

CREATE TABLE Joue (pid INTEGER REFERENCES PERSONNE(pid),
                   fid INTEGER REFERENCES FILM(fid),
                   role VARCHAR(255),
                   FOREIGN KEY (fid) REFERENCES FILM(fid),
                   FOREIGN KEY (pid) REFERENCES PERSONNE(pid));
```

Les données sont ensuite insérées via :

```
INSERT INTO Film VALUES (0, 'Trois Hommes et un couffin', 1985, 106, NULL);
INSERT INTO FILM VALUES(12, 'RoboCop 2', 1990, 117, NULL);
INSERT INTO FILM VALUES(55, 'Un poisson nommé Wanda', 1988, 108,
                'A Fish Called Wanda');
```

Bien que le type TEXT soit courant, VARCHAR(n) est généralement préféré : standardisé et permettant une gestion interne plus efficace grâce à la longueur maximale connue.

1 Sélection

1.1 Sélection simple

Comme premier exemple, supposons que l'on veuille connaître le titre et la durée des films sortis en 2017 ou plus tard. Avec notre schéma, la requête SQL répondant à la question est :

```
SELECT titre, duree FROM Film WHERE annee >= 2017;
```

Le résultat d'une telle requête est :

titre	duree
Le Crime de l'Orient-Express	114
Dunkerque	107
Moi, moche et méchant 3	90
Paddington 2	103
Deadpool 2	119
Split	117
Ghost in the Shell	107
Tout le monde debout	107

⋮

SQL étant insensible à la casse, on adopte les conventions suivantes :

— mots-clés en majuscules ;
— attributs en minuscules ;
— tables avec initiale majuscule.

La cohérence prévaut sur le choix du style[1]. Les commentaires commencent par « – » et s'étendent jusqu'à la fin de ligne.

Une requête se décompose en trois clauses : `FROM` (table interrogée), `WHERE` (filtrage des lignes), `SELECT` (colonnes retournées). Elle se lit ainsi : sélectionner `titre` et `duree` des films de `Film` dont `annee` $\geq$ 2017.

```
-- Titre et durée des films sortis après 2017
SELECT titre, duree
FROM Film
WHERE annee >= 2017;
```

Les clauses `FROM` et `WHERE` sont optionnelles : une requête minimale est un `SELECT` sans table.

```
SELECT 42, 10*10, 'Salut tout le monde !';
```

42	10*10	'Salut tout le monde!'
42	100	Salut tout le monde !

La requête précédente renvoie une table à une ligne et trois colonnes, contenant les valeurs des expressions. Les noms de colonnes, non normalisés, dépendent du Système de Gestion de Base de Données (SGBD) ; ils peuvent être fixés via le mot-clé `AS`.

```
SELECT 42 AS num, 10*10 AS prod, 'Salut tout le monde !' AS texte;
```

num	prod	texte
42	100	Salut tout le monde !

L'ajout de `FROM` permet d'extraire les valeurs d'une table.

1. Éviter des mélanges tels que : SeLeCT tiTre, DUree FRom fiLm wherE AnnEe >= 2017.

Exemple

Par exemple, pour renvoyer le titre des films et leur durée décomposée en heures et minutes :

```
SELECT titre, duree/60 AS h, MOD(duree, 60) AS min FROM Film;
```

titre	h	min
Trois Hommes et un couffin	1	46
Pouic-Pouic	1	26
Les Yeux de Laura Mars	1	44
Miss Peregrine et les enfants particuliers	2	7
Les Chevaliers du ciel	1	42
Cobra	1	26

⋮

La division renvoie un entier si les deux opérandes sont entiers, et un flottant sinon, comme en C.

```
SELECT titre, duree/60.0 AS h_min FROM Film;
```

titre	h_min
Trois Hommes et un couffin	1.7666666666666666
Pouic-Pouic	1.4333333333333333
Les Yeux de Laura Mars	1.7333333333333334
Miss Peregrine et les enfants particuliers	2.1166666666666667
Les Chevaliers du ciel	1.7
Cobra	1.4333333333333333

La clause `WHERE`, après `FROM`, filtre les lignes via une condition arbitrairement complexe, formée de :

- attributs ;
- constantes (entières ou flottantes) ;
- chaînes littérales ;
- `NULL` ;
- expressions arithmétiques (+, -, *, /, `MOD`) ;
- comparaisons (=, <>, <, <=, >, >=) ;
- opérateurs booléens (`AND`, `OR`, `NOT`) ;
- tests `IS NULL` / `IS NOT NULL`.

Exemple

```
SELECT titre, duree FROM Film WHERE annee >= 2017
    AND duree <= 105
    AND titre_orig IS NOT NULL;
```

Sélectionne les films postérieurs à 2017, de durée ≤ 105 min, avec titre original défini.

Les chaînes sont délimitées par `'`, échappées par duplication (`'C"est ...'`), sans autre échappement.

Toute comparaison avec `NULL` est fausse, y compris l'égalité ; ainsi :

```
SELECT titre FROM Film WHERE titre_orig = NULL;
```

ne renvoie rien. On utilise :

```
SELECT titre FROM Film WHERE titre_orig IS NULL;
```

Enfin, `SELECT *` renvoie tous les attributs (dans l'ordre de déclaration) :

```
SELECT * FROM Film WHERE annee < 1980;
```

équivaut à :

```
SELECT fid, titre, annee, duree, titre_orig FROM Film
WHERE annee < 1980;
```

1.2 Suppression des doublons et tris

Une opération fréquente consiste à éliminer les doublons. Par exemple, pour obtenir tous les genres, on sélectionne la colonne correspondante de la table `Genre` :

```
SELECT genre FROM Genre;
```

genre
COMÉDIE
COMÉDIE
FANTASTIQUE
THRILLER
FANTASTIQUE
ACTION
ACTION
DRAME
THRILLER
GUERRE
⋮

Sans clause `WHERE`, toutes les lignes sont renvoyées, entraînant des doublons puisque chaque genre peut apparaître plusieurs fois. L'ajout de `DISTINCT` après `SELECT` supprime ces doublons :

```
SELECT DISTINCT genre FROM Genre;
```

genre
COMÉDIE
FANTASTIQUE
THRILLER
ACTION
DRAME
GUERRE
AVENTURES
⋮

L'ordre des résultats est indéterminé et non stable : il dépend des stratégies d'évaluation du SGBD (parcours, hachage, tri, etc.). Il ne faut donc jamais s'y fier.

Le tri explicite se fait via `ORDER BY` :

```
SELECT DISTINCT genre FROM Genre ORDER BY genre;

SELECT * FROM Film WHERE annee >= 2017
ORDER BY (duree/60) DESC, titre ASC;

SELECT * FROM Personne ORDER BY nom, prenom;
```

Ces requêtes trient respectivement les genres distincts (ordre alphabétique), les films récents (heures décroissantes puis titre croissant), et les personnes (nom puis prénom).

De façon générale :

```
SELECT ... FROM ... WHERE ... ORDER BY e1 s1, ..., en sn;
```

Les résultats sont ordonnés lexicographiquement selon les expressions e_k, avec ordre naturel $\leq_k$ si $s_k =$ `ASC` et inverse $\geq_k$ si $s_k =$ `DESC`.

Pour les chaînes, l'ordre dépend de la *collation* (paramétrable en SQL) ; ici, on suppose l'ordre par défaut (ASCII/Unicode). La clause `ORDER BY` suit `WHERE` si elle est présente.

1.3 Opérateurs LIMIT et OFFSET

On ne s'intéresse souvent qu'à une partie des résultats (volume trop important ou affichage segmenté). Les clauses `LIMIT` k et `OFFSET` n permettent de restreindre la sortie : `LIMIT` renvoie les k premières lignes, `OFFSET` celles à partir de l'indice n (inclus). Elles se placent après `ORDER BY` (si présente), avec `OFFSET` après `LIMIT`.

Exemple

```
SELECT titre, annee FROM Film ORDER BY annee LIMIT 10;
```

titre	*annee*
La Naissance d'un empire	1929
La Belle et l'Empereur	1959
La Bataille de Marathon	1959
La Proie des vautours	1959
La Ballade du soldat	1959
Katia	1959
Aux frontières des Indes	1959
Le Dernier Train de Gun Hill	1959
Au risque de se perdre	1959
Ben-Hur	1959

Pour obtenir les cinq résultats suivants :

```
SELECT titre, annee FROM Film ORDER BY annee LIMIT 5 OFFSET 10;
```

Les clauses `LIMIT` et `OFFSET`, bien qu'au programme, ne sont pas standard et leur support varie selon les SGBD : PostgreSQL les utilise indépendamment, contrairement à MariaDB et SQLite.

2 Opérations ensemblistes

Dès l'origine, SQL se présente comme une implémentation pratique de l'algèbre relationnelle, où les relations (ensembles de n-uplets) sont manipulées via des opérateurs ensemblistes : union, intersection, différence et produit cartésien.

2.1 Union

L'opérateur `UNION` réalise l'union des résultats de deux requêtes R_1 et R_2. Celles-ci doivent avoir le même schéma (mêmes types et ordre des colonnes), le résultat étant une table.

Exemple

```
SELECT titre, duree FROM Film WHERE duree <= 71
UNION
SELECT titre, duree FROM Film WHERE duree >= 240;
```

Cette requête renvoie l'union des deux sous-requêtes (i) « le titre et la durée des films ayant une durée inférieure à 71 minutes » et (ii) « le titre et la durée des films ayant une durée supérieure à 240 minutes ».

titre	*duree*
1900	320
Cléopâtre	243
La Naissance d'un empire	70
Lucky Luke	71
Molière	244

Cette requête se comporte comme la requête

```
SELECT titre, duree FROM Film WHERE duree <= 71 OR duree >= 240;
```

`OR` ne suffit pas pour combiner des requêtes sur des tables différentes ; on utilise alors `UNION` :

```
SELECT pid FROM Realise
UNION
SELECT pres_id FROM Remporte;
```

Seul le domaine des colonnes importe, pas leur nom : ici, deux colonnes entières donc union valide. En revanche, des schémas différents rendent la requête invalide :

```
SELECT * FROM Realise
```

```
UNION
SELECT * FROM Remporte;
```

(car 2 colonnes vs 3).

`UNION` élimine les doublons ; `UNION ALL` les conserve :

```
SELECT titre, duree FROM Film WHERE duree <= 71
UNION ALL
SELECT titre, duree FROM Film WHERE duree <= 72;
```

titre	duree
La Naissance d'un empire	70
Lucky Luke	71
La Naissance d'un empire	70
Bambi 2	72
Lucky Luke	71
Salammbô	72
Le Livre de la jungle 2	72

Ici, l'utilisation de UNION ALL fait que certains résultats (ceux dont la durée est inférieure à 71) apparaissent en double, car leur durée est aussi inférieure à 72.

2.2 Intersection et différence

SQL fournit aussi les opérateurs `INTERSECT` et `EXCEPT`, applicables uniquement à des requêtes de même schéma.

```
SELECT pid FROM Realise
INTERSECT
SELECT pres_id FROM Remporte;
```

Cette requête renvoie les `pid` des personnes ayant à la fois réalisé un film et présenté une cérémonie des Oscars.

pid
16762

Nous pouvons aussi demander les identifiants des présentateurs qui ne sont pas des acteurs :

pres_id
3132
5348
17185
18448

2.3 Produit cartésien, sous-requêtes comme table

La dernière opération ensembliste est le produit cartésien, généralement inutile et dangereux hors cas particuliers :

```
SELECT * FROM T1, T2, ..., Tn;
```

Il génère une table combinant toutes les colonnes et toutes les combinaisons de lignes, de cardinal produit des cardinaux des T_i.

SQL permet aussi d'interroger une table issue d'une sous-requête :

```
SELECT titre, annee FROM (SELECT * FROM Film WHERE duree <= 72)
WHERE annee < 1970;
```

Interprétation :

1. construire la table des films de durée ≤ 72 ;
2. filtrer ceux d'année < 1970.

titre	annee
La Naissance d'un empire	1929
Salammbô	1960

Cette interprétation est intuitive : en pratique, la requête est évaluée sans table intermédiaire

```
SELECT titre, annee FROM Film WHERE duree <= 72 AND annee < 1970;
```

Les sous-requêtes permettent donc d'exprimer un produit cartésien.

Exemple

Par exemple, pour générer toutes les paires $(pays, genre)$:

```
SELECT pays, genre FROM (SELECT DISTINCT pays FROM Pays),
                        (SELECT DISTINCT genre FROM Genre);
```

La requête produit $54 \times 67 = 3168$ lignes (54 pays distincts, 67 genres).

Oublier `DISTINCT` entraîne $2571 \times 2601 = 6687171$ lignes, avec un coût potentiellement élevé, voire critique pour la mémoire.

En pratique, le produit cartésien est réservé à de petites tables (ex. combinaisons de créneaux). Il sert aussi à modéliser les jointures.

3 Jointure

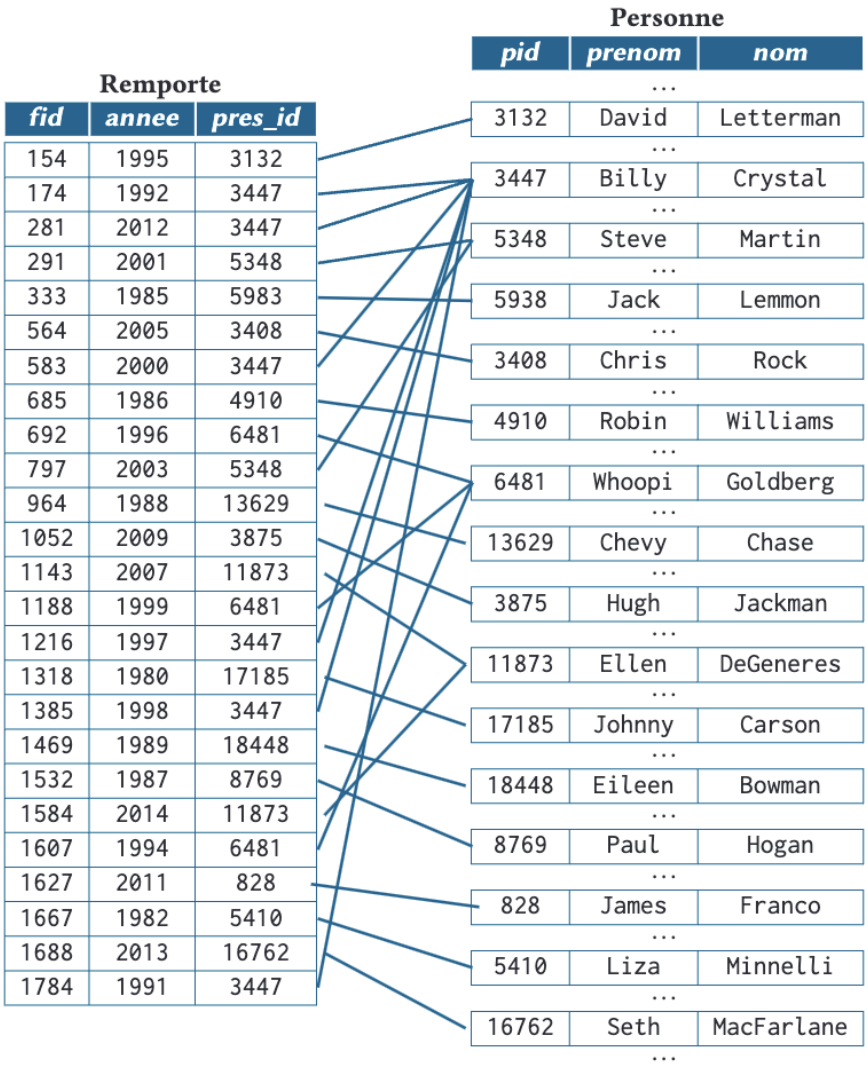
3.1 Jointure interne

Jusqu'ici, nous avons traité les entités du modèle EA et leurs associations, implémentées via clés primaires et étrangères. Nous exploitons maintenant cette modélisation pour répondre à des requêtes telles que : « pour chaque cérémonie des Oscars, renvoyer l'année, le nom et le prénom du présentateur ».

On utilise :

— `Personne` pour les noms et prénoms ;
— `Remporte` pour l'identification des présentateurs.

Les relations entre ces tables sont illustrées ci-dessous.



Dans cette figure, `Remporte` est prise entièrement, tandis que `Personne` est restreinte aux lignes vérifiant $pid = pres\_id$. Cette opération est une *jointure interne* sur cette condition :

```
SELECT annee, prenom, nom FROM Remporte JOIN Personne
ON pres_id = pid;
```

Les clauses `JOIN ... ON` se placent après `FROM` et avant `WHERE`, et se combinent avec `WHERE` et `ORDER BY`.

Exemple

```
SELECT annee, prenom, nom FROM Remporte JOIN Personne
ON pres_id = pid
WHERE annee <= 1999
ORDER BY annee DESC;
```

nous permet d'obtenir la table suivante :

annee	prenom	nom
1999	Whoopi	Goldberg
1998	Billy	Crystal
1997	Billy	Crystal
1996	Whoopi	Goldberg
1995	David	Letterman
1994	Whoopi	Goldberg
1992	Billy	Crystal
1991	Billy	Crystal
1989	Eileen	Bowman
1988	Chevy	Chase
1987	Paul	Hogan
1986	Robin	Williams
1985	Jack	Lemmon
1982	Liza	Minnelli
1980	Johnny	Carson

Les jointures se généralisent à plusieurs tables.

Par exemple, pour chaque film avant 1980, on joint `Film` avec `Joue` (sur `fid`), puis avec `Personne` (sur `pid`), avant de filtrer (`WHERE`) et projeter (`SELECT`).

Une ambiguité apparaît si les noms de colonnes coïncident :

```
SELECT titre, annee, nom, prenom, role
FROM Film JOIN Joue
     ON fid = fid
```

Elle se résout par la notation pointée `Table.colonne` :

```
SELECT titre, annee, nom, prenom, role
FROM Film JOIN Joue ON Film.fid = Joue.fid
     JOIN Personne ON Personne.pid = Joue.pid
WHERE annee <= 1980;
```

Les résultats de cette requête sont partiellement reproduits à la figure suivante.

titre	annee	nom	prenom	role
Pouic-Pouic	1963	De Funès	Louis	Léonard Monestier
Pouic-Pouic	1963	Darc	Mireille	Patricia Monestier
Pouic-Pouic	1963	Dumas	Roger	Paul Monestier
⋮				
Les Yeux de Laura Mars	1978	Dunaway	Faye	Laura Mars
Les Yeux de Laura Mars	1978	Jones	Tommy Lee	John Neville
Les Yeux de Laura Mars	1978	Dourif	Brad	Tommy Ludlow
⋮				
Trahison sur commande	1962	Palmer	Lilli	Marianna Möllendorf
Trahison sur commande	1962	Griffith	Hugh	Collins
Trahison sur commande	1962	Griffith	Hugh	dallas
⋮				
Cent Mille Dollars au soleil	1964	Ventura	Lino	Hervé Marec
Cent Mille Dollars au soleil	1964	Parisy	Andréa	Pepa
⋮				
La Grande Vadrouille	1966	De Funès	Louis	Stanislas Lefort
La Grande Vadrouille	1966	Brook	Claudio	Peter Cunningham
La Grande Vadrouille	1966	Dubois	Marie	Juliette
⋮				
Star Wars, épisode IV : Un nouvel espoir	1977	Hamill	Mark	Luke Skywalker
Star Wars, épisode IV : Un nouvel espoir	1977	Ford	Harrison	Han Solo
Star Wars, épisode IV : Un nouvel espoir	1977	Fisher	Carrie	princesse Leia Organa

On observe que la cardinalité $M:N$ de `Joue` induit des duplications : un film (resp. un acteur) apparaît autant de fois qu'il a d'acteurs (resp. de rôles).

Une jointure est un produit cartésien filtré :

```
SELECT titre, annee, nom, prenom, role
FROM Film, Joue, Personne
WHERE Film.fid = Joue.fid AND
      Joue.pid = Personne.pid AND
      annee <= 1980;
```

Conceptuellement, toutes les combinaisons sont générées puis filtrées, mais les SGBD utilisent des algorithmes efficaces évitant ce coût. La syntaxe `JOIN ... ON`, plus lisible, sépare jointure et filtrage.

La notation pointée, combinée aux alias (`AS`), simplifie l'écriture et lève les ambiguïtés :

```
SELECT F.fid, F.titre, F.annee, P.*, J.role
FROM Film AS F
JOIN Joue AS J ON F.fid = J.fid
JOIN Personne AS P ON P.pid = J.pid
WHERE F.annee <= 1980;
```

3.2 Jointure sans condition ou jointure produit

Un produit cartésien peut s'exprimer comme une jointure sans condition (réciproquement à une jointure = produit cartésien filtré) :

```
SELECT pays, genre FROM
(SELECT DISTINCT pays FROM Pays) JOIN
(SELECT DISTINCT genre FROM Genre);
```

L'absence de `ON` indique une jointure inconditionnelle. Pour plus de clarté, on utilise `CROSS JOIN` :

```
SELECT pays, genre FROM
(SELECT DISTINCT pays FROM Pays) CROSS JOIN
(SELECT DISTINCT genre FROM Genre);
```

3.3 Jointure externe

La jointure interne ne conserve que les paires vérifiant la condition :

```
SELECT titre, Remporte.annee
FROM Film JOIN Remporte ON Film.fid = Remporte.fid;
```

Elle ne renvoie que les films ayant remporté un Oscar. Pour obtenir tous les films, avec l'année ou `NULL` sinon, on utilise une *jointure externe gauche* :

```
SELECT titre, Remporte.annee
FROM Film LEFT OUTER JOIN Remporte ON Film.fid = Remporte.fid;
```

titre	*annee*
Trois Hommes et un couffin	null
Pouic-Pouic	null
Les Yeux de Laura Mars	null
Miss Peregrine et les enfants particuliers	null
Les Chevaliers du ciel	null
Cobra	null
Trahison sur commande	null
Joyeuses Pâques	null
Cent Mille Dollars au soleil	null
Les Choristes	null
⋮	
Forrest Gump	1995
Le Silence des agneaux	1992
Thor	null
Dinosaure	null
The Artist	2012
Gladiator	2001
Amadeus	1985
⋮	
Un poisson nommé Wanda	null
⋮	

La jointure gauche procède ainsi :

1. joindre toutes les paires (f, r) telles que $f.fid = r.fid$;
2. pour tout film sans correspondance, lui associer une ligne fictive de `Remporte` remplie de `NULL`.

Elle conserve donc toutes les lignes de la table de gauche (`Film`), même sans correspondant à droite. Ainsi, *Forrest Gump* est associé à 1995, tandis que *Les Choristes*, sans correspondance, est associé à `NULL`.

Le mot-clé `OUTER` est optionnel :

```
SELECT titre, Remporte.annee
FROM Film LEFT JOIN Remporte ON Film.fid = Remporte.fid;
```

Les jointures droite et externe complète existent aussi, mais sont hors programme.

4 Fonctions d'agrégation

On s'intéresse souvent non aux lignes elles-mêmes, mais à des statistiques (nombre, minimum, maximum, moyenne, somme). Ces requêtes consistent à sélectionner un ensemble puis à lui appliquer une fonction d'agrégation renvoyant une valeur unique.

Fonctions principales :

— **COUNT** : nombre de lignes ;
— **MIN** : minimum ;
— **MAX** : maximum ;
— **AVG** : moyenne ;
— **SUM** : somme.

Syntaxe générale :

```
SELECT f(e) FROM ...
WHERE ...
```

La clause `FROM` peut contenir des jointures, `WHERE` est optionnelle. Pour `COUNT`, e peut être une expression ou `*`.

Exemple

```
SELECT COUNT(*)
FROM Film JOIN Remporte ON Film.fid = Remporte.fid;
```

COUNT(*)
25

Le résultat est une table à une ligne et une colonne (nom arbitraire). Pour `COUNT`, la colonne choisie est indifférente : compter titres, années ou lignes (`COUNT(*)`) donne le même résultat.

On peut nommer la colonne avec `AS` :

```
SELECT COUNT(titre)
FROM Film JOIN Remporte ON Film.fid = Remporte.fid;

SELECT COUNT(Film.annee) AS nb_annees
FROM Film JOIN Remporte ON Film.fid = Remporte.fid;
```

Autres agrégats :

```
SELECT MIN(annee) FROM Film;
SELECT MAX(duree) FROM Film;
SELECT AVG(duree) FROM Film;
SELECT SUM(duree) FROM Film WHERE annee = 1995;
```

Interprétation en deux étapes :

1. sélectionner les valeurs ;
2. appliquer l'agrégat (les valeurs `NULL` sont ignorées).

Pour éliminer les doublons :

```
SELECT COUNT(DISTINCT genre) FROM Genre;
```

COUNT(DISTINCT genre)
67

Cette écriture est équivalente à la requête imbriquée :

```
SELECT COUNT(*)
FROM (SELECT DISTINCT genre FROM Genre);
```

La syntaxe `DISTINCT` s'applique aussi aux autres agrégats, mais n'est pertinente que pour `COUNT`, `AVG` et `SUM`, sans effet sur `MIN` et `MAX`.

4.1 Requêtes imbriquées et agrégations

Pour obtenir « le film le plus ancien » ou « le plus long » sans connaître la valeur extrême, on utilise une sous-requête scalaire (table à une ligne et une colonne) automatiquement convertie :

```
SELECT * FROM Film
WHERE annee = (SELECT MIN(annee) FROM Film);
```

fid	titre	annee	duree	titre_orig
150	La Naissance d'un empire	1929	70	Tide of Empire

La sous-requête `(SELECT MIN(annee) FROM Film)` renvoie une valeur scalaire (ici 1929), utilisée dans la comparaison. Bien que l'on puisse craindre un coût quadratique, elle est indépendante et donc évaluée une seule fois.

Une alternative équivalente utilise un produit cartésien avec une table à une case :

```
SELECT Film.* FROM Film,
     (SELECT MIN(annee) AS annee_min FROM Film)
WHERE Film.annee = annee_min;
```

On ajoute ainsi une constante à chaque ligne puis on filtre.

Les agrégats peuvent aussi apparaître dans `SELECT`.

Exemple

Par exemple, la proportion de films ayant un titre original :

```
SELECT
    (SELECT COUNT(titre_orig) FROM Film) /
    ((SELECT COUNT(*) FROM Film) * 1.0) AS prop_titre;
```

prop_titre
0.3849118942731278

La multiplication par 1.0 force une conversion en flottant ; sinon, la division entière donnerait 0.

Les sous-requêtes précédentes sont *décorrélées* (résultat unique, calculé une fois).

À l'inverse, une sous-requête *corrélée* dépend de la ligne courante :

```
SELECT * FROM Film AS F1
WHERE duree >= (SELECT AVG(duree)
                FROM Film WHERE annee = F1.annee);
```

Elle nécessite un alias (`AS`) pour distinguer les occurrences de `Film` et ne peut être précalculée, induisant un coût potentiellement quadratique. Bien que valide, ce type de requête peut devenir inefficace sur de grandes tables.

5 Requêtes de groupe

Il est souvent nécessaire d'appliquer des agrégats à des sous-ensembles. Par exemple, si on veut compter les films par année, on regroupe par année puis on applique `COUNT`.

SQL fournit pour cela la clause `GROUP BY`.

Clause GROUP BY

`GROUP BY` partitionne les lignes selon une *clé de groupe*. Ainsi :

```
SELECT annee, COUNT(*) FROM Film GROUP BY annee;
```

L'évaluation se fait en deux étapes :

1. regrouper les lignes selon l' 'expression donnée dans le GROUP BY

annee	↦	fid	titre	annee	duree	titre_orig
1929	↦	150	La Naissance d'un empire	1929	70	Tide of Empire
1959	↦	314	La Belle et l'Empereur	1959	94	null
		342	La Bataille de Marathon	1959	82	La battaglia di Maratona
		528	La Proie des vautours	1959	124	Never So Few
		741	La Ballade du soldat	1959	92	Ballada o soldate
		⋮				
2000	↦	26	Vertical Limit	2000	124	null
		54	Dinosaure	2000	79	Dinosaur
		93	Les Rivières pourpres	2000	106	null
		203	Harry, un ami qui vous veut du bien	2000	117	null
		⋮				

   Ce résultat intermédiaire *n'est pas relationnel*, puisqu'à chaque année est associé un ensemble de lignes.
2. L'agrégat `COUNT(*)` est ensuite appliqué à chaque groupe, comme à des tables distinctes ; on obtient alors une table :

annee	COUNT(*)
1929	1
1959	11
2000	31
⋮	

Syntaxe générale d'une requête groupante :

```
SELECT gi1, ..., gik, A1(a1), ..., Ak(am)
FROM ...
WHERE ...
GROUP BY g1, ..., gn
```

Évaluation :

1. construire les tables (`FROM`) ;
2. filtrer (`WHERE`) ;
3. regrouper selon les g_i ;
4. calculer, pour chaque groupe, les expressions de `SELECT`. Les g_{ij} proviennent des clés de groupe et les $A_i(a_i)$ sont des agrégats. Un `ORDER BY` peut suivre.

Exemple

```
SELECT annee, pays, COUNT(*), AVG(duree)
FROM Film JOIN Pays ON Film.fid = Pays.fid
WHERE annee >= 1980 AND annee <= 1983
GROUP BY annee, pays
ORDER BY annee;
```

annee	pays	COUNT(*)	AVG(duree)
1980	ALLEMAGNE	1	128
1980	ESPAGNE	1	97
1980	FRANCE	19	108.36842105263158
1980	ITALIE	3	100
1980	ROYAUME-UNI	3	116.66666666666667
1980	ÉTATS-UNIS	13	109.3076923076923
1981	ALLEMAGNE	1	120
1981	CANADA	2	93
1981	FRANCE	21	106.52380952380952
1981	ITALIE	2	99
1981	ROYAUME-UNI	3	119.33333333333333
1981	ÉTATS-UNIS	10	113
1982	ALLEMAGNE	2	103
1982	ESPAGNE	1	129
1982	FRANCE	19	102.21052631578948
1982	INDE	1	191
1982	MEXIQUE	1	129
1982	ROYAUME-UNI	4	126.5
1982	SUISSE	1	98
1982	ÉTATS-UNIS	14	105.42857142857143
1983	ALLEMAGNE	1	134
1983	CANADA	2	127.5
1983	FRANCE	22	107.0909090909091
1983	HONGRIE	1	145
1983	ITALIE	1	104
1983	JAPON	1	123
1983	NOUVELLE-ZÉLANDE	1	123
1983	PAYS-BAS	1	95
1983	POLOGNE	1	136
1983	ROYAUME-UNI	5	123
1983	TUNISIE	1	130
1983	YOUGOSLAVIE	1	100
1983	ÉTATS-UNIS	13	110.3076923076923

Les requêtes groupantes permettent souvent de remplacer des sous-requêtes corrélées coûteuses. Ainsi,

```
SELECT * FROM Film AS F1
WHERE duree >= (SELECT AVG(duree)
                FROM Film WHERE annee = F1.annee);
```

se reformule efficacement :

```
SELECT F1.* FROM Film AS F1
     JOIN (SELECT annee, AVG(duree) AS duree_moy
           FROM Film GROUP BY annee) AS F2
     ON F1.annee = F2.annee
WHERE F1.duree >= F2.duree_moy
```

On calcule d'abord, par `GROUP BY`, la durée moyenne par année (table `F2`), puis on joint avec `Film` pour comparer chaque film à la moyenne de son année.

5.1 Filtrage d'une requête groupante

Le filtrage après agrégation se fait via `HAVING`, car `WHERE` intervient avant le groupement. Ainsi, pour ne conserver que les groupes de taille ≥ 4 :

```
SELECT annee, pays, COUNT(*) as num_film, AVG(duree)
FROM Film JOIN Pays ON Film.fid = Pays.fid
WHERE annee >= 1980 AND annee <= 1983
GROUP BY annee, pays
HAVING num_film >= 4
ORDER BY annee;
```

et renvoie les résultats suivants :

annee	pays	num_film	AVG(duree)
1980	FRANCE	19	108.36842105263158
1980	ÉTATS-UNIS	13	109.3076923076923
1981	FRANCE	21	106.52380952380952
1981	ÉTATS-UNIS	10	113
1982	FRANCE	19	102.21052631578948
1982	ROYAUME-UNI	4	126.5
1982	ÉTATS-UNIS	14	105.42857142857143
1983	FRANCE	22	107.0909090909091
1983	ROYAUME-UNI	5	123
1983	ÉTATS-UNIS	13	110.3076923076923